Uncertainty Handling in Surrogate Assisted Optimisation of Games

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Technischen Universität Dortmund

an der Fakultät für Informatik

von

Vanessa Volz

Dortmund

2019

Tag der mündlichen Prüfung	**25. Februar 2019**
Dekan:	**Prof. Dr.-Ing. Gernot A. Fink**
Gutachter:	**Prof. Dr. Günter Rudolph** (TU Dortmund, Fakultät für Informatik)
	Prof. Dr. Mike Preuss (Universiteit Leiden, LIACS)

Executive Summary

In this thesis entitled *Uncertainty handling in surrogate assisted optimisation of games*, we started out with the goal to investigate the uncertainty in game optimisation problems, as well as to identify or develop suitable optimisation algorithms. In order to approach this problem systematically, we first created a benchmark consisting of suitable game optimisation functions (GBEA). The suitability of these functions was determined using a taxonomy that was created based on the results of a literature survey of automatic game evaluation approaches. In order to improve the interpretability of the results, we also implemented an experimental framework that adds several features aiding the analysis of the results, specifically for surrogate-assisted evolutionary algorithms.

After describing potentially suitable algorithms, we proposed a promising algorithm (SAPEO), to be tested on the benchmark alongside state-of-the-art optimisation algorithms. SAPEO is utilising the observation that most evolutionary algorithms only need fitness evaluations for survival selections. However, if the individuals in a population can be distinguished reliably based on predicted values, the number of function evaluations can be reduced. After a theoretical analysis of the performance limits of SAPEO, which produced very promising insights, we conducted several sets of experiments in order to answer the three central hypotheses guiding this thesis. We find that SAPEO performs comparably to state-of-the-art surrogate-assisted algorithms, but all are frequently outperformed by stand-alone evolutionary algorithms. From a more detailed analysis of the behaviour of SAPEO, we identify a few pointers that could help to further improve the performance.

Before running experiments on the developed benchmark, we first verify its suitability using a second set of experiments. We find that GBEA is practical and contains interesting and challenging functions. However, we also discover that, in order to produce interpretable result with the benchmark, a set of baseline results is required. Due to this issue, we are not able to produce meaningful results with the GBEA at the time of writing. However, after more experiments are conducted with the benchmark, we will be able to interpret our results in the future. The insights developed will most likely not only be able to provide an assessment of optimisation algorithms, but can also be used to gain a deeper understanding of the characteristics of game optimisation problems.

Acknowledgements

I would like to thank...

- ... Günter, my supervisor, for his feedback and support, but also for the academic freedom I have enjoyed.
- ... all my collaborators on several publications over the last years for their valuable insights, interesting discussions and constant nitpicking. I am glad I was able to use these papers (and bibfiles!) as a foundation for my thesis. Special mentions go out to my frequent co-authors Boris and Mike!
- ... everyone that agreed to proofread for their feedback. Thanks mum, dad, Andy, Günter, Mike and Boris!
- ... friends and family for their encouragements and interest in my work. Shout-out to Lisa, hope to see you in London soon!
- ... my colleagues for a great working environment at the LS11. Special thanks to Jan and Christiane, my office roommates, as well as Gundel for her great support in everything. Thanks also to the colleagues in Leiden, NYU and at Brown, I thoroughly enjoyed my stints in your labs. Thanks also for the warm welcome to the people at QMUL. The thanks also extend to the regulars at GECCO and CIG/CoG, and the attendees of the 2017 Dagstuhl seminar and the SAMCO workshop.
- ... my teachers, professors and fellow students over the years that eventually led me down the path to my PhD, of which there are too many to name.

But mostly, I'm thankful that it is done! :)

Table of Contents

CHAPTER 1

Introduction

1.1 Motivation

Real-world optimisation problems often exhibit a so called observation uncertainty, because physical measurements usually contain errors [4]. This uncertainty can cause issues, especially if the application is sensitive, such as in case of medical treatments [111]. Research in the field of noisy optimisation and real-world applications targets this and related issues.

However, additional sources of uncertainty can also be introduced by the optimisation algorithm itself. Surrogate-assisted optimisation, for instance, relies on the predictions of machine learning models, which of course can be erroneous. Still, most state-of-the-art optimisation algorithms assisted by surrogate models do not take into account the uncertainty introduced by modelling.

Both observation and prediction uncertainty are often modelled by a symmetric error distribution with a mean of 0. This type of modelling is reasonable in most applications, but there are sources of uncertainty that can likely not be described by symmetric error distributions. For example, real-world problems that, instead of the actual fitness function, use simulations to estimate the fitness of a given solution might introduce a non-symmetric bias to the evaluation.

For instance, in the research field of computational intelligence in games, many approaches require an evaluation method for a game or game content, such as a level. A common usecase are functions intended to describe the difficulty of an automatically generated game [130, 150]. We call this type of problem *game optimisation* in the remainder of this thesis. Human behaviour is then often modelled by an AI player. While the field of player modelling persistently works towards the goal of creating human-like AI, it is still unclear in most cases how to quantitatively express differences in behaviour [57, 146]. It is even less clear, how these differences affect automatic game evaluation measures based on AI behaviour.

Besides game-related optimisation problems, there is a large number of real-world problems that, too, rely on simulated evaluations. Simulations become necessary in cases where the actual fitness function is either (1) too expensive to compute or (2) carries a safety or security risk. For shape optimisation problems, for example, simulations using cheaper computational fluid dynamics models are often employed [28]. The need for simulations becomes especially prevalent when the problem involves the need to predict or react to human behaviour, as it is difficult to obtain data to train a model on.

To summarise, there is a large number of simulation-based real-world optimisation problems, where certain fitness evaluations are either non-existent or very limited. Game optimisation problems are a prominent group of problems that fall into this category. Still, existing research rarely addresses the uncertainty incurred by the various models and algorithms. This thesis is intended to fill this gap by incorporating approaches from noisy and surrogate-based optimisation, as well as detailed information on game optimisation problems to develop a uniquely suited algorithm. The main feature of the developed algorithm, SAPEO, is its usage of uncertainty information to assess the confidence of the obtained fitness estimates to dynamically decide whether to use an estimate or the correct fitness value.

The algorithm is evaluated based on its theoretical performance limits, as well as an established benchmark of diverse (albeit artificial) functions. However, artificially created functions usually have a discernible global structure that can be learned with machine learning techniques. This is not necessarily true for real-world problems. Therefore, the type and magnitude of model prediction errors likely differ between artificial and real-world functions, which can affect an optimisation algorithm based on these predictions. We therefore develop extend the existing benchmarking framework and add game optimisation function suites. The benchmark is then also used to compare the behaviour of SAPEO to state-of-the-art algorithms developed to solve computationally expensive (real-world) problems.

While this benchmark is naturally not representative for all types of problems imaginable, it serves as a demonstration of the effect of differences in uncertainties. We chose to add game optimisation problems specifically for several reasons:

1. Games describe highly complex systems, but their true state is always completely observable. This is a contrast to problems that rely on real-world measurements such as described in [28].

2. Games are designed for human decision makers and at the same time often have a player AI that allows the simulation of playthroughs.

3. The popularity of games paired with an increasing research and popular interest[1] make large datasets available[2] that are required for statistical analysis.

4. Game optimisation does not pose safety concerns.

5. Actual evaluations can be comparatively cheap, as no measurement equipment is required and typical game sessions do not last for more than a few hours at a time.

In addition to these reasons why games are a suitable test bed for researching uncertainty in optimisation problems, this study is also important in the context of games research. As demonstrated in section 3.1, the potential errors in game evaluation are rarely considered in game optimisation. This is despite the fact that the bias introduced for example by using an AI for game simulation has a demonstrable effect on the

[1]see recent successes of OpenAI's DotA AI https://openai.com/five/

[2]e.g. for StarCraft II [144] or League of Legends https://developer.riotgames.com/

discoverable solutions for the problem, as we illustrate in appendix B. The algorithm we propose considers uncertainty information and might thus lead to less biased results. We investigate this assumption further by the hypotheses specified in the following.

1.2 Hypotheses

Based on the above, we formulate the following hypotheses that are the central investigative part of this thesis:

H1 SAPEO exhibits comparable performance to comparable state-of-the-art optimisation algorithms on established benchmarks.

H2 Game optimisation problems can be used as a challenging benchmarking suite for optimisation algorithms.

H3 Due to its awareness of inherent uncertainties, SAPEO outperforms comparable state-of-the-art optimisation algorithms on the game optimisation benchmark.

These claims are investigated in detail in the remainder of this thesis. The results are described in section 5.

1.3 Contributions

The main contributions made in this thesis and specifically by the author are detailed in the following. The author is the first author and main contributor of all previously published work this thesis is based on.

Taxonomy of Game Evaluation Methods A taxonomy of approaches to game evaluation methods used for game (content) optimisation from a data-driven perspective. The taxonomy allows reasoning about uncertainties introduced by popular game evaluation approaches.

As of yet, the taxonomy is not published, but is part of a more extensive journal article including a survey of game evaluation methods currently under review. The taxonomy was developed based on discussion with the second author of this article, Boris Naujoks.

Illustrative Example of Effects of Bias in Game Optimisation A detailed illustrative example of how modelling human players using AIs can bias the obtained outcomes based on published data for the game StarCraft II (Blizzard 2010). The example conclusively shows the need for uncertainty handling in simulation-based game-related optimisation.

The example is based on an algorithm for StarCraft II winner prediction as published in [VPB18] with my co-authors Mike Preuss and Mathias K. Bonde. The analysis presented in this paper was however not published yet and is based on newly generated results from different datasets. The example also profited from discussion with Boris Naujoks.

Game-Benchmark for Evolutionary Algorithms An extension of the existing COCO benchmarking framework for numerical black-box optimisation[3] that introduces two new functions suites based on game optimisation problems. The benchmark code, results and further information is freely available[4]. Two workshops around the benchmark were also organised for GECCO 2018 and 2019 in collaboration with Tea Tušar, Boris Naujoks and Pascal Kerschke. The interface for the new functions to the existing framework was implemented by Tea Tušar.

The new function suites are based on two previous publications. The first is an optimisation problem for cards in TopTrumps that was introduced to demonstrate the feasibility of automatic game balancing with and without surrogate models in collaboration with Boris Naujoks and Günter Rudolph in [VRN16]. The second function suite is based on a significantly extended version of a procedural content generation technique for Super Mario Bros. levels proposed in [Vol+18] developed with Jacob Schrum, Jialin Liu, Simon M. Lucas, Adam Smith and Sebastian Risi[5].

SAPEO Algorithm SAPEO, an algorithm designed for the robust optimisation of games and similar complex simulation-based real-world problems. The algorithm is extensively evaluated theoretically and empirically. Benchmarks on artificial functions do attest SAPEO robust performance, especially for complex functions. However, we were not able to obtain a meaningful interpretation of the algorithm's performance on the game benchmark.

Previous versions of SAPEO have been published for both single- and multi-objective optimisation problems in [VRN17a] and [VRN17b] in collaboration with Günter Rudolph and Boris Naujoks [6]. The version of SAPEO proposed in this thesis is however improved based on findings from previous publications and extended by model validation features as suggested by Alma Rahat. The results presented in this thesis are thus novel and not previously published. The theoretical performance assessment was supported by Michael Emmerich in context of a short term scientific mission[7].

Modular Implementation of Experiments A modular and thus easily extensible implementation of the experiments available on GitHub[8]. The implementation contains a C++ interface to the COCO benchmarking framework and adds extensive logging capabilities including the ability to track prediction errors and post-processing features.

[3]NumBBO COCO https://github.com/numbbo/coco

[4]http://url.tu-dortmund.de/gamesbench

[5]The publication is the result of work at Dagstuhl Seminar 17471 - Artificial and Computational Intelligence in Games: AI-Driven Game Design (2017)

[6]The main idea was developed from discussions at Lorentz center workshop SAMCO: Surrogate-Assisted Multi-Criteria Optimization (2016)

[7]sponsored by COST Action CA15140: Improving Applicability of Nature-Inspired Optimisation by Joining Theory and Practice (ImAppNIO) in 2016

[8]https://github.com/TheHedgeify/uncertaincoco

The code uses the Shark machine learning library[9], which makes the implementation easily understandable and extensible.

