INDUCTIVE BIASES AND METAKNOWLEDGE REPRESENTATIONS FOR SEARCH-BASED OPTIMIZATION

by

STEPHEN FRIESS

A thesis submitted to
The University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Sciences
The University of Birmingham
November 2022

Abstract

“*What I do not understand, I can still create.*“

H. Sayama [124]

The following work follows closely the aforementioned bonmot. Guided by questions such as: “How can evolutionary processes exhibit learning behavior and consolidate knowledge?´´, “What are cognitive models of problem-solving?´´ and “How can we harness these altogether as computational techniques?´´, we clarify within this work essentials required to implement them for metaheuristic search and optimization.

We therefore look into existing models of computational problem-solvers and compare these with existing methodology in literature. Particularly, we find that the meta-learning model, which frames problem-solving in terms of domain-specific inductive biases and the arbitration thereof through means of high-level abstractions resolves outstanding issues with methodology proposed within the literature. Noteworthy, it can be also related to ongoing research on algorithm selection and configuration frameworks. We therefore look in what it means to implement such a model by first identifying inductive biases in terms of algorithm components and modeling these with density estimation techniques. And secondly, propose methodology to process metadata generated by optimization algorithms in an automated manner through means of deep pattern recognition architectures for spatio-temporal feature extraction. At last we look into an exemplary shape optimization problem which allows us to gain insight into what it means to apply our methodology to application scenarios. We end our work with a discussion on future possible directions to explore and discuss the limitations of such frameworks for system deployment.

ACKNOWLEDGMENTS

First of all, I foremost would like to thank Prof. Dr. Xin Yao for giving me the chance to work on a PhD thesis at the intersection of Natural Computing and Artificial Intelligence within the ECOLE project. I found the study of computing paradigms based upon findings of learning, developmental and evolutionary biology to be highly inspiring and I could not have imagined to be anywhere else exposed to such a multidisciplinary approach towards problem-solving. These were certainly very unique experiences and are life-long lasting learnings to me. I further would also like to thank especially Prof. Dr. Peter Tiňo for providing additional supervision, as well as Dr. Stefan Menzel and Dr. Zhao Xu for exchange opportunities throughout my research work. I also would like to thank Giuseppe Serra, Sibghat Ullah, Jiawen Kong and Gan Ruan for making my time in ECOLE especially enjoyable. Further, from HRI-EU, Thiago Rios for being helpful, as well as Nivesh Dommaraju, Fabian Müller, Manuel Rudolph and Felix Lanfermann for being very enjoyable office companions. Likewise, I would like to thank Dr. Nabi Omidvar, Liangpeng Zhang, Natasha Nigar and Xunzhao Yu from the School of Computer Science in Birmingham for interesting academic exchange in our bi-weekly group meetings which I organized. On this note, I would also like to thank the NaCo group at LIACS in Leiden for interesting and partly also very entertaining moments. Last but not at least, I would like to deeply thank my close friends and family who accompanied me on my journey. Which as they know best has been a strenuous one for me. Thank you for being here for me during these difficult times.

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Nature-inspired Artificial Intelligence

The focus of our work can be considered to be lying at the intersection of the often synonymously taken fields of Natural Computing, Computational Intelligence and Soft Computing. Though arguably, the notion of Natural Computing might capture best the breadth of nature- and bio-inspired computing techniques of interest for our purposes. Specifically, we take a focus within our work in techniques developed for simulated evolution and learning.

Note, that it can be reasonably argued that for natural organisms, evolution and learning pose two fundamental modes of environmental adaption. While evolution is a process occurring on large time-scales which directly acts upon the genotype of a population of organisms, learning is in comparison a process mostly restricted to individual organisms, occurring on short time-scales and giving rise to phenotypic plasticity. In principle both modes of adaption should be considered to be occurring independent of each other. However, this does not restrict them in a having a feedback on each. Thus, making them capable of giving rise to complex emergent phenomena e.g. such as the Baldwin effect and cultural knowledge exchange [11, 12]. Though, the exact nature and mechanisms underlying this mode of adaption are still open to debate.

At this point, we do not want to further elaborate on the biological side of these processes, but instead highlight attempts in harnessing their advantages and studying them computationally.

1.1.1 Simulated Evolution

First considerations on the possibility of implementing computational approaches of simulated evolution may be traced back as early as to the first computer pioneers. Noteworthy, with Alan Turing proposing in 1948 explicitly genetic and evolutionary searches as possible ways to realize intelligent machinery with the capability to demonstrate "initiative" in problem-solving [144]. Likewise, also with John von Neumann taking a keen interest in the 1950s in realizing self-replicating programs which mimic a form of artificial evolution in the form of cellular automata [95, 135]. Following this up, the subsequent decades saw the establishment of most classical foundational algorithms for simulated evolution, which primarily take an interest in being harnessable for the purpose of optimization. Noteworthy, with Genetic Algorithms (GA) emerging through various studies through the likes of George Box, Lawrence Fogel and John Holland in the late 1950s and 1960s. Likewise, Evolution Strategies (ES) emerging in the 1960s through the studies of Ingo Rechenberg and Hans-Paul Schwefel, as well as Evolutionary Programming (EP) being proposed by Lawrence Fogel in the same time-frame [135]. While the subsequent decades saw mostly the independent advancement of these different strains of research, the modern synthesis under the denominator of Evolutionary Computation only emerged within the early 90s. In this contemporary framework, a usual evolutionary algorithm (EA) is understood as consisting out of variation operators $\mathcal{V}$ (crossover and mutation) and selection operators $\mathcal{S}$, which given an objective function $f(\mathbf{x})$ iteratively evolve a population of start solutions P_0, such that to retrieve substantially improved ones at termination. Notable modern advances more or less following this outline constitute the introduction of Particle Swarm Optimization (PSO) [77], algorithms for multi-objective optimization such as NSGA-II

[39], as well as the model-based Estimation of Distribution Algorithms (EDAs) [87] and Covariance Matrix Adaption Evolution Strategies (CMA-ES) [62, 60].

1.1.2 Simulated Learning

The study of simulated learning approaches takes likewise its origin in the midst of the 20th century and can be considered to be strongly intertwined with studies on simulated evolution. With McCulloch and Pitts proposing first in 1943 a simple computational graph model to illustrate how in principle interconnected neurons in the brain can calculate simple logical functions [8]. Somewhat reflecting the present school of thought at the time, that in order for sophisticated forms of computation to arise it only requires large enough and seemingly initially randomly organized systems [144]. In the same spirit, John von Neumann meticulously began to point out the differences between deterministically operating digital computers and the forms of statistical computation occurring in brain [153] before his demise in 1957. Research continued with Frank Rosenblatt implementing the first trainable perceptron in 1962 [48]. Problems with training multi-layered perceptrons could be subsequently solved through the introduction of the backpropagation algorithm by Paul Werbos (1974). The early 1980s subsequently saw the introduction parameter-sharing for image recognition tasks by Fukushima [54] in order to mimic the hierarchical nature in which features are represented in the visual cortex [69]. This technique was later refined by Yann Le Cunn [88] in 1995 to the present day Convolutional Neural Networks (CNNs). Further, notable advancements made in the 90s concern also the introduction of Long Short-Term Memory networks (LSTMs) by Hochreiter and Schmidthuber [68] for processing sequential data. Advances since the turn of the millennium are arguably enumerate. Though, a notable nature-inspired one is for instance the Capsule Neural Network (CapsNet) [122], which was specifically designed to mimic the processing in cortical columns.

1.1.3 Advantages, Differences & Synergies

In the following, we want to briefly elaborate on the advantages of methods for simulated evolution and learning in comparison to more traditional approaches for optimization and statistical modeling. We further also briefly highlight how they are different from their biological counterparts, as well as elaborate on synergies which arise by harnessing these two modes of adaption in computational applications.

The advantage of evolutionary approaches for optimization in comparison to traditional gradient-based methods has been well-established. It can be best described as being particularly suited for problems which are NP-hard and have large and complex search spaces. Because their stochastic nature helps them in escaping local optima, it further enables them in finding unconventional and surprising solutions [89]. The complete convergence of an evolutionary algorithm to the global optimum can be theoretically guaranteed, as long as every point of the search space is accessible with non-vanishing probability and the best found solution is never removed from the population [120]. Noteworthy, the main difference between artificially simulated evolution for optimization and natural evolution is, that the latter is not explicitly an optimizing process. As exogenous fitness functions do not explicitly exist. Thus, in principle natural evolution can be subject to effects which from an perspective of optimization can be considered as being detrimental. Further, simplifications introduced in most evolutionary algorithms highly idealize actual natural evolution (e.g. [99]). While to some degree, this is necessary to ensure computational efficiency, it also introduces anatural behavior. For instance, explicit modeling of selection operators incorrectly portrays natural selection as an acting and functionally existing agent [92], rather than a mere emergent effect. One may therefore legitimately argue whether or not the explorative behavior exhibited by Bayesian optimization might come closer to how natural evolution actually achieves optimal results. Notably, the field of Artificial Life deals more explicitly with attempts to explicitly emulate natural evolution with a lesser focus on optimization.

Simulated learning approaches mimicking neural computations in the brain have especially become popular within the recent decade as they have been proven to be highly flexible predictive methods. In fact, it has been theoretically proven that the multilayer perceptron possesses the property of being an universal function approximator [35]. Thus, given enough data and a suitable layerwise configuration, neural networks can be trained to approximate any desired continuous function on a bounded interval. Therefore, they established themselves for a variety of domains ranging from speech recognition, computer vision to natural language processing. A particular advantage of them is also that these can make tedious manual pre-processing of training data and feature engineering superfluous as by mimicking how biological cognitive systems work, they can automatically learn the features which are most important for a given task. This principle has been especially exploited through parameter sharing introduced in computer vision [54, 88], which explicitly mimics the feature learning and processing capabilities of the visual cortex [69, 18]. Notably, arguments brought forward from the literature advocate that from a biological perspective, not only features, but a majority of cognitive processing capabilities and learned behaviors could be explained as arising due to the long-term effects of goal or objective-oriented reward maximization [133]. In regard to the aforementioned evolutionary computation methods, neural computing methods can be considered to be likewise highly idealized representations of their actual biological counterparts. For instance, neglecting details of synaptic transmission for computational efficiency. More strikingly though, it has been shown that single cortical neurons are capable of exhibiting the computational capabilities which have been considered to be reserved only to deep architectures [13].

The synergies between learning and evolutionary computation are various and quite natural. As in principle, evolutionary optimization methods can benefit from the incorporation of flexible predictive methods. This has been exploited in techniques for instance developed for surrogate modeling [75] or algorithm and heuristic selection [24, 103, 78]. On the flip-side, the ability of evolutionary approaches to be efficient in search through large and complex search spaces makes them interesting for neural network architecture optimization, as well as suitable for weight optimization when dealing with noisy loss functions. Surprisingly, while the latter methods for neural architecture optimization have gained significant traction within the recent years [38, 139, 44, 100, 138], advancements in the reverse direction have been comparably sparse. Even though, the complex and enumerate data produced by evolutionary search and optimization algorithms can in principle be considered to be complex and intractable for manual analysis by the practitioner.

Noteworthy, on a related note, methodology utilizing the third mode of adaption, arising through the coupling of learning and evolutionary mechanisms have been likewise mostly absent in light of current research in optimization. Even though, analogue models have been historically studied [67, 106], as well as variations of evolutionary algorithms in the form of cultural [117] and memetic algorithms [98, 109, 104] have been proposed in the literature. Though, improvements and reasonable refinements of models from cultural evolution theory, particularly under the more explicit consideration of aspects concerning the usage of learned and shared representations [20] with a modern machine learning lens to enable knowledge transfer and cooperation in distributed systems, could likely lead to further advancements in the future.

1.2 Contemporary Problems in Heuristic Optimization

Having introduced different natural computing methods for learning and optimization, we will elaborate in the following on how contemporary problems arose within the last decades concerning the latter domain. In this spirit, the contributions of our thesis aim to particularly address these issues.

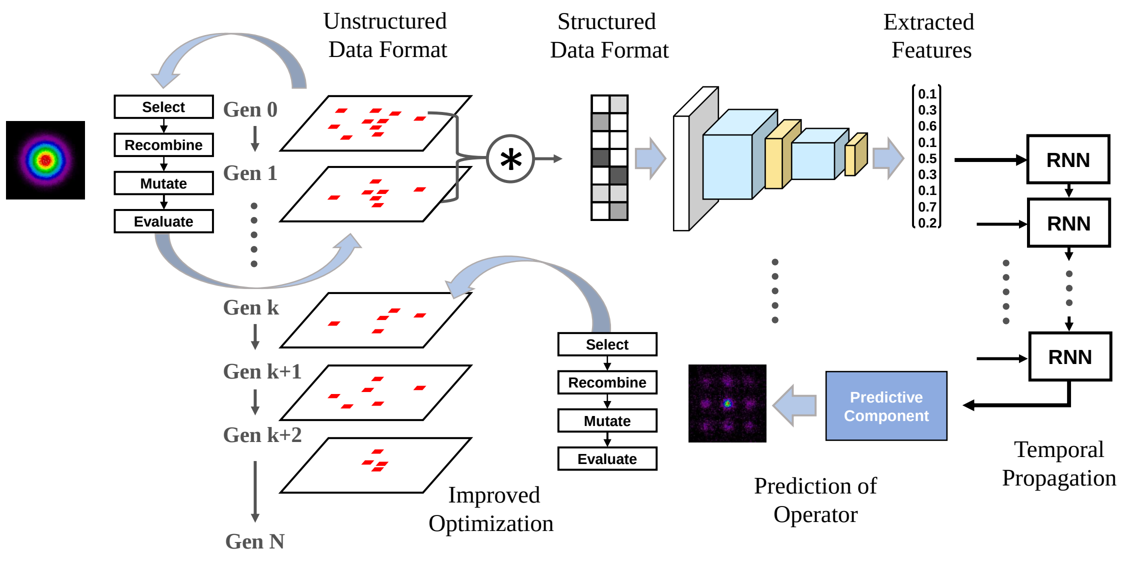

Figure 1.1: Diagram of the framework and the individual functional components it consists of that we construct within the investigations of this thesis. In principle, we aim to predict problem-tailored operators through application of predictive methods which are fed from structured inputs generated from unstructured metadata characteristic of problem-dependent algorithm behavior. A more in-depth description linking each component to the contributions of each chapter of this thesis is provided for reference within Appendix A.

While the different fields of research on optimization techniques merged within the early 90s, the early optimism for finding a high-performing universal problem-solving algorithm was quickly halted. Particularly, with the publication of the no free lunch theorems, which state that effectively all algorithms perform the same on the set of all problems [158]. Nevertheless, this chilling result did not hindered the development of many of today's successful optimization algorithms (cf. Sec. 1.1.1). Particularly this can be attributed to the fact that the no-free-lunch theorems do not provide explicit guidance on how to define characteristics and compare different optimization problems. Thus, quite naturally certain problems with common characteristics may favor certain kind of algorithmic approaches. While arguably, this proved fruitful for the establishment of many modern optimization algorithms, it allows one to commit to the logical fallacy of defining arbitrary niches on which likewise arbitrary algorithms can perform the best. What might be seen as a direct unfortunate consequence of this, is the rise of a plethora of optimization algorithms defined for specific optimization problems, taking loose inspiration of a variety metaphors. Valid criticism directed towards these algorithms is that a majority of them are either in best case minor variations, or simply just relabellings of existing algorithms [5, 150, 151, 25, 137]. Notably, with the largest repository summarizing them counting up to over two-hundred algorithms to this date [26].

Motivated by these contemporary issues within the metaheuristics community, one may legitimately ask whether or not one could move towards more general purpose algorithm frameworks that learn from the problems they solve and can be flexibly adapted for new problem domains of interest. Thus, strongly recalling the high popularity end-to-end learning architectures gained within recent years in machine learning research. However, while a variety of methods have been proposed in the literature within recent years, which in principle learn and accumulate knowledge, they unfortunately fail to recall more first-principled notions. Our work is therefore dedicated to address a dire need to clarify this issue by directly proposing methodology (cf. Fig. 1.1) and clarifying the foundations to construct such rigorously grounded frameworks.

1.3 Contents of this Thesis

As a brief foreword, we acknowledge that this thesis is written from an outside perspective of the fields of evolutionary and neural computation due to the physics background of the author. However, this gives this work the key advantage of not being overly constrained by existing notions and methodology within the literature. Practically speaking, we thus focus on combining different algorithms which we interpret as formalizing variational and approximative capabilities of natural adaptive processes. A secondary focus therefore throughout this work is also on one side on understanding what kind of underlying capabilities natural adaptive processes offer for mathematization, and on the flip-side looking at how existing algorithms offer viable formalizations that can be utilized, improved and combined.

1.3.1 Contributions and Structure

To address the outstanding issues within the literature, we particularly claim to make from Ch. 2 to 7 within our thesis the following contributions:

- **Histogram- and Density-based Modeling of Search Operators:**

  To model problem-dependent domain knowledge we propose within the following up Ch. 3, the usage of methodology for histogram construction and density estimation. Particularly, we find that this approach can be well justified for unary variation operators in the scenario of low-dimensional search spaces and problems with pronounced structures superimposed on them.

- **Graph-based Representation of Activity within Search Spaces:**

  To represent activity within continuous search spaces we need ways of converting the highly unstructured metadata generated by population-based optimization algorithms into a structured data format. We therefore introduce in Ch. 4 a pipeline to obtain structured data formats and a novel graph-based data format that can be flexibly used to this regard.

- **Spatial and Time-dependent Processing of Metadata:**

  The arbitration between different models of domain-dependent knowledge essentially requires ways of forming high-level metaknowledge representations. We therefore propose additionally within Ch. 4 and 5 specialized graph neural network architectures that can learn domain-dependent spatio-temporal characteristics from structured data formats representing activity within the search space.

- **Comprehensive Overview and Feasibility Evaluations:**

  We provide further a comprehensive overview of existing knowledge transfer techniques in the literature in Ch. 2, as well as provide exhaustive discussions concerning feasibility and limitations for system deployment additionally over Ch. 6 and 7.

Noteworthy, within the spirit of Natural Computing, as we have elaborated previously a particular emphasis within our thesis lies on approaching these contributions with a strong perspective from learning and evolutionary biology. Further, due to existing literature only insufficiently clarifying the potential theoretical limits of constructing learning optimization systems, we motivate our approaches with theoretical considerations from learning and optimization theory.

1.3.2 Relevant Publications to this Thesis

To conclude this introductory chapter, we acknowledge our published papers upon which this work is mainly based upon. While we published in total 6 conference papers in which we develop and mature our ideas, central to this thesis are the followings 4 works corresponding closely to Ch. 3 to 6:

Peer-Reviewed Conference Papers

- S. Friess, P. Tiňo, S. Menzel, Z. Xu, B. Sendhoff and X. Yao, "**Spatio-Temporal Activity Recognition for Evolutionary Search Behavior Prediction**," 2022 International Joint Conference on Neural Networks (IJCNN), 2022, pp. 1-8
- S. Friess, P. Tiňo, S. Menzel, B. Sendhoff and X. Yao, "**Predicting CMA-ES Operators as Inductive Biases for Shape Optimization Problems**," 2021 IEEE Symposium Series on Computational Intelligence (SSCI), 2021, pp. 1-7
- S. Friess, P. Tiňo, Z. Xu, S. Menzel, B. Sendhoff and X. Yao, "**Artificial Neural Networks as Feature Extractors in Continuous Evolutionary Optimization**," 2021 International Joint Conference on Neural Networks (IJCNN), 2021, pp. 1-9

- S. Friess, P. Tiňo, S. Menzel, B. Sendhoff and X. Yao, "**Representing Experience in Continuous Evolutionary Optimisation through Problem-tailored Search Operators**," 2020 IEEE Congress on Evolutionary Computation (CEC), 2020, pp. 1-7

Parts of this thesis corresponding identically to Chapters 3, 4 and 6 have also been been published within ECOLE deliverables D 2.1 and D 3.5:

- J. Kong, S. Ullah, S. Friess, G. Ruan, W. Kowalczyk, T. Bäck, **"D 2.1 Experience-based high dimensional & big data assisted optimisation"**, https://github.com/ECOLE-ITN/Deliverables/blob/main/D2.1 Experience-based high dimensional & big data assisted optimisation.pdf, 2022
- S. Friess, G. Ruan, G. Serra, L.L. Minku, Z. Xu, X. Yao, **"D 3.5 Integrated software environment and manual"**, https://github.com/ECOLE-ITN/Deliverables/blob/main/D3.5 Integrated software environment and manual.pdf, 2022

Note that we do not cover within this thesis topics concerning adaption mechanisms, transfer from low to high-dimensional problems and benchmark function design, which we also partly discussed within previous work.

CHAPTER 2

LITERATURE REVIEW

2.1 Computational Models of Cross-Problem Solving

2.1.1 Reinforcement Learning

An obvious computational model which lies at hand when discussing the construction of algorithms which accumulate experience over their lifetime is the one of reinforcement learning. From a practical point of view, this would require the implementation of finite Markov decision processes through an agent-environment interface [140]. Where the algorithm would be the agent a, and the environment e would communicate states S_t and rewards R_t to the agent, upon which the agent chooses based upon a policy π an action A_t which interacts with the environment in return, such that to maximize the value function V, which represents a sum of expected rewards. However, while rewards have an intuitive interpretation within metaheuristic and evolutionary optimization, the definition of actions A and states S is more problematic. Existing reinforcement and choice function approaches within combinatorial optimization are mostly based upon highly simplified models which neglect any notion of "state" at all, and simply arbitrate between different operators based upon which one generated better solutions in the past [24]. While "actions" can in principle be defined in terms of choosing between operators, defining a more first-principled approach approach is more problematic. Especially when considering con-

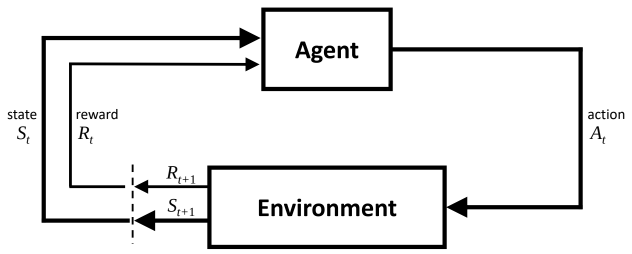

Figure 2.1: Agent-environment interaction modeled in terms of a Markov decision process within reinforcement learning. Diagrammatic illustration based upon Sutton & Barto [140].

tinuous optimization problems where in principle an infinite numbers of actions could be chosen.

2.1.2 Meta-Learning

A much more simplified and applicable model may instead be formulated through the concept of meta-learning. In principle, it allows one to significantly simplify the previously elaborated agent-environment model by means of framing it as a problem of learning inductive biases which are shared for similar problems and situations and are recalled based upon high-level representations or metaknowledge of past problems and situations. Meta-learning can in principle encompass different scales of learning, as for biological systems, the challenges posed by the environment in pursuit of specific goals also mostly emerge in different scopes and on different time scales [154]. These can be further considered to be nested in each other, such that the learning of the structure of a specific task can be found at the lowest level, while the learning of group structures occurs in-between, and the learning of highly invariant priors at the most upper levels (cf. Fig. 2.2). In terms of metaheuristic and evolutionary optimization, a reasonable bridge from machine learning research to the algorithm selection problem has been established [136]. One may reasonably argue that the algorithm selection problem might be considered to just be an extreme boundary case of an agent a which conducts a form of reinforcement learning where state

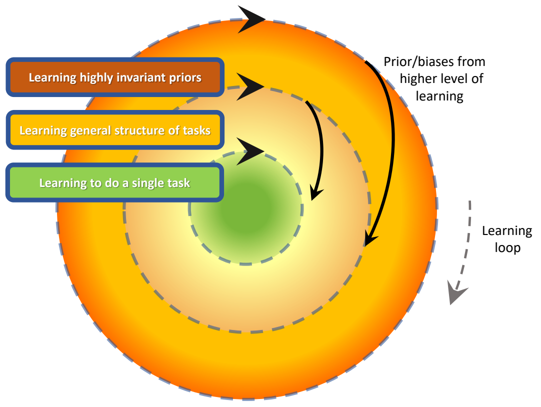
Learning highly invariant priors
Learning general structure of tasks
Learning to do a single task
Prior/biases from higher level of learning
Learning loop

Figure 2.2: Diagrammatic based upon Wang (2021) [154] of different scales of learning involved within meta-learning.

S and action A are only chosen during the early run-time of an algorithm and are not further redetermined later on.

Arguably, in the context of our work, the framework of meta-learning introduces useful simplifications and vocabulary. We will therefore reside mostly within subsequent chapters on further investigating it. Particularly, we will see that it is also well supported by learning and optimization theory and can give an interesting view on the evolutionary biological foundations and inspirations of evolutionary algorithms. However, in the following we briefly go a bit more into detail of further related approaches and also briefly elaborate on their short-comings.

2.2 Further Approaches in the Literature

2.2.1 Transfer Learning and Optimization

Transfer Learning

The strong appeal which the realization of learning-based optimization systems has makes at first glance the usage of methods from transfer learning [112, 156] very desirable. In

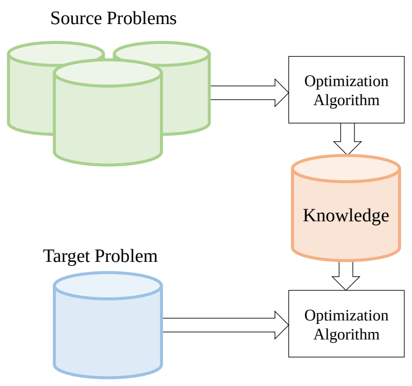

Figure 2.3: Archetypical illustration of the usual transfer learning pipeline. From a single or multiple source problems, knowledge is selectively formed from an algorithm such that help to improve its' performance on a new target problem class. Note, that due to the underlying multiple-sources-to-one-target architecture, transfer learning methods usually do not form models that exhibit cross-problem generalization capability.

principle, this bridge has been established early on within the literature through the framework of Genetic Transfer Learning (GTL) [81] which performs a form of simple transfer learning by means of using the final solutions obtained after an optimization to initialize the start population on a new optimization problem. Notable, more extensive works from Jiang et al. (2019) [72] could demonstrate the viability of the transfer component analysis technique [111] as well as forms of manifold learning [73] for scenarios of dynamic multi-objective optimization. However, otherwise the usage of more sophisticated transfer learning methods has been comparably sparse in the literature. This may be attributed to the fact that many proposed methods within machine learning research are highly application- and algorithm-specific and can be only applied with a very limited scope to optimization.

Transfer Optimization

Considering these limitations of transfer learning methodology for the application to optimization, Gupta et al. (2017) [57] have introduced the notion of transfer optimization instead. Works within this field consider for instance the use of linear rank transformations [46] to transfer solutions between different multi-objective optimization problems, or modeling solution populations from prior solved problems explicitly through mixture models [36, 163], from which solutions are sampled based upon their similarity to the current solution distribution.

2.2.2 Estimation of Distribution Algorithms

Related to the scope of this thesis are also estimation of distribution algorithms (EDAs) [87, 135]. While like evolutionary algorithms they are also based upon a population-based framework, a core distinguishing feature is that they explicitly keep track of generated high-performing solutions within each iteration. Thus, these are subsequently used to build a statistical model from which new solutions can be generated for the successor generation again. The estimation of distribution algorithm is by this procedure thus capable of learning the explicit structure of optima for a given optimization problem.

For instance, consider in the following a simple EDA [135] for the five-dimensional OneMax problem with objective function $f : \{0,1\}^5 \rightarrow \mathbb{R}$ such that $f(\mathbf{x}) = \sum_{i=1}^{5} |x_i|$ with the global maximum being at $\mathbf{x}^* = [1,1,1,1,1]^T$. Central to our exemplary EDA is a probability vector initialized as $\mathbf{p} = [0.5, 0.5, 0.5, 0.5, 0.5]^T$. To generate a new solution x from it, we generate for each component x_i a random number $r_i \in [0,1]$. If we find that $r_i \leq p_i$, we set $x_i = 1$, else we set $x_i = 0$. We generate now at each generation M new solutions and select the N highest ranking ones (i.e. $N < M$), from which we recalculate the probability vector by averaging, such that $\mathbf{p} = 1/N \sum_{i=1}^{N} \mathbf{x}_i$. Note, it is clear that within this given example that after successive generations the EDA will generate solutions ever closer to the global maximum of the OneMax problem, or any further similarly specified maximization problem. This capability of learning the structure of optima allows EDAs also to be used for the purposes for transfer learning, by means of reinitializing them with the previously obtained model on new problems that can be suspected to possess the same or similar structure of optima.

However, note that EDAs generally do not incorporate knowledge about more abstract and implicit properties of the fitness landscapes into their operators and may to this regard also exhibit only very limited domain-level generalization capability, i.e. they cannot to be trained on problems. Their internal mechanisms may also make them largely different to algorithms based upon the evolutionary framework. Though, noteworthy common implementations may share a close resemblance to the popular Covariance Matrix Adaption Evolution Strategies (CMA-ES). We discuss more explicitly the difference between EDAs and the CMA-ES within the latter Sec. 6.1.3.

2.2.3 Multitasking, Memetic and Knowledge-assisted Optimization

Further noteworthy approaches in the literature which are putting a focus on knowledge transfer within optimization are for instance techniques developed for multi-tasking [141, 108, 10, 47], memetic optimization, surrogate modeling [75] and more generally knowledge incorporation [74]. We will briefly elaborate on each in the following.

The notion of multitasking optimization aims at enabling the concurrent optimization of multiple problems at once. Multitasking Bayesian optimization [141] for instance concerns explicitly the sharing of information between in parallel constructed meta-models. Evolutionary multitasking on the other side deals more explicitly with the sharing of solutions between problems which are optimized in parallel [108, 10, 47], somewhat in analogy to the aforementioned research on transfer optimization [57].

Memetic optimization, or speaking more broadly memetic computation [104] concerns methodology which takes motivation from the cultural theory of Richard Dawkins which considers culture to be discretized into sharable and mutable bricks of knowledge or ideas, the so called 'memes'. In memetic computation these can be interpreted in terms of sharable operators and internal algorithm mechanisms. While memetic computation may not necessarily be focused upon cross-problem learning, the wide scope of it does not restrict it. Noteworthy, also hyper-heuristics for combinatorial optimization, which employ simple forms of reinforcement and online learning, are considered to lie within the wider scope of memetic computation techniques.

Within expensive optimization scenarios, surrogate- or meta-model-assisted optimization concerns the construction of a sufficiently informative model of a problem online during optimization, such that computation costs for explicitly evaluating the objective function can be saved. This approach can in principle not be considered to be knowledge transfer per-se at is only considering problems individually, however notably within the literature approaches considering cross-problem learning have been proposed [101].

A variety of approaches that we have discussed have also been previously summarized under the denominator of knowledge incorporation [74]. Methods falling under this umbrella for instance concern the aforementioned estimation of distribution algorithms, surrogate and meta-modelling techniques, but also cross-generational knowledge repositories, early cultural approaches similar to memetic computing techniques, approaches incorporating forms of Baldwinian learning, local searches and further interactive and preference-aware mechanisms. Noteworthy, within the scope of their review they had also included an implementation of the case-based genetic algorithm (CIGAR) by Louis & McDonnell (2004) [93] that may be seen as an early precursor of a transfer optimization technique and exhibits cross-problem and incremental learning capability, which we also later refer to again within this thesis in the context of problem similarity measures and long-term operability in Sec. 3.5.3 and 7.2.3.

2.3 Chapter Summary

Summarizing the state of the literature, we find that a great variety of different methodology has been developed, which in some scenarios may not necessarily concern cross-problem learning per-se, but also the re-use of knowledge acquired within an optimization on a single problem. The use of first-principled approaches however is notably lacking and therefore available methodology may often be based rather upon useful tricks and rule-of-thumbs.

For example, the use of techniques from transfer learning can be mostly considered to be in-adequate for heuristic optimization algorithms, due to them mostly being designed with a focus on predictive methods. Thus, within the literature alternative methods were developed based upon rather practical techniques to transfer found solutions between problems. However, due to a lack of understanding the quantification of problem similarity if not known a priori, it may make some techniques prone to as to what has been dubbed in the literature as 'negative transfer' [46]. Note, that similarly within memetic techniques, the issue exists of adequate characterization of optimization problems, such that algorithm components can be chosen accordingly.

At last, even though developed techniques we reviewed within the literature frequently concern evolutionary methods which mimic the variational mechanisms exhibited by natural evolution, they do give this notion very little further consideration. Thus, raising the question on what is knowledge in these in the first place. Notably, these inherently rely upon sampling random variates from symmetric statistical distributions. However, the usefulness of such distributions can be considered to be violated, when only restricted types of problems are being visited. Thus, the use of asymmetric and non-standard distributions reflecting the encountered problem structures should should inevitably lead to performance improvements.

2.3.1 Open Questions

In conclusion, from reviewing the available literature, we want to clarify within our thesis the following questions:

- **What are first-principled approaches of problem-solving and how can they be efficiently harnessed?**
  Note, that from our literature review we have already identified that reinforcement and meta-learning provide first-principled notions of problem-solving. And particularly also, that the latter offers useful simplifications. However, the question remains open what this means from an implementation perspective within optimization.
- **What is reusable knowledge within the variational mechanisms of natural evolution and how could it be used for algorithm design?**
  Essentially, considering the aforementioned meta-learning model, this is the question on what potential forms of inductive biases exist within natural evolution. We find that computational biology studies provide insights into this and thus could be harnessed as guiding principles for future algorithm design.
- **How can we characterize optimization problems such that we can easily re-identify them and use suitable previously learned knowledge?**
  Again, considering the aforementioned meta-learning model, this calls for methodology for processing metadata and form high-level abstractions from them. Particularly, we will see that for population-based optimization algorithms methodology from algorithm behavior studies can be harnessed to this regard.
- **What is the scope for application problems and beyond?**
  At last, the naturally occurring question is what it means to harness learning optimization systems for application problems. For this reason, we study a scenario which relates to shape optimization and from it draw conclusions on feasibility con-

straints. Further, we also discuss in this context limits to generalization capability as well as to long-term operability.

We will clarify these questions within the following up chapters, which in conclusion, will give us a cohesive perspective on how to design learning optimization systems and algorithms.

CHAPTER 3

INDUCTIVE BIASES IN SEARCH-BASED OPTIMIZATION

3.1 Bias as a Prerequisite for Generalization

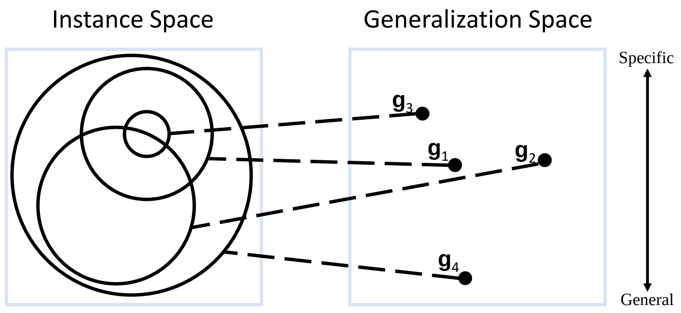

Figure 3.1: Illustration of the relationship between instance and generalization space as arising within the generalization problem [102]. Generalizations covering larger varieties of instances are considered to be more 'general' while generalizations restricted to smaller varieties are naturally considered to be more 'specific'.