The software also allows for a fair comparison of surrogate-based optimisation algorithms by allowing implemented algorithm access only to the same modelling features. These features include sampling and modelling techniques combined from various different publications. Model management strategies that are currently implemented are pre-screening, efficient global optimisation and SAPEO.

Related Publications

[VRN17a]	V. Volz, G. Rudolph and B. Naujoks. 'Surrogate-Assisted Partial Order-Based Evolutionary Optimisation'. In: *Evolutionary Multi-Criterion Optimization (EMO)*. Springer, Berlin, 2017, pp. 639–653.
[VPB18]	V. Volz, M. Preuss and M. K. Bonde. 'Towards Embodied and Interpretable StarCraft II Winner Prediction'. In: *Computer Games Workshop at International Joint Conference on Artificial Intelligence (ICJACI)*. (in press). 2018.
[VRN16]	V. Volz, G. Rudolph and B. Naujoks. 'Demonstrating the Feasibility of Automatic Game Balancing'. In: *Genetic and Evolutionary Computation Conference (GECCO)*. ACM Press, New York, 2016, pp. 269–276.
[VRN17b]	V. Volz, G. Rudolph and B. Naujoks. 'Investigating Uncertainty Propagation in Surrogate-Assisted Evolutionary Algorithms'. In: *Genetic and Evolutionary Computation Conference (GECCO)*. ACM Press, New York, 2017, pp. 881–888.
[Vol+18]	V. Volz et al. 'Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network'. In: *Genetic and Evolutionary Computation Conference (GECCO)*. ACM Press, New York, 2018, pp. 221–228.

1.4 Limitations

Due to the interdisciplinary and complex nature of game optimisation and surrogate-assisted optimisation, we have chosen to focus our study on easily observable and well-researched aspects of both research fields. Limiting the issues addressed in this thesis also serves to foster a more streamlined and in-depth analysis.

In this thesis, we therefore do not address the following aspects related to the topic of uncertainty handling in surrogate-assisted optimisation of games:

- There is a distinctive lack of formal validation of methods to evaluate game (content) in state-of-the-art research. This issue puts into question the practical applicability of this research to industrial game development. As the fitness functions

[9]http://image.diku.dk/shark/

implemented for the game benchmark are taken from previous publications, this issue is reflected in the games benchmark. We forego a thorough investigation of these fitness functions as it would require an extensive study with human participants and be beyond the scope of this thesis. However, the author is set to chair a IEEE CIS task force addressing this issue in the future. We also discuss several future work directions in this regard in section 6.2.1.1.

- Many real-world and game optimisation problems are not continuous in search- and objective space. However, most publications in surrogate-assisted optimisation address only continuous optimisation and employ Kriging models. In this thesis, we thus focus on continuous game optimisation problems. We discuss in section 6.2.3 potential ways to address mixed-integer optimisation with SAPEO by changing the surrogate model used.

1.5 Structure

In order to answer the hypotheses specified above, we first provide background information in chapter 2. In this chapter, we describe all general information regarding the optimisation algorithms used in this thesis. We also provide background on benchmarking, as well as the games intended for our function suites. Following this, we give an overview of related work in chapter 3.

Afterwards we describe our approach to analysing uncertainty handling in surrogate-assisted optimisation of games in more detail in chapter 4. An evaluation of the contributions can be found in chapter 5. We conclude this thesis with a summary of the results and a discussion of future work in chapter 6.

CHAPTER 2

Background

In the following, we describe the background information required for the description and interpretation of the experiments we conducted to investigate uncertainty handling.

2.1 Evolutionary Optimisation Algorithms

Evolutionary optimisation algorithms are optimisation algorithms that are designed based on some principles of biological evolution. They thus belong to the category of nature-inspired algorithms and are usually considered to be under the umbrella of computational intelligence methods.

In this thesis, we use three evolutionary algorithms on different optimisation tasks. In the following section, we thus first describe the underlying general concept of evolutionary algorithms for single- and multi-objective optimisation. Following that, we detail the specifics of the three algorithms in question, i.e. CMA-ES (section 2.1.3.1), MO-CMA-ES (section 2.1.3.3) and SMS-EMOA (section 2.1.3.2).

2.1.1 Concept

Evolutionary algorithms use the imagery of Darwinian evolution. Solutions of a problem are thus represented as individuals that have a fitness, i.e. the result of the objective function. The most low-level representation of an individual is usually called its genotype, after the concepts of genes in biology. The genotype is the level of representation that is modified during evolution. The phenotype is a more abstract representation of the individual and dependent on the genotype. In biology, the phenotype is the physical organism of an individual.

In the context of evolutionary algorithms, the genotype and phenotype are not always distinct. For example, when optimising real-valued functions

$$f : \mathbb{R}^n \to \mathbb{R}^m,$$

the genotype and phenotype of an individual are usually a point in the search space $x \in \mathbb{R}^n$. The corresponding fitness value of the individual is then $f(x) \in \mathbb{R}^m$. In contrast, consider the application of finding the optimal weights in a fixed artificial neural network that is used as a classifier. The genotype could then be a vector in $\mathbb{R}^n$ representing the different weights, where each index is assigned to a specific connection between neurons. The phenotype is then the resulting neural network, whereas its fitness is the resulting classification accuracy.

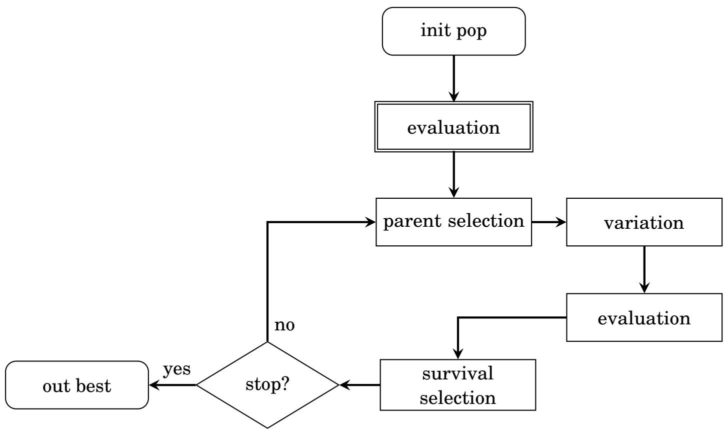


Figure 2.1: Algorithmic Skeleton of Evolutionary Algorithms

In the following, we will only address problems where the genotypic representation is a vector in $\mathbb{R}^n$. Most problems have such a representation, however, there are a number of applications where a different representation improves performance. One such application are problems where, on top of the weights in a neural network, its structure needs to be optimised as well. Several encodings have been proposed for this type of applications. One of the most popular ones was introduced in [128], where the genotype is split up into two parts. *Node genes* define nodes in the artificial neural network and their layer. *Connection genes* specify connections between these nodes as well as their weights.

Regardless of the representation chosen for the problem, an evolutionary algorithm usually still follows the same algorithmic skeleton as visualised in figure 2.1.

A population of individuals, i.e. a set of problem solutions, is generated as a first step. All individuals in the population are then evaluated to determine their fitness. From the population, a set of individuals is selected as parents to create new offspring from. These offspring are generated by variation operators that modify and/or recombine the genotypes of their parents. After the offspring are evaluated, the next generation of individuals is selected. The algorithm stops when a stopping criterion is met, for example a certain budget of function evaluations is exhausted. The output of the algorithm is finally the fittest individual or population.

2.1.1.1 Common Variations

Based on this general algorithmic skeleton, a variety of algorithms have been suggested. However, for most of the steps in the algorithm, different approaches have been suggested in the literature. In the following, we provide an overview of the most popular strategies for each of these steps. This is intended as a framework to characterise the algorithms used in the experiments for this thesis and help streamline their description. It is not intended to be exhaustive.

Selection Methods In an evolutionary algorithm, individuals are selected at two different steps for two different purposes:

- Reproduction: Parents for the creation of offspring
- Survival: Individuals that constitute the next generation

Both of these selection step should not favour worse individuals over fitter ones. While one selection step can be neutral, at least one should favour fitter individuals in order to progress the algorithm. Selection is thus typically based on the fitness values of individuals, but can also consider their genotype to control the diversity of a population.

In most algorithms, selection is either based on

- a total order of the individuals, where the top individuals are selected,
- a tournament, where in each round the better individual is chosen,
- purely chance.

In the case of survival selection, there is an important further distinction to be made. Consider an algorithm with μ parents and λ offspring. For the next generation, μ individuals need to be selected. These individuals can then either be selected from

- the offspring (μ,λ) or
- the union of parents and offspring $(\mu+\lambda)$.

$(\mu+\lambda)$ strategies have inherent elitism, that is the best solution in a generation is guaranteed to survive. This is not true for (μ,λ) selection strategies.

Variation Operators Two types of variation operators are popular, namely

- mutation: modification of one parent to create one offspring
- recombination: combination of two or more parents to create one offspring.

Either type of variation can be used exclusively in one algorithm, but typically, offspring are generated via recombination and then mutated with some probability.

Variation operators can be

- local, where only a subset of most close individuals can be reached by one variation,
- or global, where all feasible solutions can be reached by one variation.

In both cases, however, the probability of reaching more similar solutions, i.e. solutions closer in search space, is typically higher than generating more dissimilar ones.

In order to enable optimal behaviour of an evolutionary algorithm for an unknown optimisation problem, the variation operators chosen should fulfil the following three principles according to [35]

- reachability: All feasible solutions in search space should be reachable from any given solution after a finite number of repeated variation with a probability > 0.
- unbiasedness: The search direction should only depend on chance (unless favourable directions are known).
- control: The distance in search space between a given solution and any reachable solution after a single variation should be controllable.

As variation operators modify the genotype of an individual, they depend on its representation. For the most common representations, such as vectors in $\mathbb{B}^n, \mathbb{R}^n$ or $\mathbb{P}^n$, variation operators have been proposed that provably adhere to the principles described above. For less used representations, appropriate variation operators usually need to be specifically defined. For genotypes that are vectors in $\mathbb{R}^n$, the most commonly used variation operators are described below.

For mutation, the offspring is usually generated by adding noise sampled from a symmetric probability distribution with an expected value of 0. The support of this distribution can be bounded for local mutation, but is usually unbounded. In cases where some information about the fitness landscape of the problem at hand is available, the distribution can be chosen to be non-symmetric in order to bias the search into the intended direction. Popular choices of probability distributions are multivariate normal and polynomial distributions.

A popular recombination operator is simulated binary crossover (SBX). Assume we have two parents x, y and offspring z, all $\in \mathbb{R}^n$. x_i, y_i and z_i are the values of the respective individual at index $i \in \{1, n\}$. A polynomial distribution with highest and equal densities at x_i and y_i is then used to sample the value of z_i. This method is based on binary crossover popular in integer-valued optimisation.

Stopping Criteria Evolutionary algorithms are usually stopped after a predefined number of function evaluations or generations. However, since evolutionary algorithms are stochastic search algorithms, they can get trapped in local optima. In these cases, restarting the algorithm can potentially improve the best discovered solution.

However, it is not always straightforward to detect entrapment in a local optimum, especially if the fitness landscape is unknown. Instead, other methods have been suggested that hinge on performance measures. In [153], for example, convergence is determined base on statistical tests on the variance and regression trend.

2.1.1.2 Single- vs. Multi-objective Optimisation

Evolutionary algorithms are applicable to problems for both single- and multi-objective optimisation. When multiple objectives are considered at once, some individuals may be incomparable in terms of their fitness. This occurs e.g. in a two-objective minimisation problem when comparing individuals with the fitness values $(2,3)$ and $(3,2)$. As a result, the individuals cannot be sorted into a total order. This requires some alternative strategies for selection methods that depend on the fitness of individuals to make them comparable again.

Pareto dominance is a concept that induces a partial order on the individuals. A solution or individual x is said to strictly (Pareto) dominate another solution y (denoted $x \prec y$) iff x is better than y in all objectives. Considering minimisation this reads

$$x \prec y \quad \text{iff} \quad \forall i \in \{1,\ldots,m\}: \ f(x_i) < f(y_i)$$

for fitness function

$$f : X \subset \mathbb{R}^n \to \mathbb{R}^m, f(x) = \big(f_1(x),\ldots,f_m(x)\big).$$

Based on this, the set of all (Pareto) non-dominated and thus incomparable solutions as defined above is called Pareto set. The Pareto front is the image of the Pareto set under fitness function f.

The Pareto dominance relation can then be used to rank the individuals in a population. However, in order to work with the framework depicted in figure 2.1, making decisions between incomparable solutions is still necessary. Indicator-based MOEAs, as used in this thesis, decide based on information gained from performance indicators, which describe the quality of the acquired solutions at a given point in time. The most popular indicators are hypervolume contribution, additive ε and $R2$ indicator, all presented in detail by Knowles et al. [74].

In this thesis, we will be focusing on the hypervolume contribution as a secondary criterion, as dominated hypervolume is the only performance measure available in the benchmarking software we employ to date (COCO, see section 2.2). The dominated hypervolume, sometimes also called S metric, was originally proposed by [162] and the SMS-EMOA is designed around it, after it was made computationally viable [11, 36]. A definition of the hypervolume of a set of solutions M according to [36] follows. Consider the hypercubes a_i defined by their respective non-dominated point p_i and a nadir reference point x_{ref}. These hypercubes cover the range of solutions included in the solution set. The hypervolume is then the Lebesgue measure Λ of the union of hypercubes (see [36], eq. 1):

$$S(M) := \Lambda\left(\left\{\bigcup_i a_i | p_i \in M\right\}\right) = \Lambda\left(\bigcup_{p \in M} \{x | p \prec x \prec x_{ref}\}\right) \tag{2.1}$$

The hypervolume contribution of a given point $m \in M$ can be expressed by the difference between the dominated hypervolume of the set M with and without point m (see [36], eq. 3):

$$\Delta_s(m,M) := S(M) - S(M \setminus \{m\}) \tag{2.2}$$

2.1.2 Evolutionary Optimisation under Uncertainty

Real world applications in many cases signify uncertain environments for an optimisation algorithm. In their survey [67], the authors distinguish between four types of uncertainties that need to be handled. In this thesis, we will apply uncertainty handling technique from research on noise and fitness approximations.

Noise The observed fitness value is subject to additive, symmetric noise, caused, for example, by sensory measurement errors. In order to not mislead evolutionary algorithms operating on noisy fitness functions, an average of multiple evaluations is usually used to estimate the true value of an individual.

Besides explicitly evaluating an individual multiple times to combat noise on fitness functions and its implicit counterpart, where re-sampling is steered by the evolutionary algorithms, there are other strategies that modify the selection step in an evolutionary algorithm. The selection is modified in such a way, that the operator takes into account the uncertainty in the fitnesses of two individuals that are compared.

One popular way is to define relations that induce a partial order on uncertain individuals. For example, in case the uncertainty intervals are bounded as in [110], we can make the following comparisons of two uncertain individuals A and B. Let $f(X)$ be the true fitness of individual X and its measured fitness $\tilde{f}(X) \in [X_l, X_u]$, with X_l and X_u the lower and upper bounds of the uncertainty interval. Of course, it also must hold that $f(X) \in [X_l, X_u]$. [110] then defines the following relation

$$[X_l, X_u] \prec [Y_l, Y_u] \iff X_u < Y_l \tag{2.3}$$
$$[X_l, X_u] = [Y_l, Y_u] \iff X_l = Y_l \wedge X_u = Y_u \tag{2.4}$$
$$[X_l, X_u] \preceq [Y_l, Y_u] \iff X \prec Y \vee X = Y \tag{2.5}$$

The author of [110] then suggests to use evolutionary algorithms designed to find minimal elements of partially ordered sets. In case the uncertainty is not bounded, a threshold can be introduced [90]. The effect of these thresholds on the comparisons and their statistical significance has been investigated previously as well [10].

Robustness The search point of the individual might change slightly due, e.g., to manufacturing tolerances. A solution is robust if the fitness is still satisfactory for all points within the tolerance. The *effective fitness* of an individual is then the expected fitness value, considering all possible modifications of the search point and their probability.

Fitness Approximation In applications with very expensive fitness functions, so-called meta-models are sometimes used to estimate the fitness function (see section 3.2). Since these models incur a systemic bias that cannot be reduced by repeated evaluation, they are usually used in conjunction with the true fitness function. The decision which fitness function to evaluate is made based on model-management strategies surveyed in [65, 66]. The different strategies we address here are described in section 3.2 in context of the specific algorithms.

Time-varying Fitness Functions While the fitness function is deterministic at a specific point in time t, its value depends on t. As a result, the changes in fitness need to be tracked if no full restart of the algorithm is desired.

2.1.3 Algorithms

In the following, we describe the three algorithms used in this thesis in more detail.

2.1.3.1 CMA-ES

The **C**ovariance **M**atrix **A**daptation **E**volution **S**trategy (CMA-ES) is the state-of-the-art algorithm for single-objective continuous optimisation. CMA-ES uses (μ, λ) survival selection and mutation from the population mean as the only variation operator. These mutations are guided by an iteratively computed covariance matrix estimated into the direction of the negative gradient of the given problem.

The CMA-ES thus consciously introduces a bias into its variation. This does, however, not violate the unbiasedness principle for evolutionary algorithms, as favourable directions are estimated based on the covariance matrix. CMA-ES can thus also be considered an **E**stimation of **D**istribution **A**lgorithm (EDA).

symbol	value	explanation
λ	$4+\lfloor(3\log n\rfloor)$	number of offspring [49]
μ	$\lfloor\frac{\lambda}{2}\rfloor$	number of parents[49]
w_i	$\frac{1}{\mu}$	individual weights
$ccov$	$\frac{2}{n^2}$	covariance matrix update weight[2]
σ	1	step size[2]

Table 2.1: CMA-ES parametrisation

According to the authors, the values of the various parameters are also part of the algorithm design and specified as listed in table 2.1. As described in [149], the mean m and covariance matrix C are adapted in a single iteration as follows (with rank-one update):

$$x_i = m + \sigma y_i, \quad y_i \sim N_i(0, C)$$

$$m = m + \sigma y_w \quad \text{where } y_w = \sum_{i=1}^{\mu} w_i y_{i:\lambda}$$

$$C = (1 - ccov)C + ccov\ \mu_w\ y_w y_w^T \quad \text{where } \mu_w = \frac{1}{\sum_{i=1}^{\mu} w_i^2} \geq 1$$

The implementation in Shark ML[1], the framework we use, is based on [50], where more details on the algorithm and default parameters can be found.

2.1.3.2 SMS-EMOA

The **S**-**M**etric **S**election - **E**volutionary **M**ulti-objective **E**volutionary **A**lgorithm (SMS-EMOA, proposed in [11]) is popularly used for continuous multi-objective optimisation.

[1]http://image.diku.dk/shark/

It typically uses a $(\mu+1)$ selection scheme, but other schemes are also possible. According to the original publication, this decision was mostly made due to the significant computational effort of the hypervolume computation.

Like most EMOAs, the algorithm first uses non-dominated sorting to rank the solutions in a population. The secondary ranking criterion is based on the contribution of a solution to the population's hypervolume (i.e the amount of objective space covered by a Pareto front w.r.t. $(\max_{x\in X} f_1(x)+1,\dots,\max_{x\in X} f_m(x)+1)$ as reference point). See section 2.1.1.2 for more details on the hypervolume contribution measure.

SMS-EMOA is commonly used in conjunction with the most widely used variation operators in the field, namely simulated binary crossover and polynomial mutation, cf. Deb [29]. In [153], online convergence detection for the SMS-EMOA are described and improved. In [11], it was shown that the SMS-EMOA exhibits a robust performance independent of the specific implementation. Moreover, what is specifically relevant in this thesis, in the authors' experiments, the algorithm performed well on challenging real-world applications (in this case, the optimisation of airfoils).

The implementation in Shark ML[2], the framework we use, is based on [11], where more details on the algorithm and default parameters can be found.

2.1.3.3 MO-CMA-ES

The MO-CMA-ES resulted from applying the step size and covariance matrix adaptation from CMA-ES (see section 2.1.3.1) to a multi-objective evolutionary optimisation framework [61]. In order to rank the solutions in a given population, MO-CMA-ES employs non-dominated sorting, as most EMOAs do. Like the SMS-EMOA described in the previous section, the original publication uses hypervolume contribution as a secondary ranking criterion. However, other indicators popular for multi-objective optimisation have also been employed.

The extensive study in [61] suggests a $(\mu+1)$ selection scheme for the MO-CMA-ES, just like in the SMS-EMOA. In this case, the two algorithms only differ in terms of their variation operators and strategy adaptation. In a later publication, further refinements were made to the MO-CMA-ES. One that resulted in major performance improvements is a new step size adaptation procedure that besides the evaluation of the individual in question, also considers the success of their parents and / or the whole population [152].

The implementation in Shark ML[3], the framework we use, is based on [61, 152], where more details on the algorithm and default parameters can be found.

2.2 Benchmarking with the COCO framework

Benchmarks are a commonly used tool to compare different algorithms. The goal is often to determine the best algorithm for a given problem type. In this thesis, we use the **CO**mparing **C**ontinuous **O**ptimisers framework [51] for several reason:

[2]http://image.diku.dk/shark/

[3]http://image.diku.dk/shark/

- COCO is a popular framework with regular workshops in the EA research community.
- COCO enables easy comparisons with other algorithms without the need for re-implementation and fine-tuning. It includes data from well-known and best-performing algorithms, which ensures that the benchmark is not biased by insufficient hyperparametrisation for some algorithms.
- COCO measures performance of algorithms in relation to the number of function evaluations.

2.2.1 Core Concepts

One core intent of the COCO benchmarking framework is to measure anytime performance. This means that the framework does not only measure the performance of the algorithm after a set number of function evaluations, but instead can record each evaluation made. The progress of the optimiser can thus be expressed.

Precision Targets Furthermore, this progress is always expressed in terms of so-called precision targets. These targets are defined a-priori and specify the difference between the target value and the known globally best fitness fitness value. For multi-objective problems, the optimal value is expressed as the dominated hypervolume of the best population observed instead of the actual value of the fitness functions. If the global optimum is unknown, an estimate can be given. The framework allows for exceeding the previously estimated optimum. For each algorithm, the number of function evaluations needed to reach a given target can then be observed.

The advantage of this target-based approach is that it allows a meaningful comparison between different algorithms. To illustrate this, let's assume algorithm A requires 100 function evaluations to reach a precision of 10^{-3} on function f. Algorithm B requires 200 function evaluations to do the same. We can therefore observe that algorithm A is twice as fast as B to reach precision 10^{-3}.

Now let's further assume that algorithm A reaches a precision of 10^{-4} after 200 function evaluations. We now know that after the same number of function evaluations, algorithm A is closer to the global optimum than algorithm B (targets 10^{-4} vs. 10^{-3}). However, it is unclear how much more difficult reaching a higher precision really is. Algorithm B could potentially reach 10^{-4} at the very next function evaluation. A meaningful comparison of fitness values is thus not easily possible.

Average Runtime and Bootstrapping However, there are further caveats even when using the target-based approach. COCO has to account for the fact that many optimisation algorithms are stochastic. Therefore, a single set of observations does not suffice to express the behaviour of an algorithm. Additionally, since stochastic optimisation algorithm such as evolutionary algorithms cannot guarantee convergence to the global optimum, common practice is to restart the algorithm after either a fixed budget of function evaluations or after some form of convergence was detected. This practice increases the chance of finding the global optimum

COCO enforces simulated restarts by providing multiple instances for the same function and will automatically restart an algorithm when it stops before its allocated budget of function evaluations is exceeded. It can thus be safely assumed that all algorithms run on COCO are restarted. This assumption is used to give a more accurate estimate of the expected runtime of an algorithm until the optimum is found. The average runtime measure (aRT) of a restart algorithm can be measured by dividing the number of function evaluations conducted in all trials divided by the number of successful runs [52].

A further benefit of using aRT is that it allows for bootstrapping the estimated performance after a given amount of function evaluations, even when the algorithm was stopped earlier. For example, let's assume we evaluate algorithm A with 5 restarts. In our imaginary scenario, after 100 function evaluations, we observed the following final precision values 10^{-3}, 10^{-4}, 10^{-3}, 10^{-3}, 10^{-4}. Target 10^{-3} was reached after $100, 50, 100, 100, 50$ function evaluations.

The aRT for 10^{-3} is thus $\frac{100+50+100+100+50}{5} = 80$, while for 10^{-4} it is $\frac{100+100+100+100+100}{2} = 250$, a value larger than the 100 function evaluations we ran the algorithm for. The aRT can only be computed for a target that has been reached in at least one run, as this proves that the specific target is obtainable within a finite number of restarts.