The concept of inductive bias first emerged from the study of the generalization problem [102] . Which can be formulated in an adapted form as follows: Given 1) a language of instances, 2) a language to represent knowledge through generalizations (or synonymously hypotheses), 3) a matching predicate for matching generalizations to instances and 4) a set of training instances, the goal of the generalization problem is to determine generalizations such that they are consistent with predicting the properties of the training instances. A key argument on the analysis of the generalization problem is, that in order for any kind of learning algorithm to perform an inductive leap such that it can achieve high performance on new instances from the same instance domain, the algorithm needs to essentially form a bias in the form of a domain-specific knowledge representation, which is preferred to a default knowledge representation due to its effectiveness. Removing any biases completely from a learning systems in forming a generalization is futile, as in principle such a system would simply degenerate into its default behavior. Thus, it would not be able to exhibit generalization capability.

In principle, one can identify two different sources of biases in a learning systems: 1) The incapability of the generalization language to be adapted to all possible classes of instances and 2) a bias in the generalization procedure that searches through the space of knowledge representations. In this sense, an unbiased generalization language would impose no restrictions on forming a knowledge representation over arbitrary subsets of the instance space, and an unbiased generalization procedure would consider all possible knowledge representation formed from the training set when encountering new instances as equally valid. Considering a completely unbiased learning system, it becomes very clear that the only way such that all possible generalizations hold for true for a given instance, is when it is already contained in the training set. Thus, a completely unbiased generalization procedure would always lead to the learning algorithm to degenerate into its default behavior, which in the original study of the generalization problem by Mitchell (1980) [102] would mean that it completely overfits on the training data.

However, the necessity for learning biases for the inductive leap is in principle not something undesirable and it can be argued [102] for possible useful biases that can be explicitly included to make the formation of generalizable knowledge representations more efficient in learning systems. For instance, popular built-in inductive biases of many modern architectures of learning systems in the generalization language are various forms of parameter sharing through e.g. convolutional filters and gating mechanisms [88, 68], that can be particularly attributed to be a key success factor for their current popularity in processing tensor and sequential data. Likewise, for optimization algorithms inductive biases are posed for instance by diversity maintenance schemes and self-adaption mechanisms for multi-objective [39] and convex optimization problems [62]. For the generalization procedure itself, the bias can be formed through a beneficial parameter initialization or configuration of the procedure itself, which enables the training or optimization to converge faster than with a default setting.

3.2 Evolutionary Search Operators as Inductive Biases

Concluding the importance of inductive biases for generalizing domain knowledge gained from learning tasks, we need to identify in the following ways of forming them within metaheuristic optimization algorithms. The framework of evolutionary optimization is particularly charming to this regard, as it reduces the optimization algorithm to two essential key components: variation and selection operators. Following up their original inspiration from evolutionary biology, the variation processes consist out of mutation and crossover. However, with key understanding being that mutation is the the prime source of variation [92]. While the utility of crossover processes can be supposedly understood as aiding the rapid adaption of a population to an environment in the scenario of small population sizes.

Notably, many modern algorithms implement mutation operators by sampling variates from internal models based upon symmetric distributions (cf. right panel of Fig. 3.2) which could be considered to unsuitable when only specific subsets of optimization problems are considered to be solved by an optimization algorithm. Thus, biasing these distributions in a problem-dependent manner should pose one way of making a meta-

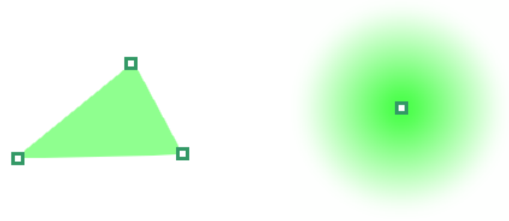

Figure 3.2: Spatial probability distributions in the form of a simplex as used by a generalized arithmetic crossover operator (left) and isotropic normal distribution for a mutation operator (right) used to generate new solutions from old ones (green squares). Surfaces of equal shades of green represent same probability density.

heuristic search capable of retaining knowledge. We can find particular guidance on how inductive biases can be formed from basic research in computational biology and adaptive systems research. Essentially, evolvability studies from Kounios et al. (2016) [83] have explored synergies between learning theory and evolutionary biology. A key result of their simulation studies on gene-regulatory networks is, that when repeatedly exposed to rugged fitness landscapes, evolutionary variation processes can be re-framed as learning processes, by accumulating knowledge about short-term beneficial mutations. However, it is notable to point out that these developmental biases may not only foster adaption to specific environments [146], but exhibit a broader learning capability such that they are able to generalize adaptive trends to future unseen environments. Thus, in light of these computational biology studies, one can consider unary mutation operators within evolutionary optimization to be a key algorithm component to form the model space of possible algorithms. In fact, their importance has been solidified within convergence theorems from Rudolph (1998) [120], as only through them parts of the search space can be accessed which would otherwise remain unreachable. Essentially, acting as novelty generators.

The role of crossover operators is in principle less dramatic. From an evolutionary biology perspective, genetic crossover is not a necessity, but as previously mentioned can be considered to be merely aiding rapid adaption in the scenario of small population sizes [92]. In principle, they work by generating new genomic variants from mixing previously high-performing found ones. From a mathematical point this can be viewed as an interpolation operation. An example of a probability distribution for sampling new solutions through an arithmetic crossover of three solutions is illustrated in Fig. 3.2. Note, that due to it being a technical abstraction, in principle anatural behaviors can be easily introduced by allowing multiparent inheritance.

3.3 Designing Operators through Density Estimation

As in the previous section, we have given a probabilistic interpretation to the search operators, we want to reside in the following to modeling them in terms of empirically obtained probability densities. In principle, different techniques to this regard can be considered, from which we will however review in the following only the most relevant ones.

3.3.1 Histograms

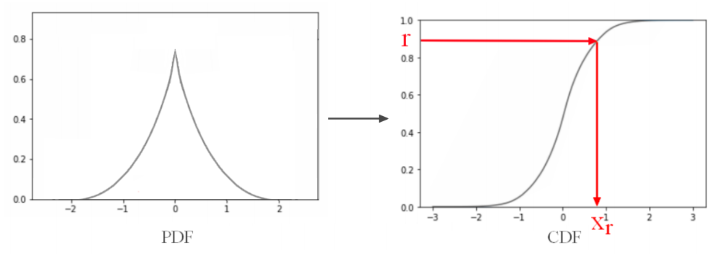

Figure 3.3: Illustration of the inverse transform sampling technique [41]. By means of calculating a cumulative density function CDF(j), any random variate r sampled from a uniform distribution $r \in [0, 1]$ can be easily transformed into a corresponding to sample from a non-uniform distribution $x_r = \text{CDF}^{-1}(r)$.

These are among one the most intuitive techniques to be used. Given a data set $\mathcal{D} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ of size $N = |\mathcal{D}|$ with $\mathcal{D} \subseteq \mathbb{R}^d$, one partitions the data space into a number of equisized bins $\mathcal{B} = \{b_1, \cdots, b_n\}$. Subsequently, for every bin $b_i \in \mathcal{B}$ we count the number c_i of data points $\mathbf{x}_j \in \mathcal{D}$ for which $\mathbf{x}_j \in b_i$ and calculate from these probabilities $p_i = c_i/N$. Based upon the probabilities we can easily calculate pseudo-random numbers from the modeled distribution using the inverse transform sampling technique [41]. For example, in the case of a discrete distribution we first sample a random number $r \in [0, 1]$. Subsequently, we calculate the cumulative density function $\text{CFD}(j) = \sum_{i=1}^{j} p_i$. To generate a pseudo-random number we find the j such that $\text{CFD}(j) < r \leq \text{CFD}(j+1)$ and based upon the associated part of the data space defined by bin b_j we generate the random number. For example, for $b_j = [a, b]$, we uniformly sample a random number from $[a, b]$. While histograms are at first glance intuitive and therefore tempting, their use is on the flip-side rather cumbersome due to them being parametric, meaning that they require the explicit definition of bin positions and widths. Further, often times additional pre-processing steps, such as the cropping of the data space to ensure that those parts are covered at good resolution which reveal the essential structure the modeled distribution.

3.3.2 Kernel Density Estimation

This technique, which is also known as *Parzen windows*, is among the most accurate methods one can use to model a data distribution [125, 28]. Essentially, for a given data set $\mathcal{D}$ we place a kernel function $K_{\mathbf{H}}(\mathbf{x}, \mathbf{x}_i)$ on every data point $\mathbf{x}_i \in \mathcal{D} \subseteq \mathbb{R}^d$, where $\mathbf{H} \in \mathbb{R}^{d \times d}$ is a parameter or bandwidth matrix shared by all data points and controlling the shape of the kernel function. The complete data set can then be modeled as the sum $f(\mathbf{x}) = 1/N \sum_{i=1}^{N} K_{\mathbf{H}}(\mathbf{x}, \mathbf{x}_i)$. For the kernel function different choices can be made. A popular one is the use of the Gaussian kernel $K_{\mathbf{H}}(\mathbf{x}, \mathbf{x}_i) = |\mathbf{H}|^{1/2}/(2\pi)^{d/2} \text{exp}[-(\mathbf{x}-\mathbf{x}_i)^T \mathbf{H}(\mathbf{x}-\mathbf{x}_i)]$. A key drawback of kernel density estimation method however is the determination of the bandwidth matrix $\mathbf{H}$. The mathematically correct way to determine $\mathbf{H}$ would be by minimizing the mean integrated squared error of f to the true distribution f^* using a bandwidth selection method [28]. However, the former distribution is in most cases not known. One simple way would be to estimate the bandwidth matrix using the log-likelihood in a cross-validation training scheme. However, this approach is easily leading to overfitting. Alternatively also bandwidth selection rules can be employed, however these introduce additional complexities and thus are difficult to get right for off-the-shelf application [66]. For sampling from a kernel density estimate, one simply randomly selects a point $\mathbf{x}_r \in \mathcal{D}$ from the data set and samples succeedingly from the kernel function $K_{\mathbf{H}}(\mathbf{x}, \mathbf{x}_r)$. While accurate, sampling within kernel density estimates can be intensive in regards to memory requirements.

3.3.3 Gaussian Mixture Model

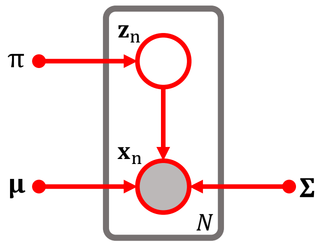

Figure 3.4: Probabilistic graph of the Gaussian mixture model based upon [17]. By performing ancestral sampling on $p(\mathbf{x}|\mathbf{z})\,p(\mathbf{z})$ we can easily generate random numbers behaving according to the modeled data distribution.

The Gaussian mixture model is in some way similar to the kernel density estimation technique with Gaussian kernel, however is more sparing in its complexity. In the Gaussian mixture model, the data distribution is modeled as $f(\mathbf{x}) = \sum_{i=1}^{K} \pi_i \, \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, where π_i are the so called mixture coefficients and $K \ll |\mathcal{D}|$ is the number of mixture coefficients, which naturally is chosen such that it is much smaller than the total size of the data set $\mathcal{D}$. The parameters π_k, $\boldsymbol{\mu}_i$ and $\boldsymbol{\Sigma}_i$ are determined using the expectation-maximization (EM) algorithm. The number of mixture components has to be chosen by the practitioner first and is in principle likewise to kernel density estimation a non-trivial topic. However, one can similarly reside from a pragmatic point of view to maximizing the log-likelihood using a cross-validation training scheme. The mixture coefficients are determined such that $\sum_{i=1}^{K} \pi_i = 1$, thus each coefficient π_i can be interpreted as a probability for choosing the specific component i. In this framework, pseudo-random numbers can be generated using ancestral sampling. Meaning that for a given distribution $p(\mathbf{x}|\mathbf{z})\,p(\mathbf{z})$, we first sample $\mathbf{z}_1 \sim p(\mathbf{z})$ and based upon it sample $\mathbf{x}_1 \sim p(\mathbf{x}|\mathbf{z}_1)$. For the Gaussian mixture model this would equate to $p(\boldsymbol{\mu}, \boldsymbol{\Sigma}|i)p(i)$, where $p(i) = \{\pi_1, \cdots, \pi_N\}$ and $p(\boldsymbol{\mu}, \boldsymbol{\Sigma}|i) = \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$. Where $p(i)$ can be sampled using the previously mentioned inverse transform sampling technique and the normal distribution $\mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ using readily available methods.

3.4 Extracting Knowledge from Evolutionary Searches

In the usual evolutionary search routine, given a start population of solutions with known fitness values, crossover is conducted first, followed by mutation, recalculation of fitness values, and subsequent selection to form the new population. Thus, the direct effect of crossover and mutation operators on obtaining good solutions and ensuring fast convergence is rather obscured.

Mechanically, unary mutation operators in their most basic form can be considered to be comparably simple. As they rely simply upon adding random variates on solutions, sampled from normal distributions (cf. right panel of Fig. 3.2). N-ary crossover operators on the flip-side are particularly more sensitive to specific chosen solution pairs, as the offspring created from them directly depends upon their position in the search space (cf. left panel of Fig. 3.2). Selection operators can in principle introduce further stochasticities, however it has been suggested in the literature that so called (μ, λ) and (μ+λ) selection schemes lead to performance boosts in mutation-based algorithms [9]. In the (μ, λ) scheme, λ new solutions are generated at each iteration from which only the μ best are accepted into the successor generation. In comparison, in the latter (μ+λ) selection scheme the set of old μ and λ new solutions are merged such that together from both only the μ best solutions are accepted into the successor generation.

3.4.1 The $(\mu + \lambda)$ Evolutionary Algorithm

Considering the simplicity and importance of unary mutation operators, as well as the nature of the $(\mu{+}\lambda)$ selection scheme, it is particularly tempting to consider an algorithmic template solely based upon these two operations. In principle, it is obvious that within such a framework only sampled random variates are accepted which improve the solution quality, while any worsening ones are discarded by the selection scheme. Thus, one is inclined to keep statistics about the quality of the performed mutations during algorithm runs, and construct from these inductive biases. These can be represented in the form of improved operators, such that they can be used to realize performance improvements on re-runs of similar problems.

Even though such a framework may spare out many of the complexities of modern algorithms, it can still serve as an effective model to understand the pitfalls and opportunities in regards to constructing experience-based approaches from a procedural view. For this reason, we therefore shall consider an algorithm based upon an isotropic mutation operator using a normal distribution, as well as the $(\mu{+}\lambda)$ selection scheme. An outline of our considered framework is given in Alg. 1. Noteworthy, this framework highly resembles algorithms such as Evolutionary Programming and Evolution Strategy [7]. We define in the following our objective function as follows as $f : \chi \subset \mathbb{R}^d \to \mathbb{R}$, i.e. a mapping from a d-dimensional search space χ. The optimization problem is then posed by the condition that we want to find a solution $\mathbf{x}^*$ such that it satisfies

$$\mathbf{x}^* \equiv \arg\min_{\mathbf{x}\in\chi} f(\mathbf{x}), \tag{3.1}$$

i.e. it corresponds to the global optimum. Note, that by convention within this work we consider optimization problems to be minimization problems. To solve our optimization problems, we initialize the evolutionary algorithm with a start population $P_0{=}\{\mathbf{x}_1^{(0)}, \cdots, \mathbf{x}_\mu^{(0)}\}$ of size $\mu = |P_0|$ with $P_0 \subset \chi$. Note, that we do not explicitly distinguish between genotype

Algorithm 1: $(\mu+\lambda)$ Evolutionary Algorithm

```
$g = 0$, initialize $P_g$=$\{\mathbf{x}_1^{(g)}, \cdots, \mathbf{x}_\mu^{(g)}\}$ and $\mathbf{\Sigma}$
repeat
    $Q_g \leftarrow \emptyset$
    for $k = 1, \ldots, \lambda$ do
        $\mathbf{x}_k^{(g)} \leftarrow$ selected randomly from $P_g$
        $\mathbf{x}_k^{(g)\prime} \sim \mathcal{N}(\mathbf{x}_k^{(g)}, \mathbf{\Sigma})$
        $Q_g \leftarrow Q_g \cup \{\mathbf{x}_k^{(g)\prime}\}$
    end
    $P_{g+1} \leftarrow$ select $\mu$ best from $P_g \cup Q_g$
    $g \leftarrow g + 1$
until *stopping criterion is met*;
```

and phenotype, such that the search space directly corresponds to the solution space, i.e. $\chi \equiv \mathcal{S}$, where each solution $\mathbf{x}_k^{(g)} \in P_g$ is expressed as a d-dimensional vector

$$\mathbf{x}_k^{(g)} = [x_k^{(g)}(1), x_k^{(g)}(2), \cdots, x_k^{(g)}(d)]^T, \tag{3.2}$$

where the variable k simply indicates the k-th solution and g the g-th iteration or generation. Subsequently one introduces variation operators which modify the solutions. As we have previously elaborated, we will consider for our studies only unary mutation operators based upon normal distributions (cf. right panel of Fig. 3.2). These mutation operators work by drawing differential increments $\Delta\mathbf{x}$ from a normal distribution $\mathcal{N}$ such that

$$\Delta\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}) \tag{3.3}$$

where the distribution is parametrized with mean $\boldsymbol{\mu} = \mathbf{0}$ and covariance $\mathbf{\Sigma}$. In the following, we will specifically consider a spherical covariance $\mathbf{\Sigma} = \mathbb{1} \cdot \sigma^{-2}$ with fixed sampling width σ. Upon application, the operator shifts solutions $\mathbf{x}_k^{(g)\prime}$ such that

$$\mathbf{x}_k^{(g)\prime} := \mathbf{x}_k^{(g)} + \Delta\mathbf{x} \sim \mathcal{N}(\mathbf{x}_k^{(g)}, \mathbf{\Sigma}). \tag{3.4}$$

To foster the use of experience by means of learning improved operators, we keep in our framework track of mutations performed. In principle, we do this by extending the algorithmic framework in Alg. 1 by a repository, which is filled with copies of pairs of objective function values, or better called *fitness values* f and solution positions $\mathbf{x}$ from before and after application of the mutation operator. This mutation tracking allows us in the following to further distinguish between *improving*

$$f(\mathbf{x}_k^{(g)}) - f(\mathbf{x}_k^{(g)\prime}) \geq 0 \tag{3.5}$$

and *worsening mutations*

$$f(\mathbf{x}_k^{(g)}) - f(\mathbf{x}_k^{(g)\prime}) < 0. \tag{3.6}$$

As we have previously elaborated, that we only consider the $(\mu{+}\lambda)$ selection scheme, it is quite clear that within the optimization itself only solutions $\mathbf{x}_k^{(g)\prime}$ generated from improving mutations are accepted into the successor generation. We therefore want to harness this property in the following, by building new variation operators from these collected statistics. These can be considered to be tailored to the specific problems from which they were derived, as they incorporate the problem structure, and therefore should quite naturally result into performance improvements.

Noteworthy, we do not explicitly consider to suppress worsening solution modifications. This is on one hand in reflection of arguments for developmental biases in evolvability studies [83, 146], but on the other hand can be also justified from essential convergence theorems [120]. As to guarantee convergence of an evolutionary optimization algorithm all areas of the search space should remain reachable for the unary mutation operator by a finite and positive probability.

3.4.2 Properties of the Retrieved Distributions

We start our first series of experiments with an investigation into the algorithm-problem interaction because we want to understand how the properties of retrievable mutation distributions rely upon the interplay between algorithm and problem. Quite naturally, the characteristic properties of both should lead to noticeable differences in the statistical distributions we can extract. For this reason, we run experiments using the previously extended evolutionary algorithm with the configuration as detailed previously, a mutation rate of 1, as well as for the sake of experimentation a population and offspring size of $\mu = \lambda = 10$. Note, that this particular setting ensures that improving mutations are always accepted, while only worsening ones are discarded for the formation of the successor generation.

In the first experiment we consider Griewank's benchmark function from Eq. (B.7). We keep the problem parameters constant and only vary the sampling width for mutations from $\sigma = 1.5$ to 4. In the second one we keep the sampling width constant and consider exclusively Ackley's benchmark function from Eq. (B.6) and with the depth parameter being usually defined as $a = 20$. However, in the following we vary a in the range from 1 to 20, thus varying the depth and steepness of the funnel while essentially keeping the positions of local extrema the same. In the third experiment, we simply visualize different generational intervals from 0 and 100, as well as 100 to 1000. While in the last fourth experiment, we simply show the distribution of all mutations, as well as the worsening ones for Griewank's function for comparison. Histogram representations of all retrieved distributions are illustrated from the first to fourth column of Figure 3.5.

We find on Griewank's function where we only vary the sampling width σ, that for $\sigma = 1.5$, the algorithm only retrieves a Gaussian multivariate distribution. After significantly enhancing the sampling to $\sigma = 4$, we can however retrieve a neighborhood structure of peaks arranged in a hexagonal grid akin to the structure of local optima in the fitness landscape of the benchmark function Eq. (B.7). Note, that we can interpret the recovered distribution as consisting out of a central part for local improvements and an outer part for long-range exploration of the neighborhood.

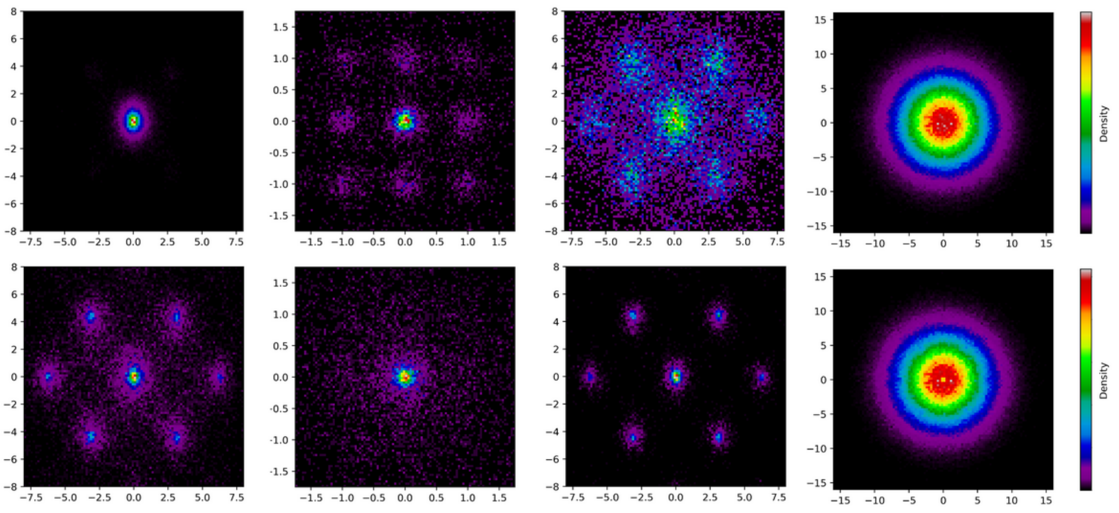

Figure 3.5: Different mutation distributions retrieved from running the $(\mu+\lambda)$ Evolutionary Algorithm under variable settings: Different sampling widths (Column 1), different problem parameters (Column 2), different generational intervals (Column 3), as well as all mutations and only worsening mutations (Column 4, top to bottom).

On Ackley's function for the steepness parameter $a = 20$ of the funnel, the retrieved distribution resembles a simple multivariate normal distribution and does not seem to encode any problem specific information. Setting the parameter to $a = 1$, the retrieved distribution strongly differs from a Gaussian bell shape by having further peaks akin to to the structure of local optima of the benchmark Eq. (B.6). The chosen sampling width is thus able to resolve notable problem-specific information. In the third experiment on Griewank's function again, we find that in the initial generations the retrieved distribution strongly resembles a multivariate normal distribution, and only in the latter phase, the structure of local optima becomes very prominent. The fourth experiment is merely to verify that the full distribution is Gaussian as expected, however putting it in comparison to the distribution of worsening mutations, the similarity between both is striking, as the latter is barely encoding problem-specific information. Thus, this validates our assumption that building operators by suppressing mutations may not be considered to be a viable strategy.

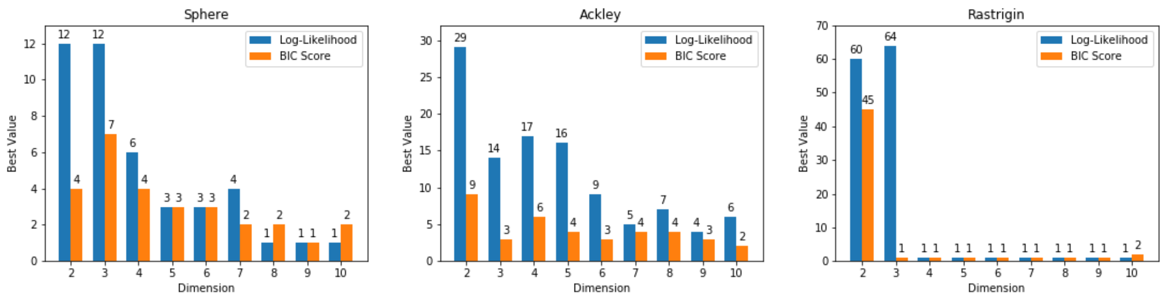

Figure 3.6: Comparison of the average number of mixture components retrieved by either optimizing the Log-Likelihood (blue) or BIC Score (orange) over variable dimensionality from 2 to 10 for the symmetric Sphere, Rastrigin and Ackley function (left to right panel).

3.4.3 Hyperparameter Optimization and Model Selection

We have previously used histograms for visualization of our retrieved mutation distributions and in principle, they could be also used to model probability densities. However, as previously elaborated from a practical point of view their application is rather cumbersome because they explicitly parametrize a density model, thus introduce additional model complexities which are difficult to get right in an off-the-shelf manner. In principle, we therefore would desire for a general purpose method to use instead non-parametric methods. For this purpose different techniques are available.

As previously mentioned, kernel density estimation is the most accurate technique, however may at times suffer from the problem that it can be memory intensive. The Gaussian mixture model offers instead a viable compromise in which the data set is reduced to a set of n descriptive clusters each modeled by a normal distribution. Sampling in this framework can be done using the previously mentioned ancestral sampling technique. While Gaussian mixture models are known to be an efficient method for density estimation and clustering, determining the best number of components n in an automatic fashion is a non-trivial topic. In principle, also highly dependent upon the particular application of choice [27]. The off-the-shelf approach is to directly maximize the log-likelihood given by

$$l(\mathbf{X}|\boldsymbol{\theta}, n) := \sum_{i=1}^{k} \ln \left\{ \sum_{j=1}^{n} \pi_j \mathcal{N}(\mathbf{x}_j|\boldsymbol{\theta}_j) \right\} \tag{3.7}$$

in respect to the number of components n, where $\boldsymbol{\theta}$ are the parameters determined by EM algorithm. Alternatively, to prevent overfitting one may instead consider the so called Bayesian Information Criteria (BIC)

$$\mathrm{BIC} = -2\, l(\mathbf{X}|\boldsymbol{\theta}, n) + v_n \ln(k), \tag{3.8}$$

which modifies the log-likelihood through an additional term which explicitly penalizes more complex models through the factor $v_n = n(1 + d + d(d+1)/2) + 1$, scaling in the dimension d of the data space, as well as size k of the fitted data set. The ideal model n is then the one which minimizes the respective BIC value. We compare the found model complexity (cf. Fig. 3.6) by maximizing the log-likelihood in a 10-fold cross validation training scheme with the one suggested by the BIC score over a range of three different benchmark functions of varying dimensionality. Overall, we can confirm the expectation that the BIC score suggests the usage of low-complexity models. However, both methods show a tendency to break down for more than three dimensions, thus suggesting the use of the lowest complexity model with only a single component. However, the log-likelihood estimate is in most cases still more optimistic. Thus, we will preferably use it in our follow-up studies.

On a last note, we again recall that the goal of density modeling for us is merely to have a method available to build reliably operators. One might therefore legitimately object that such a model should just aid in reducing the complexity of a data set. Thus, we could by this willfully neglect any model selection procedure and force the mixture model simply to have a fixed number of n components. While in principle this has the danger of overfitting, one might acknowledge that this still a much more desirable property than underfitting with low-complexity model. We will put both approaches, log-likelihood estimate and fixed number of components, into further comparison within Sec. 3.5.2, but for the time being will stick to the former.

3.5 Effectiveness of the Learned Operators

3.5.1 Operators as Domain Knowledge Representation

For the following experiments, we use a set of 9 different benchmarks functions taken from Appendix B. Specifically, we further divide them three different groups, where the first one consists out of unimodal problems without local optima from Eq. (B.1), (B.3) & (B.5), the second out of regularly structured multimodal problems taken from Eq. (B.6), (B.7) & (B.8) and the last one out of multimodal functions with highly irregular structure from Eq. (B.9), (B.10) & (B.11).

We set the population size to $\mu = 10$ and at each iteration generate $\lambda = 10$ new offspring solutions through randomly selecting individuals and applying the mutation operator. To initialize the population, we select randomly positions within the search space. Further, we additionally use for the multimodal functions with highly irregular structure a penalization by rejecting solutions generated outside of the search space boundary. This measure is mandatory because else on these benchmark functions better optima could be reached outside of the search space. We run each experiment for 1000 generations and average all obtained data from in total 100 runs per benchmark function. For all experiments, the dimension is set to $d = 2$, as this avoids problems with data sparsity and makes obtained distributions still manually interpretable. For the problems corresponding from Eq. (B.1) - (B.8), we initialize the sampling width with $\sigma = 4$, whereas for the problems from Eq. (B.9) - (B.11) corresponding to the difficult multimodal functions we choose accordingly $\sigma = 100$ for Schaffer's function, $\sigma = 220$ for Schwefel's function and $\sigma = 320$ for the Eggholder function. Note, that the choice of σ is in reflection of the search space sizes. However, we will elaborate in the follow-up paragraph more on the reasons for these particular choices of parameters. For the first 6 functions the mixture model is constructed by explicitly tuning the number of components using the log-likelihood in a Bayesian optimization routine [6] in the range from 1 to 100. For the latter functions from

Table 3.1: Performance values for the default operator ($\mathcal{N}$) based upon a multivariate normal distribution and improved operators ($\mathcal{M}$) that incorporate problem structures. Particularly, we list minimum median $\tilde{f}_{\text{min}}$ and mean function values $\overline{f}_{\text{min}}$ after 1000 generations, as well as variances s_{min} and a statistical significance comparison of both operators using the p-value.

	Operator: $\mathcal{N}$			Operator: $\mathcal{M}$			
Function	$\tilde{f}_{\text{min}}$	$\overline{f}_{\text{min}}$	s_{min}	$\tilde{f}_{\text{min}}$	$\overline{f}_{\text{min}}$	s_{min}	p-value
Sphere	2×10^{-3}	4×10^{-3}	4×10^{-3}	2×10^{-5}	3×10^{-5}	3×10^{-5}	4×10^{-32}
Bohachevsky	6×10^{-2}	7×10^{-2}	6×10^{-2}	3×10^{-3}	5×10^{-3}	5×10^{-3}	3×10^{-26}
Rosenbrock	3×10^{-2}	3×10^{-2}	3×10^{-2}	5×10^{-4}	8×10^{-4}	8×10^{-4}	2×10^{-30}
Rastrigin	4×10^{-1}	5×10^{-1}	4×10^{-1}	5×10^{-3}	8×10^{-3}	9×10^{-3}	3×10^{-32}
Ackley	2×10^{-1}	2×10^{-1}	2×10^{-1}	2×10^{-1}	2×10^{-1}	1×10^{-1}	1×10^{-33}
Griewank	2×10^{-3}	3×10^{-3}	3×10^{-3}	7×10^{-5}	8×10^{-5}	7×10^{-5}	1×10^{-32}
Schaffer	$1 \times 10^{+0}$	$1 \times 10^{+0}$	3×10^{-1}	7×10^{-1}	6×10^{-1}	2×10^{-1}	9×10^{-31}
Schwefel	$2 \times 10^{+0}$	$5 \times 10^{+1}$	$6 \times 10^{+1}$	2×10^{-2}	3×10^{-2}	3×10^{-2}	4×10^{-33}
Eggholder	3×10^{-3}	1×10^{-2}	2×10^{-2}	6×10^{-5}	$2 \times 10^{+1}$	$3 \times 10^{+1}$	5×10^{-8}

Eq. (B.9) - (B.11) we use explicitly the kernel density estimation technique, as otherwise the performance of the operator depends too heavily upon the retrieved mixture by the expectation maximization algorithm.

We give the experimental results for each of the three groups of benchmark functions in Fig. 3.7, 3.8 and 3.9 accordingly, where the upper figure corresponds to the unimodal functions, the central figure to the regularly structured multimodal functions and bottom row to the irregularly structured multimodal functions. Each panel illustrates the minimum fitness costs f per generation for in total 1000 generations over a given experimental setting, where minimum function values for individual runs are indicated in light blue, while median and mean values are corresponding to dark blue and grey lines. Notably, for all of the investigated benchmark functions we find that the new operators improve the convergence behavior. This can be partly attributed to the fact that the operators alleviate late convergences resulting from premature convergence in local optima.

We can even demonstrate that the approach can also work on the irregularly structured

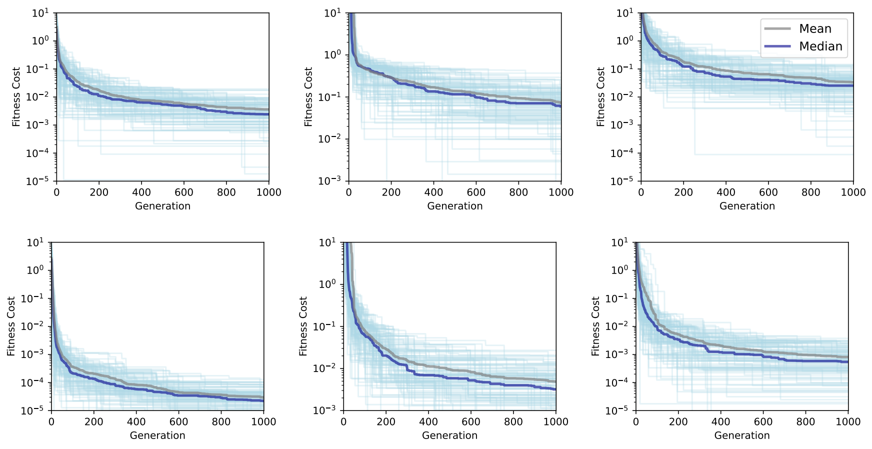

Figure 3.7: Fitness costs for the Sphere, Bohachevsky and Rosenbrock function (left to right) for default (top) and improved operator (bottom).

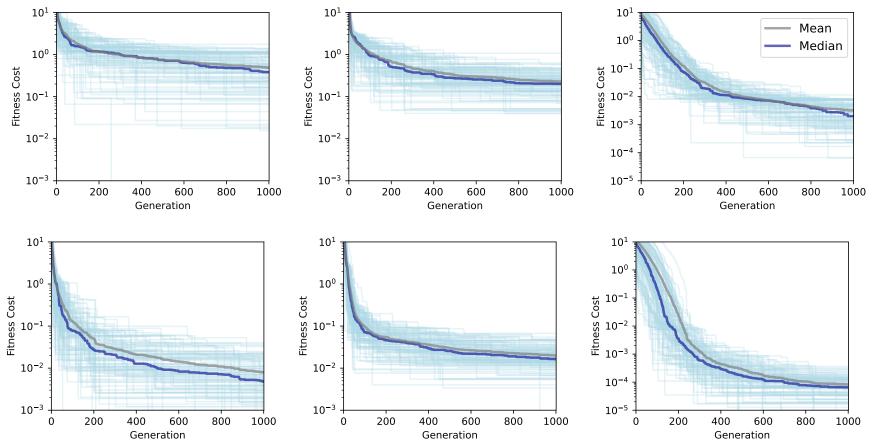

Figure 3.8: Fitness costs for the Rastrigin, Ackley and Griewank function (left to right) for default (top) and improved operator (bottom).