Instances While COCO will automatically restart algorithms that stop before their allocated budget of function evaluation is exceeded, it also enforces repeated runs by implementing a feature called instances. Instances are shifted versions of a given function that are intended to have only slightly modified fitness landscapes from the original function. Algorithms have to run on all instances of a function, where each instance is interpreted as a restart.

Using these instances instead of restarting on the same function has the added benefit of providing a more robust measurement of performance by increasing the difficulty of overfitting to a function. A deterministic optimiser that was by chance started close to the global optimum of the original function would thus only have an advantage in one run. This avoids observing an uncharacteristically high performance on all restarts.

2.2.2 Post-Processing

The COCO framework provides a set of post-processing features based on the aRT-measure described above. Chief among them are ECDF plots, an example is depicted in figure 2.2.

The graphs show the expected average runtime of an algorithm for all precision targets. The targets are displayed in decreasing order on the y-axis, normalised to $[0,1]$. The respective average runtime for each of the targets is displayed on the x-axis in log-scale. The large cross signifies the number of function evaluations the algorithms were run for. The plots can be generated to compare the performance of a single algorithm

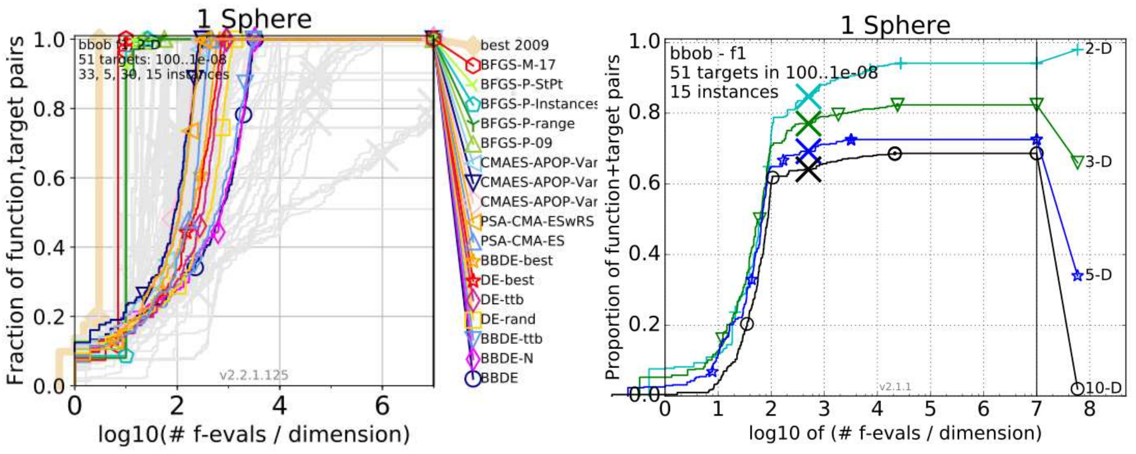


Figure 2.2: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/n) for 51 targets with target precision in $10^{[-8..2]}$ for the sphere function (fid 1). The “best 2009” line corresponds to the best aRT observed during BBOB 2009 for each selected target. Left: Comparison between various algorithm from the 2018 BBOB competitions on dimension 2. Right: Comparison between CMA-ES performance on different dimensions (2, 3, 5, 10).

across different dimensions 2.2 (right) or to compare different algorithms on the same function and dimension 2.2 (left).

Scaling behaviour is additionally visualised in separate figures, see figure 2.3 for an example. In these plots, the aRT in log-scale is plotted on the y-axis and the dimension of the problem is indicated on the x-axis. This way, scaling behaviour over search space dimension can be visualised for selected targets. The example plot shows 7 targets and the corresponding scaling behaviour. It can be clearly seen, that for lower target, i.e. less precise optimisation, the algorithm in question (CMA-ES) scales really well (almost linearly, as parallel to the horizontal lines). This observation is not true, however, for target 10^{-5}, which was not reached for the 10-dimensional problem and took significantly longer for dimension 5 when compared to dimension 2 and 3.

The values visualised in the scaling figures are also available in a table format, alongside additional statistics regarding a slew of statistical tests regarding the significance of the results obtained as well as their distribution. These tables provide the opportunity for further and more detailed analysis where the visualisations produced by the existing post-processing plots are not sufficient.

2.2.3 Function Suites

Different function suites can be integrated into COCO in order to test different types of algorithms and/or scenarios. Besides the standard single-objective BBOB function suite, there is a multi-objective, a noisy and an expensive version. In this thesis, we use BBOB and BBOB-BIOBJ, which are described below.

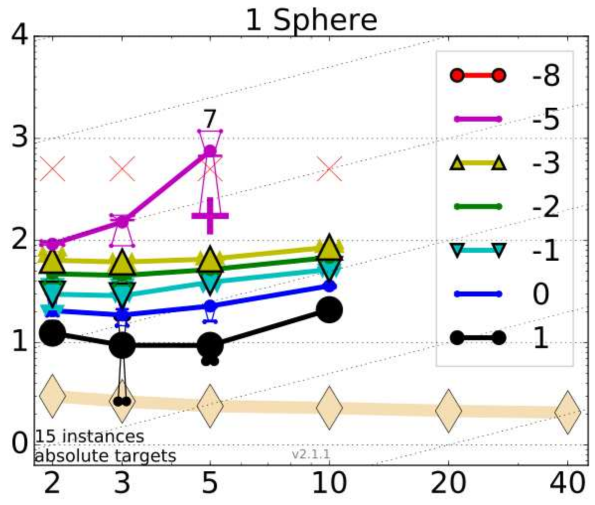


Figure 2.3: Average running time (aRT in number of f-evaluations as $\log_{10}$ value), divided by dimension for target function values versus dimension. Slanted grid lines indicate quadratic scaling with the dimension, while horizontal lines indicate linear scaling. Light symbols give the maximum number of function evaluations from the longest trial divided by dimension. Black stars indicate a statistically better result compared to all other algorithms with $p < 0.01$ and Bonferroni correction number of dimensions. Plot shows CMA-ES performance on the sphere function (fid 1).

2.2.4 BBOB Suite

with small modifications from [145]

BBOB is a single-objective **B**lack-**B**ox **O**ptimisation **B**enchmarking test suite [53] which contains 24 functions. In order to measure general algorithm performance across function types, the functions were selected such that the resulting benchmark would be diverse in terms of separability, conditioning, modality and global structure [53].

The test suite contains 15 instances for each function, which are generated using a combination of various transformations (e.g. linear, local non-linear, rotations) on the original functions. All of the functions in the test suites are defined for search spaces of multiple dimensions $d \in \{2,3,5,10,20\}$ in order to be able to evaluate a wide range of problem sizes. The global optimum of each of the functions is located in $[-5,5]^d \subset \mathbb{R}^d$.

2.2.5 BBOB-BIOBJ Suite

with small modifications from [145]

BBOB-BIOBJ is a **bi-obj**ective **B**lack-**B**ox **O**ptimisation **B**enchmarking test suite [141]. It consists of 55 bi-objective functions that are a combination of 10 of the 24 single-objective functions in the BBOB test suite. In order to measure general algorithm performance across function types, single-objective functions were chosen such that the resulting benchmark would be diverse in terms of separability, conditioning, modality and global structure [53]. Based on these properties, the single-objective functions are divided into 5 function groups, from which 2 functions are chosen each. The resulting

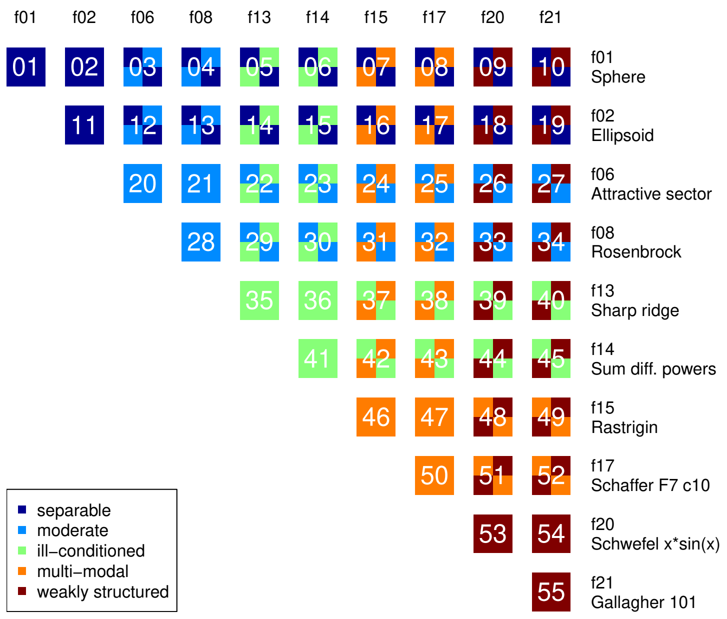


Figure 2.4: The 55 BBOB-BIOBJ functions are combinations of 10 single-objective functions (on the top and right). The groups the single-objective and the resulting bi-objective functions belong to are colour-coded according to the legend.

problems and corresponding properties are visualised in figure 2.4.

Each of the functions in this test suite has 10 instances, which are combinations of the existing instances for the single-objective functions. Like in the BBOB test suite, all functions are defined for dimensions $d \in \{2,3,5,10,20\}$. The global optimum of each of the separate single-objective function is contained in $[-100,100]^d \subset \mathbb{R}^d$.

The performance of an algorithm on the benchmarking suite is measured using a quality indicator expressing both the size of the obtained Pareto set and the proximity to a reference front. Since the true Pareto front is not known for the functions in the test suite, an approximation is obtained by combining all known solutions from popular algorithms. The ideal and nadir points are known, however, and used to normalise the quality indicator to enable comparisons across functions [15]. The metric reported as a performance measure for the algorithm is called precision. It is the difference of the quality indicator of the reference set I_{ref} and the indicator value of the obtained set. 58 target precisions are fixed and the number of function evaluations needed to achieve them is reported during a benchmark run. This way, the COCO platform enables an anytime comparison of algorithms, i.e. an evaluation of algorithm performance for each target precision and number of function evaluations [15]. In its current version (2018), the framework uses the hypervolume of all evaluated individuals as a performance indicator.

2.3 Kriging

The Kriging model was originally proposed as a geostatistical estimator to find gold based only on the locations of previous gold finds [91]. Kriging, also called Gaussian process regression, models a given fitness landscape using Gaussian processes. A prediction for a given point x is then made by interpolation. A more detailed description is given in the following based on [91].

Let f be a single-objective, continuous optimisation problem. We assume that f is a minimisation problem without loss of generality. Further, let $X \in \mathbb{R}^{M \times n}$ be a set of M samples from the n-dimensional search space. Let $D = \{(x_i, f(x))\}$, where $x_i \in \mathbb{R}^n$ is the i-th sample, be the *initial design*, where the sampled points in search space are paired with their function value. The initial design is the basis for the construction of a Kriging model.

The Kriging model is based on the assumption that a collection of random variables representing observations follows a joint Gaussian distribution with kernel κ. A kernel defines the assumed correlation between the fitness of two values based on their proximity in search space. The kernel can be characterised further by a set of hyperparameters θ that control the nature and flexibility of the chosen kernel function. We denote $\kappa(\mathbf{x}', \mathbf{x}'', \theta)$ as the covariance between two vectors $x', x'' \in X$. Further, we define $\boldsymbol{\kappa}(\mathbf{x}, X, \theta)$ as the vector of covariances $(\kappa(x, X_i, \theta)), i \in \{1, \ldots, M\}$. Finally, $K \in \mathbb{R}^{M \times M}$ is the resulting covariance matrix consisting of the covariance vectors $\boldsymbol{\kappa}(X_i, X, \theta), i \in \{1, \ldots, M\}$.

We know that for any finite set of random variables exists a joint Gaussian distribution [108]. With the kernel chosen, we can now formulate the predictive density of a Kriging model for a point in search space x as:

$$P(\hat{f}(\mathbf{x}) | \mathbf{x}, D, \theta) = \mathcal{N}(\mu(\mathbf{x}), \sigma(\mathbf{x})), \tag{2.6}$$

where the predicted mean and the variance are given by

$$\mu(\mathbf{x}) = \boldsymbol{\kappa}(\mathbf{x}, X, \theta) K^{-1} \mathbf{f} \tag{2.7}$$

$$\sigma(\mathbf{x}) = \kappa(\mathbf{x}, \mathbf{x}, \theta) - \boldsymbol{\kappa}(\mathbf{x}, X, \theta)^\top K^{-1} \boldsymbol{\kappa}(X, \mathbf{x}, \theta). \tag{2.8}$$

Technically, any function $\kappa(\mathbf{x}', \mathbf{x}'', \theta)$ can be interpreted as a kernel, as long as the derived covariance matrix K is positive semi-definite [108]. The Kriging implementation used in this thesis[4] uses a composite covariance function as a default, in this case the sum of two different Covariance functions, namely CovSEiso and CovNoise. The former is a squared exponential covariance function with isotropic distance measure, with the characteristic length scale $\Lambda = \mathrm{diag}(l^2, \ldots, l^2)$ and signal variance α:

$$\kappa(x, y) := \alpha^2 \exp\left(-\frac{1}{2}(x-y)^T \Lambda^{-1} (x-y)\right)$$

[4]https://github.com/mblum/libgp

The latter is the independent covariance function, representing simple white noise controllable by parameter σ^2. We use the default values for the parameters as suggested in the documentation[5], i.e. $l^2 = 1$, $\alpha = 1$, and $\sigma^2 = e^{-4}$.

In order to train a Kriging model, the hyperparameters θ of the chosen kernel need to be determined. This is done in accordance with the initial design D and usually with maximum likelihood estimation. The maximum likelihood estimate for θ is thus the θ' that maximises the following equation:

$$\log P(D|\theta) = -\frac{1}{2}\log|K| - \frac{1}{2}\mathbf{f}^{\intercal}K^{-1}\mathbf{f} - \frac{M}{2}\log(2\pi). \tag{2.9}$$

Other methods to compute θ have been proposed in the literature, such as the Markov Chain Monte Carlo method [124], but the implementation used in this thesis uses maximum likelihood estimation, as it is more precise.

2.4 Game Optimisation

As explained in section 1.4, this thesis tackles handling uncertainty in surrogate optimisation applied to games optimisation specifically. For the purposes of this thesis we define game optimisation as follows:

Definition 2.1. Game Optimisation. A game optimisation problem can be formalised as a function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$, where the search space modifies some configurations of a game. The function f can be computed directly from the input or based on AI playthroughs of the game.

In this thesis, we choose to focus on a subset of game optimisation problems in order to allow for a clear analysis. The two problems we focus on have been published previously ([148], [150]) along with their respective results. In the following we describe just the problems that form the basis of the function suites implemented for our benchmark described in 4.3. According to the framework proposed in [83], both problems can be classified as *level* generation methods with *embedded input*. Both problems are intended to also allow for an *interactive process* with input from *human-based computation*.

2.4.1 TopTrumps Deck Generation

This problem is based on the card game TopTrumps and the task of generating a deck for the game. In the following, we first describe the game. We then introduce a formalisation which we use to define fitness functions for a card deck.

[5]In the code, the parameters are computed as $\hat{l}^2 = 0, \hat{\alpha} = 0, \hat{\sigma}^2 = -2$, respectively, as they are processed as $l^2 = \exp(\hat{l}^2)$, $\alpha = \exp(2\hat{\alpha})$ and $\sigma^2 = \exp(2\hat{\sigma}^2)$.

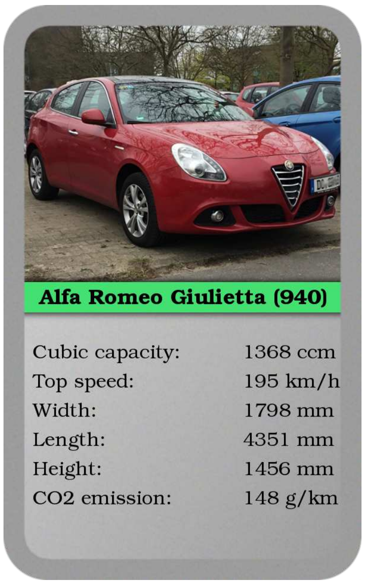


Figure 2.5: Example card from a car-themed TopTrumps deck with 6 categories

2.4.1.1 Game Description verbatim from [148]

TopTrumps is a themed card game originally published in the 1970s and relaunched in 1999. Popular themes include cars, motorcycles, and aircrafts. Each card in the deck corresponds to a specific member of the theme (such as a specific car model in a car-themed deck) and displays several of its characteristics, such as cubic capacity, top speed, or width. An example can be found in Fig. 2.5.

At the start of a game, the deck is shuffled and distributed evenly among players. The starting player chooses a characteristic whose value is then compared to the corresponding values on the cards of the remaining players. The player with the highest value receives all cards played in this round (called trick) and then continues the game by selecting a new attribute from their next card. The game usually ends when at least one player has lost all their cards. However, for the purpose of this benchmark, we end the game after all cards have been played once in order to avoid possible issues of non-ending games.

2.4.1.2 Formalisation verbatim from [148]

For the remainder of this thesis, we denote the number of cards in a deck as K and the number of characteristics (categories) displayed on a card L. Two representations are used for a deck, a vector $x \in \mathbb{R}^{KL}$ for the evolutionary algorithm and a $K \times L$ matrix V for easier comprehensibility.

The value of the k-th card in the l-th category is $v_{k,l}$ with $k \in \{1,\dots,K\}, l \in \{1,\dots,L\}$. The values on the k-th card in a deck are $v_{k,\cdot} = (v_{k,1},\dots,v_{k,L})$. A partial order for the cards can be expressed with $v_{k_1,\cdot} \prec v_{k_2,\cdot}$ meaning that card $v_{k_2,\cdot}$ beats $v_{k_1,\cdot}$ in all categories (dominant cards have larger values, since higher values win according to the game rules).

We only consider decks that fulfil two basic requirements we deem existential for entertaining gameplay:

- all cards in the deck are unique: $\nexists (k_1, k_2) \in \{1, \ldots, K\}^2$, $k_1 \neq k_2$ with $v_{k_1,\cdot} = v_{k_2,\cdot}$
- there is no strictly dominant card in the deck: $\nexists k_1 \in \{1, \ldots, K\}$ with $v_{k_2,\cdot} \prec v_{k_1,\cdot} \forall k_2 \in \{1, \ldots, K\}$

We consider two agents p_4, p_0 with different knowledge about the played deck in order to investigate how much of the game is based on skill vs. luck:

- p_4 knows the exact values of all cards in the deck
- p_0 only knows the valid value range for all values $v_{k,l}$

Both agents are able to perfectly remember which cards have been played already. Player p_4 is expected to perform better than p_0 on average on a balanced deck. In order to reduce the number of simulations needed to verify this, only games of a player p_4 against p_0 will be considered here.

In our simulation, both agents compute the probability to win with each category on a given card with consideration of their respective knowledge about the deck as well as the cards already played. p_0 therefore has to assume a uniform distribution and will only take the values of their current card into account. p_4, in contrast, is able to model the probability more precisely by accounting for the number of cards with a higher value in each category and still in play.

Let R_G be the number of simulation runs. The number of tricks that p_4 received at the end of the r-th game ($r \in \{1, \ldots, R_G\}$) with deck V will be called $t_4^{(r,V)}$ henceforth, and thus iff $t_4^{(r,V)} > \frac{K}{2}$, p_4 won the game, iff $t_4^{(r,V)} = \frac{K}{2}$ the game was a draw, and else, p_4 lost. $t_c^{(r,V)}$ is the number of times the player choosing the category did not win the trick in round r of the game with deck V, i.e. the number of times the player announcing the categories changed.

2.4.1.3 Fitness Functions

with small modifications from [148]

Without loss of generality, all problems are transformed into minimisation problems.

- Single-objective optimisation according to the dominance-related (D) measure proposed in [21] which describes the distance of the cards in a deck V to the Pareto front of the deck, where categories are interpreted as objectives:

$$f_D(V) : \mathbb{R}^{KL} \rightarrow [-K, 0] \in \mathbb{R}$$
$$f_D(V) = -\frac{1}{K} \sum_{k=1}^{K} \sum_{i=1}^{K} (1 - \mathbb{1}(v_k \prec v_i))$$

- Multi-objective optimisation with simulation-based measures developed with expert knowledge that are supposed to express the decks V's fairness (high p_4 win rate), excitement (high average # trick changes, low average trick difference), and resulting balance (B):

$$
\begin{aligned}
f_B(V) &: \mathbb{R}^{KL} \to \left([-1,0] \times \left[-\frac{K}{2},0\right] \times \left[0,\frac{K}{2}\right]\right) \in \mathbb{R}^3 \\
f_B(V) &= \left(-\frac{1}{R_G}\sum_{r=1}^{R_G} \mathbb{1}\left(t_4^{(r,V)} > \frac{K}{2}\right),\right. \\
&\quad \left. -\frac{1}{R_G}\sum_{r=1}^{R_G} t_c^{(r,V)}, \frac{1}{R_G}\sum_{r=1}^{R_G}\left|2t_4^{(r,V)} - \frac{K}{2}\right|\right).
\end{aligned}
$$

- Multi-objective optimisation with simulation-independent measures developed in the pre-experimental planning phase. With an appropriate mapping, the objectives can be used as a surrogate (S) for (the simulation-based) fitness f_B of different decks used for speed-up and interpretation purposes:

$$
\begin{aligned}
&f_S(V) : \mathbb{R}^{KL} \to \mathbb{R}^2 \\
&f_S(V) = (-hv(V), -sd(\{avg(v_{\cdot,l}) | l \in \{1,\ldots,L\}\})),
\end{aligned}
$$

  with the dominated hypervolume hv of a deck V, sd the empirical standard deviation and avg the average.

2.4.2 Mario Level Generation

This problem is based on a procedural level generation method proposed in [150]. The method is generally applicable, but is applied here to a platformer heavily based on Super Mario Bros.. In the following, we first describe the concept of latent variable evolution, which is central to the publication. We then explain the approach used in [150] in greater detail. Following that, we describe the level representation and the training process. Afterwards, we introduce the fitness functions used to evaluate the generated levels.

2.4.2.1 Generative Adversarial Networks verbatim from [150]

Generative Adversarial Networks (GANs) were first introduced by Goodfellow et al. [44] in 2014. Their training process can be seen as a two-player adversarial game in which a generator G (faking samples decoded from a random noise vector) and a discriminator D (distinguishing real/fake samples and outputting 0 or 1) are trained at the same time by playing against each other. The discriminator D aims at minimizing the probability of misjudgement, while the generator G aims at maximizing that probability. Thus, the generator is trained to deceive the discriminator by generating samples that are good enough to be classified as genuine. Training ideally reaches a steady state where G reliably generates realistic examples and D is no more accurate than a coin flip.

GANs quickly became popular in some sub-fields of computer vision, such as image generation. However, training GANs is not trivial and often results in unstable models. Many extensions have been proposed, such as Deep Convolutional Generative Adversarial Networks (DCGANs) [107], a class of Convolutional Neural Networks (CNNs), Auto-Encoder Generative Adversarial Networks (AE-GANs) [89], and Plug and Play Generative Networks (PPGNs) [96]. A particularly interesting variation are

Wasserstein GANs (WGANs) [6, 48]. WGANs minimize the approximated Earth-Mover (EM) distance (also called Wasserstein metric), which is used to measure how different the trained model distribution and the real distribution are. WGANs have been demonstrated to achieve more stable training than standard GANs.

At the end of training, the discriminator D is discarded, and the generator G is used to produce new, novel outputs that capture the fundamental properties present in the training examples. The input to G is some fixed-length vector from a latent space (usually sampled from a block-uniform or isotropic Gaussian distribution). For a properly trained GAN, randomly sampling vectors from this space should produce outputs that would be mis-classified as examples of the target class with equal likelihood to the true examples. However, even if all GAN outputs are perceived as valid members of the target class, there could still be a wide range of meaningful variation within the class that a human designer would want to select between. A means of searching within the real-valued latent vector space of the GAN would allow a human to find members of the target class that satisfy certain requirements.

2.4.2.2 Latent Variable Evolution verbatim from [150]

The first *latent variable evolution* (LVE) approach was introduced by Bontrager et al. [13]. In their work the authors train a GAN on a set of real fingerprint images and then apply evolutionary search to find a latent vector that matches with as many subjects in the dataset as possible.