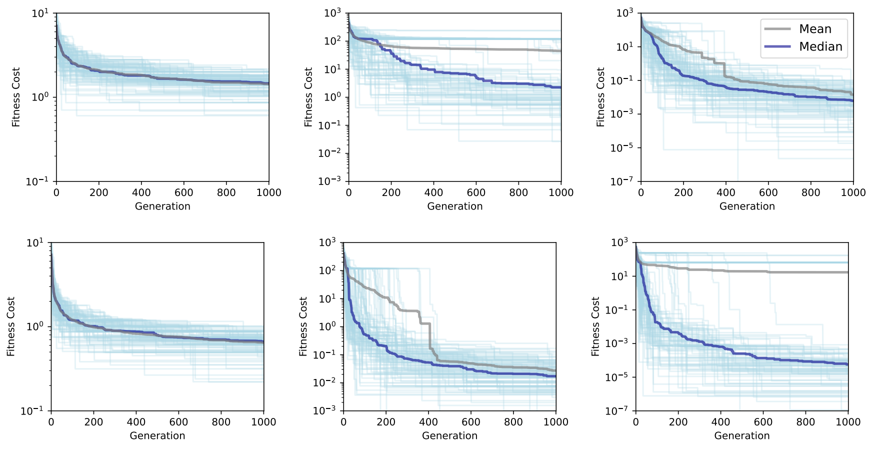

Figure 3.9: Fitness costs for the Schaffer, Schwefel and Eggholder function (left to right) for default (top) and improved operator (bottom).

benchmark functions in Fig. 3.9 to a limited degree. However, further precautions must be taken into consideration beforehand. In particularly, the sampling width must be tuned such that the optimization algorithm exhibits good convergence behavior with the default operator first, as otherwise we found that the retrieved operators can even be detrimental to the algorithm performance by encouraging convergence into local optima.

To additionally substantiate our claims, we further give p-values in Tbl. 3.1 obtained using a Wilcoxon rank-sum test. Taking $\alpha = 0.05$ as statistical significance level into consideration, our results clearly show that the null hypothesis is rejected in all considered experimental settings and thus the performance improvements are indeed statistical significant.

3.5.2 Upscaling to High Dimensional Problems

Having seen that model selection fails at high dimensions and tends to be biased towards low-complexity models, we want to find out in the following whether we can generate any significant improvements using alternative approaches for density modeling.

Particularly, we will look in the following into modeling operators by means of either directly resampling from the mutation storage without applying any density modeling, enforcing component-wise marginalization, or using mixtures with a preset and fixed number of components. As an optimization problem of choice we will use in the following the Rastrigin function from Eq. (B.8). This benchmark function has the beneficial property that while being multimodal, it is still comparably simple constructed. Particularly, featuring a quadratic funnel structure upon which a regular sinusoidal periodicity is superimposed on top. Due to the function being fully separable, upscaling it to higher dimensions just is equivalent to a summation of the one-dimensional versions of the function for each component.

Table 3.2: Comparison of the different modeling techniques in terms of repository-based sampling, marginalized mixture model with $n\times$GMM$[d/n]$ and fixed component mixtures, over different settings in terms of dimensionality d and sampling width σ relative to the default behavior of the $(\mu{+}\lambda)$ Evolutionary Algorithm, in terms of percentual fitness reduction Δf_{med}, median fitness f_{med} and p-value. Note, that statistically significant improvements in comparison to baseline behavior are printed in bold.

Setting		$\sigma = 1$				$\sigma = 3$				$\sigma = 5$			
Dimension	Model	Δf_{med} [%]	f_{med}	p-value	Samples	Δf_{med} [%]	f_{med}	p-value	Samples	Δf_{med} [%]	f_{med}	p-value	Samples
6	Default	-	11.70	-	5425	-	24.36	-	3610	-	34.86	-	2612
6	Repository	**45.06**	6.43	6×10^{-30}	-	**40.64**	14.46	5×10^{-28}	-	**44.34**	19.40	2×10^{-30}	-
6	3×GMM[2]	**39.04**	7.13	4×10^{-27}	-	**40.36**	14.53	3×10^{-28}	-	**40.22**	22.43	2×10^{-26}	-
6	2×GMM[3]	**36.11**	7.47	9×10^{-24}	-	**22.20**	18.95	1×10^{-14}	-	**35.66**	20.84	1×10^{-29}	-
6	100	**16.63**	9.75	1×10^{-9}	-	**24.15**	18.48	3×10^{-16}	-	**36.10**	22.28	9×10^{-28}	-
6	50	**10.17**	10.51	3×10^{-4}	-	**23.97**	18.18	9×10^{-17}	-	**34.30**	22.90	1×10^{-26}	-
6	25	**4.73**	11.14	4×10^{-3}	-	**25.37**	18.52	4×10^{-15}	-	**35.48**	22.49	1×10^{-27}	-
6	1	-2.30	11.97	$1\times10^{+0}$	-	**25.81**	18.07	6×10^{-16}	-	**35.08**	22.63	4×10^{-27}	-
12	Default	-	56.71	-	5847	-	96.35	-	3181	-	137.40	-	1367
12	Repository	**11.42**	50.24	1×10^{-11}	-	**17.48**	79.50	3×10^{-22}	-	**24.30**	104.01	9×10^{-30}	-
12	6×GMM[2]	**10.89**	50.54	1×10^{-10}	-	**15.90**	81.03	3×10^{-20}	-	**24.91**	103.18	6×10^{-31}	-
12	4×GMM[3]	**11.00**	50.47	4×10^{-9}	-	**5.12**	81.78	3×10^{-17}	-	**25.36**	102.55	6×10^{-31}	-
12	100	0.42	56.47	7×10^{-1}	-	**13.97**	82.88	2×10^{-17}	-	**24.31**	104.00	1×10^{-30}	-
12	50	0.08	56.67	9×10^{-1}	-	**16.39**	80.55	2×10^{-20}	-	**23.38**	105.27	1×10^{-29}	-
12	25	0.78	56.27	5×10^{-1}	-	**15.34**	81.57	2×10^{-18}	-	**26.89**	100.46	2×10^{-31}	-
12	1	2.65	55.21	1×10^{-1}	-	**13.73**	83.13	3×10^{-18}	-	**25.20**	102.77	4×10^{-31}	-
24	Default	-	182.80	-	6484	-	299.60	-	1933	-	372.11	-	137
24	Repository	**3.63**	176.15	6×10^{-3}	-	**10.83**	267.14	2×10^{-21}	-	**4.38**	355.81	4×10^{-3}	-
24	12×GMM[2]	**3.43**	176.53	7×10^{-3}	-	**9.36**	271.57	1×10^{-20}	-	**8.32**	341.16	6×10^{-12}	-
24	8×GMM[3]	**2.77**	177.73	2×10^{-2}	-	**9.33**	271.64	7×10^{-19}	-	**7.28**	345.02	5×10^{-11}	-
24	6×GMM[4]	0.73	181.45	6×10^{-1}	-	**11.21**	266.02	2×10^{-23}	-	**6.52**	347.86	4×10^{-9}	-
24	100	0.03	182.73	4×10^{-1}	-	**11.26**	265.86	1×10^{-22}	-	**9.72**	335.95	2×10^{-16}	-
24	50	2.24	178.70	1×10^{-1}	-	**10.00**	269.65	3×10^{-20}	-	**14.43**	318.43	3×10^{-26}	-
24	25	2.63	177.99	7×10^{-2}	-	**11.67**	264.63	2×10^{-23}	-	**12.69**	324.89	4×10^{-23}	-
24	1	1.34	180.35	5×10^{-1}	-	**10.62**	267.77	4×10^{-22}	-	**9.97**	335.00	4×10^{-17}	-

In the following up experiments, we initialize likewise our $(\mu{+}\lambda)$ Evolutionary Algorithm again randomly on the entire search space, with the population size and new solutions per generation as elaborated previously. However, we take the liberty of varying the sampling width σ over the range $\sigma \in \{1, 3, 5\}$ simply to test different settings. We run each experiment for 1000 generations and take the median over 100 runs. To see whether the performance difference by the newly retrieved operators is statistically significant, we apply again a Wilcoxon rank-sum test and compare it to the default runs. All performance values, meaning percentual median fitness reduction Δf_{med}, as well as median fitness values f_{med} and p-values for different settings in terms of dimension, modeling technique and sampling width σ are listed in Tbl. 3.2.

From the results we can draw the following conclusions: First of all, comparing all modeling techniques and sampling widths, performance improvements are most pronounced for the lower dimensional settings. Ranging up to $\approx 45\%$ for $d = 6$ and $\sigma = 1$ considering resampling based upon the mutation storage, while being ≈ 40 to 44% for $\sigma = 3$ and 5. Going to higher dimensions, the percentage of performance improvements shrinks to $\approx 11\%$ for $\sigma = 1$ for repository-based sampling, to $\approx 17\%$ for $\sigma = 3$ and ≈ 24 to 27% across all statistically significant modeling techniques for $\sigma = 5$. The improvements shrink drastically from 3 to 4% for $\sigma = 1$, from 9 to 12% for $\sigma = 3$ and from 4 to 14% for $\sigma = 5$. In conclusion, we find that the utility of the operators rapidly declines with increasing dimensions.

Comparing the modeling techniques altogether, we find that repository-based resampling dominates for $\sigma = 1$ or $d = 6$. However, radically declines in its efficiency beyond these parameters. We find that the marginalization-based approach works likewise well for a sampling width of $\sigma = 1$, but tends to decrease in its performance for $\sigma = 3$ and $\sigma = 5$, especially when considering higher dimensions. To be on point: The more spread out densities that would need to be resolved from the samples are, the less effective repository and marginalization-based re-sampling become. Notably, at larger sampling widths and higher dimensions, density models based upon a fixed and preset number of components become more effective. This can be seen as being a direct consequence of the inaccuracy generated by the marginalization approaches. Thus, a mixture with a fixed number of components can be considered to give in this regime a more accurate density estimate. At last, we remark that the approximation using only a single Gaussian does not lead to significantly dominating results in all considered scenarios.

To conclude, the effectiveness and trade-offs of each of the alternative modeling techniques can be interpreted in terms of spread of the density that given a particular setting would need to be resolved. Though, practically we could observe that in all scenarios, the efficiency of the operators rapidly declined with increasing dimensions. Ranging for

$\sigma = 1$ only from 3 to 4%, for $\sigma = 3$ from 9 to 11% and for $\sigma = 5$ from 4 to 14%. In all considered settings, we are not able to identify a single technique which outperforms significantly the other ones.

3.5.3 Cross-Problem Knowledge Transfer

The quantification of problem similarity remains to be an outstanding problem within research on knowledge transfer methods for optimization. Existing methods within the literature largely rely upon pragmatic methods to this regard. The first method in the literature encountering this issue is CIGAR [93] in the context of combinational problems, which maintains a case-base of solutions from problems it solved within its past to solve new ones more efficiently. Particularly, by means of querying them based upon problem similarity. However, the authors openly acknowledge, that in many scenarios distinguishable problem characteristics may not be directly accessible. Therefore, suggest as a way to cope with it, to reflect problem similarity by means of solution similarity instead.

Similar measures have also been proposed in more recent work where the AMTEA framework from Da et al. (2018) [36] is particularly noteworthy to this regard. Within their work, a Gaussian distribution is fitted to each final population obtained from previously solved multi-objective optimization problems and the obtained parameters are stored within a database. Thus, when encountering new and previously unseen problems, the algorithm halts at regular intervals of Δ and performs a stacked density estimation of the current solution population using the Gaussian distributions stored within the database constructed from the previously solved problems. Effectively, the obtained mixture weights subsequently encode the solution similarity between the single target and the multiple source problems. In a similar fashion also the work from Zhang et al. (2019) [163] has employed Gaussian distributions. However, in their proposed multisource selective transfer framework, the similarity between solution distributions of the current target problem and the ones stored from the source problems in the repository are directly cal-

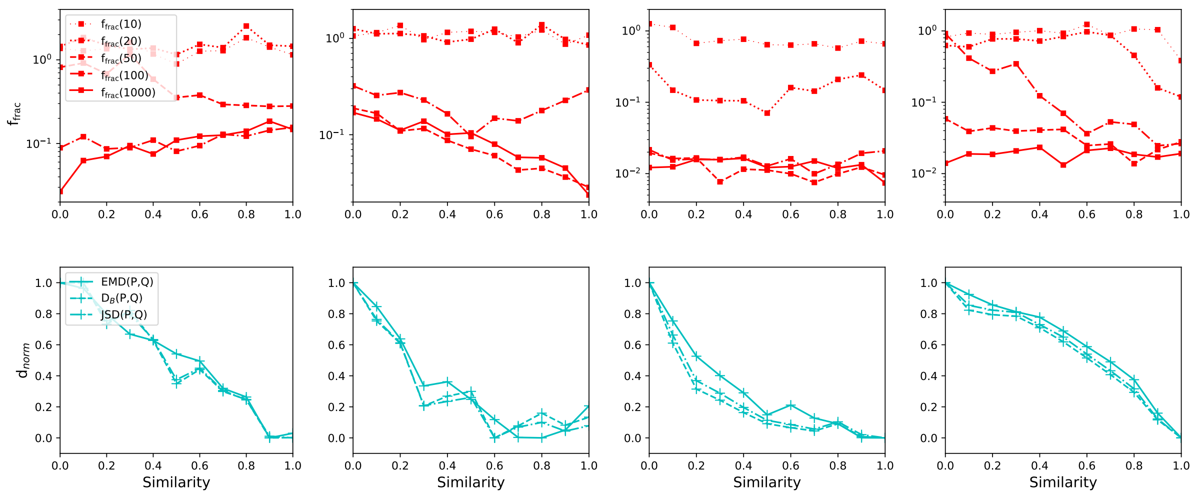

Figure 3.10: Percentual fitness fraction f_{frac} at generation 10, 20, 50, 100 and 1000 (upper row), as well as normalized statistical distances of the mutation distributions d_{norm} in terms of $D_B(P, Q)$, JSD(P, Q) and EMD(P, Q) (lower row) corresponding to Bhattacharyya distance, Jensen-Shannon divergence and Wasserstein distance, over varied source-to-target function similarity parameter s, for the transfer scenarios of Ackley-to-Rastrigin, Rastrigin-to-Ackley, Rastrigin-to-Sphere and Sphere-to-Rastrigin (from left to right column) .

culated through the Wasserstein distance. Based upon the calculated similarities, different source selection strategies can subsequently be employed.

Note however, that a variety of approaches within the literature completely neglect to look more extensively into measures of problem similarity. Thus, fostering what has been dubbed as 'negative transfer' [46]. However, as we have looked previously into the modeling of operators explicitly through taking a procedural view, our approach in principle allows a slightly different way of characterizing optimization problems. Particularly, in terms of the mutation distributions which we used to build the operators. Notably, this allows a less absolute view of characterizing optimization problems. Thus, to test our assumption, that similarity in mutation distributions also encodes problem similarity, we will consider the following scenario for investigation: We take three benchmark functions from Eq. (B.6), (B.8) & (B.5) and form two groups of each. The first group consists out of the Rastrigin and Sphere function, while the second group consists out of the Ackley and Rastrigin function. We build new optimization problems from these, by considering in the following the sum $f(\mathbf{x}; s) = (1-s)g(\mathbf{x}) + sh(\mathbf{x}))$, where s is a similarity parameter $s \in [0, 1]$, such that $f(\mathbf{x}; 0) = g(\mathbf{x})$ and $f(\mathbf{x}; 1) = h(\mathbf{x})$. Within our experiments, we first build operators for the function $h(\mathbf{x})$, and subsequently apply them to $f(\mathbf{x}; s)$, while we vary the similarity parameter s.

Additionally, we also consider a second series of experiments, in which we consider the same setup, but instead of tracking performance improvements, we compare the obtained operators on $h(\mathbf{x})$ and $f(\mathbf{x}; s)$, in terms of statistical similarity measures, to test the assumption on whether similarity of mutation distributions corresponds also to problem similarity, while again we vary the parameter s. We use to this regard three different similarity measures which have different underlying interpretations. The Bhattacharyya distance [15] given by

$$D_B(P, Q) = -\ln\left(\sum_{\mathbf{x}\in X} \sqrt{P(\mathbf{x})Q(\mathbf{x})}\right), \tag{3.9}$$

measures the overlap between two distributions P and Q, where $\mathbf{x}$ corresponds to the center position of a bin and X being the set of all bins; the Jensen-Shannon divergence [53] given by

$$\mathrm{JSD}(P, Q) = \frac{1}{2}[D_{\mathrm{KL}}(P\|M) + D_{\mathrm{KL}}(Q\|M)], \tag{3.10}$$

is an information theoretic metric that measures the summed difference of P and Q to an average mutual distribution $M = 1/2(P + Q)$, where D_{KL} is the Kullback-Leibler divergence; and at last the Wasserstein or Earth Mover's distance [119]

$$\mathrm{EMD}(P, Q) = \left(\sum_{i=1}^{m}\sum_{j=1}^{n} f_{i,j} d_{i,j}\right) / \left(\sum_{i=1}^{m}\sum_{j=1}^{n} f_{i,j}\right), \tag{3.11}$$

which can be interpreted as measuring the minimum energy necessary to transform P into Q and vice versa, where d_{ij} are Euclidean distances between the bins of both the distributions and f_{ij} are flow coefficients which are calculated by solving the optimal transport problem [50].

All results from our experiments are illustrated in the plots of Fig. 3.10. Where the upper row are plots of performance improvements in terms fitness fraction over variable similarity parameter s, and in the lower row are plots of the normalized statistical distances calculated through 2d histograms in the range of $[0, 1]$, where 0 encodes highest similarity, over variable similarity value s, as calculated by Eq. (3.9), (3.10) and (3.11). We elaborate in the following on the conclusions we can draw from these experiments. First of all, it is quite evident, that all statistical distance measures behave very similar, and that there is a clear negative correlation with the similarity parameter s evident in all plots. Thus, if the mutation distributions of two operators are statistically similar, it also implies that the problems are similar. However, relating these to performance improvements is problematic. While we find that the transfer of operators is generally beneficial in our considered benchmark problems, any gains in efficiency itself are at best only ambiguously correlated with problem similarity. Thus, the more similar the target problem is to the source problem in terms of s, it does not necessarily imply that the performance improvements are also higher. In fact, we only found that this scenario holds true for either $g(\mathbf{x})$ and $h(\mathbf{x})$ being Rastrigin's and Ackley's function (cf. second column in Fig. 3.10), or the Sphere and Rastrigin function (cf. fourth column). However, the same judgment cannot be made for the reverse case (cf. for the former the first, and the latter the third column).

Noteworthy, the no-free-lunch theorems for optimization [158] explicitly caution that performance improvements of any algorithm a_j in comparison to a baseline configuration a on a restricted function set $\mathcal{F}^*$ are only guaranteed when it is known, that it also explicitly incorporates problem-structure. However, the reverse direction does not necessarily hold true, and thus making judgments about the similarity of two problems by comparing the performance of operators on thereof as well as vice versa can be considered to be a logical fallacy. And while we have used a similarity parameter s to quantify problem similarity, it may only poorly reflect actual structural similarity from an algorithm-dependent view.

3.6 Chapter Summary

Contributions

To conclude this chapter, we wrap up the key contributions from our studies of modeling evolutionary search operators:

- **We proposed histogram and density estimation based methodology to model mutation operators in the framework of a (μ+λ) Evolutionary Algorithm.**
- **We introduced different methods for upscaling the operator construction to higher dimensional problems.**
- **We framed problem similarity in terms of statistical similarity of preferences in generated random variates and evaluated different statistical distance measures as a possible means to quantify it.**

Summary

First of all, by looking into literature of evolutionary biology, we pointed out that within the variational calculus of natural evolution, mutations are the most important and the prime source of variation [92], as they effectively act as novelty generators [89]. The importance of mutation as a variation operation has been further solidified within convergence theorems of evolutionary computation, which suggest that to ensure convergence every point of the search space should be accessible with non-vanishing probability [120]. It is thus quite natural to consider this variational operation as a means to model domain knowledge. Essentially, this also connects the variational calculus of evolutionary computation with the concept of inductive biases in machine learning [102]. Meaning, that in order to generalize performance improvements, any kind of learning systems must necessarily form biases towards one model of domain knowledge over others within the hypothesis space. Residing again to studies in computational biology, it has been well argued that the variational calculus of natural evolution exhibits learning capabilities through mirroring the aforementioned inductive biases through developmental biases [146]. However, these may not only foster adaption to specific environments, but may exhibit a much broader learning capability such that they are able to generalize performance gains to future unseen environments [83].

Motivated by the insights from these literature findings, we looked into mutation-based evolutionary algorithms to find ways to explicitly model domain knowledge. From comparing Evolutionary Programming and Evolution Strategies [9], we concluded that a framework based upon elitist selection schemes is most beneficial for further scrutiny. Specifically, we argued for the utility of the $(\mu+\lambda)$ selection scheme, as in principle it is clear that within such a framework only mutations are accepted that are improving the solution quality. Thus, we proceeded in the following with modeling the operator. For this reason, we specifically resided to a probabilistic interpretation thereof and used to this regard density estimation methods. We can demonstrate the efficiency of this approach on unimodal, as well as structured multimodal, as well as to a certain degree also irregularly structured optimization problems. While density estimation becomes more problematic at higher dimensions, we were nevertheless still able to generate performance improvements with the upscaling approaches in Sec. 3.5.2. Though, admittingly the effectiveness of the retrieved operators fastly declines with increasing dimensions. We further also looked into a scenario concerning cross-problem knowledge transfer and specifically potential ways of quantifying problem similarity. While we found, that in the considered scenarios, the transfer of operators is generally beneficial, we were not able to relate increases in problem similarity to increases in performance improvements. Admittingly, in some cases even found the reverse relationship to be in effect. An explanation for this may be partly attributed to the fact, that our considered similarity measures in terms of statistical distances $d(P, Q)$ and similarity parameter s may only poorly reflect actual problem similarity from an algorithm-dependent view. Thus making judgments on the similarity of them, or any generated performance improvements of the operators, may be considered a logical fallacy [158].

In the following chapters, we therefore want to look into further ways on how to obtain a problem-dependent perspective of optimization problems themselves. In principle, within the aforementioned meta-learning model, this is the question on how to obtain metaknowledge representations which serve as high-level abstractions of procedural metadata generated from the interplay of algorithm and optimization problem. We will also revisit in Chapter 6 a scenario in which we explicitly predict operators for optimization problems. However, we reframe it in terms of a problem of predicting a more compact parametrization in terms of an operator configuration instead. This scenario, has the advantage that it is less complex as density estimation, due to it requiring a comparably smaller number of parameters to be determined, while at the same time sparing the need of high numbers of samples, which we previously required to be in the first place able to model the mutation operators.

CHAPTER 4

FEATURE EXTRACTION FROM PROCEDURAL METADATA

4.1 Implications of the NFL Theorems in Optimization

We elaborated in the previous chapter on the importance of inductive biases in enabling algorithms to generalize performance gains from training instances to arbitrary new instances from a given problem domain. We formed inductive biases by explicitly incorporating information about the problem structure into variation operators. While the performance gains generated through this approach are obvious from an intuitive perspective, it is important to understand its theoretical justification.

Specifically, we recall at this point the first no free lunch theorem, which is stated as follows [158]: Given 1) an objective function $f : \mathcal{X} \rightarrow \mathcal{Y}$, 2) the space of all possible objective functions $\mathcal{F}$ with $f \in \mathcal{F}$, 3) a set d_m of m distinct function evaluations $(d_m^x(i), d_m^y(i))$ in ascending time-order with $d_m^x(i) \in \mathcal{X}$, $d_m^y(i) \in \mathcal{Y}$ at step i, 4) algorithms $a : d \in \mathcal{D} \rightarrow \{x | x \notin d^x\}$ mapping a sample d from the space of all possible samples $\mathcal{D}$ to unvisited ones $x \notin d^x$, and 4) the space of all algorithms $\mathcal{A}$ with $a \in \mathcal{A}$. Then, for any pair of algorithms a_1, $a_2 \in \mathcal{A}$, it holds true that: $\sum_{f \in \mathcal{F}} P(d_m^y | f, m, a_1) = \sum_{f \in \mathcal{F}} P(d_m^y | f, m, a_2)$. Meaning the probability averaged over all possible objective functions $f \in \mathcal{F}$ to retrieve a sample set d_m^y for any pair of algorithms a_1, $a_2 \in \mathcal{A}$ is the same.

First of all, the most obvious implication is that if we find that for an algorithm $a \in \mathcal{A}$, that on two subsets $\mathcal{F}_1, \mathcal{F}_2 \subset \mathcal{F}$ of the function space with $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2$, that a demonstrates a high-performance on e.g. $\mathcal{F}_1$, it necessarily implicates in return performance degradation on the complementary function set $\mathcal{F}_2$ such that the total performance on $\mathcal{F}$ remains constant and thus independent of the particular algorithm a. Secondly, the mere knowledge about the existence of structure does not justify the choice of one optimization algorithm a_1 over another algorithm a_2 unless the structure is explicitly known and reflected in the design of a chosen algorithm. We elaborated on the implications of the latter already previously. But in the following, will take a closer look on the former.

In principle it acknowledges the impossibility of constructing a high-performing universal problem solver. As any performance gains realized on a restricted problem domain comes in turn with performance degradation on the complementary problem set. However, the no free lunch theorems allow one to circumvent this problem. Given a partition of a function set into different problem structures $\mathcal{F}_1, \cdots, \mathcal{F}_N$, as well as a base algorithm a, one may attempt to specialize the base algorithm into distinctive configurations $a_1, \cdots, a_N$ corresponding to each function set such that performance improvements are guaranteed on each. Thus, framing it in terms of a meta-learning model, the difficulty lies in finding ways to arbitrate between the different biased algorithm components in a domain dependent manner. As we have previously elaborated, this requires the active generation and processing of metaknowledge. We will therefore look into it in the following chapter.

4.2 Approaches within the Literature

Within the literature, we can particularly identify two lines of research which are relevant for deriving methodology, that allows us to harness metadata from heuristic optimization algorithms. The first one being attempts to analyze behavioral optimization data using various supervised and unsupervised learning methods. The second one employs deep pattern recognition for classification and regression as a way to short-cut traditional algorithm selection frameworks and predict solvers and solutions based upon inputs directly generated from optimization problems.

4.2.1 Algorithm Behavior Studies

This line of research originates from early attempts at moving away from theoretical models of search behavior to more pragmatic ones which enable the analysis of algorithms through means of empirical measures as done by Turkey & Poli (2012) [145]. While in principle, this has been already done in the past by relating analytically calculated properties of white-box optimization problems with algorithm performance (e.g.: [76]), the approach taken Turkey & Poli (2012) attempts to develop a new tool based upon directly assessing the behavior of an algorithm by keeping track of the problem-dependent movement of candidates solutions in the search space as a whole (cf. Fig. 4.1).

Central to their method is the use of a self-organizing map (SOM) [82] which is a clustering technique that superimposes a neighborhood structure upon the cluster nodes, such that every cluster can be identified through a set of tuples $(n_1, \cdots, n_N)$ of size N within a regular structured coordinate system. Usual implementations of the self-organized map rely upon two-dimensional and rectangular coordinate systems with $N = 2$. Turkey & Poli (2012) use the self-organized map as a way to model the population structure within the search space of continuous single-objective optimization problems. In an initial training phase, the self-organized map is fitted to the start population. Based upon the obtained map, one can associate fitness values, solution counts and solution distances to the nearest cluster centroid. Subsequently, the evolutionary algorithm is iterated for another generation and a second training phase is initiated to refit the map to the new population. Based upon changes to cluster assignments, as well as population and fitness

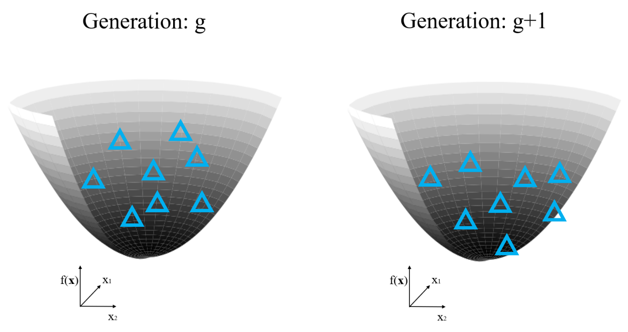

Figure 4.1: In works on algorithm behavior studies, a key idea is that algorithms and problems can be characterized by the specific changes imposed on the solution distribution from $P_g \rightarrow P_{g+1}$, through the interplay between algorithm and problem over successive generations $g \rightarrow g+1$.

density, Turkey & Poli (2012) define empirical measures quantifying exploration and exploitation behavior. Note, that their focus is solely to describe different search behaviors of evolutionary algorithms from a qualitative perspective.

More loosely based upon their work, Pang et al. (2016) [113] follow up this approach using likewise behavioral optimization data. However, this time with a clear focus on learning features which allow them to explicitly differentiate the behavior of different evolutionary algorithms and optimization problems, and not to describe them. Likewise they adapt the self-organized map within their work, but instead to obtain a low-dimensional representation of the search space itself. Thus, in advance fit it to training data uniformly and exhaustively generated within the search space. Experiments are setup with a fixed initial population and subsequently the evolutionary algorithm or problem of interest is varied and the generated offspring population is recorded. Using the offspring generation, the cluster assignments on the self-organized map are recorded and post-processing is done using PCA and the slow-feature analysis technique. Where the latter is explicitly used to construct a feature space in which the different evolutionary algorithms and optimization problems separate. Within their experiments, Pang et al. (2016) can show that they are able to construct feature spaces which can sufficiently separate different

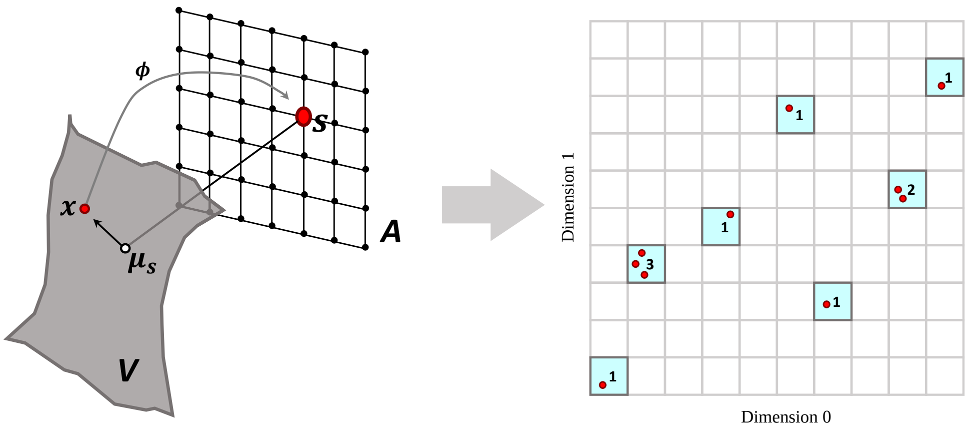

Figure 4.2: Illustration of the approach as proposed by Liu et al. (2017) [91]. Points are converted from a high-dimensional space V to a low-dimensional 2d space A by means of a mapping $\phi : V \rightarrow A$ constructed through the self-organized map. Counting the number of solutions closest for each neuron in the self-organized map, a structured data format in the form of two-dimensional histograms can be obtained, which can be subsequently used as input for the training of learning algorithms.

evolutionary algorithms. However, they are incapable of sufficiently separating different optimization problems for given fixed optimization algorithms, unless the given problems contain obvious asymmetries. Notably, their work neglects any information about fitness values or distances of the candidate solutions to their assigned cluster node.

Work from Liu et al. (2017) [91] has been further following up the slow-feature analysis based approach. However, motivated by the computational expensiveness of the former, their work explores the question whether or not a modern deep network architecture can be used to learn features capable of separating different evolutionary algorithms. Likewise, to previous work, the self-organized map is used once again as a way of obtaining a low-dimensional representation of a search space. However, they explicitly harness the two-dimensional structure of the self-organized map by recording generational changes in the assigned number of candidate solutions to every cluster node in a matrix representation (cf. Fig. 4.2). Once these are obtained, they can subsequently be labeled and fed for training to a neural network architecture based upon convolutional layers [88]. Note, that the latter are used in an attempt to explicitly exploit the two dimensional structure of the input matrices. Their study shows, that they can indeed achieve competitive results to the previous SFA-based method, by means of obtaining a latent space in which behavioral data of different algorithm is significantly disentangled. However, note their work does not reconsider the problem of distinguishing different continuous optimization problems.

4.2.2 Feature-Free Algorithm Selection

The application of pattern recognition methods has a longer tradition within the field of heuristic optimization. For example, in the field of hyper-heuristics and combinatorial optimization, they can be used to learn a mapping from a known problem state to the best known optimal solver [24]. They may also find their application within traditional studies of algorithm selection pipelines [103, 78]. However, in many cases obtaining good and descriptive characteristics is the problem which needs to be solved in the first place [96].

Recent ambitions within the field of combinatorial optimization have attempted to shortcut this step by employing deep neural networks (cf. Fig. 4.3). Thus, the step of calculating problem characteristics is done implicitly by the network architecture, which is trained to predict algorithms and solutions to efficiently solve optimization problems in the first place. From an intelligent systems design point of view, this reflects the notions which features also have in biological cognitive systems [69, 18]. Meaning that any kind of models of problem characteristics solely arise out of environment and goal-dependent learning [133]. Noteworthy, particularly the recent work of Seiler et al. (2020) [127] explicitly uses deep neural networks to predict an optimal solver for traveling salesperson problems (TSP). However, instead of calculating problem characteristics of the different TSP instances, they generate 512x512 raster images of visual representations from plots of point clouds, minimum spanning trees and nearest neighbor graphs. Based upon these generated images, a CNN-based classifier is trained to predict the best known solver for a given TSP problem instance from two available ones. Through their results they can prove, that

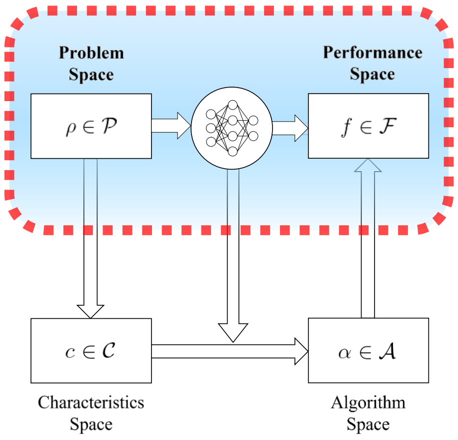

Figure 4.3: Recent works on algorithm selection frameworks have attempted to shortcut the traditional pipeline, by introducing a direct mapping from problem space to performance space, where the calculation of problem characteristics is encapsulated into deep neural network architectures, from which directly the best performing algorithms and solutions are predicted.

with the feature-free neural network based approach they can achieve results which are competitive and partly surpassing classical algorithm selection frameworks. Similarly, recent work also explored the use of transformer networks to this regard [22].