In another paper Bontrager et al. [14] present an interactive evolutionary system, in which users can evolve the latent vectors for a GAN trained on different classes of objects (e.g. faces or shoes). Because the GAN is trained on a specific target domain, it becomes a compact and robust genotype-to-phenotype mapping (i.e. most produced phenotypes do resemble valid domain artifacts) and users were able to guide evolution towards images that closely resembled given target images. Such target based evolution has been shown to be challenging with other indirect encodings [156].

2.4.2.3 Approach verbatim from [150]

The approach is divided into two main phases, visualised in Figure 2.6. First, a GAN is trained on existing Mario levels (Figure 2.7). The levels are encoded as multi-dimensional arrays as described in Section 2.4.2.4 and depicted in the yellow box. The generator (green) operates on a Gaussian noise vector (red) and is trained to output levels using the same representation. The discriminator is then employed to tell the existing and generated levels apart. Both the generator and discriminator are trained using an adversarial learning process as described in Section 2.4.2.1.

Once this process is completed, the generator network of the GAN, G, can be viewed as our learned genotype-to-phenotype mapping: Based on a latent vector (blue) of real numbers (of size 32 in the experiments in this paper), it produces a tile-level description of a Mario level. Instead of simply drawing independent random samples from the latent space, we put exploration under evolutionary control. In other words, we search through

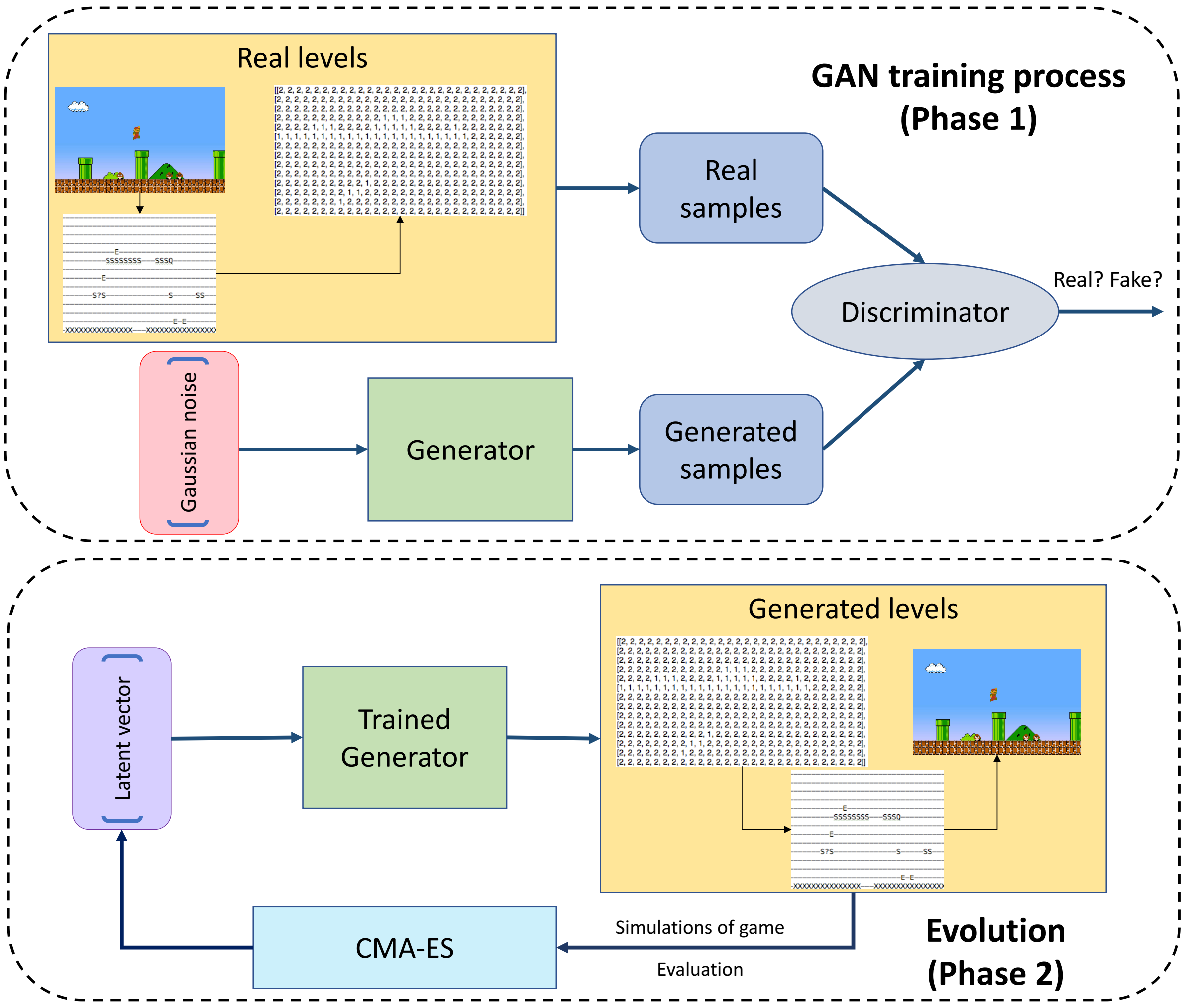


Figure 2.6: Overview of the GAN training process and the evolution of latent vectors. The approach is divided into two distinct phases. In Phase 1 a GAN is trained in an unsupervised way to generate Mario levels. In the second phase, we search for latent vectors that produce levels with specific properties.

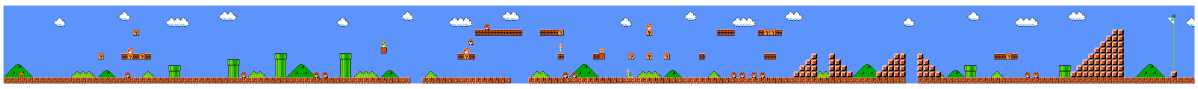


Figure 2.7: The Training Level. The training data is generated by sliding a 28 × 14 window over the level from left to right, one tile at a time.

Table 2.2: Tile types used in generated Mario levels. The symbol characters come from the VGLC encoding, and the numeric identity values are then mapped to the corresponding values employed by the Mario AI framework to produce the visualisation shown. The numeric identity values are expanded into one-hot vectors when input into the discriminator network during GAN training. Taken from [150]

Tile type	Symbol	Identity	Visualisation
Solid/Ground	X	0	⬜
Breakable	S	1	🟨
Empty (passable)	-	2	
Full question block	?	3	🟨
Empty question block	Q	4	🟫
Enemy	E	5	●
Top-left pipe	<	6	🟩
Top-right pipe	>	7	🟩
Left pipe	[	8	🟩
Right pipe	]	9	🟩

the space of latent vectors to produce levels with different desirable properties such as distributions of tiles, difficulty, etc..

2.4.2.4 Level Representation

As Mario levels are tile-based, the most straightforward way to represent them is as a matrix where each cell encodes a different tile. In the training levels taken from the Video Game Level Corpus (VGLC) [131], the tiles are encoded as ASCII symbols. These are then translated to an integer value and further to a one-hot encoding for GAN training. The integer value can also be read into the MarioAI framework to achieve a playable level and visualisation. The encoding used is specified in table 2.2.

2.4.2.5 GAN Training

with small modifications from [150]

Our Deep Convolutional GAN (DCGAN) is adapted from the model in [6] and trained with the WGAN algorithm. The network architecture is shown in Figure 2.8. Following the original DCGAN architecture, the network uses strided convolutions in the discriminator and fractional-strided convolutions in the generator. Additionally, we employ *batchnorm* in the generator and discriminator after each layer. In contrast to the original architecture in [6], we use ReLU (Rectified Linear Units) activation functions for all layers in the generator, even for the output (instead of Tanh), which we found gave better results. Following [6], the discriminator uses LeakyReLU activation in all layers.

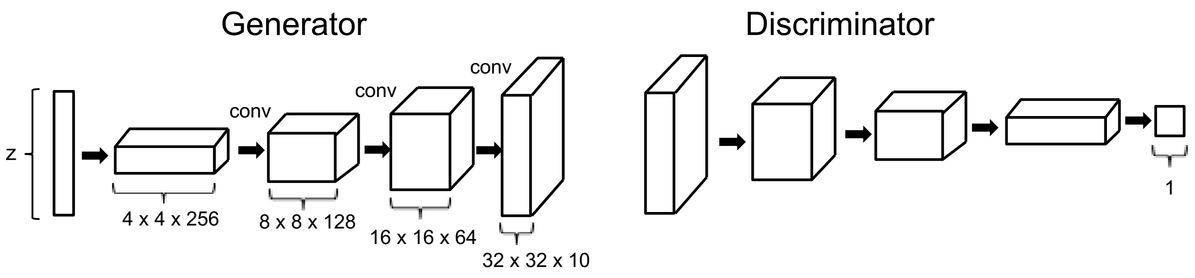


Figure 2.8: The Mario DCGAN architecture.

When training the GAN, each integer tile was expanded to a one-hot vector. Therefore the training inputs for the discriminator are 10 channels (one-hot across 10 possible tile types) of size 32×32 (the DCGAN implementation we used required the input size to be a multiple of 16 so the levels were padded). For example, in the first channel, the location of ground titles are marked with a 1.0, while all other locations are set to 0.0. The size of the latent vector input to the generator has a length of 32.

Once training of the GAN is completed the generator represents our learned genotype-to-phenotype mapping. When running evolution, the final $10 \times 32 \times 32$ dimensional output of this generator is cropped to $10 \times 28 \times 14$ and each output vector for a tile is converted to an integer using the argmax operator, resulting in a level that can be decoded by the Mario AI framework.

The GAN input files were created by processing a level file from the VGLC for the original Nintendo game *Super Mario Bros*, which is shown in Figure 2.7. Each level file is a plain text file where each line of the file corresponds to a row of tiles in the Mario level. Within a level all rows are of the same length, and each level is 14 tiles high. The GAN expected to always see a rectangular image of the same size, hence each input image was generated by sliding a 28 (wide) x 14 (high) window over the raw level from left to right, one tile at a time. The width of 28 tiles is equal to the width of the screen in Mario. In the input files each tile type is represented by a specific character, which was then mapped to a specific integer in the training images, as listed in Table 2.2. This procedure created a set of 173 training images.

2.4.2.6 Fitness Functions

In the following section, we describe the fitness functions used for optimising the Mario levels. We propose two types of fitness functions, the first solely based on the level tiles and the second one based on AI playthroughs.

Representation-based Fitness Functions with small modifications from [150] In the representation-based scenarios we directly optimize for a certain distribution of tiles. In more detail, we test (1) if the approach can generate levels with a certain number of ground titles, and (2) a combination of ground titles and number of enemies. We seek to minimize the following functions.

- Distance between produced fraction of ground tiles g and the targeted fraction t

$$F_{ground} = \sqrt{(g-t)^2}.$$

- Combination of ground coverage and maximising the total number n of enemies:

$$F = F_{ground} + 0.5 \cdot (20.0 - n).$$

This particular weighting was found through prior experimentation.

Simulation-based Fitness Functions with small modifications from [150] While being able to generate levels with exactly the desired number of ground tiles and enemies is one desirable feature of a level generator, a fitness function based entirely on the level representation has two inherent weaknesses:

- Levels with maximal fitness value might not be playable, especially if they are optimized for a small number of ground tiles and/or a large number of enemies.
- The number of ground tiles and enemies does not necessarily affect the playthrough of a human or AI agent, and may thus not result in levels with the desired difficulty. E.g., the enemies might fall into a hole before Mario can reach them or there might exist an alternative route that avoids difficult jumps.

These problems can be alleviated by using an evaluation that is based on playthrough data instead of just the level representation. This way, playability can be explicitly tested and characteristics of a playthrough can be observed directly.

To this end, we implemented agent-based testing using the Mario AI competition framework, as there are a variety of agents already available [138]. To evaluate a level, the latent vector in question is mapped to $[-1,1]^n$ with a sigmoid function and then sent to the generator model in order to obtain the corresponding level. The level is then imported into the Mario AI framework using the encoding detailed in Table 2.2, so that agent simulations can be run.

While there are a variety of properties that can be measured using agent-based testing, for this proof-of-concept we chose to specifically focus on the two weaknesses of representation-based fitness functions mentioned above. As before, our use case is to find playable levels with a scalable difficulty.

Given that the A* agent by Robin Baumgarten[6] (winner of the 2009 Mario AI competition) performs at a super-human level, we use its performance to determine the playability of a given level. For an approximation of experienced difficulty, we use the number of jump actions performed by the agent. The correlation between the number of

[6] https://www.youtube.com/watch?v=DlkMs4ZHHr8

jumps and difficulty is an assumption, however, jumping is the main mechanic in Mario and is required to overcome obstacles such as holes and enemies. The fitness function we seek to minimize is:

$$F_1 = \begin{cases} -p & \text{for } p < 1 \\ -p - \#jumps & \text{for } p = 1, \end{cases}$$

where p is the fraction of the level that was completed in terms of progress on the x-axis.

In order to investigate the controllability of the level generation process, we introduce the following fitness function

$$F_2 = \begin{cases} -p + 60 & \text{for } p < 1 \\ -p + \#jumps & \text{for } p = 1, \end{cases}$$

where p is the fraction of the level that was completed in terms of progress on the x-axis. The offset of 60 for the incomplete levels was chosen after preliminary experiments so that unbeatable levels where the agent is trapped and repeatedly jumps are discouraged. As a result, passable levels will always score a higher fitness than impassable ones.

Since the exact number of jumps is non-deterministic and can produce outliers if the agent gets stuck under an overhang, the actual fitness value in both cases is the average of 10 simulations.

2.4.3 StarCraft II Winner Prediction

This problem is based on the popular *real-time strategy* (RTS) game StarCraft II (Blizzard Entertainment, 2010). It is not a game optimisation problem per se, but closely related. The task is to predict the winner of a StarCraft II game in real-time based on statistics observable to only a single player. This predictor can then be used as a fitness function in game optimisation, for example to automatically adjust the difficulty of an AI opponent. The predictor could be used here as a way to assess how frustrated the human player might feel at any given point in the game. Of course, the predictor could also be used as a model to evaluate game states, as is e.g. required in some popular AI approaches such as Monte-Carlo Tree Search [17].

In the following, we first briefly describe the game StarCraft II. Following that, we introduce the pre-processing methods used on the acquired player data.

2.4.3.1 Game Description

verbatim from [147]

StarCraft Series StarCraft II[7] is a popular real-time strategy (RTS) game with a science-fiction theme released by Blizzard in 2010, which was followed up with further expansion packs in 2013, 2015, and 2016. It is the second game in the series, the first StarCraft game was published in 1998. StarCraft II was designed as an E-Sport [16] and has a massive following, regular tournaments (e.g. World Championship Series) and professional players.

[7]https://starcraft2.com

StarCraft II features three playable races (Terran, Protoss, Zerg) and several game modes (1v1, 2v2, 3v3, 4v4 and campaign). Each player spawns with a town-hall building and a small number of workers at a predetermined location on a map, their base. The players can construct additional buildings, which can be used to produce more workers and military units. The properties of the available buildings and units are determined by the race played. There are additional upgrade mechanisms available to buildings as well as units. Buildings, units, and upgrades require different amounts of minerals and vespene gas, the two resources in the game. Both can be gathered by worker units. The supply value, which may be increased by additional buildings, poses a limit to the number of units that can be built by a player.

The player that successfully destroys all their opponent's buildings has won the game. The game also ends if a player concedes or if a stalemate is detected by the game.

StarCraft as Research Environment The first game version, and specifically its expansion pack *StarCraft: Brood War*, have been used in research as a benchmark and competition framework[8] for AI agents since 2009 [143]. In 2017, DeepMind and Blizzard published the *StarCraft II Learning Environment (SC2LE)* [144]. The SC2LE provides an interface for AI agents to interact with a multi-platform version of StarCraft II and supports the analysis of previously recorded games.

Specifically, the SC2LE offers an interface through which a large set of game state observations[9] can be made available for every game tick in a replay or in real-time. The information that can be obtained includes raw data on features such as unit health and unit type in the form of heatmaps. At the same time, it also includes aggregated information that is usually displayed to game observers that can help to characterise a player's progress. Examples include the resource collection rate and the number of units destroyed represented as their value in resources. The SC2LE consists of multiple sub-projects, which include, among other things, a python wrapper library `pysc2`[10].

Even before releasing SC2LE, Blizzard has been allowing players to save their own StarCraft II games to a file using the `.S2Replay` format. These replays can then be watched using the StarCraft II software and even analysed using the *S2 Protocol* published by Blizzard. `s2protocol`[11] is a Python library that provides a standalone tool to read information from `.S2Replay` files. The files contain repositories with different information. The `metadata` repository, for example, contains general information on the game and the players, such as the result, the selected races, the game map, and the duration of the game as well as technical details such as the StarCraft II build number.

2.4.3.2 Data Pre-Processing

A model for winner prediction can theoretically be trained on any type of obtained StarCraft II data, that is playthroughs from AI or human tournaments, or ladder games. In

[8] http://bwapi.github.io/

[9] https://github.com/deepmind/pysc2/blob/master/docs/environment.md

[10] https://github.com/deepmind/pysc2

[11] https://github.com/Blizzard/s2protocol

either case, these game should be pre-processed in order to remove any uncharacteristic examples, such as player disconnects. This is especially important in ladder games with participants from lower leagues.

We thus remove games

1. where at least one player performed 0 actions per minute,
2. that lasted 30 seconds or less,
3. where at least one player spent less than 50 minerals and already destroyed one of their own buildings (player is losing intentionally).

Chapter 3

Related Work

In the following section, we present related work for several topics addressed in this thesis. We start with a review of research on game optimisation in section 3.1 and highlight persisting issues with uncertainty handling in the state-of-the-art. This is our main motivation for proposing the application of concepts established in surrogate-assisted optimisation and noisy optimisation research. We describe work related to these fields in sections 3.2 and 3.3.

Finally, we also describe publications on benchmarking evolutionary algorithms in order to put the benchmark we propose in this thesis into context.

3.1 Numerical Game Optimisation

Game optimisation problems according to the definition given in section 2.4 can be mainly found in research on automatic game balancing and search-based procedural generation of different creative artefacts, such as levels. Both types of problems have two major characteristics in common:

- They require an automatic evaluation of the game or a specific part of its content.
- An optimisation algorithm is operating on the evaluation function in order to find an optimal game configuration or piece of content.

The evaluation function is a common bottleneck in game optimisation, as it is often based on *simulations*, i.e. AI players playing the game. This observation is corroborated by [137], a survey on search-based PCG. This issue is the main focus of this thesis, and we propose a surrogate-based algorithm with dynamic uncertainty handling as described in section 4.2. In the following, we provide an overview of how related publications have approached the issue of expensive evaluation functions in game optimisation. In order to focus the discussion and improve comparability, we will mainly highlight work on numerical game optimisation, i.e. where the optimisation algorithm was run on continuous real-world inputs. Furthermore, we will not discuss the question of the validity of the fitness functions chosen for game optimisation here, as it is not relevant for the topics in this thesis. However, we address this issue in section 6.2.1.1.

There are many publications that fit into the category of game optimisation in the context of platformers [58]. In [129], for example, metrics like the number of jumps and the number of enemies killed by an AI player are used to evaluate generated levels. The search algorithm used is a combination of Markov chains and Monte Carlo Tree

Search. However, the authors ran into issues of unmanageably large runtimes when using full simulations with a state-of-the-art player AI. To combat this problem, they replaced the full simulations by a lower fidelity function using a simplified AI. Similarly, as the approach proposed in [63] requires numerous simulations, the agents chosen are very simple and fast. In *Ludi*, a famous system that generates board game rules, the computational budget for the agents and the number of moves allowed are restricted in order to allow manageable computation times [18].

Approaches that speed up computation by simplifying the simulation itself have the benefit of allowing the designer to explicitly model (in greater detail) the behaviour that the evaluation function is based on. The experiments can therefore be very focused on selected aspects of the game. However, in most reasonably complex and long games, the designer is not acutely aware of all interactions between the various components of a game. Therefore, any decisions on how a game can be simplified or reduced to certain aspects will incur a bias which is not easily understood or expressed. As a consequence, it is difficult to reason about the optimality of solutions found on a simplified fitness function and its transferability to the complete game as intended.

A similar problem occurs in approaches that seek to avoid AI playthroughs completely to save computational resources. Numerous of these approaches can be found in literature. In fact, a majority of the publications surveyed for this thesis fall into this category.[1] For instance, a survey of fitness functions used for Mario and related platformers can be found in [130]. In [92], the fitness function used is based on the similarity to already existing levels. *Ludi* also includes a multitude of measures without simulation [18]. Especially when targeting more complex games such as real-time strategy games, for example, evaluation concepts that do not require simulation are very prevalent [81, 82, 103, 136].

Of course, without any simulations at all, these approaches rely all the more on the correctness of the assumptions the game designer makes when creating the evaluation functions. This issue can be observed in [148], for example. In this paper, we first used an optimisation algorithm to create optimal decks for a card game with a multi-objective evaluation function based on AI playthroughs. Through observations made on these playthroughs and considering our knowledge of the AIs we implemented, we then created bi-objective evaluation function that did not require simulations. We called this newly created function a *surrogate* for the original, simulation-based one. The solutions found on the surrogate function achieved very good results, even when evaluated with the original function. Additionally, they were achieved much faster without the need for simulation. However, while we found numerous solutions on the Pareto front (as visualised in figure 3.1), they occupy a very different region than the solutions found using the original function. While it is certainly beneficial to be able to explore different regions of the Pareto front in order to find the best solutions, it is also clear that the fitness landscapes of the original and surrogate functions differ. Even if this fact can be beneficial in some cases, such as in [148], the intention was to model the original function as closely as possible. We therefore have to conclude that biases were introduced

[1]For complete survey, see the appendix A

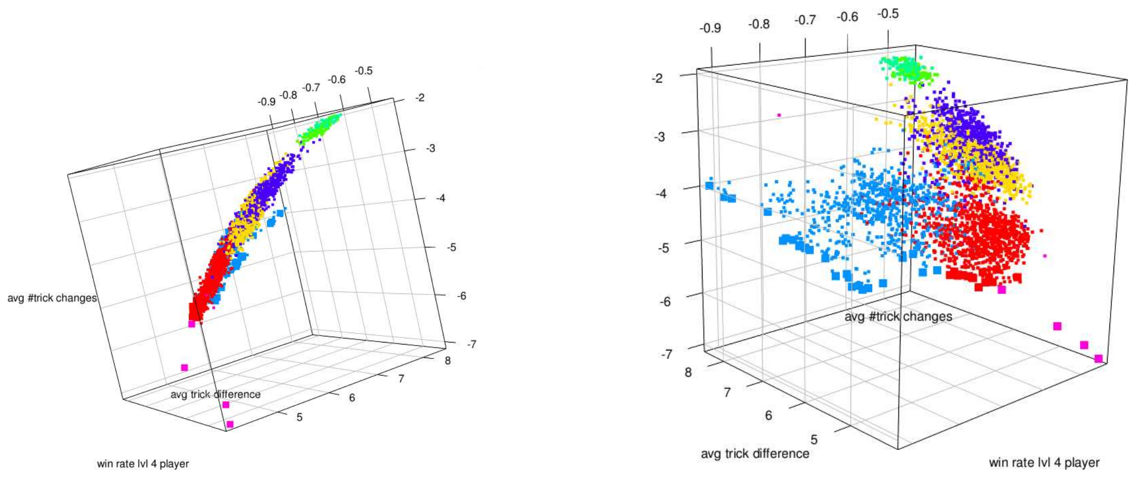


Figure 3.1: Final solutions from multiple optimisation runs of TopTrumps using various evaluation functions encoded by colour. Original function: 🟥 and 🟨, surrogate function: 🟦 and 🟪, alternative surrogate function: 🟩 and 🟩. Existing decks for comparison: 🟪. Larger squares depict solutions on shared Pareto front. Taken from [148].

into the evaluation which we were unaware of, even though we had complete access to all sources for a relatively simple game. An alternative surrogate function suggested in [21] was also tested, but did not achieve comparable results in terms of the performance of the discovered solutions as shown in figure 3.1.