A similar and parallel line of work [2, 3] has investigated the use of sequential models, particularly implementations of LSTMs [68], to solve 1d bin packing problems. In principle, with an interest in either predicting the optimal solvers for a given problem instance or optimal solutions. For the former case, they can show that their approach is capable of achieving higher performance than the single best solver (SBS). While for the latter, they find that their method can predict solutions very accurately to heuristics used to generate the training data, and at times even generates them with comparably higher performance. Though, while such a learned predictive model might not always be able to fully substitute handcrafted solvers, it can be considered to be a useful complementary [4]. The comfort of combinatorial problems is that in principle the search spaces thereof are well-structured and problem characteristics may often be accessible in advance.

Noteworthy, it has been pointed out early-on by the comparative work of Smith-Miles

(2009) [136] that essentially algorithm selection and configuration problems constitute meta-learning problems. Thus, research on thereof establishes a bridge to our aforementioned question on the nature of inductive biases within optimization algorithms and possible ways to arbitrate between them.

4.2.3 Conclusions

Summarizing the reviewed work, we can come to the following conclusions: First of all, as previously mentioned within the literature, algorithm selection and configuration problems have been identified to constitute meta-learning problems [136]. While we approached the question of building experienced-based and learning optimization algorithms and systems initially rather from intuitive notions and through these related them to the aforementioned meta-learning model, the framework of algorithm selection and configuration offers answers for some of the technical challenges when constructing such algorithms and systems. Though, quite notably, our line of research still approaches this problem from a different angle, and to our knowledge the question on the nature of and ways to construct inductive biases in optimization algorithms has of yet not been studied sufficiently in the literature yet.

Secondly, we found that the use of the feature learning capabilities of deep network architectures gained especially an interest in arbitrating between solvers or predicting explicitly solutions in combinatorial optimization [2, 127, 3, 4]. However, surprisingly most of the latter reviewed work in regards to feature-free algorithm selection has mostly been only done within the domain of combinatorial optimization. Even though, prior research on methods from algorithm behavior studies [145, 113, 91] can be considered to have laid a certain foundation for further investigation. Therefore, we want to take in the following the opportunity to advance the state of the art by investigating whether we can propose a pipeline to learn features capable of distinguishing different continuous optimization problems from procedural optimization data. Showing the efficiency of such an approach, could enable us to likewise integrate it into a framework to learn and predict problem-tailored algorithm components from inputs generated during the run-time of an optimization algorithm.

We adopt the procedure of partitioning the search space from prior works [113, 91], however we lay a special focus on a comparative view by imposing different kinds of neighborhood relationships upon the retrieved partitions. By keeping track of problem-specific changes within each cells of the search space partitions, we subsequently train classifiers based upon specialized neural network architectures to learn features capable of separating the different continuous optimization problems within a latent space. We will also include in the following a channel for fitness values, as previous work solely based upon changing solution counts has shown ambiguity for symmetric functions [113] for the task of problem identification.

4.3 Partitioning the Search Space

In the following, we will elaborate on different methods to partition the search space of continuous optimization problems. Specifically, with an emphasis on the different ways how to retrieve search space partitions through clustering methods, and ways to impose a neighborhood relationship on them. The necessity of such a step might not seem obvious within low dimensions, as one might be simply inclined to equally divide the space among each axis into p pieces. However, as the number of partitions would scale exponentially with p^d at high dimensions d through this approach, it would become infeasible if one still wants to maintain a sufficient resolution.

Our principle approach is outlined in Fig. 4.4. For a search space volume $\chi \subset \mathbb{R}^d$, we first generate $D_T = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ samples uniformly random within it. This data set D_T can then be subsequently used as training data for clustering methods to partition the

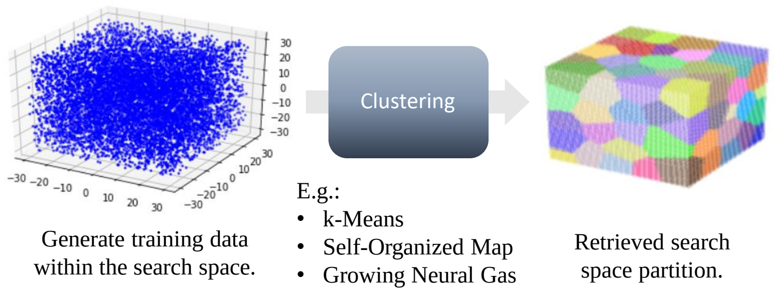
Clustering
Generate training data within the search space.
E.g.:
• k-Means
• Self-Organized Map
• Growing Neural Gas
Retrieved search space partition.

Figure 4.4: Illustration of the search space partitioning step. First training data is generated within the search space, to which subsequently a clustering method is applied to such that to obtain a partitioning of the search space into disjoint and discrete parts.

search space homogeneously. The number of preset clusters N_C can be seen as regulating the resolution of the retrieved partition. Specifically, we investigate in the following as clustering methods *k-means*, the *self-organized map* and the *growing neural gas*.

4.3.1 Unstructured Partitions

Applying the k-means algorithm to a given training data set D_T, with a preset value $k = N_C$ usually results in the algorithm retrieving unstructured clusters after N iterations with centroids $\{\boldsymbol{\mu}_i\}_{i=1}^{N_C}$ and without any further neighborhood relationship being imposed on them. The cluster centroids are usually initialized randomly on the data set and iteratively updated such that

$$\boldsymbol{\mu}_i = (\Sigma_n\, r_{nk} \cdot \mathbf{x}_n)/(\Sigma_n\, r_{nk}), \tag{4.1}$$

where $r_{nk} = 1$ if the closest $\boldsymbol{\mu}_j$ for given $\boldsymbol{x}_n$ has $j = k$, otherwise it is 0. As the k-means algorithm is usually part of many introductory literature [17] as well as standard software packages and libraries for machine learning and statistics [115], we neglect in the following any further more elaborate discussion of it.

4.3.2 Structured Partitions as Maps

Structured clusters can be retrieved using the self-organized map (SOM) technique [82]. As mentioned, this approach has been used in prior work [113, 91] as a way to partition the search space, originally motivated by work on modeling solution populations [145] within continuous evolutionary optimization. In its usual formulation, the SOM imposes a 2d grid structure upon the clusters, such that the total number of clusters is $N_c = N_x \cdot N_y$ and the clusters can be identified through tuples $n_c = (n_x, n_y)$ with $n_c \in [1, N_x] \times [1, N_y]$. In the recursive formulation [82], each of the N_c clusters is identified by a centroid $\boldsymbol{\mu}_i$ (sometimes also called weights or model vectors), being likewise to k-means randomly initialized on the training data set D_T, and updated at each iteration t for a given training data point $\mathbf{x}$ by

$$\boldsymbol{\mu}_i(t+1) = \boldsymbol{\mu}_i(t) + h_{ci}(t)[\mathbf{x}(t) - \boldsymbol{\mu}_i(t)], \tag{4.2}$$

where c is the index of the best matching unit (BMU), i.e. likewise to the k-means algorithm $c = \arg\min_i(||\mathbf{x}(t) - \boldsymbol{\mu}_i(t)||)$, and i being the index of its topological neighbors. Here, h_{ci} is a neighborhood function with

$$h_{ci}(t) = \alpha(t)\ \exp(-||\boldsymbol{\mu}_c - \boldsymbol{\mu}_i||^2/2\sigma^2(t)), \tag{4.3}$$

where $\sigma(t)$ and $\alpha(t)$ are monotonically decreasing functions of t. For the former, according to literature [82] its exact form does not matter, as long as $\sigma(t)$ is a monotonically decreasing function with its value being about half of the grid diameter in the beginning and reduced after about 1000 steps to only a fraction of it. The use of the SOM to partition a high-dimensional space can be motivated as an attempt to topologically 'fold' a low-dimensional space into a higher dimensional one (cf. central column of Fig. 4.5). Note, that this different to exploiting the manifold hypothesis.

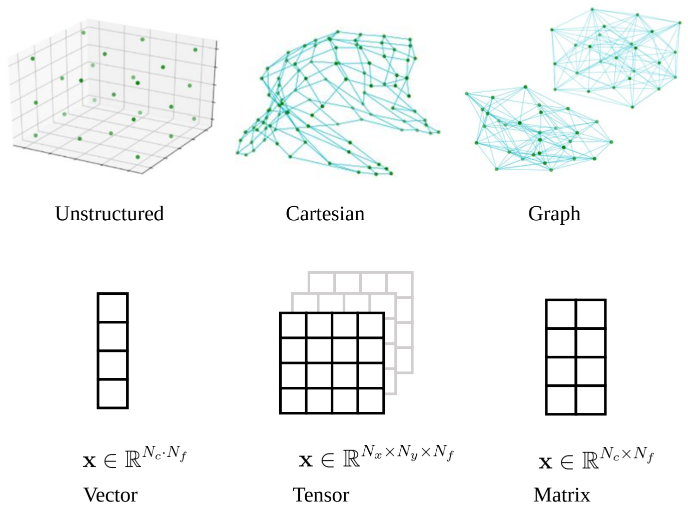
Unstructured
Cartesian
Graph
$\mathbf{x} \in \mathbb{R}^{N_c \cdot N_f}$
Vector
$\mathbf{x} \in \mathbb{R}^{N_x \times N_y \times N_f}$
Tensor
$\mathbf{x} \in \mathbb{R}^{N_c \times N_f}$
Matrix

Figure 4.5: Neighborhood-relationships which can be enforced through different techniques (i.e. k-Means, SOM, GNG & Delaunay). Bottom row: Different data formats which are obtained using the different neighborhood relationships.

4.3.3 Structured Partitions as Graphs

Delaunay Triangulations

The cluster centroids $\{\boldsymbol{\mu}_i\}_{i=1}^{N_C}$ retrieved on the basis of the k-means algorithm can be reinterpreted as nodes of a graph structure. To construct a graph, one simply considers for each cluster with centroid $\boldsymbol{\mu}_i$ its associated decision volume $V(i)$, finds all neighboring volumes $V(j)$ and subsequently builds an adjacency matrix $\mathbf{A}$, with $a_{ij} = 1$ for neighboring pairs (i, j) and $a_{ij} = 0$ for unneighbored pairs (i, j). This procedure is known as Delaunay triangulation. In principle, implementing it into an algorithmic form is not trivial and requires a less simplified approach. However, library implementations are readily available [152].

The Growing Neural Gas

The growing neural gas (GNG) [52] can be considered to be a variation of the former SOM [82]. However, its focus is on evolving a graph of vertices and edges (V, E) which describe the topology of the given data set D_T. Thus, in principle the total number of clusters and edges can dynamically change during the training process. Likewise to k-means and the SOM, the training starts with N_c clusters with positions $\boldsymbol{\mu}_i$ being randomly initialized on the data set D_T. Based upon a randomly drawn data point $\mathbf{x} \in D_T$, the nearest cluster $\boldsymbol{\mu}_1$ and second-nearest cluster $\boldsymbol{\mu}_2$ are determined. If the cluster $\boldsymbol{\mu}_1$ has edges, the ages of the edges are incremented and an error variable $\Delta\text{error}(1) = ||\boldsymbol{\mu}_1 - \mathbf{x}||^2$ is calculated. The cluster $\boldsymbol{\mu}_1$ and its topological neighbors $\boldsymbol{\mu}_n$ are subsequently moved towards the drawn data point $\mathbf{x}$ by fractions ϵ_b and ϵ_n with

$$\Delta\mathbf{w}_{s_1} = \epsilon_b(\mathbf{x} - \boldsymbol{\mu}_1) \quad \text{and} \quad \Delta\mathbf{w}_{s_n} = \epsilon_n(\mathbf{x} - \boldsymbol{\mu}_n), \tag{4.4}$$

analogue to the self-organized map. If $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$ possess an edge, its age is set to 0 and if no edge exists, it is created anew. Edges with an age larger than a_{max} are subsequently removed, and likewise, clusters without an edge are removed from the gas. After a certain number of iterations λ, the gas will insert a new cluster $\boldsymbol{\mu}_r$. This is done, by selecting the cluster $\boldsymbol{\mu}_q$ with the highest error, and subsequently inserting a new cluster half-way at $\boldsymbol{\mu}_r = \frac{1}{2}(\boldsymbol{\mu}_f + \boldsymbol{\mu}_q)$ between the neighbor with highest error $\boldsymbol{\mu}_f$. Subsequently, the old edges are removed and new ones are created. Errors of q and f are lowered by a multiplicative factor α. The new cluster r subsequently inherits the updated error of q. At last, the neural gas decreases all errors by multiplication with a constant d. The algorithm terminates as soon as it has achieved a predefined network size or performance goal.

4.4 The Data Post-Processing Pipeline

The full data post-processing pipeline is illustrated in Fig. 4.6. From running an evolutionary optimization algorithm on different single-objective optimization problems of the form $f : \chi \subset \mathbb{R}^d \to \mathbb{R}$, we first extract raw data in the form of tuples $(\mathbf{x}, f(\mathbf{x}))$ of generated solutions $\mathbf{x} \in \mathbb{R}^d$ and fitness values $f(\mathbf{x})$. We further organize these into tuples

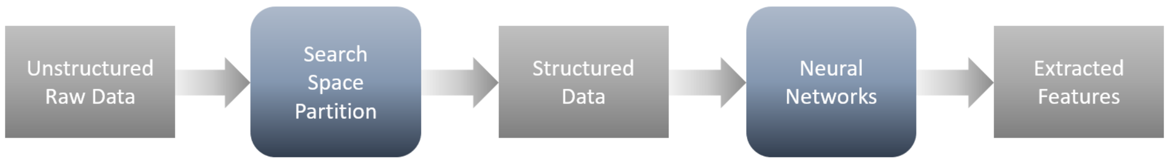

Figure 4.6: Illustration of the data post-processing pipeline. Unstructured raw data descriptive of a solution population P_g^r at generation g and run r in the form of tuples $(\mathbf{x}, f(\mathbf{x}))$ of candidate solutions and fitness values are converted by a search space partitioning method into a structured data format $\mathbf{z}$, which subsequently can be fed to an adequate neural network architecture for feature extraction.

(P_g^r, y), where P_g^r is the population, it is the set of all $(\mathbf{x}, f(\mathbf{x}))$, at generation g and run r and y is a label for a particular optimization problem.

By applying a search space partition of N_c clusters to a solution population P_g^r, we obtain a structured data format in either the form of a vector $\mathbf{z} \in \mathbb{R}^{N_c \cdot N_f}$ for an unstructured k-means partition, the form of a tensor $\mathbf{z} \in \mathbb{R}^{N_x \times N_y \times N_f}$ for a structured map, or a feature matrix $\mathbf{z} \in \mathbb{R}^{N_c \times N_f}$ for structured graphs, where the latter are also further supplied with an adjacency matrix $\mathbf{A}$. Note that N_f is the number of cluster node features, and $N_x \cdot N_y = N_c$. After having converted all solution populations P_g^r into representations $\mathbf{z}_g^r$, we can use these for the subsequent feature learning and extraction step.

Note that in our approach we further process these by explicitly taking the differences $\Delta\mathbf{z}^r = \mathbf{z}_0^r - \mathbf{z}_1^r$. We also consider at most only two features with $N_f = 2$ for each $\mathbf{z}_g^r$, i.e. the sum of all solutions associated to a cluster and the sum of all fitness values associated to a cluster, to which we will refer in the following as *solution channel* and *fitness channel*. Effectively, we can interpret the $\Delta\mathbf{z}^r$ as finite differences, i.e. discrete derivatives in generational change $\Delta g = 1$, of total solutions and total fitness per cluster.

4.5 Neural Nets for Feature Learning

Depending upon the previously used search partition method, we choose in the next step of the processing pipeline the neural network architecture most suitable to process the obtained data format.

4.5.1 Processing of Vector Data

We use for vector data $\mathbf{z} \in \mathbb{R}^{N_c \cdot N_f}$ the multilayer perceptron (MLP) [17] with stacked dense layers of the form

$$\mathbf{h}^{(n)} = \sigma^{(n)}(\mathbf{W}^{(n)}\mathbf{h}^{(n-1)}), \tag{4.5}$$

where $\mathbf{h}^{(n)}$ is the output of the n-th hidden layer, where we have for the input layer $h^{(0)} = \mathbf{z}$, non-linear activation functions $\sigma^{(n)}$, in our case either $\mathrm{ReLU}(\mathbf{x})$:=$\max(\mathbf{0}, \mathbf{x})$ or $\mathrm{SoftMax}(\mathbf{x}) = \exp(\mathbf{x}) / \sum_j \exp(x_j)$ and $\mathbf{W}^{(n)}$ being a trainable weight matrix.

4.5.2 Processing of Tensor Data

For tensor data $\mathbf{z} \in \mathbb{R}^{N_x \times N_y \times N_f}$ extracted using the SOM, we use the convolutional neural network (CNN), which is a special architecture that has been designed to process tensorial data, e.g. such as time-series and images [88]. Key ingredient of it are name-giving convolution operations [55] which can be written as

$$\mathbf{H}^{(n)}_{i,j,k} = \sigma\left(\Sigma_{l,m,p}\mathbf{H}^{(n-1)}_{(i-1)\times s+l,(j-1)\times s+m,p}\mathbf{W}^{(n)}_{l,m,p,k}\right) \tag{4.6}$$

where $\mathbf{H}^{(n-1)}$ is an input tensor, the parameter s is a so called stride and $\mathbf{W}^{(n-1)}$ is a kernel with trainable weights which can be parametrized by (l, m, p), and k being an index for the number of pre-defined filters. Further, we use pooling layers which are defined by

$$\mathbf{H}^{(n)}_{i,j,k} = \max_{m,n}\{\mathbf{H}^{(n-1)}_{(i-1)\times s+m,(j-1)\times s+n,\,k}\}. \tag{4.7}$$

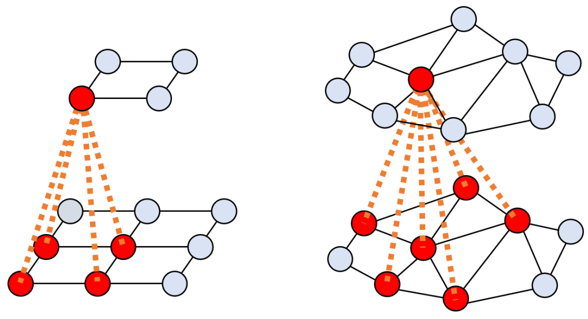
Figure 4.7: In traditional convolution layers (left side), filters have a smaller dimension than the given input domain, and can aggregate features from patches of pre-defined arbitrary size from the input. In Kipf & Welling's graph convolution [80], filters have the same dimension as the input graph, and only aggregate features from the direct neighborhood of a given node.

Both operations are employed in specialized layers as a means to capture the underlying structural correlations hidden within the training data. We neglect a more elaborate discussion and refer to available literature instead [55], in favor of fostering in the following a comparison to their novel analogues for graph data.

4.5.3 Processing of Graph Data

For graph data represented by feature matrices $\mathbf{z} \in \mathbb{R}^{N_c \times N_f}$, we use in our work the recently developed techniques for graph neural networks (GNN) [23, 159]. While a variety of methods [159, 105] have been developed within the recent years, we will employ within our work particularly operations which have been defined in analogy to traditional operations, and became comparably popular. Specifically, we consider graph convolutions [80] as defined by

$$\mathbf{H}^{(n)} = \sigma^{(n)}(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(n-1)}\mathbf{W}^{(n)}), \tag{4.8}$$

with a weight matrix $\mathbf{W}^{(n)} \in \mathbb{R}^{N_f \times N_{f'}}$, further $\mathbf{H}^{(0)} = \mathbf{z}$, the adjacency matrix $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ with self-connections, as well as the degree matrix $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$. The first four multiplicative terms can be interpreted as an aggregation operation over all features of the neighbors of a node (cf. Fig. 4.7). Further, we also use pooling layers, based upon a graph

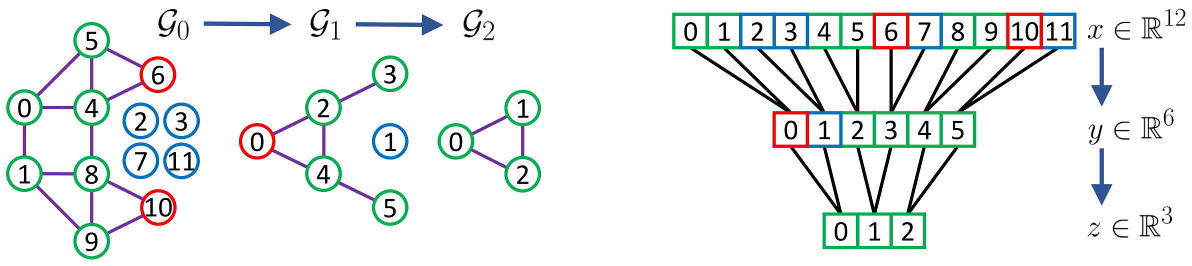

Figure 4.8: The proposed graph pooling operation by Defferrard (2016) [40] based upon the Graclus algorithm [42]. Left panel: First, the input graph is extended by fake nodes (blue), such that subsequently it enables the iterative calculation of graph cuts. Right panel: The node relationships arising through the graph coarsening operation can subsequently be used to convert the graphs into a balanced binary tree, on which node features can be simply pooled at each step in analogy to 1d signals.

coarsening step using the Graclus algorithm [42, 40]

$$[\mathbf{A}_0^*, \cdots, \mathbf{A}_l^*; \mathbf{P}] = GraphCoarsening(\mathbf{A}, l), \tag{4.9}$$

with coarsened adjacency matrices $\mathbf{A}_j^*$ of size $N_C^{*j} \times N_C^{*j}$, with $N_C^{*j} = N_C^*/2^j$, where $j = 0, \cdots, l$ indicates the coarsening level up until $l \leq \ln(N_C^*)/\ln(2)$ and a permutation matrix $\mathbf{P}$ likewise of dimension $N_C^* \times N_C^*$. Note, that the Graclus algorithm extends any given input graph with N_C nodes by adding Δ feature-less fake nodes, such that $N_C^* = N_C + \Delta$ and further permuting the original node arrangement, such that the graph can be converted into a balanced binary tree [40]. Thus, the original feature matrix must be converted by means of applying a permutation matrix $\mathbf{P}$ such that $\mathbf{X}^* = \mathbf{PX}$. Based upon the permuted feature matrix, which is ordered according to the balanced binary tree, pooling operations are then simply conducted branch-wise in analogy to 1d signals with (cf. Fig. 4.8)

$$\mathbf{H}_{i,j}^{*(n)} = \max\{\mathbf{H}_{2i-1:2i,j}^{*(n-1)}\}. \tag{4.10}$$

Note that, even though recent work [97] has been questioning the utility of pooling operations in graph neural networks, we insist on including them nevertheless into our work to keep the analogy to traditional architectures. This is a reasonable decision, as there

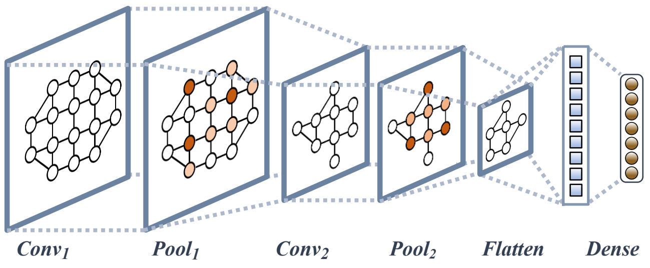

Figure 4.9: The graph-based spatial feature extraction we propose as elaborated in Sec. 4.5.3. Based upon two graph convolutions [80] with 25 and 16 filters, as well as graph pooling operations [40], structured data formats descriptive of a solution distribution are processed such that low-dimensional features descriptive of search behavior are extracted.

does not seem to be a clear consensus established upon this topic yet.

To conclude this section, we remark that we provide additional explanations on the interpretation of graph convolution and pooling operations in Appendix C.

4.6 Experimental Studies

The neural network architectures we use for feature extraction within our study are elaborated in Tbl. 4.1. We implement them based upon available standard frameworks [31] and use custom implementations [80], as well as available ones [40] for the new layer-wise operations. We choose our GNN architecture in analogy to the tried-and-tested architecture for the CNN from Liu et al. (2017) [91], however take for both the liberty of using as classification layers a MLP architecture with bottleneck. We train each network using the Adam optimizer [79] for 1000 epochs in a classification task, where we use for our exploratory studies a training to cross-validation data set split of 80-20 and a batch size of 250, with the categorical cross-entropy loss

$$\mathcal{L} = \sum_{i=1}^{|\mathcal{D}|} \sum_{j=1}^{C} -y_{ij} \log(\hat{y}_{ij}) \tag{4.11}$$

where $\hat{y}_{ij}$ is being the network output for the i-th sample and j-th class, as well as y_{ij} being the true label in the form of a one-hot encoding for each sample i. For the prior search space partitioning step, we generate data uniformly random within a volume of $[-30, 30]^d$ with in total 10,000 training data points. Note that many of the standard single-objective optimization problems are defined on variable search space sizes. To accommodate for them within our experiments, we rescale any extracted solution populations to the size of the aforementioned training volume. This is a reasonable decision, as changing the search space sizes of the problems themselves would otherwise distort the original properties of the benchmark functions. Subsequently, using the obtained partition, we apply the data post-processing pipeline as elaborated in Sec. 4.4. For the clustering methods which partition the search space, we use optimized implementations [115, 129] and train each for 1000 epochs when there is no self-termination criterion implemented by default. The number of clusters is set to $N_C = 100$ for each, which allows us to cover the search spaces in the follow-up at sufficient resolution while not making our experiments overly computationally expensive.

As evolutionary algorithm from which we study the learning of characteristics of the metadata generated thereof we use within our following study a $(\mu+\lambda)$ Evolution Strategy [7, 51]. We set the population and offspring size again to $\mu = 10$ and $\lambda = 10$ to keep the analogy to our experiments in Sec. 3.4.2, as well as use strategy parameters of $\sigma_i \in [0.1, 4]$ and a mutation and crossover probability of 0.5. We randomly initialize the population on the entire search space with the given benchmark functions being of dimensionality $d = 3$, and further generate for each 1000 pairs of parent and offspring generation.

4.6.1 Comparison of Network Performances

In the following, we compare the performances of our approaches in terms of training stability, achieved accuracy and cluster separation. We train the networks upon data

Data Type	Vector	Tensor	Graph
Input Size	$N_c \cdot N_f$	$N_x \times N_y \times N_f$	$N_c \times N_f$
Layer0	Dense(10)*	Conv(5×5×25)	GraphConv(25)
Layer1	ReLU	ReLU	ReLU
Layer2	Dense(50)	MaxPooling(2×2)	MaxPooling(4)
Layer3	ReLU	Conv(3×3×16)	GraphConv(16)
Layer4	-	ReLU	ReLU
Layer5	-	MaxPooling(2×2)	MaxPooling(4)
Layer6	-	Dense(10)*	Dense(10)*
Layer7	-	ReLU	ReLU
Layer8	-	Dense(50)	Dense(50)
Layer9	-	ReLU	ReLU
Output	SoftMax(#C)	SoftMax(#C)	SoftMax(#C)

Table 4.1: The neural network architectures used for the different data types within our study, with the number of classes $\#C$ and * indicating the visualized layer.

generated on a set of symmetric function based upon Eq. (B.5) - (B.8). For the combined solution and fitness channel (S+F), we find that all networks exhibit stable training performance (cf. Fig. 4.10) and are capable of achieving high accuracies in the range of $\sim 80-90\%$. Achieved values for the networks are listed in Tbl. 4.2. Note that within prior available work [113, 91], different search spaces are used and results are only discussed on a qualitative basis. Thus, these do not enable a direct quantitative comparison.

Overall, we find that the GNNs, trained upon graph-representations of the search space, obtained through the GNG and Delaunay triangulations, exhibit highest training performance on the validation sets with accuracies of about $\approx 96\%$. Followed up by the CNN and MLP with about $\approx 84\%$. Looking at the obtained feature spaces in Fig. 4.11, all compared methods exhibit clearly a high ability to separate data inputs generated on the different optimization problems. However, one may argue, that the GNNs have a slightly better capability in retrieving more pronounced and better separated clusters. Note, that by comparing the unstructured as well as Delaunay-based approach, we can cross-check that the higher performance of the GNNs can be particularly attributed to considering additional knowledge about the topological structure of neighboring partition cells. As

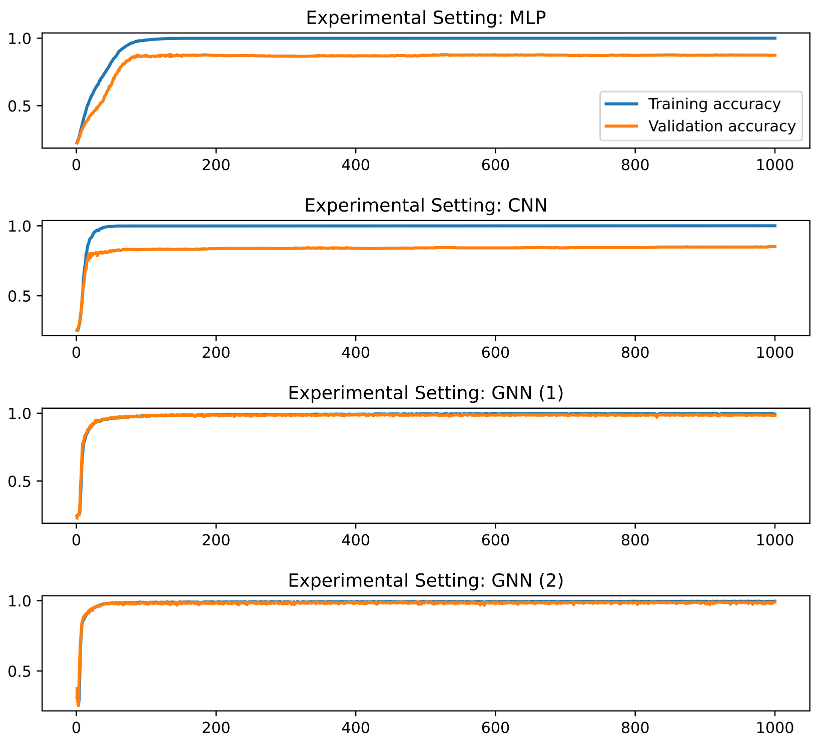

Figure 4.10: Accuracy over the training epochs for the MLP and CNN, as well as the GNN architecture trained on structured data generated using the GNG (1) and Delaunay triangulation (2) as input (from top to bottom). The plots allow us to verify that all architectures exhibit stable and asymptotically convergent training performance.

Architecture	Accuracy (S)	Accuracy (F)	Accuracy (S+F)
MLP	30.06 ± 0.75	83.89 ± 1.45	84.19 ± 0.88
CNN	28.00 ± 2.77	67.19 ± 2.90	84.20 ± 1.54
GNN(1)	26.95 ± 1.48	94.01 ± 0.49	96.90 ± 0.47
GNN(2)	27.06 ± 1.24	93.90 ± 0.92	96.46 ± 0.90

Table 4.2: Accuracy values averaged over 10 training runs from the neural network architectures used for the different data types within our study. Again, for the GNNs (1) indicates input data generated from the GNG, while (2) indicates the Delaunay triangulation.

otherwise, the partition cells are in both approaches the same.

A particularly interesting question is to which regard the solution channel (S) and fitness channel (F) contribute to the training of the networks. We therefore trained all networks separately on each channel and collected likewise accuracy values averaged over 10 training runs on each. The resulting values are listed again in Tbl. 4.2. We find, that training the networks solely based upon changes in the solution channel (S) makes them

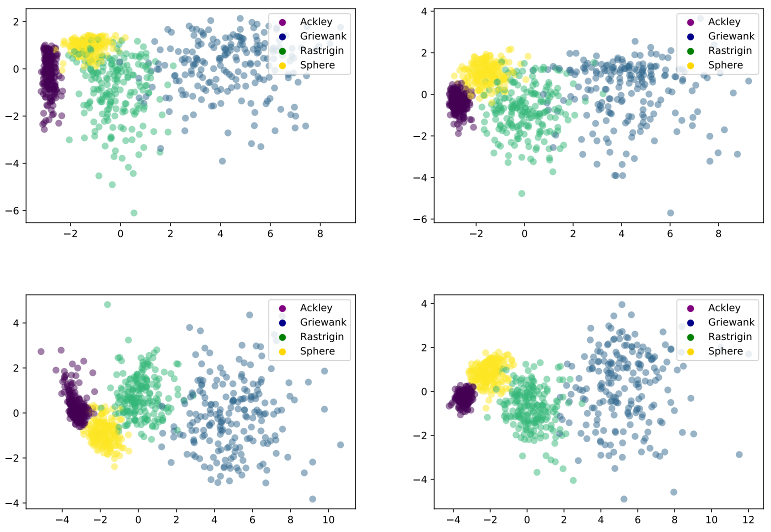

Figure 4.11: LDA-plots of the feature spaces obtained on the symmetric function set from Eq. (B.5) - (B.8) for the MLP, the CNN and GNN (GNG & Delaunay) (from left to right and top to bottom).

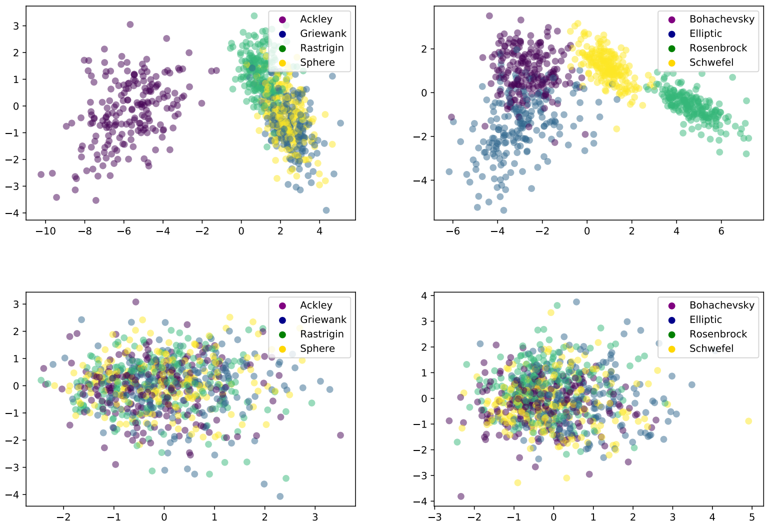

Figure 4.12: Upper row: LDA-plots of the feature spaces for the GNG-based trained GNN using rescaled search space sizes and normalizes fitness values on the symmetric (left) and asymmetric function set (right). Lower row: Obtained feature spaces for training on the solution channel (S).

incapable of separating the inputs from the symmetric function set. With accuracies being only in the range of $\sim 26-30\%$. The bulk of performance gain in the network training can therefore be attributed to the fitness channel (F). With the difference in accuracy for the MLP and the GNNs to the combined channel (S+F) being only about $\approx 1-3\%$. But arguably, the inclusion of the solution channel still contributes to performance improvements. This is most striking for the CNN, where the accuracy gain is about $\approx 17\%$.

While this seems surprising at first glance, considering the fact, that by means of 'folding' the SOM into the higher dimensional space, neighborhood relationships are created which don't reflect the actual structure of the search space, including the solution channel (S) therefore can be considered as helping the network in learning more faithful neighborhood relationships between different search space regions.

4.6.2 Rescaling of Benchmark Functions, Fitness Values and the Set of Asymmetric Functions

At last, we test the behavior of our approach in regards to rescaling the benchmark functions, fitness values and its behavior on asymmetric functions. For the symmetric function set, we find that rescaling the benchmarks to a uniform search space size of $[-5.12, 5.12]^d$ while keeping the algorithm configuration fixed has only a negligible effect. We thus neglect a further discussion of it, however keep it within the further parts and suggest the practitioner to generally consider such an approach, as hyperparameters are mostly tuned to characteristic length scales of a given set of optimization problems. Normalizing the fitness values such that for every benchmark function $0 \leq f(x) \leq 1$, we find that on the symmetric function set the clusters within the feature space order themselves according to the different funnel structures of their benchmark functions (cf. upper left panel in Fig. 4.12). Particularly, clusters are separated into exponential $\sim 1 - \exp(-|x|)$ and quadratic $\sim x^2$ funnel structure. But notably, we find that an intra-cluster separation is still evident. Particularly, between functions with low (Sphere & Griewank) and strong periodic modulation (Rastrigin) superimposed on them in relation to their search space sizes.

At last, we consider a set of asymmetric function set as given by Eq. (B.1) - (B.4). Training our graph neural network upon data generated from these benchmarks, we find initially, that the training does not properly converge. Therefore, we apply the previously elaborated fitness normalization step. Subsequently, we find that the network training properly converges and we likewise find within the feature space, that the clusters separate according to the different funnel structures (cf. upper right panel in Fig. 4.12). However, in comparison to the symmetric function set (cf. lower left panel in Fig. 4.12), we find that training the network solely on the solution channel likewise does not retrieve a feature space in which the optimization problems can be separated (lower right panel).

4.7 Chapter Summary

Contributions

To conclude this chapter, we reiterate again on the contributions we made and motivations we started out originally with.

- **We introduced a pipeline to convert unstructured raw data descriptive of solution populations into a structured data format that can be readily used for processing by learning algorithms.**
- **We proposed a structured graph-based data format which quantizes high-dimensional continuous search spaces and more faithfully reflects the neighborhood relationships thereof.**
- **We proposed a specialized neural network architecture as well as a fitness channel that can be used in conjunction with the structured graph-based data format for feature extraction.**

Summary

First of all, we elaborated on the no free-lunch theorems in optimization. Essentially, from these it is clear that an algorithm can only realize performance improvements reliably in comparison to a baseline configuration, if it incorporates the structure of the problem domain of interest. On the flip-side, this also means that it is not possible to build one monolithic and high-performing universal problem-solving optimization algorithm. As any realized performance gains on a specific problem domain come at the expense of performance degradation on the complementary one. We connected these implications of the no-free-lunch theorems, with the aforementioned meta-learning model. Which required the formation of domain-specific inductive biases that reflect problem structure, as well as metaknowledge representations to mitigate between these such that to realize cross-problem learning.

Looking into the literature in regards to possible ways to realize the latter component, we specifically found an answer within works on algorithm behavior studies and feature-free algorithm selection. Noteworthy, within comparative studies algorithm selection and configuration problems have been identified to constitute meta-learning problems within optimization [136]. However, we find that existing methodology for continuous optimization is specifically lacking. Particularly, as most work on algorithm behavior studies is rather interested in characterizing optimization algorithms themselves, while feature-free algorithm selection is mostly only concerned with problems in combinatorial optimization. We therefore decided to advance the methodology to this regard. Specifically, by proposing a pipeline to convert unstructured solution populations into structured data formats, a graph-based data format to represent solution populations in search spaces of continuous optimization problems, methodology to generate these graph structures using either a growing neural gas or Delaunay triangulation, as well as novel graph neural networks architectures for feature extraction and the explicit inclusion of an input channel to take fitness values into consideration for identifying different optimization problems.

We found that the graph-based format in conjunction with the new input channel is capable of achieving higher performance than more conventional approaches. And we demonstrated, that these are indeed capable of learning features that reflect global structural properties of problem-dependent search behavior on different benchmark problems.

Interestingly, the effect of our considered graph convolution operation [80] can be interpreted as calculating an interpolation of fitness values and solution distribution by means of propagating fitness values and calculating virtual population members within the graph structure. For the time-being, we will only make a brief note of this fact and refer for a more elaborate discussion to Chapter 7 of this thesis instead.

To follow-up the work of this chapter, we could consider different directions. First of all, the most obvious one would be to consider more benchmark functions. While in principle, this would be desired to fully stress test our method, we have also already seen some limits to this regard. Specifically, we saw that the set of asymmetric benchmark functions only shows good training performance when the inputs in the fitness channel are rescaled to the range of 0 to 1. However, in practical scenarios the maximum fitness value of each optimization problem may not be directly deducible. Alternatively, we may consider only to normalize fitness values generation-wise. However, we have found that the feature spaces retrieved by this procedure are only barely more informative than when we completely drop the fitness channel. We could further consider different input transformations or architectures which include additional processing steps and operations for the spatial feature extractor. But for the time being, the most natural option which lies at hand would be to instead work with what is already given. Thus, we will therefore look into the following aspects: Improving the learning of spatial anisotropies already contained in the raw data, as well as using more inputs by going forward in time through processing the temporal component of metadata generated by the optimization algorithm.

CHAPTER 5

SPATIO-TEMPORAL ACTIVITY RECOGNITION OF SEARCH BEHAVIOR

5.1 Spatio-Temporal Data Processing

We previously elaborated in Chapter 4 on spatial feature extraction from metadata generated by optimization algorithms. Particularly, we were looking into this by considering finite differences of structured data formats generated from start and successor population P_0 and P_1 of the optimization algorithm on different optimization problems. While we have seen, that it can be an effective approach to enable predictive algorithms to learn features that are capable of distinguishing between different optimization problems, we have also seen that it struggles when the inputs in the fitness channel are not sufficiently informative.

To improve the performance in such scenarios, we consider in this chapter the usage of additional inputs by going forward into time. Particularly, we propose network architecture extensions to enable spatio-temporal data processing. This kind of modeling approach for instance has popularity for problems e.g. concerning urban mobility and activity recognition in multimedia data-streams, however can be considered to be natural to be extended to problems concerning population-based optimization, as fitness landscape measures [76] and convergence theorems emphasize spatio-temporal notions [120].

We therefore will consider the following approach: We keep our aforementioned network proposed in Chapter 4 for spatial feature extraction at each time step, but explicitly extend it with methodology to model the temporal component. Specifically, we will look in the following into methods proposed for time series classification, as well as activity recognition in multimedia data-streams. Additionally we also look into whether the spatial feature extractor can be improved, by extending it to explicitly learn spatial anisotropies.

5.2 Learning Spatial Anisotropies

5.2.1 Graph Attention Operations

In the following, we will use as mentioned for spatial feature extraction the graph-based approach as introduced in Chapter 4. Besides graph-based data formats in the form of tuples of adjacency matrices and feature matrices $(\mathbf{A}, \mathbf{X})$ with $\mathbf{A} \in \mathbb{R}^{N_c \times N_c}$ and $\mathbf{X} \in \mathbb{R}^{N_c \times N_f}$, we used a specialized graph neural network architecture [23, 159], particularly based upon graph convolutions (GCNs) [80] as defined by

$$\mathbf{H}^{(n)} = \sigma^{(n)}(\hat{\mathbf{A}}\ \mathbf{H}^{(n-1)}\mathbf{W}^{(n)}), \tag{5.1}$$

where $\mathbf{W}^{(n)} \in \mathbb{R}^{N_f \times N_{f'}}$ is a weight matrix, $\mathbf{H}^{(0)} = \mathbf{X}$, as well as $\hat{\mathbf{A}}$ being an effective adjacency matrix descriptive of the structure of the graph. While in principle, different methods to perform convolution operations on graphs exist (e.g. [105, 159]), the approach based upon the operation by Kipf & Welling [80] is particularly elegant, because in principle it is based upon a low-order heat diffusion model. Thus, the first two multiplicative terms in Eq. (5.1) can be interpreted as computing 'heat' propagation within the graph structure. Note, that due to this physical interpretation, these convolutions do not take in account the importance which difference node features might have.

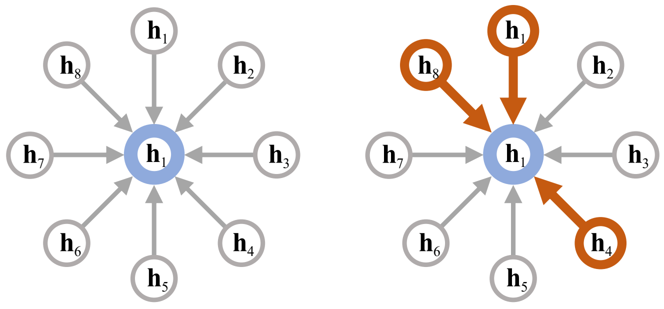

Figure 5.1: Illustration of the difference between graph convolution [80] (left) and graph attention operations [147] (right). While the graph convolution only propagates node features based upon the number of neighborhood connections, graph attention operations explicitly allow nodes with more important features to be valued higher when features are propagated in a neighborhood.

To address this issue, Veličković et al. (2017) [147] introduced graph attention operations (cf. Fig. 5.1) in which the elements of the effective adjacency matrix are explicitly learned through a regression function $\hat{A}_{ij} := f(\mathbf{h}_i, \mathbf{h}_j)$ during training. The regression function can parametrized as given by

$$f(\mathbf{h}_i, \mathbf{h}_j) = \frac{\exp\left(\text{LeakyReLU}\left([\mathbf{h}_i^T\mathbf{W}||\mathbf{h}_j^T\mathbf{W}]\mathbf{a}\right)\right)}{\exp\left(\sum_{k\in\mathcal{N}_i}\text{LeakyReLU}([\mathbf{h}_i^T\mathbf{W}||\mathbf{h}_k^T\mathbf{W}]\mathbf{a})\right)}, \tag{5.2}$$

where $\mathbf{a} \in \mathbb{R}^{2f'}$ is a vector with trainable weights, $||$ a concatenation operation, $\mathcal{N}_i$ the neighborhood of a reference node i inclusive of itself, $\mathbf{h}_i, \mathbf{h}_j, \mathbf{h}_k$ are row vectors of the feature matrix $\mathbf{X}$ and the LeakyReLU activation function is being given by $\sigma(\mathbf{x})=\max(\alpha\mathbf{x}, \mathbf{x})$ with $\alpha = 0.2$. To stabilize the training of a graph attention network, Veličković et al. (2017) [147] additionally suggest to use instead an average of K individual attention operations, so called 'attention heads', such that Eq. (5.1) is modified to

$$\mathbf{H}^{(n)} = \sigma^{(n)}\left(\frac{1}{K}\sum_{k=1}^{K}\hat{\mathbf{A}}_k\ \mathbf{H}^{(n-1)}\mathbf{W}_k^{(n)}\right). \tag{5.3}$$

In principle, given the aforementioned equations for the graph attention operation, we have all ingredients together to apply it in our graph neural network architecture from

Chapter 4 in place of the original graph convolution operation by Kipf & Welling [80].

5.3 Temporal Data Processing

Because we are interested in the following to process metadata in a time-dependent manner, we want to use ideally spatial feature extraction methods in conjunction with methodology for temporal data processing. In principle, two lines of research can be identified which are relevant to this regard. The first one being approaches which have been specifically proposed for time series classification tasks, while the second one being ANN-RNNs stemming from activity recognition tasks in multimedia data streams.

5.3.1 CNN-based Time Series Classification

While a great variety of different approaches have been proposed within the literature [161], we focus in the following particularly on discriminative deep network methods that can be trained in an end-to-end manner [45]. Particularly popular to this regard are architectures which are centered upon simple 1d convolution operations. Given a multi-variate time series signal $\mathbf{X} \in \mathbb{R}^{T \times d}$ of d dimensions of length T, with $\mathbf{H}^{(0)} = \mathbf{X}$ convolution operations are then iteratively applied such that

$$\mathbf{H}_{ij} = \sigma(\Sigma_{l,p}\mathbf{H}_{(i-1)\times s+l,p}\mathbf{W}_{l,p}) \tag{5.4}$$

with non-linearity $\sigma(\cdot)$, s being a stride and the filter matrix being given by $\mathbf{W} \in \mathbb{R}^{T \times d}$. Note that, within this framework the multi-variate nature of the time series is accounted for by means of treating these as multi-channeled signals, while the time dimension is kept being treated as 1-dimensional. For our purposes, two particular CNN-based architectures are of most interest. Where the first one has been dubbed within the literature as *Fully Connected Network* (FCN) [155] and the second one simply as *Encoder* (ENC) [128]. Both architectures are high-performing for either multivariate or short-time series, while at the same time lightweight in terms of trainable parameters [45].

We briefly elaborate in the following upon the structures of the FCN and ENC architectures, however will refer the interested reader for in-depth details to available literature instead [155, 128]. Both architectures are based upon three convolution blocks, which incorporate convolution and normalization operations as well as an activation function. The ENC architecture additionally applies pooling operations between the first, second and third block. The FCN architecture ends with a global average pooling layer applied to each filter of the last convolution block, while the ENC architecture ends with an attention layer, followed up by a dense layer with a sigmoid activation function and at last a normalization step. The FCN architecture has demonstrated within the comparative study conducted by Fawaz et al. (2019) [45] high performance on multivariate times-series, while the latter ENC architectures strives on short-time series.

5.3.2 ANN-RNN-based Spatio-Temporal Activity Recognition

Combinations of ANNs for feature extraction and RNNs for sequential processing have established themselves for applications of activity recognition in multimedia data-streams [123, 43], vehicle behavior prediction [160] and pre-miRNA classification [142]. The most common instantiation is in the form of a CNN-LSTM which is directly trained in an end-to-end manner. Within our work, we specifically propose GNN-RNN architectures for processing metadata generated by optimization algorithms. Particularly, using the aforementioned GCN and GAT operations from Sec. 5.2 for spatial feature extraction. The particular RNN we use is either a simple recurrent neural network (sRNN), a gated recurrent unit (GRU) or long short-term memory (LSTM). Considering an input sequence $\mathbf{x}_1, \cdots, \mathbf{x}_T$, for T time-steps, a simple recurrence equation can be formulated through [55, 14]

$$\begin{aligned} \mathbf{h}_t &= \tanh(\mathbf{b} + \mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t), \\ \text{and } \hat{\mathbf{y}}_t &= \sigma(\mathbf{c} + \mathbf{V}\mathbf{h}_t), \end{aligned} \tag{5.5}$$

where $\mathbf{h}_t$ is the hidden output and $\hat{\mathbf{y}}$ the predicted label for input element $\mathbf{x}_t$, as well as $\mathbf{b}$, $\mathbf{c}$, $\mathbf{W}$, $\mathbf{V}$ being bias vectors and weight matrices and $\sigma(\cdot)$ an activation function. This architecture transforming the entire input sequence into an output sequence of equal length T is known as being many-to-many. Neglecting the second equation and only calculating the last hidden output $\mathbf{h}_T$, we retrieve an architecture which is known as many-to-one. Within our implementation [31], we will neglect the second equation and only use $\mathbf{h}_t$ as output. In comparison to this simple RNN (sRNN), the state-of-the-art can be considered to be posed by the long short-term memory (LSTM) network. The LSTM can be described in total by six update equations

$$\begin{aligned}
\mathbf{f}_t &= \sigma(\mathbf{W}_f \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f), \\
\mathbf{i}_t &= \sigma(\mathbf{W}_i \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i), \\
\tilde{\mathbf{C}}_t &= \tanh(\mathbf{W}_C \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_C]), \\
\mathbf{C}_t &= \mathbf{f}_t * \mathbf{C}_{t-1} + \mathbf{i}_t * \tilde{\mathbf{C}}_t, \\
\mathbf{o}_t &= \sigma(\mathbf{W}_o \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o)
\end{aligned} \tag{5.6}$$

and at last the computation of the hidden state

$$\mathbf{h}_t = \mathbf{o}_t * \tanh(\mathbf{C}_t), \tag{5.7}$$

which can be used analogous to the second equation in (5.5) to calculate the step-wise output. Note, that $*$ denotes a component-wise product and $[\cdot, \cdot]$ is a concatenation operation. A key difference of the LSTM to the sRNN is the explicit introduction of gating mechanisms in the first three lines of Eq. (5.6) through memory cells $\mathbf{C}_t$. These not only dynamically control the influence of past hidden states on future ones, but are also an essential milestone in enabling and mitigating problems with the training of RNNs on data sets with long-term dependencies. However, due to the large amount of parameters introduced, LSTMs cannot always be considered to be a reasonable choice. To keep the novelties introduced by LSTMs, but at the same time prune their complexities, the so called Gated Recurrent Unit (GRU) has been alternatively introduced. Defined by in total four equations

$$\begin{aligned}\mathbf{z}_t &= \sigma(\mathbf{W}_z \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]),\\ \mathbf{r}_t &= \sigma(\mathbf{W}_r \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]),\\ \tilde{\mathbf{h}}_t &= \tanh(\mathbf{W}_h \cdot [\mathbf{r}_t * \mathbf{h}_{t-1}, \mathbf{x}_t]),\\ \mathbf{h}_t &= (1 - \mathbf{z}_t) * \mathbf{h}_{t-1} + \mathbf{z}_t + \tilde{\mathbf{h}}_t,\end{aligned} \tag{5.8}$$

the gated recurrent unit shares obvious similarities to Eq. (5.6), however skips intermediate steps introduced by the calculation of the memory cell $\mathbf{C}_t$.

5.4 Experimental Studies

5.4.1 Generation of Synthetic Metadata

As no agreed upon reference data sets exist of yet, we again have to reside in the following to create one explicitly by ourselves. For this reason, we use in the following the symmetric optimization function set from Chapter 4 based upon Eq. (B.5) - (B.8), which consists out of four functions with either quadratic $\sim x^2$ or exponential $\sim 1{-}\exp(-|x|)$ global symmetric funnel structure as well as different periodicities superimposed on them. As evolutionary algorithm (EA) we use again the $(\mu{+}\lambda)$ Evolution Strategy [135]. However, we initialize the start population only in the corner of the search space defined by $[s_b/2, s_b]^d$, where s_b is the upper boundary and $d = 3$. As otherwise, the crossover operator will lead to rapid convergence due to the centered position of the global optimum.

We configure the EA with $\mu{=}\lambda{=}10$, i.e. a population size of 10, strategy parameter $\sigma \in [0.1, 2.0]$ for a reference size of $s_b = 5.12$ and crossover and mutation probability of 0.5. To convert metadata in a structured data format, we explicitly use the graph-structure evolved using the growing neural gas from Chapter 4 to map out the search space. Using the nodes of this graph structure, a solution population P_g can then be

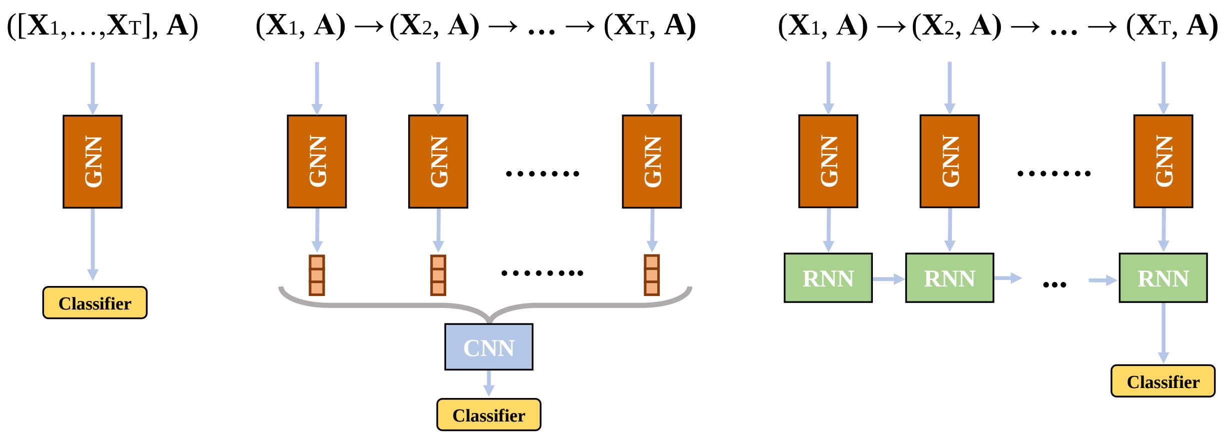

Figure 5.2: Simplified schematics of the end-to-end architectures used within our work. From left to right: Graph neural network under simple concatenation of input graphs (GNN), graph neural network as spatial feature extractor with CNN-based time series classification architecture on top (GNN-FCN & GNN-ENC) and at last GNN-based spatial feature extraction with recurrent network for temporal processing on top (GNN-RNN).

converted into a structured data format $\mathbf{z}_g \in \mathbb{R}^{N_c \times N_f}$ by finding the closest graph node to each solution in P_g, and summing up solutions associated with them. Note, that we do not consider a fitness channel as introduced in Chapter 4, but only concentrate on the channel of solution counts. Similar to Turkey & Poli (2012) [145], we distinguish between growth and non-growth regions. Explicitly, by calculating for successive generations $g \to g+1$ the vector $\Delta \mathbf{z}_g$ (cf. Chapter 4) and further to enrich the training data set also $\mathbf{z}^*_g$. Where the first one encodes changes in the solution distribution under application of the EA, and the second one invariant parts. We explicitly use within our experiments time-series generated from running the EA for 20 iterations, and generate 1000 samples for each benchmark. To accommodate for the varying search space size, we rescale σ by a factor $s_b/5.12$ accordingly like suggested in Chapter 4.

5.4.2 Network Configuration and Training

Because we will evaluate and compare in the following in total 12 different neural networks architectures, we have to ensure that they are configured and trained in a way such that to enable a fair comparison. Though, admittingly our emphasis is not on finding out

Table 5.1: Hyperparameters and testing ranges for the different architectures. Note that for simplicity RNN* denotes sRNN, GRU and LSTM, as well as Args* hidden nodes and attention heads.

	Epochs	**Batches**	**Loss Function**	**Optimizer**	**Learning Rate**	**Args***	
Range	-	2^n	-	-	$q \times 10^z$	2^n	n
GCN	300	256	Cross-Entropy	Adam	0.001	-	-
GAT	300	256	Cross-Entropy	Adam	0.001	-	3
GCN-FCN	300	16	Cross-Entropy	Adam	0.001	-	-
GCN-ENC	300	12	Cross-Entropy	Adam	0.00001	-	-
GCN-RNN*	300	256	Cross-Entropy	Adam	0.001	8	-
GAT-FCN	300	16	Cross-Entropy	Adam	0.001	-	3
GAT-ENC	300	12	Cross-Entropy	Adam	0.00001	-	3
GAT-RNN*	300	256	Cross-Entropy	Adam	0.001	8	3

which architecture is in theory the best, but instead which one can be considered to be reasonably efficient and performant under a fixed budget.

Therefore, at best we only minimally modify the previously elaborated architectures. Within all experiments, we use a GNN either based upon GCN or GAT operations for spatial feature extraction. The architecture of the spatial feature extractor is based upon the originally proposed one by Liu et al. (2017) [91], however with the adjustments for processing graph data as suggested by us in Chapter 4 (cf. Fig. 4.9) using graph convolution and coarsening layers [42, 40], as well as 10d output features. For the GAT-based feature extraction, we simply replace the convolution layers. Note, that GAT architectures explicitly introduce the number of attention heads K as a hyperparameter. For temporal data processing, we either use a CNN or RNN-based component as illustrated in Fig. 5.2. For training the architectures, we set an upper fixed budget of 300 epochs and use for all architectures the Adam optimizer [79]. We vary the batch size according to 2^n between 8 to 256. In principle, the usage of large batch sizes of 256 can be justified for most architectures. However, the FCN and ENC architectures exhibit unstable training behavior when doing so. We therefore can verify as previously suggested [45], that smaller batches of size 16 and 12 exhibit significantly better training performance for each. For the learning rate, we likewise can confirm a previous suggestion made for the ENC architecture [45]. For the number of attention heads in the GAT architectures, we consider K in the interval $[3, 8]$. We find that $K = 3$ proves to be a good choice. As the performance gain with more attention heads is at best only minimal and mostly accompanied with significant computational overhead. Noteworthy, the GAT architectures may also benefit from smaller learning rates. However, in the investigated range of $q \times 10^z$ for $q \in \{0.5, 1.0\}$ and $z = -5, \cdots, -3$, this gain was at best only minimal. At last, we tune the hyperparameter for the number of hidden nodes in the RNN architectures in the range of 4 to 128. We find that the usage of 8 hidden neurons exhibits the best performance. Note, that this seems to somewhat reflect the product of number of classes $\#C$ and node features F. Effectively, it can be interpreted as calculating two scores for each benchmark function at successive steps. All final hyperparameters are listed in Tbl. 5.1.

5.4.3 Statistical Performance Comparison

For performance comparison we reside in the following to the validation accuracy after 300 epochs under consideration of a 80-20 training to test data split and the estimated wall clock time in reference to a NVIDIA Tesla V100-SXM2-16GB provided as a cloud instance [116]. The latter wall clock time is a necessity, as even though if a particular configuration may exhibit high training performance, it can be potentially rendered infeasible for practical applications by the produced computational overhead.

Because retrieved validation accuracies are subject to stochastic variation, we therefore reside in the following towards taking the mean from in total 30 training runs. As however, most of the the performance values are distributed non-Gaussian, we further need to conduct additional statistical testing. Particularly, we reside in the following to what one may dub in analogy to Fawaz et al. (2019) [45] as a Kruskal-Wallis-Holm test, meaning that we first use the Kruskal-Wallis test to reject the null hypothesis, and subsequently perform tuplewise unpaired Wilcoxon rank-sum tests, for which we additionally use the Holm-Bonferroni correction.

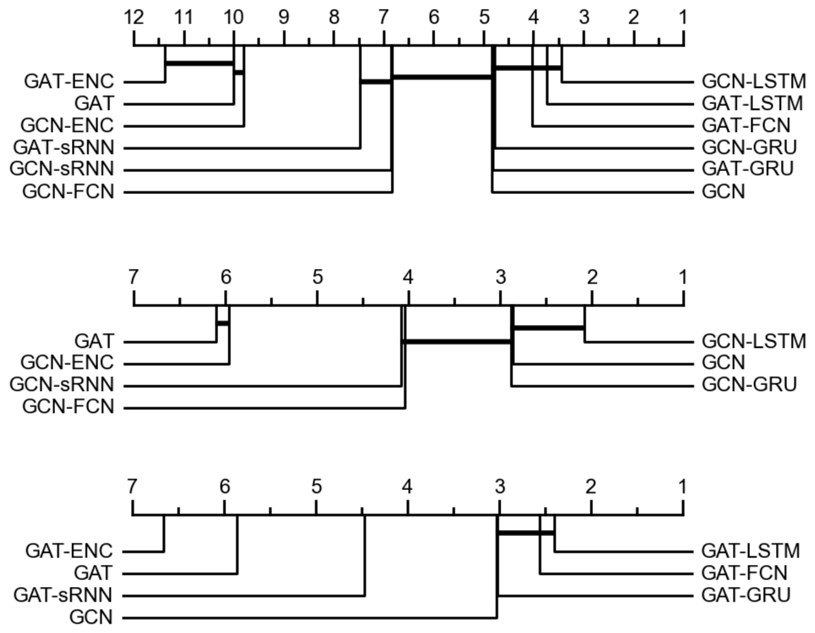

Figure 5.3: Critical difference diagrams for the comparison of all GCN and GAT architectures from Tbl. 5.2 and 5.3 on the 20 time-steps long data set (top), as well as the subset of GCN (center) and GAT architectures (bottom).

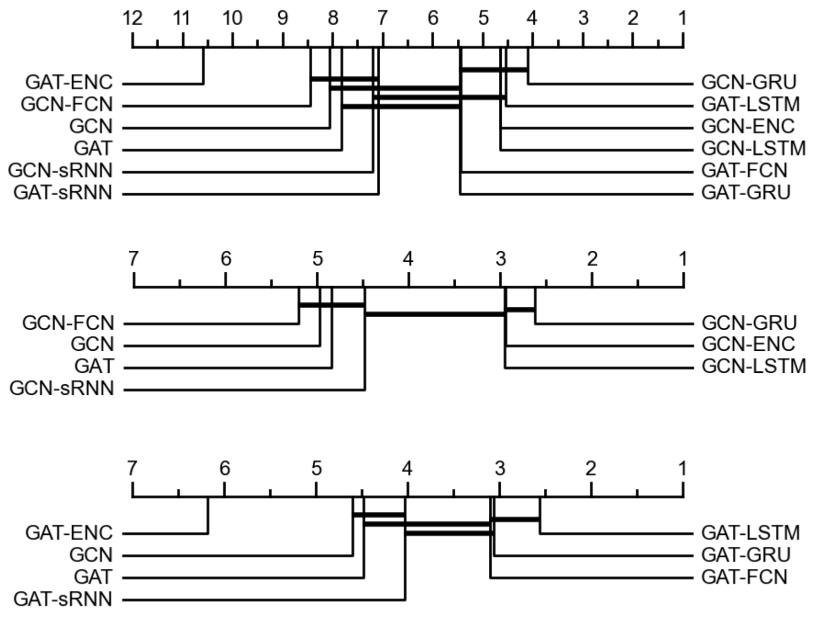

Figure 5.4: Critical difference diagrams for the comparison of all GCN and GAT architectures from Tbl. 5.2 and 5.3 on the 10 time-steps long data set (top), as well as the subset of GCN (center) and GAT architectures (bottom).

To visualize the performance of the different architectures we use critical difference diagrams. Meaning that, architectures are ranked from 1 to M, where M is the total number of compared architectures, and statistically similar architectures are indicated by a connected line. To calculate the ranking $r(\mathcal{D})$ of a performance data set $\mathcal{D}$ for each architecture, we derived

$$r(\mathcal{D}) = M - \frac{M-1}{|\mathcal{D}|(N-|\mathcal{D}|)} \sum_{i=1}^{|\mathcal{D}|} W_i, \tag{5.9}$$

in which we perform the sum over the total number of observations of a performance data set $|\mathcal{D}|$ with N being the total number of measurements from all performance data sets, further W_i being the number of observations a particular observation i in $\mathcal{D}$ dominates in the remaining $M-1$ data sets.

5.4.4 Discussion of Results

Recorded performance values and critical difference diagrams are given in Tbl. 5.2 & 5.3 and Fig. 5.4. First of all, the most obvious observation is that the GAT architectures create a large computational overhead with an increase in training time of up to ≈ 2800 mins in reference to the GPU instance. Note however, that due to practical reasons we use for computations of the GAT architectures TPU instances provided by the cloud computing service instead. Thus reducing the computational overhead by at least 50-70%. Comparing the architectures with GCN and GAT as domain-level feature extractors altogether, we find that the performance increase in regards to mean validation accuracies is at best only minimal. Ranging for the GAT-FCN and GAT-LSTM from $1.83 - 0.14\%$ on the 20 time-step long series, where for the latter these are not statistical significant, and on the 10 time-steps long series from 1.48% for the GAT-FCN, 0.05% for the GAT-sRNN and 0.14% for the GAT-LSTM, where however none is statistical significant.

Table 5.2: Comparison of all tested architectures in terms of rank (R) of the performance data sets, mean validation accuracy and estimated wall clock time (WCT) in reference to a cloud GPU instance calculated from in total 30 trials each.

Architecture	**WCT (20)**	**Acc (20)**	**Acc (10)**	**R_Σ(20)**	**R_Σ(10)**
GCN	$\approx$ 0.78 min	68.84%	62.53%	4.84	8.06
GAT	$\approx$ 130.00 min	63.56%	62.51%	10.01	7.82
GCN-FCN	$\approx$ 20.00 min	65.67%	62.22%	6.83	8.45
GCN-ENC	$\approx$ 21.36 min	63.95%	64.05%	9.81	4.64
GCN-sRNN	$\approx$ 8.75 min	65.82%	62.86%	6.85	7.19
GCN-GRU	$\approx$ 6.25 min	68.42%	64.33%	4.77	4.09
GCN-LSTM	$\approx$ 6.25 min	67.96%	64.03%	3.44	4.65
GAT-FCN	$\approx$ 2600 min	67.50%	63.70%	4.03	5.43
GAT-ENC	$\approx$ 2400 min	62.13%	60.65%	11.38	10.59
GAT-sRNN	$\approx$ 2775 min	65.38%	62.91%	7.48	7.09
GAT-GRU	$\approx$ 2750 min	67.00%	63.61%	4.82	5.45
GAT-LSTM	$\approx$ 2460 min	67.68%	64.17%	3.74	4.54

Table 5.3: Ranks of the subsets of either GCN or GAT architectures for the 20 step or 10 step long time-series.

	Time Steps	**GCN**	**GAT**	**FCN**	**ENC**	**sRNN**	**GRU**	**LSTM**
GCN	10	4.97	4.84	5.20	2.95	4.48	2.62	2.95
	20	2.86	6.10	4.04	5.97	4.08	2.88	2.08
GAT	10	4.60	4.47	3.10	6.18	4.03	3.06	2.56
	20	3.03	5.85	2.56	6.66	4.47	3.02	2.40

Overall, considering the large computational overhead and the minimal performance improvement, the usage of GAT-based architectures cannot to be justified. Thus, due to to their high practicality, we therefore take in the following our GCN architectures under closer scrutiny. We find that the GCN-GRU and GCN-LSTM exhibit highest performance in the 10 and 20 time-steps long data sets, followed up by either the GCN architecture for the long time-series or ENC architecture for the short-time series. Interestingly, the ENC architecture performs significantly worse on the long time-series data set, while the FCN architecture performs slightly better. This is somewhat surprising, as we are dealing in both scenarios with comparably short time-series. Though, somewhat reflecting results from [45]. An interesting observation is also, that the GCN architecture under input concatenation significantly strives on longer time-series, while the input concatenation with

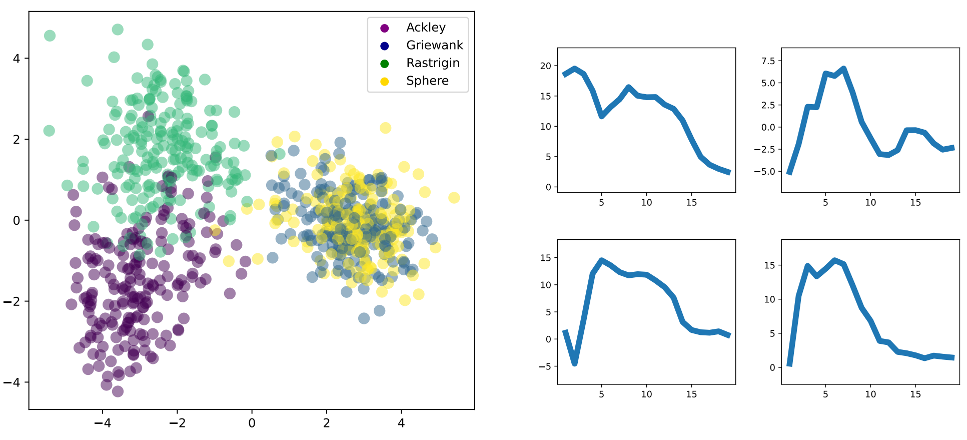

Figure 5.5: Left panel: Visualization of learned features from the concatenated GCN architecture for the 20 time-steps long data set. Notably, the benchmark functions are separated according to whether they have a strongly pronounced quadratic funnel structure or not. Right panel: Median CAM values learned for the different benchmark functions over 20 time-steps (from top left to bottom right: Ackley, Griewank, Rastrigin & Sphere). We find that the benchmark functions with quadratic funnel structure share similar functional dependencies between the 2nd and 8th time-step.

GAT layer performs in comparison in both scenarios suboptimal.

5.4.5 Interpretation of the Learned Metadata Characteristics

A natural interest arising when dealing with learning tasks involving complex data sets is to interpret what a predictive method has actually learned. We therefore analyze obtained feature spaces and time-dependent characteristics. A representative feature space obtained using LDA and the GCN architecture under input concatenation is plotted in the left panel of Fig. 5.5. Notably, the network learned to separate benchmark functions according to whether they have globally a strong quadratic funnel structure with $\sim x^2$ or not. While for instance, the Rastrigin function in Fig. 5.6 has the same funnel structure, it also features a strong periodicity superimposed on top of it. Thus, significantly distorting the behavior of the EA on a global level. Note, that this is unlike to the Griewank function, which has a less pronounced periodicity.

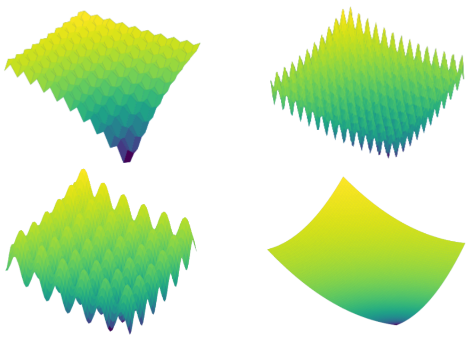
Figure 5.6: Slope topologies of the different benchmarks with global optimum in the bottom corner of which we analyze the learned metadata characteristics by the FCN architecture of the problem-dependent search behavior with the CAM. From top left to bottom right: Ackley, Griewank, Rastrigin & Sphere.

At last, we use the so called class activation map (CAM) for the GCN-FCN to peek into what parts of the time-series are making the most important contribution for predicting a certain class c. The CAM value for a class c is calculated by

$$\mathrm{CAM}_c(t) = \sum_m w_m^c A_m(t), \tag{5.10}$$

where $A_m(t)$ is the output of the m-th filter of the FCN architecture and w_m^c is the weight connecting the filter output with the output neuron z_c of the SoftMax classification layer. Note, that for this analysis we explicitly removed intermediate hidden layers of the classification head used in the prior experiments.

For the calculated CAM values in the right panel of Fig. 5.5, we find that for the three functions with quadratic funnel structure $\sim x^2$ the CAM assigns a high importance to the interval between the 2nd and 8th time-step which reflects convergent behavior. This phase is elongated to the 15th generation for the Rastrigin function, which features significant distortions on a global level. While for Griewank, which is globally similar to Sphere, the local distortions become only important around the 12th step due to the algorithm becoming stuck. Note, that for the Ackley function with a flat and concave funnel structure of $\sim 1-\exp(-|x|)$, the early regions of the time-series have higher importance. Overall, we find that our calculated median CAM values in the right panel of Fig. 5.5 can be interpreted as reflecting slope characteristics of the different benchmarks.

5.5 Chapter Summary

Contributions

In conclusion of this chapter, we summarize again our key contributions and findings:

- **We introduced the learning of spatial anisotropies of activity within continuous search spaces with graph attention operations to our previously proposed graph neural network architecture.**
- **We compared extensions to our graph-based neural network architecture for temporal data processing using CNN and RNN-based approaches for time series classification and activity recognition.**
- **We used techniques for interpretable machine learning to understand what time series parts are most important for a learning task and suggest that we can also draw implications from it about problem-dependent search behavior of an optimization algorithm.**

Summary

First of all, we found that among our proposed architectures GCN-GRUs and LSTMs demonstrate most consistent performance on variable time series lengths and high efficiency in terms of accuracy and training time. While the GAT architectures which we proposed can show performance improvements in the case of the GAT-FCN, GAT-sRNN and GAT-LSTM, these are only statistical significant for the GAT-FCN on the 10 time-steps long data set, however in no case it is significantly outperforming any other architectures. Thus, the use of the GAT architectures can be largely considered to be infeasible.

We further also demonstrated that the CAM for interpretable learning with time series data can help us to understand, as well as reflects global properties of the different problem-dependent behaviors of an optimization algorithm. Specifically, we saw that for the benchmark problems with quadratic $\sim x^2$ and exponential $\sim 1 - \exp(-|x|)$ funnel structure significantly different regions of the time series are emphasized by the CAM, while variations in the local structure of the former, depending upon their severity either lead to elongations or additional smaller regions of interest in the plot of the CAM values over time.

In the following-up chapter we will take a look at a shape optimization problem relating to simulation-based design optimization. Within this study, we aim to explicitly predict an operator as inductive bias in terms of configurations for the CMA-ES algorithm. This scenario has the advantage, that unlike with our previously investigated methodology in Chapters 3 and 4, it allows the construction of predictive components and biases in a joint manner. Further, as it effectively emulates a possible application problem, it allows us to understand some of the pitfalls we could encounter when building and deploying such a system.

CHAPTER 6

PREDICTING OPERATOR CONFIGURATIONS

6.1 Design Optimization

In this chapter we will emulate problems encountered in simulation-based design optimization. The goal of simulation-based design optimization can be formulated as being the improvement of physical properties of real-world objects to satisfy desirable performance goals and external design constraints [131]. In principle to setup experiments we need a way to represent objects, a method to manipulate object shapes and generate new designs, an algorithmic routine which can optimize design representations in an automated manner and at last a way to generate performance and constraint values of the physical objects which are being optimized.

6.1.1 3D Object Representations in Computer Graphics

While there are different ways to represent physical objects in 3d computer graphics, such as voxel or boundary representations [19], we reside in the following to polygonal surface meshes, which model the surfaces of 3d objects by means of a collection of edge-wise interconnected polygons. In our case, based upon a triangulated surface mesh (cf. Fig. 6.1), where each polygon is a triangle consisting out of vertices described by 3d coordinates and their connecting edges. This representation is necessary for the following up applica-

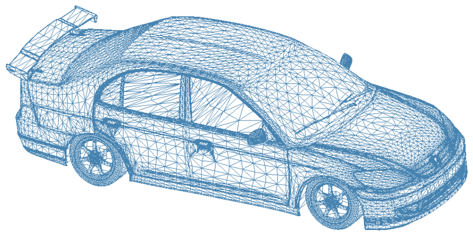

Figure 6.1: Wireframe representation of a three-dimensional model from the ShapeNet library [29].

tion of shape deformation techniques, which have the advantageous property that for the calculation of e.g. aerodynamic properties, they still guarantee that the modified object representation maintains mesh continuity. As in the following, we will only operate on vertex renderings of objects, we additionally perform a Poisson-disk sampling [34] using MeshLab [32], such that we obtain more uniformly covered surface representations.

6.1.2 Shape Deformation Techniques

The Free-Form Deformation (FFD) technique [126] is a particular type of shape deformation technique which has been well-established and popularized within the literature. While originally motivated by mimicking the process of sculpturing for computer-aided design systems, mathematically the idea of FFD can be described as embedding an object which is to be deformed into a parallelepiped lattice as control structure, colloquially just referred to as the 'control volume'. The control points in the lattice grid are parametrized by triples (i, j, k) such that

$$\mathbf{c}_{ijk} = \mathbf{x}_0 + \frac{i}{l}\mathbf{u}_1 + \frac{j}{m}\mathbf{u}_2 + \frac{k}{n}\mathbf{u}_3, \tag{6.1}$$

where $\mathbf{x}_0$ is the origin, $\mathbf{u}_i$ are the basis vectors of the lattice, and $l+1$, $m+1$ and $n+1$ are the number of lattice layers in each respective direction. The lattice imposes a local coordinate system into the control volume such that every point $\mathbf{x} \in \mathcal{V}$ can be described by a triplet $(\alpha_1, \alpha_2, \alpha_3)$, where the coefficients α_i can be be calculated by means of a

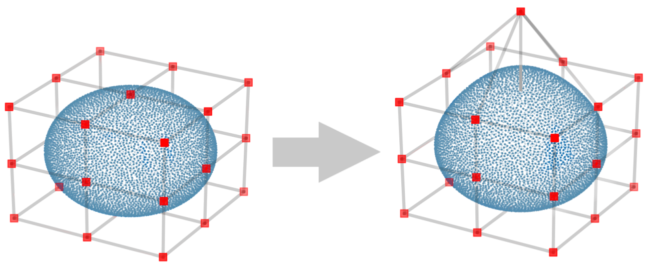

Figure 6.2: The Free-Form Deformation technique (FFD) applied to a test model based upon a control volume spanned by a 3x3x3 lattice. Left: The unaltered sphere and control volume based upon the initialized 3x3x3 lattice. Right: Sphere deformed by an altered control volume which was created by displacing one control point upwards.

triple vector product [126]. Pointwise free-form deformations are then expressed in terms of a trivariate tensor product of the shifted lattice control points $\mathbf{c}'_{ijk}$ with basis functions $N_i^l(\alpha)$ [131] such that

$$\mathbf{x}' = \sum_{i=0}^{l} \sum_{j=0}^{m} \sum_{k=0}^{n} \mathbf{c}'_{ijk}\, N_i^l(\alpha_1) N_j^m(\alpha_2) N_k^n(\alpha_3). \tag{6.2}$$

The basis functions are usually expressed within the free-form deformation in terms of Bernstein polynomials [126]

$$N_i^l(\alpha) = \begin{pmatrix} l \\ i \end{pmatrix} (1-\alpha)^{l-i} \alpha^l. \tag{6.3}$$

The calculation of shape deformations is now straightforward within this framework. We first modify the control points $\mathbf{c}'_{ijk}{=}\mathbf{c}_{ijk}{+}\delta\mathbf{c}_{ijk}$, and subsequently use Eq. (6.2) to retrieve the updated points descriptive of the deformed object such that $\mathbf{x}' \in \mathcal{M}'$. An example of a simple free-form deformation is given in Fig. 6.2.

6.1.3 The Covariance Matrix Adaption Evolution Strategy

As within design optimization we want to improve the physical properties of objects by generating new designs from manipulating digital representations of real-world objects, we ideally want to achieve this in an automated and systematic manner. Thus, avoiding any further manual input required by the design engineer.

Particularly helpful to this regard are within design optimization evolutionary optimization algorithms. This is largely due to their stochastic nature helping them in escaping local optima. However, they also have the advantage that they demonstrate the capability to retrieve unconventional and surprising solutions [89].

While we have previously already seen that a large variety of different algorithms have been proposed within the literature, a particularly popular state-of-the-art algorithm within design optimization is the Covariance Matrix Adaption Evolution Strategy (CMA-ES) [62]. It is particular useful for optimization problems which have a pronounced global convex structure, as its internal mechanism can in principle be interpreted as an attempt in fitting a normal distribution to the global optimum of a given problem (cf. Fig. 6.3).

A simplified version of the algorithm is outlined in Alg. 2. In the following, we briefly outline its main functionality but refer for more details to available literature instead [62, 59, 60]. The principle mechanism of the (μ_w, λ)-CMA-ES is to iteratively sample λ solutions from a multivariate normal distribution centered at a centroid $\mathbf{m}^{(g)}$ and parametrized by a variance or 'step-size' σ and covariance matrix $\mathbf{C}$. Based upon the λ sampled solutions the μ best solutions are selected and used to recalculate a new weighted mean $\mathbf{m}^{(g+1)}$. Subsequently, the step-size σ and covariance $\mathbf{C}$ are updated. Notably, the CMA-ES also uses so called evolution paths $\mathbf{p}_\sigma$ and $\mathbf{p}_c$, which accumulate adaptive trends from successive iterations. This mechanism makes it largely different to related model-based optimization algorithms such as EDAs [87].

6.2 Experimental Studies

In our following experimental studies we investigate the possibility of predicting operator configurations as inductive biases in a problem-specific manner for the CMA-ES algorithm. Specifically, we will proceed within our study in three steps: First, we investigate whether we can derive operator configurations, i.e. pairs of step-size and covariance matrix $(\sigma, \mathbf{C})$, which can serve as inductive biases, i.e. good re-initializations on synthetic benchmark functions. In a second step, we setup two target shape optimization scenarios in which the cabin size of a car shape is increased by either heightening or widening the model. Thus, the subsequent task in a third step is to jointly construct predictive component and bias and subsequently estimate the bias component in a problem-specific manner merely from procedural data generated during the run-time of the CMA-ES algorithm.

6.2.1 Studies on Synthetic Benchmark Functions

To investigate whether we can derive operator configurations which can serve as the bias component, we consider in the following a study on synthetic benchmark functions.

To model the bias we explicitly keep statistics about individual algorithm runs and derive from them improved configurations for the step-size and covariance matrix $(\sigma^{(g)}, \mathbf{C}^{(g)})$. Note, that we explicitly neglect the evolution paths $\mathbf{p}_\sigma$ and $\mathbf{p}_c$ such that we do not overly constrain the adaptiveness of the CMA-ES algorithm during run-time.

In principle, different schemes on how to retrieve more optimal configurations $(\sigma,\mathbf{C})$ as initializations can be tested to this regard. The first obvious one would be to select the operator configuration after the maximum amount of iterations $g^* = N$ has passed. The second one would be to select the configuration at generation g at which the fitness reduction is maximal, i.e. $g^* = \arg\max_g \Delta f_{\min}(g)$, where $\Delta f_{\min}(g) = f_{\min}^{(g)} - f_{\min}^{(g+1)}$ and $f_{\min}^{(g)}$ is the minimum fitness at generation g. A third option would be to select median values

Algorithm 2: (μ_w, λ)-CMA-ES

```
Initialize $\mathbf{m} \in \mathbb{R}^n$, $\sigma \in \mathbb{R}_+$
Initialize $\mathbf{p}_\sigma$=0, $\mathbf{p}_c$=0, $\mathbf{C}$=$\mathbf{1}$, $g$=0
repeat
    for $k = 1, \ldots, \lambda$ do
        $\mathbf{x}_k \sim \mathbf{m}^{(g)} + \sigma \cdot \mathcal{N}(\mathbf{0}, \mathbf{C})$
    end
    $\mathbf{m}^{(g+1)} = \sum_{i=1}^{\mu} w_i \mathbf{x}_{i:\lambda}$, $\sum_{i=1}^{\mu} w_i$=1, $w_i > 0$
    $\mathbf{\Delta} = \mathbf{m}^{(g+1)} - \mathbf{m}^{(g)}$
    $\mathbf{p}_\sigma, \sigma, \mathbf{p}_c, \mathbf{C} \leftarrow$ update($\mathbf{p}_\sigma, \sigma, \mathbf{p}_c, \mathbf{C}, \mathbf{\Delta}, \mathbf{x}_{i:\lambda}$)
    g ← g+1
until stopping criterion is met;
```

over g for $\sigma^{(g)}$ and $\mathbf{C}^{(g)}$ each.

We test each of these different configurations on three different benchmark functions from Eq. (B.5), (B.7) & (B.8) corresponding to the Sphere, Griewank and Rastrigin function. Note, that while this set of functions is not exhaustive, its purpose is instead to have sufficiently different characteristics such that we can gain necessary key insights on the properties of the different approaches. To get additional insight, we further vary the given evaluation budget from $g = 10$ to 50 and the dimensionality from $d = 3$ to 10. We initialize for the experiments the CMA-ES at the slope of each benchmark function, i.e. such that $\mathbf{m}^{(0)} = (0, \cdots, s_b, \cdots, 0)$, where s_b is the search space boundary of each interval $[-s_b, s_b]$ and we set $\mu = \lambda = 10$, as well as $\sigma^{(0)} = 1$ and $\mathbf{C}^{(0)} = \mathbf{1}$ for the initial runs. Notably, we use a large population size such that we exhaustively sample the search space.

In total, we consider for each benchmark function three different experimental setups with parameter settings $(g, d) \in \{(10, 3), (50, 3), (50, 10)\}$. The results of our study are illustrated in Fig. 6.4. In the first setup, we find that on all benchmark functions, the reinitialization methods are capable of generating performance improvements. The effect is most pronounced for the Griewank and Sphere function, which have a strong pronounced global quadratic $\sim x^2$ funnel structure. For Rastrigin's function, which has a periodicity superimposed on top of it, the performance gain is still evident, but comparably less sig-

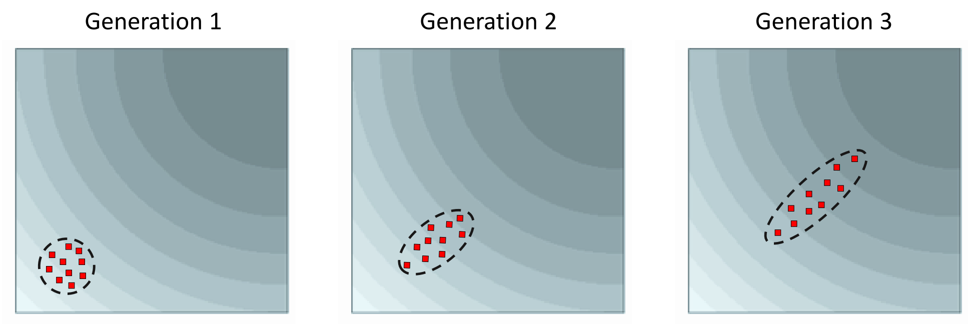

Figure 6.3: Illustration of the covariance matrix adaption mechanism of the CMA-ES algorithm. From an initial sampling derived using an isotropic multivariate normal distribution, the best found solutions are selected upon which the covariance matrix and subsequently mean are recalculated, such that the parameters of the multivariate normal distribution are iteratively adapted to the problem-structure.

nificant. Within our second series of experiments at increased evaluation budget, we find that particularly the reinitialization based upon choosing the final parameters and median tend to lead to ill-convergence behavior. For the Sphere function this is particularly evident, due to the default run converging too well within the given evaluation budget, thus the majority of the budget is spent on generating parameter configurations which are suboptimal for reinitialization. For the Rastrigin function, the multimodal structure becomes more evident at increased evaluation budget and thus parameter configurations are retrieved which are globally unfit. In the last series of experiments, we increase the dimensionality of the problems. We find that this largely helps in improving the convergence properties of the different reinitialization methods again. This can be attributed due to the fact that increased dimensionalities largely negate the effects of increased evaluation budget by enlarging the search space.

In conclusion, in all experiments the most consistent behavior was shown by choosing as initialization the operator configuration which leads to the greatest reduction in fitness costs. More generally, reinitializing the operators seems to be particularly efficient when dealing with a scenario of large search spaces. This was especially evident for Griewank's function, where we did not observed any pathological behavior of the reinitializations in

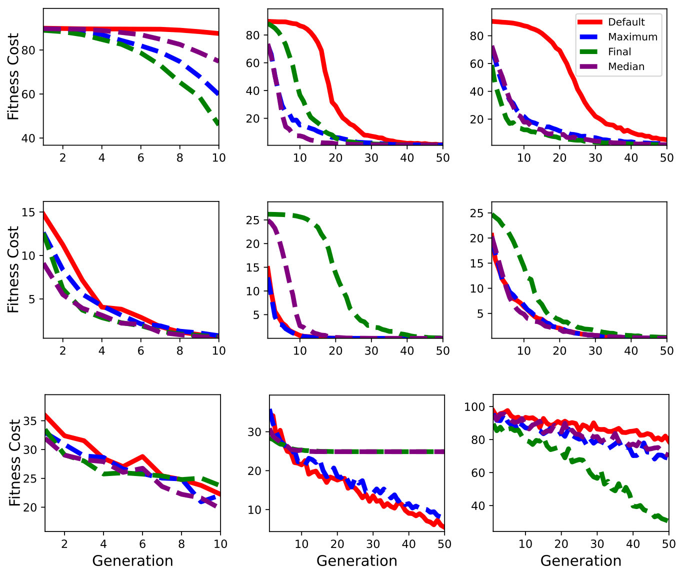

Figure 6.4: Median fitness cost curves over 100 runs for the Griewank, Sphere and Rastrigin function (top to bottom) for experimental settings of evaluation budget g and dimension d of $(10, 3)$, $(50, 3)$ and $(50, 10)$ (left to right). Continuous red curves: Default runs. Dashed curves: Reinitialization methods. Particularly: Median configuration (purple), final configuration (green) and maximum fitness decreasing configuration (blue). Notably, across all tested experimental settings the last reinitialization method demonstrates the most consistent performance improvements.

all of the experiments. But also for the Sphere and Rastrigin at increased dimension. We note, that for the former Griewank function, pathological initializations can be also retrieved when significantly increasing the evaluation budget. But otherwise, we observe that the aforementioned most fitness cost reducing parameter initialization is still most effective.

6.2.2 Target Shape Optimization Problems

As previously elaborated, within design optimization physical objects are usually represented through point clouds by means of meshes. Meaning that essentially, one can quantify the similarity between two point clouds X and Y through distance measures $d(X, Y)$.

We consider for this reason in the following the modified Hausdorff distance [164]:

$$d_H(X, Y) = \max \left\{ \sum_{x \in X} \min_{y \in Y} d(x, y), \sum_{y \in Y} \min_{x \in X} d(x, y) \right\}. \tag{6.4}$$

In principle, this metric measures the similarity between two shapes, by means of minimizing the maximum distance of the sum of the distances $d(x, y)$ of mutually nearest points $x \in X$ and $y \in Y$. As performing design optimization experiments based upon computational simulation models can be considered to be a highly expensive venture, one commonly resides for benchmarking to computationally cheap target shape optimization problems with objective functions which are defined based upon the Hausdorff distance, such that:

$$f(\mathbf{x}) := f(\mathbf{x}; Y) = d_H(t(\mathbf{x}; X), Y), \tag{6.5}$$

i.e. we inscribe the source shape X into a control volume defined by a deformation function t which maps design variables to positions of control points in the lattice. Notably, the consideration of such a simplified problem is also beneficial as it allows one to constrain the degrees of freedom in simulation-based optimization through an additional objective such that design solutions can be retrieved which are overall more meaningful.

6.2.3 Control Lattices for Free-Form Deformation

In the following, we design two target shape optimization problems using the free-form deformation technique. For this purpose, we first need to setup a control point lattice. We take a representative car model [29] and setup a control volume around it with a lattice of $2 \times 9 \times 2$ in width, length and height (cf. Fig. 6.5). Within the design problem we construct, we enforce the requirement of increasing the cabin volume. Depending on the particular final target shape, this could either require an increase in the height or width of the midsection of the control volume.

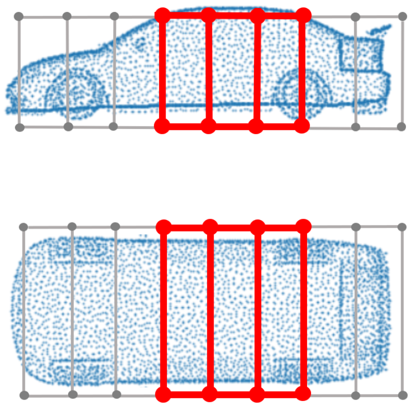

Figure 6.5: Control point setup used within our study. Because we are only interested modifying the mid-section of the shape, we keep the control points in the front and rear section fixed (grey) while only the ones in the mid-section are movable (red).

To generate the control lattice, we first calculate the mean $\mathbf{c}$ of the point set X representative of the car model $\mathcal{M}$. Subsequently, we calculate the norms of the basis vectors $\|\mathbf{u}_i\|$ =max($\{x_i|\mathbf{x} \in X\}$)−min($\{x_i|\mathbf{x} \in X\}$). We place the origin of the lattice at $\mathbf{x}_0 \equiv \mathbf{c} - \frac{1}{2}\sum_{i=1}^{3} \mathbf{u}_i$. Subsequently, the control lattice can be parametrized using Eq. (6.1), where we set $l = 2$, $m = 9$ and $n = 2$ accordingly. Thus, the complete unconstrained control lattice has 108 degrees of freedom with 36 control points. In the following, we significantly reduce the degrees of freedom to 3 for the purpose of our studies. First, by concentrating on the mid-section $4 \leq j - 1 \leq 7$, the degrees of freedom can be reduced by a factor 4/9 to in total 48 with 16 control points. By enforcing mirror symmetry on the width axis we half them to 24 with 8 control points. Fixing the control points on the length axis and fixing the lower control points in height we retrieve 12 degrees. Finally, at each width-length plane among the height dimension, we subsume the control point to one each. Thus, retrieving the aforementioned 3 design variables distributed to 2 control points.

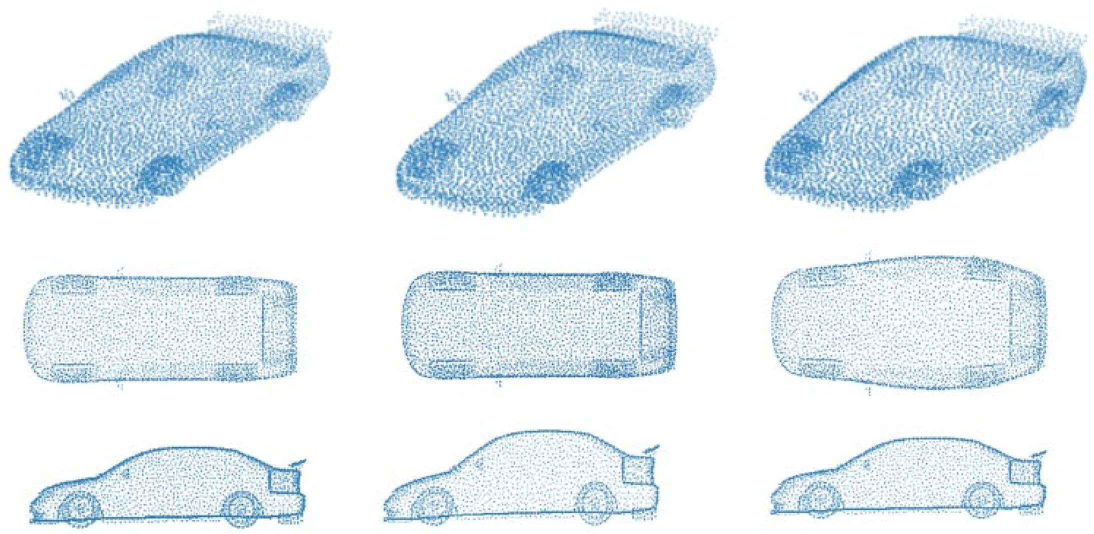

Figure 6.6: Column 1: Original source shape which is used within our studies to generate target shapes. Column 2-3: Two different target shapes used within our study generated by either increasing the height (column 2) or width of the cabin volume (column 3).

6.2.4 Predicting Operators for Shape Optimization Problems

We begin our first experiments by constructing two target shape optimization problems reflecting the previously elaborated scenario of increasing the cabin volume along different dimensions. To construct the first one, we set the design variable for the height to $\mathbf{c} = (0, 0, 20)$, and for the second one the design variables for the width to $\mathbf{c} = (20, 20, 0)$ (cf. Fig. 6.6). We setup our optimization problems using Eq. (6.5) and calculate the deformed target shapes $Y_1 = Y(\mathbf{c}_1)$ and $Y_2 = Y(\mathbf{c}_2)$ accordingly. The objective is then to retrieve the design variables $\mathbf{x}^*$ such that

$$\mathbf{x}^* \equiv \arg\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x}; Y_i), \tag{6.6}$$

where $Y_i \in \{Y_1, Y_2\}$. To model the bias component we select the operator configuration which created the highest decrease in fitness. We collect data from 200 runs where one half of it are from the first target shape problem and the other are from the second, and subsequently use it to train a network to directly predict improved operator configurations.

Explicitly, by converting procedural optimization data into a structured format $\mathbf{z}_g^r$ for each generation g and run r, based upon methodology developed within our previous work of Chapter 4 using search space partitions. Particularly, we use a partition created using the growing neural gas, which divides up the search space using a graph-structure. Based upon the retrieved partitioning, we can count the number of solutions accumulated in each partition cell and aggregate them into a vector $\mathbf{z}_g^r \in \mathbb{R}^{N_C}$ to obtain a structured data format, where N_C is the number of partition cells. Note, that we pre-process the obtained unstructured optimization data first by explicitly centering it through shifting it with a vector $\mathbf{v} = (2.5, 2.5, 2.5)$ and subsequently rescaling it by a factor 12. This ensures that after converting it into a structured data format that the retrieved entries thereof are less sparsely populated, i.e. the part of the search space where the most action is happening is covered at a better resolution.

As we previously denoted the algorithm configuration as $(\sigma, \mathbf{C})$, we want to predict in the following the step-size and covariance matrix accordingly. Note, that without further thought this would require us to predict $1+d^2$ parameters. However, due to the symmetry properties of the covariance matrix $\mathbf{C} = \mathbf{C}^T$ we can reduce the number of parameters to effectively $1+d(d+1)/2$. Additionally, to ensure positive definiteness, we use the activation functions $\mathbf{C}_{ij} = \mathrm{ReLU}(\overline{\mathbf{C}}_{ij})$ for the diagonal elements $i = j$ such that $\mathbf{C}_{ij} \geq 0$, and linear outputs $\mathbf{C}_{ij} = \overline{\mathbf{C}}_{ij}$ to allow negative values for the off-diagonal elements $i \neq j$. Similarly, we can use for the step-size $\sigma = \mathrm{ReLU}(\overline{\sigma})$ to ensure positiveness. Note, that this approach is somewhat similar though more pragmatic than mixture density networks [16, 84]. In principle, we find it to be working sufficiently well for our considered scenario.

We train the multilayer perceptron to approximate the regression function $f(\Delta\mathbf{z}) = (\sigma, \mathbf{C})$, where we use the finite difference $\Delta\mathbf{z}^r = \mathbf{z}_0^r - \mathbf{z}_t^r$. After trying out different values, we find that the offset $t = 2$ is most effective, as it still captures enough change of activity within the search space, while still allowing us to make early enough a prediction for an operator

$$\sigma = \mathrm{ReLU}(\overline{\sigma}) \qquad \mathbf{C} = \begin{bmatrix} \mathrm{ReLU}(\overline{\mathbf{C}}_{11}) & \overline{\mathbf{C}}_{21} & \dots & \overline{\mathbf{C}}_{d1} \\ \overline{\mathbf{C}}_{12} & \mathrm{ReLU}(\overline{\mathbf{C}}_{22}) & \iddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ \overline{\mathbf{C}}_{1d} & \dots & \dots & \mathrm{ReLU}(\overline{\mathbf{C}}_{dd}) \end{bmatrix}$$

Figure 6.7: Prediction strategy we use to retrieve operator configurations $(\sigma, \mathbf{C})$ in conjunction with the multilayer perceptron. We use for the step-size and diagonal elements of the covariance matrix ReLU activation functions to ensure positiveness, while for the off-diagonal elements we use linear ones to allow for negative values.

with sufficient accuracy. Note, that while in Chapter 4 we used both, a solution and fitness channel, we will consider only the former to keep the input size small. We also will not use the time-dependent architectures from Chapter 4, as within our considered scenario only small amounts of training data are accessible. To keep the number of parameters low, we use in the following a multilayer perceptron. The input size of it is naturally dictated by the one of the input file format which in this case has 100 dimensions. To keep the analogy to the previous chapters, we first calculate again 10 dimensional features in the first layer followed-up by a ReLU activation function. Note, that this layer does not have to perform sophisticated computations as we only want to get a hint about the directionality the optimization is taking (i.e. whether the control points are moving towards the first or second target shape). Subsequently, we perform the high-dimensional regression task to predict the algorithm configuration for which we dedicate 200 neurons that are followed up by a ReLU activation function. We find this setting to work well for our experiments, however acknowledge that possibly less neurons might be also sufficient. For the output layer, we choose $1+d(d+1)/2$ neurons, with activation functions as elaborated previously for the prediction of the step-size and covariance matrix.

6.2.5 Discussion of Results

After having jointly constructed predictive and bias component, we can conduct our experiments in the following. We therefore predict operator configurations as biases in a

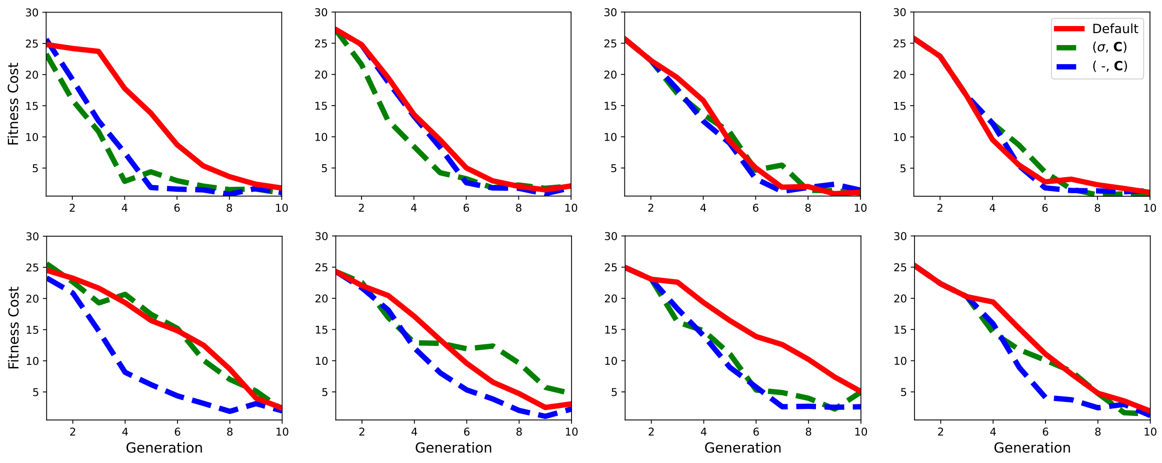

Figure 6.8: Retrieved median fitness curves over 10 runs for the operator configuration prediction scenario. From left to right: With generational offset from -3 to 0. From top to bottom: For the first target shape from Fig. 6.6 in column 2 (top) and the second target shape in column 3 (bottom) from Fig. 6.6. Red continuous curves: Default runs. Dashed curves: For predicted configurations, with full configuration $(\sigma, \mathbf{C})$ (green), and only covariance matrix $\mathbf{C}$ (blue).

	Shape #1			Shape #2		
Offset	Default	Fixed	Same	Default	Fixed	Same
-3	126.28	142.69	**73.52**	147.61	150.17	**87.72**
-2	108.34	143.00	**102.75**	123.59	137.82	**100.34**
-1	103.54	115.42	**97.36**	155.40	**125.42**	**105.45**
0	91.69	**90.20**	**90.20**	131.58	**107.16**	**107.16**

Table 6.1: Table of integrated fitness values (IF) for the predicted operators for the first and second shape (from left to right) corresponding to the values plotted in Fig. 6.9.

problem-specific manner during run-time. Our results are plotted in Fig. 6.8, where in the top row we display median results over 10 predictions on the first target shape problem, and the bottom row the predictions on the second target shape problem. We further consider scenarios in which we either predict the full configuration $(\sigma, \mathbf{C})$ (dashed green) or just the covariance matrix $\mathbf{C}$ (dashed blue), as well as reset the CMA-ES with the predicted configurations at different generational offset from -3 to 0 (left to right).

Overall, we find that modeling the bias component by predicting the full operator configuration, i.e. the covariance matrix $\mathbf{C}$ and step-size σ, leads to a result which is partly

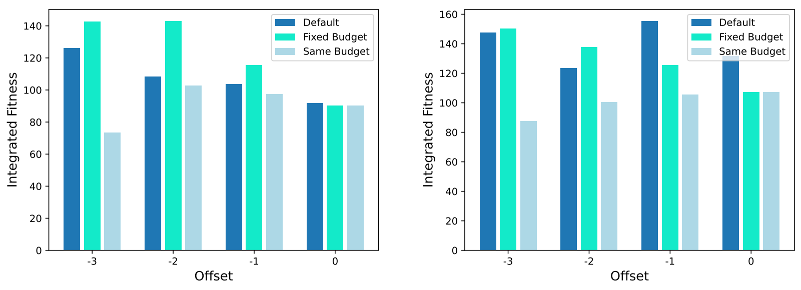

Figure 6.9: Bar plots of integrated fitness values (IF) for the predicted operators for the first and second target shape, for default configuration (blue), prediction with fixed evaluation budget (cyan) and with same evaluation budget (light blue), initialized at different generational offsets, corresponding to the plots from Fig. 6.8

performing worse in comparison to the baseline (red continuous) due to overshooting. More consistent performance increases were realized using just the predicted covariance $\mathbf{C}$ component. This effect can be reasonably explained, as a high step-size prediction can in principle amplify the effects of wrongly predicted covariance matrix $\mathbf{C}$. Thus, only focusing on the latter allows the adaptive properties of the CMA procedure to correct any wrong prediction during run-time. However, notably this additional performance gain realized through resetting the CMA-ES is partly negated by the expended function evaluations required to generate a prediction first. To compare the efficiency of the operators for a fixed evaluation budget, we use the area under the convergence curves in terms of integrated fitness values in Fig. 6.9 and Tbl. 6.1.

6.3 Chapter Summary

Contributions

To conclude this chapter, we reiterate again on the contributions we made and motivations we started out with as well as emphasize the implications we can draw from our experiences in regards to designing experiments and the results we subsequently achieved from them.

- **We introduced a scenario focusing on algorithm configuration which has**

the advantage that it allows the construction of both, predictive and bias component, in a joint manner.

- **We used our previously proposed structured data format to directly predict operator configurations for the CMA-ES, for which we also proposed a prediction strategy for step-size and covariance matrix $(\sigma, \mathbf{C})$.**
- **We adapted the approach in regards to a shape optimization scenario relating to simulation-based design optimization and evaluated the feasibility of our methodology for application problems.**

Summary

First of all, the motivation for the study of a scenario concerning the prediction of operator configurations was to consider a different way of framing the meta-learning model from Chapter 2. Particularly, this scenario allows the construction of predictive and bias component in a joint manner. For the lack of a better alternative, we emulated the operator configuration using pairs of step-size and covariance matrix $(\sigma, \mathbf{C})$ within the CMA-ES algorithm. As the CMA-ES algorithm also enjoys great popularity for design optimization problems, we leveraged the opportunity to gain additional insight into potential gains and pitfalls in regards to application scenarios. Thus, to construct a simple scenario which can serve as a benchmark, we considered a shape optimization scenario in which the cabin volume of a car model is increased by either widening or heightening the shape of the body. The task is therefore to train a predictive method to suggest a suitable operator configuration in terms of pairs of step-size and covariance matrix $(\sigma, \mathbf{C})$ for each target shape from metadata generated during initial iterations of the algorithm.

While we could in principle see that this approach can demonstrate efficacy, we were also able to encounter significant limitations when designing the experiments and constructing the predictive and bias component. First of all, the most outstanding one is the difficulty of obtaining training data in the first place. While in the previous chapters, we mostly considered synthetic benchmark functions, the considered target shape optimization problems are comparably much more expensive. With training data generation and benchmarking taking from several up to ~ 10 hours of time. While calculating a single deformation can be done in about a minute and evaluating several deformations in parallel is no problem, calculating successive ones when considering several iterations of the optimization algorithm becomes very quickly expensive. For example, considering an evaluation budget of 10 iterations, this means a ten fold increase in computational time. As we want to also have training data sufficiently available, this increases the required computational cost at worst by a factor of 100. In principle, these associated computational costs impede on the ease of performing exhaustive experiments. And notably including a fluid dynamic simulation would further significantly worsen as the time for performing a single experiment would increase would increase up to several hours. These issues would be further amplified when increasing the dimensionality of the considered optimization problems. As we note that the number of parameters of the CMA-ES algorithm scale quadratically in $\mathcal{O}(d^2)$, thus effectively more training data would be required to train the predictive component.

Having now gained a complete overview over different aspects concerning the construction of learning optimization algorithms, we will end this thesis with a conclusive summary of the most important results we retrieved, a discussion of limitations that need to be taken into consideration for system deployment, and suggestions on future advancements for the framework as a whole as well as the specific methodology proposed within this work.

CHAPTER 7

CONCLUSIONS & OUTLOOK

The final chapter of this thesis is structured as follows: First, we concisely summarize in Sec. 7.1 the main technical contributions we made throughout Chapter 3 to 6. Following this up, we give a comprehensive overview of their limitations. We take to this regard two distinct perspectives: The first one in Sec. 7.2 is in regard to more general limitations from architectural considerations. The second one in Sec. 7.3 is with a focus on limitations specifically to our proposed methodology within this work. For both viewpoints, we also suggest potential ways to overcome their limitations. Either, by pointing towards promising lines of research existing within the literature, or through directly suggesting specific follow-up studies and work that could be done as based upon the results of this thesis. We end this chapter in Sec. 7.4 at last with a final conclusive summary of goals, proposed methods, obtained results and promising future lines of study. Note, that a separate overview of our proposed framework with a description of its functional components linked to the specific contributions of the previous chapters can be additionally found in Appendix A.

7.1 Summary of Contributions

Within this thesis, we argued for the meta-learning model as a framework to realize learning optimization systems that can avoid issues as existing in works within the literature. Particularly, we reasoned that such a model necessitates the implementation of two components: inductive biases and metaknowledge representations. We argued, that the former may be represented by the operators themselves, while the latter must inevitably be derived from the unstructured data generated as a by-product of the optimization algorithm during run-time. Thus, to satisfy these two requirements, we developed within this thesis methodology to implement them. Particularly, we claim to have made the following technical contributions by this:

- **We proposed histogram and density estimation based methodology to model inductive biases through mutation operators in the framework of a (μ+λ) Evolutionary Algorithm.**
- **We introduced a structured graph-based data format to quantize high-dimensional continuous search spaces and more faithfully reflect the neighborhood relationships thereof such that it enables us to harness generated unstructured procedural data of optimization algorithms for the training of learning algorithms.**
- **We suggested a specialized neural network architecture as well as fitness channel that can be used in conjunction with the structured graph-based data format as feature extractor for problem-dependent algorithm/bias component selection as well as extensions for temporal data processing based upon CNN and RNN-based components.**
- **We used our framework to directly predict operator configurations for the CMA-ES in the context of a shape optimization scenario for which we further provided a feasibility evaluation.**

Note, that we remarked that the overall architecture which we proposed constitutes an algorithm selection system. Thus, to comprehensively discuss the limitations of our work we take as previously elaborated a two-fold perspective:

In Sec. 7.2 we focus on more general limitations from the architecture itself, which can be considered to also apply to any algorithm selection systems in general. These concern constraints on the feasibility of constructing them, their generalization capability as well as long-term operability. While trivial solutions to these may not exist, we can nevertheless point to possible conditions under which these short-comings are alleviated as well as literature in which promising first steps are being made.

The second one in Sec. 7.3 is with a focus on limitations specifically to our proposed methodology within this work, particularly relating to the topics of inductive biases in optimization, graph-based data formats and feature extraction, as well as spatio-temporal modeling of search behavior. We specifically point within this discussion also to open problems and questions that can be easily picked up by any reader interested into follow-up studies.

7.2 Limitations to System Deployment

7.2.1 Constraints on Feasibility

Within our work we investigated the construction of operators and predictive components for an algorithm selection system which can be considered to be a particular intuitive and simple implementation of a meta-learning framework. In principle, we were already able to encounter within our research limitations in constructing such a system which need to be taken into consideration for system deployment. Foremost, the most striking one is that large amounts of training data are needed to build such a system in the first place. This was particularly a hurdle for the conduction of more flexible and extensive studies in the shape optimization scenario considered within Chapter 6. Specifically, the expensiveness of it makes a further incorporation computational fluid dynamic simulations impractical.

Thus, effectively the only feasible way would be by warm starting with an algorithm selection system pre-built and pre-trained on synthetic benchmark functions. Naturally, raising the question on how much the synthetic training problems can correspond to the ones in the application environment.

Also remark that the construction of our operators was especially very expensive in terms of required functions evaluations and we further had to introduce certain simplifications such that to neglect advanced adaption mechanisms as present in modern state-of-the-art algorithms [62]. While one may therefore arguably question the utility of this approach, we can still rationalize it as a possible blueprint in light of more general purpose optimization frameworks based upon some principles of evolutionary optimization, with less specific algorithms existing for solving them. We elaborate on this aspect in more detail in Sec. 7.3.1.

7.2.2 Generalization Capability

We argued from a review of optimization and learning theory over Chapters 2, 3 and 4, that a possible strategy to circumvent the limitations dictated by the no-free-lunch theorems is to create models which are adapted to each niche of the problem space and use predictive components which arbitrate between them. Thus, this especially motivated the usage of the previously elaborated computational model of meta-learning from Chapter 2. While such a system exhibits the capability to generalize performance improvements over each problem niche it has been constructed on, the system does not exhibit predictable performance gains on previously unencountered ones. Formulating, it in a more concise manner: While possessing intra-domain generalization capability, it does not possess inter-domain generalization capability (cf. Fig. 7.1). In principle, the no-free-lunch theorems do not provide explicit guidance on what happens in this scenario. Thus, applying such a constructed algorithm selection system to an unseen problem domain can result in either replicating the baseline behavior, performance degradation or improvements. How-

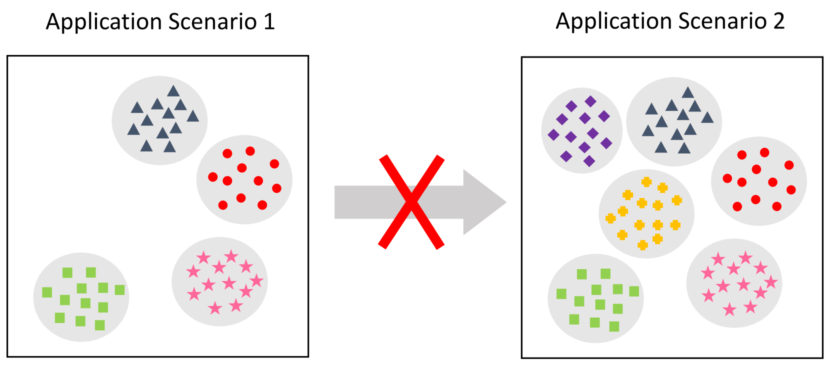

Figure 7.1: Even though the second application scenario possesses all domains from the problem space in the first application scenario, the algorithm selection system cannot be naively transferred to the new scenario as the performance of it on the newly introduced domains can be considered to be unpredictable.

ever, at no costs should performance improvements generated nevertheless be understood as an indication of problem similarity and the NFL theorems explicitly advise against it. Notably, this problem concerning the lack of performance guarantees resulting in a limited applicability of trained computational models to new encountered problems in the test domain is also recognized within reinforcement learning research [49].

7.2.3 Long-Term Operability

Ideally, we want to aim towards systems that can be constructed from the scratch and are capable of maintaining their operability by themselves in long-term. Thus, challenges are posed by implementing mechanisms which allow one to incrementally build such a system and continuously refine and adapt it for new problems that are being encountered (cf. Fig. 7.2).

From a practical point of view, these requirements can be difficult to be realized. In principle, existing methods within the domain of combinatorial optimization to predict solvers and solutions rely mostly upon warm-started models through pre-generated data. And as elaborated in Sec. 7.2.1, it is even more questionable whether in continuous optimization, specifically concerning the scenario of expensive optimization, one can construct a system which doesn't rely upon warm-starting, aside from a few exceptional scenarios. Noteworthy, e.g. the framework of AMTEA [36] for multi-objective optimization is capable of incrementally building up a repository of solutions from previously solved problems. Likewise, within combinatorial optimization, the CIGAR system [93] is also incrementally building a solution repository. However, notably both systems do not build explicit predictive models for regions of the problem space. Which we previously considered to be an implicit requirement for the implementation of computational models that exhibit generalization capability.

Life-Long Learning Systems

More in-depth has the aforementioned problem been studied under the denominator of life-long learning [132, 64, 110, 134, 30, 157, 63] in the recent decades. In principle, in its widest sense the notion of life-long learning aims towards the careful orchestration of different learning and knowledge refinement techniques [132, 30, 157] to realize long-term adaptiveness of a system when encountering new and different tasks. It is important to emphasize that this notion is not arising out of a mere academic desire, but is quite natural as human-made infrastructure such as e.g. in agriculture [157], telecommunications or cloud-based services [70], possess the quality of turning into legacy systems that need to maintained with a long-term scope.

Notions from Adaptive Systems Research

The mechanisms to support life-long learning in biological adaptive systems have coincidentally been well summarized within a very recently released article published by Kudithipudi et al. (2022) [85]. While there are acknowledgedly a great variety of mechanisms which biological systems implement to handle different problems encountered in a life-long scenario, we focus in the following on the most relevant ones for the context of our work.

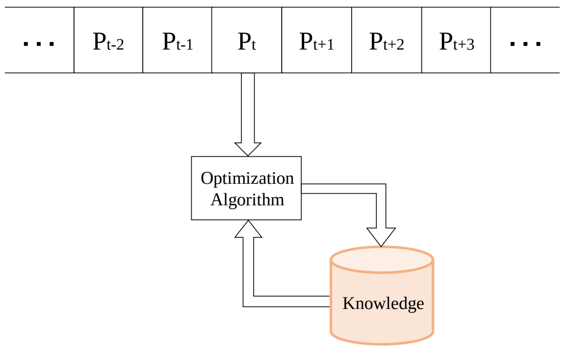

Figure 7.2: In its most simplest form, a life-long learning system can be considered to encounter a stream of different tasks during its lifetime from which it builds a knowledge base which it actively maintains and refines such that to incrementally improve its performance for future unseen tasks in the stream.

Mechanisms concerning neurogenesis allow the dynamic adaption of model capacity of a life-long learning system, by either generating new neurons when new problems are encountered, or performing active forgetting [56] and pruning of unnecessary connections such that to ensure more efficient model usage. Metaplasticity, meaning a mechanism to enable learning at different hierarchies through different degrees of plasticity, as well as neuromodulation, meaning a mechanism to dynamically regulate plasticity and learning on local and global level, helps a life-long system not be subjected to catastrophic forgetting of previous acquired knowledge when acquiring new knowledge and helps fostering flexible adaption to unseen tasks and environments. In a general sense, these mechanisms can be realized with architectures that implement context-dependent gating, hierarchical and distributed processing [85]. At last, episodic replay in rest states is an important mechanism to refine and improve the performance of a system. Ranging from simple replay of previous recorded experiences to aid memory consolidation and counter catastrophic forgetting, to more randomly and synthetically generated ones which can be considered to play the role of out-of-distribution examples that improve generalization capability of the system, to resource efficient replays on a small subset of experiences, to ones only relating to new learning tasks or restricted to abstract high-level representations.

Applications within Optimization

Life-long learning systems within optimization have been proposed within the previous decade in the context of combinatorial optimization problems. Particularly, for streaming constrained satisfaction [110] and bin packing problems [64, 134]. For the former constrained satisfaction problem (CSP), Ortiz-Bayliss et al. (2015) specifically propose a hyper-heuristic selection framework that is based upon a three component architecture. In principle, we can readily analyze it with the former elaborated systematics from Sec. 7.2.3.

The architecture, implemented is based upon distributed, hierarchical and context-dependent processing. Where the most important component is the main solver, which upon encountering a new CSP instance, based upon its available features $\mathbf{f}$ calls the prediction module, from which a predictor at point $\mathbf{p}$ is retrieved which satisfies a minimal distance requirement $\|\mathbf{f}-\mathbf{p}\| \leq d_{\min}$, which in context-dependent manner suggests the best known heuristic h from a pool $\mathcal{H}$ for the main solver to use. The third component, called learning module, is given information about features of the encountered CSP instance, as well used predictor. While the main solver is solving the given instance, the learning module uses the time to conduct episodic replays in a parallel thread, by testing other available heuristics from the pool out, such that the predictor is refined and can suggest better suitable heuristics if available in the future. In principle, if a predictor satisfying the distance constraint $\|\mathbf{f}-\mathbf{p}\| \leq d_{\min}$ cannot be found, a new predictor is created at point $\mathbf{p}$ and a random heuristic $h \in \mathcal{H}$ is chosen for the main solver, while the learning module in parallel refines the predictor. In the long run, the system continuously constructs and refines a hyper-heuristic which learns to associate different regions of the problem space $\mathcal{P}$ with the best matching heuristic in $h \in \mathcal{H}$.

The approaches from Hart & Sim (2014 & 2015) [64, 134] for solving streaming 1d bin-packing problems are very similar to this regard. However, based instead upon an artificial immune system, which in the long-run aims to maintain a set of mutually highly stimulating heuristics and set of problems.

7.3 Future Directions for Research

Before we end this thesis with a final conclusive summary, we provide a further outlook on the future most promising directions for advancing the methodology as proposed within our research. Specifically, we directly highlight open problems and questions that could be easily picked up by any reader interested into doing follow-up work.

7.3.1 Inductive Biases in Optimization

We proposed in Chapter 3 the construction of inductive biases in the form of problem-tailored mutation operators through histogram and density estimation methods. Particularly, we were able to justify this approach adhoc from finding work on evolvability studies in computational biology from Kounios et al. (2016) [83], suggesting that indeed the underlying variational mechanisms of natural evolution possess, in theory, the ability to learn the structures of rugged fitness landscape from repeated exposure. Thus, it helps avoid getting the population stuck in local optima of future unseen environments, which is a behavior that we could also partly replicate within our studies on the regularly structured multimodal problems (cf. Sec. 3.5.1). This is indeed a very intriguing perspective we could relate our research to. However, as our experiments preceded the discovery of this work, we were unfortunately within the scope of this thesis not able to fully explore the full complexities of it, e.g. by the study of artificial gene regulatory networks or through the assistance of evolvability measures from fitness landscape analysis. But as within our approach constructing the mutation operators was particularly laborious and had reduced effectiveness at higher dimensionalities, a future study could explore for instance predicting instead operator configurations based upon problem-dependent evolvability characteristics.

Noteworthy, putting evolvability considerations aside, more generally speaking some of the hurdles we encountered when constructing the operators could very likely be alleviated when either considering parametrizations in the form of algorithm configurations with less degrees of freedom instead, as we did in Chapter 6, or more well-behaved optimization problems with more of a coarser structure. For the former, recall that within Chapter 6 the scenario of predicting algorithm configurations also gave us the advantage that we could jointly construct bias and predictive component. In principle, the prediction of algorithm configurations or portfolios may be generally considered a more reasonable venture in the domain of continuous optimization. However, from our studies in Chapter 6, we suggest to focus on algorithms which do not scale in $\mathcal{O}(d^2)$ in the number of parameters. The approach of constructing explicitly operators, however might still make sense in the framework of EDAs [87]. Also note, that as we have previously elaborated in Sec. 4.2.2, the problem of constructing operators can generally be considered to be relaxed in combinatorial optimization. Concerning the suggested usage of more well-behaved optimization problems with coarser structure, one could for instance realize this case by either restricting oneself only to specific use cases that satisfy this requirement, the construction of low-resolution versions of given optimization problems, or through the explicit use of dimensionality reduction techniques. On this note, we also need to emphasize that within our study we particularly considered a framework based upon (μ+λ) selection, which enabled us to be able in the first place to collect exhaustive statistics about the performed variation operations by an algorithm. However, while elitist selection has proven to be useful in mutation-based algorithms [9], recent studies suggest that non-elitist selection excels on deceptive fitness landscapes [37]. It would be interesting to consider within further work what this means for algorithm design and how one would proceed in this scenario with the ideas that we have outlined within this thesis.

At last, we elaborate more generally on the nature of inductive biases in optimization. Jethwani et al. (2021) [71] investigated forms of inductive biases towards simplicity for evolving Boolean functions with a genetic algorithm. Within their study they explicitly model the inductive bias either through a statistical distribution or explicit constraint. In principle, they find that the former one is more effective. Thus, unknowingly validate the argument we made previously in Chapter 3 for inductive biases in the sense of biases in the generation of random variates. Intrinsic symmetries and invariances of optimization problems may further foster quite naturally certain forms of biases. The first step towards this line of investigation has been made somewhat similar to our work of Chapter 3 by Tian et al. (2021) [143]. It could be interesting to consider this in a more generalized framework in the context of learning of invariance properties. Further, already known forms of biases can be considered to be posed by the simplexes spanned by crossover operators (cf. Fig. 3.2), self-adaption mechanisms [62], which are notably very similar to the adaptiveness developmental biases foster as envisioned by Uller et al. (2018) [146], as well as explicit forms of diversity maintenance in multi-objective optimization [39] and implicit ones through resource limitation as introduced recently in co-evolutionary frameworks [21].

In any case, the identification of mechanisms and qualities which can serve as potential forms of inductive biases in optimization algorithms remains to be an intriguing topic for which we argue that it warrants further investigation in the future. We stress at this point, that it does not simply constitute a mere re-framing of existing methodology, but essentially raises the question on how far domain knowledge can be incorporated through derandomizing the variational mechanisms of existing algorithms, such that to allow them to be tailored to specific problems from past experiences of solving them. It also establishes a bridge and enables vitalizing exchange with similar ongoing research within the broader scope of artificial intelligence research [136].

7.3.2 Graph-based Data Formats and Feature Extraction

We introduced within Chapter 4 a pipeline to convert unstructured raw data derived from metadata of population-based optimization algorithms in the form of tuples $(\mathbf{x}, f(\mathbf{x}))$ of generated solutions $\mathbf{x} \in \mathbb{R}^d$ and fitness values $f(\mathbf{x})$, which we further organized into sets P_g^r for each population at generation g and experimental run r into a structured data format $\mathbf{z}$ by applying a search space partition to the solution set P_g^r. Subsequently, the structured data format could be used in conjunction with neural network architectures for feature extraction. In particularly, we introduced a graph-based data format such that $\mathbf{z} \in \mathbb{R}^{N_c \cdot N_f}$, where N_c are the number of nodes and N_f the number features. We could show that in addition to a channel for solution counts, considering a further one for fitness values ($N_f = 2$), the graph-based data format in conjunction with specialized network architectures achieves higher validation accuracies and retrieves more pronounced and better separated clusters.

In principle, there is a vast amount of combinatorial possibilities to further extend these studies. However, we will focus in the following on highlighting the most intriguing aspects of this proposed approach and from it show the most promising directions for future possible investigation. First of all, a key advantage of the graph-based approach is that it allows a structured representation of problem-dependent search behavior of an optimization algorithm, thus fosters explainability, which can in principle be considered to be a key advantage in comparison to potential alternative ways of feature extraction utilizing unstructured data formats. Noteworthy, for this reason graph-based search trajectory networks [107] have likewise been recently introduced for the analysis of algorithm behavior on grey box optimization problems.

Secondly, particularly very interesting properties can be observed when considering the graph-based data format in conjunction with specialized neural network architectures for processing them. As we elaborated over Chapters 4 and 5, the graph convolution operations by Kipf & Welling (2016) [80] can in principle be interpreted as a low-order approximation of a 'heat' diffusion model. Thus, the application of the convolution operation is in principle extrapolating from known values of solution counts and fitness values, by conducting in-filling of empty parts through radiating fitness values and creating 'virtual' population members of the solution distribution within the graph structure. It is clear that these properties which graph operations conduct on the fly exhibit notions with intrinsic similarities to surrogates and uncertainty measures in model-based optimization. A more mathematically formalized analysis of these properties in conjunction with the development of specialized graph operations for specifically processing this specific data type, could therefore lead to further promising synergies.

At last, to conclude we briefly elaborate on lines of investigation which can be considered to be more accessible and could be picked up at ease without requiring a more formal theory-based approach. First of all, different input transformations could be further considered. While to some degree we did this already within our work, further investigations could for instance consider instead ranks of solutions as input and also look in more detail how different combinations of input formats can enrich the training of the architectures for feature extraction. Finally, the process of generating a knowledge graph of activity in the search space itself could be further improved. While we used clustering methods fitted to randomly generated data within the search space in Chapter 4 to keep the analogy to the original work by Liu et al. (2017) [91], one could explore further alternatives, such as e.g. random graphs, or graphs generated based upon the distribution of solutions in the training data set for the network architectures.

7.3.3 Spatio-Temporal Modeling of Search Behavior

We addressed within Chapter 5 the challenge of spatio-temporal modeling of search behavior using the metadata generated by an optimization algorithm. For this reason, we looked into methods for learning spatial anisotropies of the knowledge graph representing activity within the search space, as well as considered specialized CNN and RNN-based network architectures lent from methodology developed for time-series classification and activity recognition for processing the temporal component. In the following, we want to show further directions which can be considered to be valuable to be taken up as based upon our work.

The algorithm selection problem within population-based optimization, reframed as a meta-learning problem, is within the context of methods for time-series processing undeniably an early-classification problem. Meaning the predictive component should make a decision as soon as enough data from the optimization problem has been collected. However, within our currently proposed methodology within Chapters 4 and 5, we either only considered data generated from the first iterations, or from time-series with lengths of 10 and 20 iterative steps. Thus, to satisfy the aforementioned requirement, we ideally want to avoid sticking to a fixed and preset interval, but instead have a mechanism available that automatically decides on the best interval for us to make a prediction. Rußwurm et al. (2019) [121] have to this regard proposed a method to extend sequential models any future follow-up work can easily pick up to extend our feature extractors proposed in Chapter 5. Alternatively to our work on meta-learning in algorithm selection systems, recall from Chapter 2, that we criticized that a major implementation hurdle for reinforcement learning is the definition of states and actions in the first place. While we were able to identify within our research earlier work by Zhang et al. (2008) [162] that implements reinforcement learning where different operators pose as different choosable actions, a state representation is still missing out. To this regard, the flexibility of our proposed RNN architectures in Chapter 5 in conjunction with the graph-based data format we have proposed in Chapter 4 can be considered to be very helpful solutions for future ventures into this direction.

At last, note that the reinforcement learning approach by Zhang et al. (2008) [162] which frames different actions in terms of different operators, can in principle be framed as a memetic optimization technique, in which operators take the roles of the memes. Thus, this emphasizes valuable synergies and cross-connections between these two different fields to be taken into consideration.

7.3.4 Further Remarks

To end this section of the thesis, we want to discuss in the following briefly some of the more intriguing insights we have obtained during our research. Following up these directions might not directly be associated with any significant short-term gains, however can be considered to have more general benefits on a longer time scale.

Particularly, we highlight the work done by Rudolph (1998) [120], which analyzed finite Markov chain models and from these formulated requirements on operators and convergence theorems for evolutionary optimization algorithms, in terms of probabilistic spatio-temporal relationships. This work essentially substantiates and validates the spatio-temporal modeling approach for the construction of end-to-end frameworks. It is thus very tempting to be able to replicate in long-term successes achieved in reinforcement learning, such that ideally entire algorithm components can be purely derived from problem-dependent goal and reward-oriented behavior of a learning optimization system [133]. Note, that this notion is also currently being popularized under the denominator of 'neural algorithmic reasoning' [148], offering aside from the previous mentioned works on feature-free algorithm selection from Sec. 4.2.2 new applications for e.g. minimum cut [148] or graph recovery problems [130]. How far can we abandon notions of hand-crafted methodology? While we strongly advocated for it over Chapters 3, 4 and 5, and further substantiated it with arguments brought forward within adaptive systems [69, 18] and artificial intelligence research [133], evidence from the studies of Alissa et al. (2021) [4] on bin-packing problems suggest that such built models of solvers may not fully be able to replace handcrafted ones, but can still be considered to be useful complementaries. Likewise, while the study from Seiler et al. (2020) [127] on the traveling salesperson problem attempted to abandon analytical fitness landscape techniques, their results nevertheless indicate that the usage of their feature-free approach in conjunction with analytical techniques gives the best results. It is interesting to consider how the aforementioned notions could more generally be extrapolated to more free-form frameworks e.g. such as incorporating mechanisms for open-endedness [149, 99].

At last, we remark on more general potential future synergies between computer science and model-oriented research in the natural sciences. While we proposed over Chapters 4 and 5, graph-based data formats and architectures for processing them, and emphasized beneficial properties of these, to a similar degree such an approach for relational modeling has been previously introduced under the denominator of evolutionary graph theory (e.g. [90, 114, 94]) in theoretical biology. Even though, rather concerned with modeling evolutionary dynamics of spatially separated populations modeled through graphs, instead of activity within a search space in the context of optimization, the closeness between the underlying notions of both are intriguing. It is therefore very obvious and worthwhile to keep under observation whether similarities between developed approaches from theoretical biology research and rather applied work on evolutionary computation could enable similar synergies in the future as has been argued to exist for neural computation research [65].

7.4 Closing Summary

To end this thesis, we give a closing summary, which in a conclusive manner highlights and compares goals and achievements of this thesis to each other, as well as gives a compressed recapitulation of the most promising research directions to be further taken up as we have previously outlined in Sec. 7.3. Note, that we provide for the interested reader an overview of our proposed framework with a description of its functional components linked to specific contributions of our chapters separately within Appendix A.

Before we continue, we briefly reiterate on the main technical contributions of the thesis as we have listed in Sec. 7.1:

- **We proposed histogram and density estimation based methodology to model inductive biases through mutation operators in the framework of a (μ+λ) Evolutionary Algorithm.**
- **We introduced a structured graph-based data format to quantize high-dimensional continuous search spaces and more faithfully reflect the neighborhood relationships thereof such that it enables us to harness generated unstructured procedural data of optimization algorithms for the training of learning algorithms.**
- **We suggested a specialized neural network architecture as well as fitness channel that can be used in conjunction with the structured graph-based data format as feature extractor for problem-dependent algorithm/bias component selection as well as extensions for temporal data processing based upon CNN and RNN-based components.**
- **We used our framework to directly predict operator configurations for the CMA-ES in the context of a shape optimization scenario for which we further provided a feasibility evaluation.**

We started out this thesis by first reviewing foundations from Natural Computing in regards to simulated evolution and learning in Chapter 1.1. From these, we headed over to discussing the current state of research and contemporary issues within metaheuristic optimization from which we argued that there is a dire need to move towards more general purpose algorithm architectures that learn from the problems they solve and can be flexibly adapted for new problem domains of interest. Thus, we decided to dedicate our work to particularly address this issue by directly proposing methodology and clarifying the foundations to construct such first-principled frameworks.

For this reason, we first reviewed different computational models of cross-problem solving within Chapter 2. While we argued that Markov decision processes as popular in reinforcement learning are difficult to implement, useful simplifications can be instead found in a meta-learning model. In principle, the latter has been also identified within the literature as corresponding to algorithm selection and configuration systems. We also inspected within our literature review potential other options. Particularly interesting to this regard was the notion of transfer learning in optimization. However, unfortunately it has only limited applicability, as most transfer learning methods that have been proposed within the literature are mostly domain-specific. The related notion of transfer optimization, while highly pragmatic, however suffers from the definition of sound problem-similarity measures and more broadly speaking lack of generalization capability.

We formulated subsequently four questions within this thesis that aim to answer these issues. While in particularly, we answered the first one on the nature of models of computational problem-solvers partly in the literature review of Chapter 2, we gave an answer for the second on what the reusable knowledge within the variational mechanisms of natural evolution is and how it could be used for algorithm design, in the following up Chapter 3. Particularly, we argued that for this reason we need to introduce the concept of inductive biases from learning theory within optimization. We identified to this regard specifically mutation as most important and prime variational mechanism that should be put under further scrutiny. We therefore, explicitly considered a $(\mu+\lambda)$ Evolutionary Algorithm, for which we proposed explicitly methodology to build mutation operators for different benchmark functions using histogram and density estimation based methodology. We were able to demonstrate realizable performance gains with this proposed methodology, and were able to justify the approach with studies done in computational biology arguing for the learning capability of evolutionary processes.

The third question, in regards to how we can characterize optimization problems such that we can easily re-identify them with suitable previously learned knowledge, was answered by us in Chapters 4 & 5. Particularly, we introduced a pipeline to convert unstructured raw data representative of activity within the search space into a structured data format, that can subsequently be used for feature extraction in conjunction with specialized neural network architectures. Novelties that we introduced in Chapter 4 to this regard, are a novel graph-based data format, a specialized graph neural network architecture for processing it, as well as the additional inclusion of an input channel to take fitness values into consideration. We were able to show that through these measures, the graph-based approach demonstrated higher performance in comparison to more traditional ones. However, because it showed still ambivalent performance in the case where inputs in the fitness channel are not informative, we extended this approach in Chapter 5 with graph attention operations (GAT) to learn spatial anisotropies, as well as introduced specialized CNN- and RNN-based components such that to enable temporal data processing. Particularly, we were able to demonstrate that our proposed GCN-GRU and GCN-LSTM network architectures demonstrated highest and most stable performance on time-series of variable length, while still being computationally efficient.

To conclude the last question in regard to the scope for application problems and beyond, we studied a scenario in Chapter 6 relating to shape optimization, which enabled us to gain some insight into opportunities and potential hurdles for application problems. It also allowed us to reframe the problem of predicting operators in terms of predicting operator configurations, which we did so through step-size σ and covariance matrix $\mathbf{C}$ of the CMA-ES algorithm. Particularly, due to the expensiveness of the studied problem, we were not able to exhaustively collect training data and therefore were limited in conducting more extensive studies. We therefore see these experiments as indicative, that in the foreseeable future the study of such frameworks on rich and varied sets of synthetic functions mimicking application problems will be a more valuable direction to explore.

In the last Chapter 7 of this thesis, we discussed limitations for system deployment and future viable directions to advance the methodology we have proposed within this work. Particularly, in regards to the former, we briefly elaborated on how to realize a long-term scope of operability in learning optimization systems, and particularly elaborated on notions of adaptive system research, from which we drew valuable criteria that enabled us to make a brief analysis and evaluation of available existing work.

To conclude, we summarize the most promising research directions as suggested within Sec. 7.3 to be taken up by potential future follow-up work as based upon this thesis:

- **The investigation of operator design based upon evolvability characteristics and the consideration of potentially more suitable optimization scenarios possibly constructed in conjunction with dimensionality reduction techniques.**
- **Properties of explainability of the graph-based data format and its relationship to surrogates and uncertainty measures in model-based optimization.**
- **The extension of the spatio-temporal approach to frame algorithm selection as an early-classification problem and its usage as state representation and memory for reinforcement learning based operator selection.**

We hope that this thesis and the broad range of topics covered within it helps any reader of it as much as creating it helped ourselves in understanding the current state of methodology, open issues and opportunities such that to enable and encourage them likewise in the construction of sophisticated and on first-principles based learning optimization algorithms and frameworks.

APPENDIX A

OVERVIEW OF THE CONSTRUCTED FRAMEWORK

We describe in the following the general purpose algorithm framework that we construct throughout Chapters 3 to 6 in this thesis as illustrated in Fig. A.1. A special focus is thereby paid on linking the description of each functional component with the specific contributions of the respective chapters.

First of all, we discussed within Chapter 2 different computational models of cross problem-solving. We argued from this literature review for the meta-learning model, which frames cross problem-solving in terms of domain-level knowledge through inductive biases, as well as metaknowledge representations to arbitrate between them. In principle, our framework therefore must implement these two design requirements and thus the framework we construct over the course of this thesis as given in Fig. A.1 overall corresponds to an algorithm selection system [136].

As a first step, such a system must extract features that are able to sufficiently characterize different optimization problems. We propose our method to achieve this within Chapter 4. In comparison to the literature, we take likewise the view that is popularized within recent years, that handcrafted methods could be neglected to this regard in favor of a pure data-driven approach that harnesses the feature learning capabilities of deep

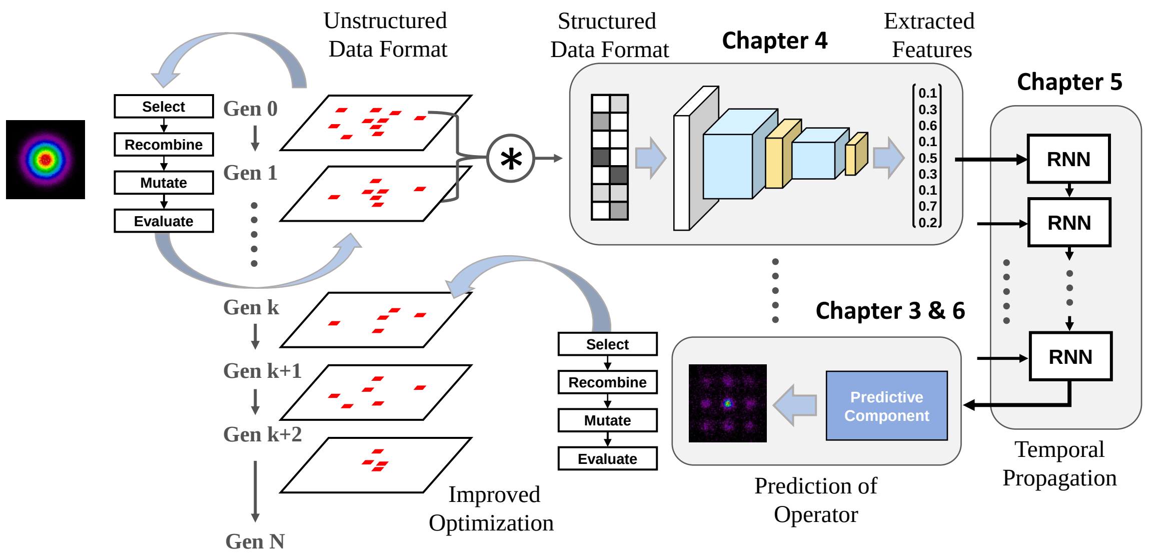

Figure A.1: Diagram of the framework we construct within this thesis, as well as it's functional components separated according to chapters in which the respective contributions to the methodology are made.

network architectures. We argue that this should be addressed specifically in continuous optimization by means of methodology from algorithm behavior studies, such that it allows us to extract characteristics descriptive of problem-dependent behavior of optimization algorithms. The novelties we introduce to this regard specifically within Chapter 4 are a graph-based structured data format to represent activity within high-dimensional continuous search spaces, a fitness channel to enrich this activity information as well as specialized graph neural network (GNN) architectures for feature extraction.

While the feature extractor we introduced can in principle already be considered to be sufficient for extracting features that are descriptive to distinguish different optimization problems, it however may fail in scenarios when the generated inputs are not sufficiently informative. We propose therefore in Chapter 5 ways to improve this framework component by enhancing the capabilities of the feature extractor to explicitly model the time-dependent nature of the metadata the optimization algorithm generates during a run. We specifically find within Chapter 5, that extending our GNNs with a recurrent component to a GNN-RNN-based architecture, particularly GCN-GRU and GCN-LSTM, proves to be the most viable strategy.

Having now methods available to extract spatio-temporal features sufficiently descriptive of problem-dependent algorithm behavior, we can move over to make a recommendation for the optimal operator to use. Naturally, raising the question on how we can obtain these operators in the first place, as well as how we can train such a framework as a whole. We answer both of these questions at the beginning and the end of this thesis within Chapters 3 and 6. First of all, note that we identified the question of modeling and choosing the optimal operator as corresponding to the question of modeling and choosing a domain-dependent bias. Thus, established a bridge to the same concept as existing within machine learning research [102, 154].

We argued that within a (μ+λ) Evolutionary Algorithm we can readily model domain-knowledge by means of incorporating problem-structure into the mutation operator. In principle, we also demonstrated within our experiments that this strategy can work. Note, that this approach can also be justified more or less from studies in computational biology [83].

However, due to the particular expensiveness in modeling them, as well as hurdles to jointly construct the predictive architecture as depicted in Fig. A.1 together with the operator itself, a more practical alternative may instead be found in the prediction of a high-performing algorithm configuration, as we demonstrate in Chapter 6. For the example of the CMA-ES, as well as using a multi-layer perceptron as stand-in for the architecture in Fig. A.1, we can show the efficacy of this scenario as well as find overall better handleable properties for the implementation of such frameworks in an end-to-end manner.

APPENDIX B

BENCHMARK FUNCTIONS

B.1 Unimodal Functions

Bohachevksy Function

$$f(\mathbf{x}) = \sum_{i=1}^{d-1} [x_i^2 + 2\,x_{i+1}^2 - 0.3\cos(3\pi x_i) - 0.4\cos(4\pi x_i) + 0.7] \tag{B.1}$$

Properties

- Funnel Structure: Asymmetric, quadratic, convex
- Search Space: $\chi = [-100, 100]^d$
- Reference: [61]

Ellipsoidal Function

$$f(\mathbf{x}) = \sum_{i=1}^{d} (10^6)^{\frac{i-1}{d-1}} x_i^2 \tag{B.2}$$

Properties

- Funnel Structure: Asymmetric, quadratic, convex
- Search Space: $\chi = [-100, 100]^d$
- Reference: [61]

Rosenbrock Function

$$f(\mathbf{x}) = \sum_{i=1}^{d} [100\,(x_{i+1} - x_i^2) + (x_i - 1)^2] \tag{B.3}$$

Properties

- Funnel Structure: Asymmetric, polynomial
- Search Space: $\chi = [-5, 10]^d$
- Reference: [61]

Schwefel 1.2 Function

$$f(\mathbf{x}) = \sum_{i=1}^{d} \left(\sum_{j=1}^{i} x_j \right)^2 \tag{B.4}$$

Properties

- Funnel Structure: Asymmetric, polynomial
- Search Space: $\chi = [-65.536, 65.536]^d$
- Reference: [135]

Sphere Function

$$f(\mathbf{x}) = \sum_{i=1}^{d} x_i^2 \tag{B.5}$$

Properties

- Funnel Structure: Symmetric, convex, quadratic
- Search Space: $\chi = [-5.12, 5.12]^d$
- Reference: [61, 135]

B.2 Regularly Structured Multimodal Functions

Ackley Function

$$f(\mathbf{x}) = -20\exp\left(-0.2\sqrt{1/d\sum_{i=1}^{d} x_i^2}\right) - \exp\left(-\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi x_i)\right) + 20 + e \tag{B.6}$$

Properties

- Funnel Structure: Symmetric, concave, exponential
- Search Space: $\chi = [-32.768, 32.768]^d$
- Reference: [135]

Griewank Function

$$f(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{d} x_i^2 - \prod_{i=1}^{d}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \tag{B.7}$$

Properties

- Funnel Structure: Symmetric, convex
- Search Space: $\chi = [-600, 600]^d$
- Reference: [135]

Rastrigin Function

$$f(\mathbf{x}) = 10\,d + \sum_{i=1}^{d}\left[x_i^2 - 10\cos(2\pi x_i)\right] \tag{B.8}$$

Properties

- Funnel Structure: Symmetric, convex, quadratic
- Search Space: $\chi = [-5.12, 5.12]^d$
- Reference: [61, 135]

B.3 Irregularly Structured Multimodal Functions

Eggholder Function

$$f(\mathbf{x}) = -(x_2 + 47)\sin(\sqrt{|x_2 + x_1/2 + 47|}) - x_1 \sin(\sqrt{|x_1 - (x_2 + 47)|}) \tag{B.9}$$

Properties

- Funnel Structure: Asymmetric
- Search Space: $\chi = [-512, 512]^d$
- Reference: [135]

Schaffer Function

$$f(\mathbf{x}) = \sum_{i=1}^{d-1} (x_i^2 + x_{i+1}^2)^{0.25} \{\sin^2[50(x_i + x_{i+1})^{0.10}] + 1\} \tag{B.10}$$

Properties

- Funnel Structure: Symmetric
- Search Space: $\chi = [-100, 100]^d$
- Reference: [1]

Schwefel 2.26 Function

$$f(\mathbf{x}) = 418.9829\, d - \sum_{i=1}^{d-1} x_i \sin\left(\sqrt{|x_i|}\right) \tag{B.11}$$

Properties

- Funnel Structure: Asymmetric
- Search Space: $\chi = [-500, 500]^d$
- Reference: [86, 135]

APPENDIX C

INTERPRETATION OF GRAPH OPERATIONS

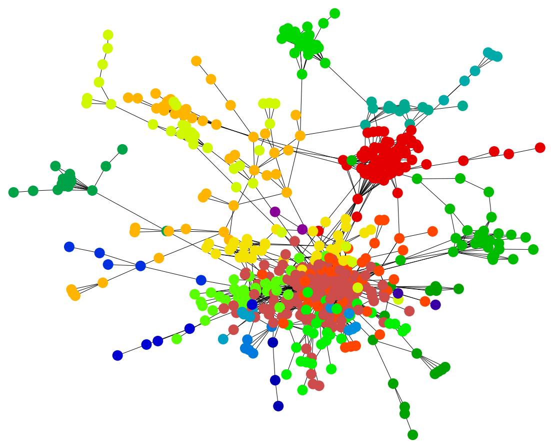

Figure C.1: Social network graph from a data set of Facebook pages with mutual likes [118] visualized using the NetworkX library [58]. Different communities existing within the graph have been obtained using greedy modularity maximization [33] and are indicated by different colors.

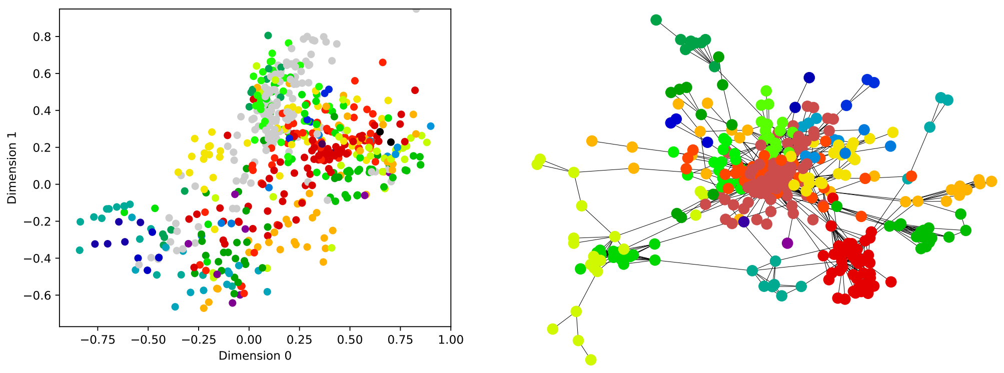

Figure C.2: The graph convolution as proposed by Kipf & Welling (2016) [80] can be interpreted as calculating low-dimensional node embeddings in which the communities are roughly preserved (left panel), while the graph pooling operation based upon Defferrard et al. (2016) [40] simply reduces the number of nodes and edges in the graph (right panel).

LIST OF REFERENCES

[1] DEAP 1.3.1 documentation. https://deap.readthedocs.io/en/master/index.html, accessed: May 2022.

[2] Mohamad Alissa, Kevin Sim, and Emma Hart. Algorithm selection using deep learning without feature extraction. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 198–206, 2019.

[3] Mohamad Alissa, Kevin Sim, and Emma Hart. A deep learning approach to predicting solutions in streaming optimisation domains. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 157–165, 2020.

[4] Mohamad Alissa, Kevin Sim, and Emma Hart. A neural approach to generation of constructive heuristics. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 1147–1154. IEEE, 2021.

[5] Claus Aranha, Christian L Camacho Villalón, Felipe Campelo, Marco Dorigo, Rubén Ruiz, Marc Sevaux, Kenneth Sörensen, and Thomas Stützle. Metaphor-based metaheuristics, a call for action: the elephant in the room. *Swarm Intelligence*, pages 1–6, 2021.

[6] The GPyOpt authors. GPyOpt: A Bayesian Optimization framework in Python. http://github.com/SheffieldML/GPyOpt, 2016.

[7] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms.* Oxford University Press, 1996.

[8] Thomas Bäck, Joost N Kok, and G Rozenberg. *Handbook of Natural Computing.* Springer, Heidelberg, 2012.

[9] Thomas Bäck, Günter Rudolph, and Hans-Paul Schwefel. Evolutionary programming and evolution strategies: Similarities and differences. In *In Proceedings of the Second Annual Conference on Evolutionary Programming.* IEEE, 1993.

[10] K. K. Bali, A. Gupta, L. Feng, Y. S. Ong, and Tan Puay Siew. Linearized domain adaptation in evolutionary multitasking. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 1295–1302, June 2017.

[11] Richard K Belew. Evolution, learning and culture: Computational metaphors for adaptive algorithms. *Technical Report CS89-156, Computer Science, Univ. Calif. San. Diego*, 1989.

[12] Richard K Belew. *Adaptive Individuals In Evolving Populations: Models And Algorithms*. Routledge, 1st edition, 1996.

[13] David Beniaguev, Idan Segev, and Michael London. Single cortical neurons as deep artificial neural networks. *Neuron*, 109(17):2727–2739, 2021.

[14] JD Bermúdez, P Achanccaray, ID Sanches, L Cue, P Happ, and RQ Feitosa. Evaluation of recurrent neural networks for crop recognition from multitemporal remote sensing images. In *Anais do XXVII Congresso Brasileiro de Cartografia*, pages 800–804, 2017.

[15] Anil Bhattacharyya. On a measure of divergence between two statistical populations defined by their probability distributions. *Bull. Calcutta Math. Soc.*, 35:99–109, 1943.

[16] Christopher M Bishop. Mixture density networks. Technical report, Aston University, 1994.

[17] Christopher M Bishop. *Pattern Recognition and Machine Learning*. Springer, 1st edition, 2006.

[18] Colin Blakemore and Grahame F Cooper. Development of the brain depends on the visual environment. *Nature*, 228(5270):477–478, 1970.

[19] Barry G Blundell. *An Introduction to Computer Graphics and Creative 3-D Environments*. Springer Science & Business Media, 2008.

[20] Maarten Boudry and Steije Hofhuis. Parasites of the mind. why cultural theorists need the meme's eye view. *Cognitive Systems Research*, 52:155–167, 2018.

[21] Jonathan C Brant and Kenneth O Stanley. Diversity preservation in minimal criterion coevolution through resource limitation. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 58–66, 2020.

[22] Xavier Bresson and Thomas Laurent. The transformer network for the traveling salesman problem. *arXiv preprint arXiv:2103.03012*, 2021.

[23] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[24] Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

[25] Christian Leonardo C. Villalón, Marco Dorigo, and Thomas Stützle. The intelligent water drops algorithm: why it cannot be considered a novel algorithm. *Swarm Intelligence*, 13(3):173–192, 2019.

[26] Felipe Campelo and Claus Aranha. EC Bestiary: A bestiary of evolutionary, swarm and other metaphor-based algorithms. https://doi.org/10.5281/zenodo.1293352, June 2018.

[27] Gilles Celeux, Sylvia Frühwirth-Schnatter, and Christian P Robert. Model selection for mixture models–perspectives and strategies. In *Handbook of Mixture Analysis*, pages 117–154. Chapman and Hall/CRC, 2019.

[28] José E Chacón and Tarn Duong. *Multivariate Kernel Smoothing and its Applications*. Chapman and Hall/CRC, 2018.

[29] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository, 2015.

[30] Zhiyuan Chen and Bing Liu. Lifelong machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 12(3):1–207, 2018.

[31] François Chollet and others. Keras: The Python Deep Learning library. Astrophysics Source Code Library, record ascl:1806.022, June 2018.

[32] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, Guido Ranzuglia, et al. Meshlab: an open-source mesh processing tool. In *Eurographics Italian Chapter Conference*, volume 2008, pages 129–136. Salerno, Italy, 2008.

[33] Aaron Clauset, Mark EJ Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical Review E*, 70(6):066111, 2004.

[34] Massimiliano Corsini, Paolo Cignoni, and Roberto Scopigno. Efficient and flexible sampling with blue noise properties of triangular meshes. *IEEE Transactions on Visualization and Computer Graphics*, 18(6):914–924, 2012.

[35] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[36] B. Da, A. Gupta, and Y. Ong. Curbing negative influences online for seamless transfer evolutionary optimization. *IEEE Transactions on Cybernetics*, no. 99:1–14, 2018.

[37] Duc-Cuong Dang, Anton Eremeev, and Per Kristian Lehre. Non-elitist evolutionary algorithms excel in fitness landscapes with sparse deceptive regions and dense valleys. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1133–1141, 2021.

[38] Omid E David and Iddo Greental. Genetic algorithms for evolving deep neural networks. In *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 1451–1452, 2014.

[39] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and Tanaka Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *International Conference on Parallel Problem Solving from Nature*, pages 849–858. Springer, 2000.

[40] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in Neural Information Processing Systems*, 29:3844–3852, 2016.

[41] L. Devroye. *Non-Uniform Random Variate Generation.* Springer-Verlag New York, 1986.

[42] Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(11):1944–1957, 2007.

[43] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2625–2634, 2015.

[44] Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

[45] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, 2019.

[46] Liang Feng, Yew-Soon Ong, Siwei Jiang, and Abhishek Gupta. Autoencoding evolutionary search with learning across heterogeneous problems. *IEEE Transactions on Evolutionary Computation*, 21(5):760–772, 2017.

[47] Liang Feng, Lei Zhou, Jinghui Zhong, Abhishek Gupta, Yew-Soon Ong, Kay-Chen Tan, and Alex Kai Qin. Evolutionary multitasking via explicit autoencoding. *IEEE Transactions on Cybernetics*, 49(9):3457–3470, 2018.

[48] Emile Fiesler and Russell Beale. *Handbook of Neural Computation.* CRC Press, 1997.

[49] Chelsea Finn. Conceptual Understanding of Deep Learning Workshop, May 2021. Principles for Tackling Distribution Shift: Pessimism, Adaptation, and Anticipation.

[50] Rémi Flamary and Nicolas Courty et al. POT Python Optimal Transport library. *Journal of Machine Learning Research*, 22(78):1–8, 2021.

[51] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, Jul 2012.

[52] Bernd Fritzke. A growing neural gas network learns topologies. In *Advances in Neural Information Processing Systems*, pages 625–632, 1995.

[53] Bent Fuglede and Flemming Topsoe. Jensen-shannon divergence and hilbert space embedding. In *2004 IEEE International Symposium on Information Theory*, page 31. IEEE, 2004.

[54] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and Cooperation in Neural Nets*, pages 267–285. Springer, 1982.

[55] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press Cambridge, 2016.

[56] Lauren Gravitz. The importance of forgetting. *Nature*, 571(July):S12–S14, 2019.

[57] Abhishek Gupta, Yew-Soon Ong, and Liang Feng. Insights on transfer optimization: Because experience is the best teacher. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(1):51–64, 2017.

[58] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[59] Nikolaus Hansen. The CMA evolution strategy: a comparing review. In I. Inza E. Bengoetxea J.A. Lozano, P. Larrañaga, editor, *Towards a New Evolutionary Computation*, pages 75–102. Springer, 2006.

[60] Nikolaus Hansen. The CMA Evolution Strategy: A Tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[61] Nikolaus Hansen, Steffen Finck, Raymond Ros, and Anne Auger. Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. Technical report, INRIA, 2009.

[62] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[63] Emma Hart. *Lifelong Learning Machines: Towards Developing Optimisation Systems That Continually Learn*, pages 187–203. Springer International Publishing, Cham, 2022.

[64] Emma Hart and Kevin Sim. On the life-long learning capabilities of a NELLI*: A hyper-heuristic optimisation system. In *International Conference on Parallel Problem Solving from Nature*, pages 282–291. Springer, 2014.

[65] Demis Hassabis, Dharshan Kumaran, Christopher Summerfield, and Matthew Botvinick. Neuroscience-inspired artificial intelligence. *Neuron*, 95(2):245–258, 2017.

[66] Nils-Bastian Heidenreich, Anja Schindler, and Stefan Sperlich. Bandwidth selection for kernel density estimation: a review of fully automatic selectors. *AStA Advances in Statistical Analysis*, 97(4):403–433, 2013.

[67] Geoffrey E Hinton and Steven J Nowlan. How learning can guide evolution. *Adaptive Individuals in Evolving Populations: Models and Algorithms*, 26:447–454, 1996.

[68] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[69] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–154, 1962.

[70] Thomas Jatschka, Günther R Raidl, and Tobias Rodemann. A general cooperative optimization approach for distributing service points in mobility applications. *Algorithms*, 14(8):232, 2021.

[71] Hetvi Jethwani and Sumeet Agarwal. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 101–102, 2021.

[72] Min Jiang, Zhongqiang Huang, Liming Qiu, Wenzhen Huang, and Gary G Yen. Transfer learning-based dynamic multiobjective optimization algorithms. *IEEE Transactions on Evolutionary Computation*, 22(4):501–514, 2017.

[73] Min Jiang, Zhenzhong Wang, Liming Qiu, Shihui Guo, Xing Gao, and Kay Chen Tan. A fast dynamic evolutionary multiobjective algorithm via manifold transfer learning. *IEEE Transactions on Cybernetics*, 51(7):3417–3428, 2020.

[74] Yaochu Jin. *Knowledge Incorporation in Evolutionary Computation*. Springer, 2005.

[75] Yaochu Jin, Handing Wang, Tinkle Chugh, Dan Guo, and Kaisa Miettinen. Data-driven evolutionary optimization: An overview and case studies. *IEEE Transactions on Evolutionary Computation*, 23(3):442–458, 2018.

[76] Terry Jones and Stephanie Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the 6th International Conference on Genetic Algorithms*, volume 95, pages 184–192, 1995.

[77] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, 1995.

[78] Pascal Kerschke, Holger H Hoos, Frank Neumann, and Heike Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, 27(1):3–45, 2019.

[79] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations*, 2015.

[80] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[81] Barış Koçer and Ahmet Arslan. Genetic transfer learning. *Expert Systems with Applications*, 37(10):6997–7002, 2010.

[82] Teuvo Kohonen. Essentials of the self-organizing map. *Neural Networks*, 37:52–65, 2013.

[83] Loizos Kounios, Jeff Clune, Kostas Kouvaris, Günter P Wagner, Mihaela Pavlicev, Daniel M Weinreich, and Richard A Watson. Resolving the paradox of evolvability with learning theory: How evolution learns to improve evolvability on rugged fitness landscapes. *arXiv preprint arXiv:1612.05955*, 2016.

[84] Jakob Kruse. Technical report: Training mixture density networks with full covariance matrices. *arXiv preprint arXiv:2003.05739*, 2020.

[85] Dhireesha Kudithipudi, Mario Aguilar-Simon, Jonathan Babb, Maxim Bazhenov, Douglas Blackiston, Josh Bongard, Andrew P Brna, Suraj Chakravarthi Raja, Nick Cheney, Jeff Clune, et al. Biological underpinnings for lifelong learning machines. *Nature Machine Intelligence*, 4(3):196–210, 2022.

[86] Manuel Laguna and Rafael Marti. Experimental testing of advanced scatter search designs for global optimization of multimodal functions. *Journal of Global Optimization*, 33(2):235–255, 2005.

[87] Pedro Larrañaga and Jose A Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, volume 2. Springer Science & Business Media, 2001.

[88] Yann Lecun and Yoshua Bengio. *Convolutional networks for images, speech, and time-series.* MIT Press, 1995.

[89] Joel Lehman, Jeff Clune, Dusan Misevic, Christoph Adami, Lee Altenberg, Julie Beaulieu, Peter J Bentley, Samuel Bernard, Guillaume Beslon, David M Bryson, et al. The surprising creativity of digital evolution: A collection of anecdotes from the evolutionary computation and artificial life research communities. *Artificial Life*, 26(2):274–306, 2020.

[90] Erez Lieberman, Christoph Hauert, and Martin A Nowak. Evolutionary dynamics on graphs. *Nature*, 433(7023):312–316, 2005.

[91] Lei Liu, Chengshan Pang, Weiming Liu, and Bin Li. Learning to describe collective search behavior of evolutionary algorithms in solution space. In *Asia-Pacific Conference on Simulated Evolution and Learning*, pages 196–207. Springer, 2017.

[92] Jonathan B Losos. *The Princeton Guide to Evolution.* Princeton University Press, 2017.

[93] Sushil J Louis and John McDonnell. Learning with case-injected genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 8(4):316–328, 2004.

[94] Loïc Marrec, Irene Lamberti, and Anne-Florence Bitbol. Toward a universal model for spatially structured populations. *Physical Review Letters*, 127(21):218102, 2021.

[95] Norman McRae. John von Neumann. *New York, NY: Pantheon*, 1992.

[96] Olaf Mersmann, Bernd Bischl, Heike Trautmann, Mike Preuss, Claus Weihs, and Günter Rudolph. Exploratory landscape analysis. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, pages 829–836, 2011.

[97] Diego Mesquita, Amauri Souza, and Samuel Kaski. Rethinking pooling in graph neural networks. *Advances in Neural Information Processing Systems*, 33:2220–2231, 2020.

[98] Ryan Meuth, Meng-Hiot Lim, Yew-Soon Ong, and Donald C Wunsch. A proposition on memes and meta-memes in computing for higher-order learning. *Memetic Computing*, 1(2):85–100, 2009.

[99] Risto Miikkulainen and Stephanie Forrest. A biological perspective on evolutionary computation. *Nature Machine Intelligence*, 3(1):9–15, 2021.

[100] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pages 293–312. Elsevier, 2019.

[101] Alan Tan Wei Min, Yew-Soon Ong, Abhishek Gupta, and Chi-Keong Goh. Multiproblem surrogates: Transfer evolutionary multiobjective optimization of computationally expensive problems. *IEEE Transactions on Evolutionary Computation*, 23(1):15–28, 2017.

[102] Tom M Mitchell. The need for biases in learning generalizations. Technical Report CBM-TR-117, Laboratory for Computer Science Research, Rutgers University, 1980.

[103] Mario A Muñoz, Yuan Sun, Michael Kirley, and Saman K Halgamuge. Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges. *Information Sciences*, 317:224–245, 2015.

[104] Ferrante Neri and Carlos Cotta. Memetic algorithms and memetic computing optimization: A literature review. *Swarm and Evolutionary Computation*, 2:1–14, 2012.

[105] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International Conference on Machine Learning*, pages 2014–2023. PMLR, 2016.

[106] Stefano Nolfi and Dario Floreano. Learning and evolution. *Autonomous Robots*, 7(1):89–113, 1999.

[107] Gabriela Ochoa, Katherine M Malan, and Christian Blum. Search trajectory networks: A tool for analysing and visualising the behaviour of metaheuristics. *Applied Soft Computing*, 109:107492, 2021.

[108] Yew-Soon Ong and Abhishek Gupta. Evolutionary multitasking: a computer science view of cognitive multitasking. *Cognitive Computation*, 8(2):125–142, 2016.

[109] Yew-Soon Ong, Meng Hiot Lim, and Xianshun Chen. Memetic computation - past, present & future. *IEEE Computational Intelligence Magazine*, 5(2):24–31, 2010.

[110] José Carlos Ortiz-Bayliss, Hugo Terashima-Marín, and Santiago Enrique Conant-Pablos. Lifelong learning selection hyper-heuristics for constraint satisfaction problems. In *Mexican International Conference on Artificial Intelligence*, pages 190–201. Springer, 2015.

[111] Sinno Jialin Pan, Ivor W Tsang, James T Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks*, 22(2):199–210, 2011.

[112] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[113] Chengshan Pang, Mang Wang, Weiming Liu, and Bin Li. Learning features for discriminative behavior analysis of evolutionary algorithms via slow feature analysis. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, pages 1437–1444, 2016.

[114] Andreas Pavlogiannis, Josef Tkadlec, Krishnendu Chatterjee, and Martin A Nowak. Construction of arbitrarily strong amplifiers of natural selection using evolutionary graph theory. *Communications Biology*, 1(1):1–8, 2018.

[115] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[116] Google Research. Colaboratory. https://colab.research.google.com/, accessed: 2022.

[117] Robert G Reynolds. An introduction to cultural algorithms. In *Proceedings of the Third Annual Conference on Evolutionary Programming*, pages 131–139. World Scientific, 1994.

[118] Ryan Rossi and Nesreen Ahmed. The network data repository with interactive graph analytics and visualization. http://networkrepository.com, 2015.

[119] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. A metric for distributions with applications to image databases. In *Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271)*, pages 59–66. IEEE, 1998.

[120] Günter Rudolph. Finite markov chain results in evolutionary computation: A tour d'horizon. *Fundamenta Informaticae*, 35(1-4):67–89, 1998.

[121] Marc Rußwurm, Sébastien Lefèvre, Nicolas Courty, Rémi Emonet, Marco Körner, and Romain Tavenard. End-to-end learning for early classification of time series. *arXiv preprint arXiv:1901.10681*, 2019.

[122] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. *arXiv preprint arXiv:1710.09829*, 2017.

[123] Tara N Sainath, Oriol Vinyals, Andrew Senior, and Haşim Sak. Convolutional, long short-term memory, fully connected deep neural networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4580–4584. IEEE, 2015.

[124] Hiroki Sayama. Guiding designs of self-organizing swarms: Interactive and automated approaches. In *Guided Self-Organization: Inception*, pages 365–387. Springer, 2014.

[125] David W Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization.* John Wiley & Sons, 2nd edition, 2015.

[126] Thomas W Sederberg and Scott R Parry. Free-form deformation of solid geometric models. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, pages 151–160, 1986.

[127] Moritz Seiler, Janina Pohl, Jakob Bossek, Pascal Kerschke, and Heike Trautmann. Deep learning as a competitive feature-free approach for automated algorithm selection on the traveling salesperson problem. In *International Conference on Parallel Problem Solving from Nature*, pages 48–64. Springer, 2020.

[128] Joan Serrà, Santiago Pascual, and Alexandros Karatzoglou. Towards a universal neural network encoder for time series. In *21st International Conference of the Catalan Association for Artificial Intelligence*, pages 120–129. IOS Press, 2018.

[129] Y Shevchuk. NeuPy: Neural Networks in Python. http://neupy.com/pages/home.html, accessed: November 2020.

[130] Harsh Shrivastava, Xinshi Chen, Binghong Chen, Guanghui Lan, Srinvas Aluru, Han Liu, and Le Song. Glad: Learning sparse graph recovery. *arXiv preprint arXiv:1906.00271*, 2019.

[131] Daniel Sieger, Stefan Menzel, and Mario Botsch. A comprehensive comparison of shape deformation methods in evolutionary design optimization. In *Proceedings of the International Conference on Engineering Optimization*, pages 1–5, 2012.

[132] Daniel L Silver, Qiang Yang, and Lianghao Li. Lifelong machine learning systems: Beyond learning algorithms. In *2013 AAAI Spring Symposium Series*, 2013.

[133] David Silver, Satinder Singh, Doina Precup, and Richard S Sutton. Reward is enough. *Artificial Intelligence*, page 103535, 2021.

[134] Kevin Sim, Emma Hart, and Ben Paechter. A lifelong learning hyper-heuristic method for bin packing. *Evolutionary Computation*, 23(1):37–67, 2015.

[135] Dan Simon. *Evolutionary Optimization Algorithms*. John Wiley & Sons, 1st edition, 2013.

[136] Kate A Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR)*, 41(1):6, 2009.

[137] Kenneth Sörensen. Metaheuristics—the metaphor exposed. *International Transactions in Operational Research*, 22(1):3–18, 2015.

[138] Kenneth O Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35, 2019.

[139] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Deep neuroevolution: Genetic algorithms are a competitive

alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 2017.

[140] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction.* MIT Press, 2018.

[141] Kevin Swersky, Jasper Snoek, and Ryan P Adams. Multi-task bayesian optimization. *Advances in Neural Information Processing Systems*, 26, 2013.

[142] Abdulkadir Tasdelen and Baha Sen. A hybrid CNN-LSTM model for pre-miRNA classification. *Scientific Reports*, 11(1):1–9, 2021.

[143] Ye Tian, Xingyi Zhang, Cheng He, Kay Chen Tan, and Yaochu Jin. Principled design of translation, scale, and rotation invariant variation operators for metaheuristics. *arXiv preprint arXiv:2105.10657*, 2021.

[144] Alan Turing. Intelligent machinery. 1948. *The Essential Turing*, page 395, 1969.

[145] Mikdam Turkey and Riccardo Poli. An empirical tool for analysing the collective behaviour of population-based algorithms. In *European Conference on the Applications of Evolutionary Computation*, pages 103–113. Springer, 2012.

[146] Tobias Uller, Armin P Moczek, Richard A Watson, Paul M Brakefield, and Kevin N Laland. Developmental bias and evolution: A regulatory network perspective. *Genetics*, 209(4):949–966, 2018.

[147] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[148] Petar Veličković and Charles Blundell. Neural algorithmic reasoning. *Patterns*, 2(7):100273, 2021.

[149] Aymeric Vie, Alissa M Kleinnijenhuis, and Doyne J Farmer. Qualities, challenges and future of genetic algorithms: a literature review. *arXiv preprint arXiv:2011.05277*, 2020.

[150] C L Camacho Villalón, T Stützle, and M Dorigo. Cuckoo Search$\equiv(\mu+\lambda)$–Evolution Strategy. Technical Report No. 2021-006, IRIDIA, Université Libre de Bruxelles, 2021.

[151] Christian Leonardo Camacho Villalón, Thomas Stützle, and Marco Dorigo. Grey wolf, firefly and bat algorithms: Three widespread algorithms that do not contain any novelty. In *International Conference on Swarm Intelligence*, pages 121–133. Springer, 2020.

[152] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3):261–272, 2020.

[153] John von Neumann. *The Computer and the Brain.* Yale University Press, 2012.

[154] Jane X Wang. Meta-learning in natural and artificial intelligence. *Current Opinion in Behavioral Sciences*, 38:90–95, 2021.

[155] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 1578–1585. IEEE, 2017.

[156] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big Data*, 3(1):1–40, 2016.

[157] Danny Weyns, Thomas Bäck, Renè Vidal, Xin Yao, and Ahmed Nabil Belbachir. Lifelong computing. *arXiv preprint arXiv:2108.08802*, 2021.

[158] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[159] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[160] Haipeng Xiao, Miguel Angel Sotelo, Yulin Ma, Bo Cao, Yuncheng Zhou, Youchun Xu, Rendong Wang, and Zhixiong Li. An improved LSTM model for behavior recognition of intelligent vehicles. *IEEE Access*, 8:101514–101527, 2020.

[161] Zhengzheng Xing, Jian Pei, and Eamonn Keogh. A brief survey on sequence classification. *ACM SIGKDD Explorations Newsletter*, 12(1):40–48, 2010.

[162] Huaxiang Zhang and Jing Lu. Adaptive evolutionary programming based on reinforcement learning. *Information Sciences*, 178(4):971–984, 2008.

[163] Jun Zhang, Weien Zhou, Xianqi Chen, Wen Yao, and Lu Cao. Multisource selective transfer framework in multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 24(3):424–438, 2019.

[164] Pan Zhang, Xin Yao, Lei Jia, Bernhard Sendhoff, and Thorsten Schnier. Target shape design optimization by evolving splines. In *2007 IEEE Congress on Evolutionary Computation*, pages 2009–2016. IEEE, 2007.