To combat the issue of unintended and obscure biases introduced by surrogate evaluation functions, some authors choose a data-driven machine learning approach instead. For example, in [70], the authors used an automatic game evaluation function based on performance statistics of two opposing AIs to balance a simplified shooter game. The game is modelled after *Team Fortress 2* (Valve 2007) and thus supports multiple character classes, which makes the problem of balancing the classes against each other more complex and requires a multitude of evaluations. Since the simulations, even though simplified, would otherwise become forbiddingly expensive, a surrogate model is used in [70] to replace the fitness function. The model chosen is a *Convolutional Neural Network* (CNN) which is trained before optimisation starts and using a deep learning approach using the results of $2 \cdot 10^5$ simulated playthroughs. The input for the CNN are the modifiable parameters of the game to be balanced.

However, while the models are evaluated in terms of their accuracy, the obtained information is not considered during the optimisation at all. The model is pre-trained and not changed at all during the actual optimisation. As a result, any solutions found are entirely reliant on the model, which can produce large errors at times. In the field of surrogate-based optimisation, it is common practice to combine results from the original and surrogate functions in order to avoid the issue of finding optima on the surrogate function that do not align with the optima in the original one. See *Fitness Approximation* in section 2.1.2 and the following section 3.2 for more details. The approach we propose in this thesis uses these best practices from surrogate-assisted optimisation, and additionally has the ability to improve the model in interesting regions close to the optimum and identify low-fidelity predictions during runtime.

Several approaches have been published using pre-trained surrogates as the sole evaluation function, which of course induces the same issues as described above. In [117], for instance, a neural network trained on human feedback is used as a surrogate function to optimise the aesthetics of platform games. Integrating human feedback into an evaluation model is a very attractive approach to reduce the amount and influence of assumptions made by the designer. Despite the additional effort required for collecting enough data, this approach can be observed in several publications. In [54], for example, the feedback considered is the popularity of specific items within the game. In [154], the game is evaluated based on player retention statistics.

However, while reducing the amount of assumptions required for game evaluation is certainly a step in the right direction, pre-trained models still have the issue of questionable fidelity for newly discovered solutions. Additionally, they cause significant upfront costs. To address this issue, in surrogate-assisted optimisation algorithms (see section 3.2), the model is usually constructed during the runtime of the algorithm. The only paper taking this approach on game-related optimisation we found was [75], where a random mutation hill climber is used in conjunction with a surrogate model. This approach greatly resembles a simplified and potentially more efficient version of the popular iterative sampling methods discussed in more detail in section 3.2.1 and specifically GP-UCB [7, 8]. The surrogate model used is a *multi-armed bandit model* in conjunction with *Upper Confidence Bounds* (UCB1, see [17]) as the criterion to determine which point to sample in each iteration. The bandit model makes this approach very promising for noisy optimisation problems, but unfortunately is only designed for finite search spaces. It is thus not applicable to game optimisation as defined in this thesis.

Besides fully automatic approaches as discussed above, there are also integrated approaches that address the issue of model fidelity by including newly generated feedback from human testers into the algorithm during runtime. For example, we recently proposed an integrated process for game balancing that involves manual and automatic game balancing, as well as strategic reduction of the search space [104]. However, the tools and processes proposed in the paper still require a large amount of human involvement and decisions.

The proposed method falls into the umbrella of *mixed-initiative design*. Mixed-initiative approaches can be broadly clustered into two main concepts; *computer-aided design* and *interactive evolution* [80]. In Computer-aided design, the computer helps support the human creative process by evaluating the human-designed content automatically, e.g. in terms of playability. These approaches therefore use the same type of evaluation procedures as fully-automatic ones and thus carry the same issues discussed above. One famous example would be restricted play [64], where test cases are manually defined by a designer and then automatically executed.

In interactive evolution methods, however, the interaction is framed the other way around, i.e. one or multiple humans evaluate content generated by a computer system. These methods thus resemble another extreme of game evaluation methods, sacrificing runtime for the ability to avoid modelling errors. They therefore come with a different set of issues, mainly *user fatigue* caused by the continued cognitive effort required from a human for generating a large amount of feedback. A common approach to combat *cognitive overload* is through conscious design of the user interface and efficient and intuitive representations of potential solutions [80]. Furthermore, crowdsourcing is sometimes used to reduce the cognitive effort for a single user [113]. In some instances, the population size of the evolutionary algorithm is kept relatively small to ensure fast convergence [133].

In any case, however, interactive evolution methods require runtimes that are disproportionately larger than conventional optimisation methods. In order to alleviate this issue, the number of decisions delegated to the human decision maker can be reduced by automatically evaluating some of the individuals in a generation or all individuals in certain generations. To do this, [60] propose to use a fitness value that is inversely proportional to the distance of unevaluated individuals to evaluated ones. However, in this case, the validity of the model and the assumptions necessary to build it are not considered. This resembles previously proposed model management strategies popular for surrogate assisted evolutionary optimisation, which we discuss in section 3.2. Unfortunately, none of these methods involves checks on model validity either. This is one of the main motivations for the algorithm proposed in this thesis.

3.2 Surrogate-Assisted Evolutionary Optimisation

In evolutionary optimisation, algorithms generally follow the algorithmic skeleton depicted in figure 2.1 inspired by Darwinian theories on evolution, as described in section 2.1. Most evolutionary algorithms do not make any assumptions about the convexity and differentiability of the objective and constraint functions [24, 29]. This makes them a likely choice for real-world optimisation problems where, in many cases, the fitness landscape is not or only partially known. As a result of their typically exploratory approach, however, evolutionary algorithms tend to require a relatively large number of function evaluations until convergence or until a suitable solution is found.

This becomes an issue when evaluating a solution is computationally / economically expensive or otherwise time-consuming. This is a common issue in real-world applications and in game optimisation specifically, as discussed in the previous section 3.1. To alleviate this problem, surrogate-assisted evolutionary algorithms have been proposed. Broadly speaking, there are two types of popular approaches in literature:

- iterative sampling (e.g. [68, 73])
- evolution control (survey in [66])

In the following, we first briefly outline both approaches in general. Following that, we describe the specific publications and corresponding algorithms relevant to this thesis in greater detail. In theory, any kind of optimisation algorithm can be coupled with any kind of predictive model. However, we will only be covering evolutionary algorithms (see section 2.1) working in conjunction with Kriging models (see section 2.3) in this thesis.

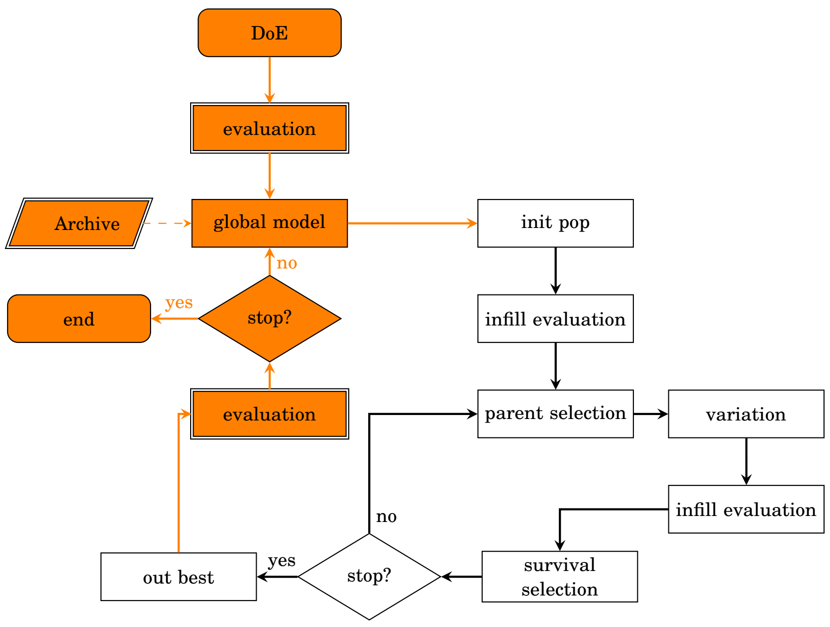


Figure 3.2: Iterative Sampling methods. The expensive objective function is optimised by iteratively improving a surrogate model guided by an infill criterion. The optimiser is used to select solutions for evaluation with the expensive objective function. The model is then updated accordingly and the next iteration starts. The process terminates after the budget of expensive function evaluations is exhausted. Evaluations indicated with double borders are added to an archive. Additions to the EA skeleton from figure 2.1 are marked in orange.

As the predicted values from the model act as a surrogate to evaluations of the objective, the model is also often called a surrogate model. Hence the name surrogate-assisted evolutionary optimisation.

Iterative Sampling Iterative sampling methods seek to improve a surrogate model throughout the runtime of the algorithm. This process is usually guided by a function (also called *infill criterion*) that expresses both the predicted accuracy of the model after the new sample as well as the estimated progress regarding the original expensive fitness function. If an evolutionary algorithm is used to optimise the infill criterion, iterative sampling methods fall under the umbrella of surrogate-assisted evolutionary optimisation. A visualisation of iterative sampling methods can be found in figure 3.2. Iterative sampling methods usually start from a sample using a space-filling *design of experiments* (DoE) method. For a clear distinction it is important to note that in iterative sampling approaches, the optimisation algorithm works on the model exclusively and does not automatically trigger evaluations of the true expensive fitness function during its execution.

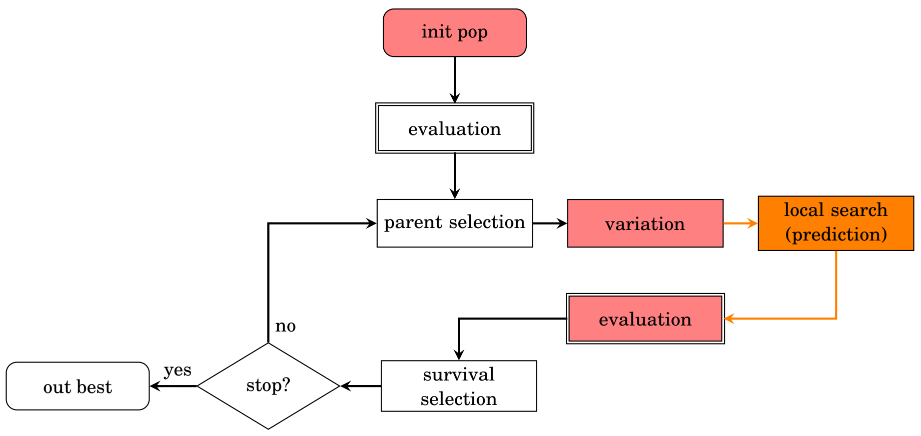


Figure 3.3: Evolution control methods. Steps where surrogate models can be helpful according to [66] are coloured in red. Additional steps are indicated in orange. Figure adapted with modifications from [66].

Evolution Control In evolution control methods, one or multiple steps in the algorithmic skeleton as depicted in figure 2.1 are supported by a surrogate model. For example, the random generation of offspring for a new generation might be biased using information from the model. Figure 3.3 depicts those steps where a surrogate model can be helpful according to the survey in [66]. The method of integration of surrogate model and evolutionary algorithm is often called *model management strategy* and can generally be classified as either individual-based, generation-based or population-based. The survey by Y. Jin [66] also gives an overview of popular model management strategies.

3.2.1 Efficient Global Optimisation of Expensive Black-Box Functions (EGO) [68]

Efficient Global Optimisation (EGO) is an iterative sampling framework (cf. figure 3.2) that was proposed by D. Jones, M. Schonlau and W. Welch in [68], which is popular in research (see [114] and references therein). Since the original publication, modifications have been suggested [27, 100], as well as adaptations to multi-objective problems, for example ParEGO [73]. We will mainly present EGO according to its original publication [68] in the following, but add notes on the most popular applications and modifications.

EGO starts with an initial sample of the fitness landscape, which it constructs a Kriging model from. This initial experimental design is supposed to be *space-filling*, which is why Latin hypercube designs [93] are often used. In [68], the authors suggest to use about $10d$ points for the initial sample, where d is the dimension of the search space.

After the model is fit to the data using maximum likelihood estimation, diagnostic tests are performed in order to ensure the fit of the model is satisfactory. These tests include multiple plots and a test whether the cross-validated standardised residuals are less than 3. If these tests fail, transformations to the dependent variable (such as log or inverse $-\frac{1}{y}$) are applied and used for the remainder of the algorithm, if the models can thus be improved. The model validation step is skipped in many of the more recent publications. However, recent investigations seem to suggest that the choice of model and its fitness do not have a significant effect on the performance of EGO [23]. Still, further experiments need to be conducted to validate these counter-intuitive observations.

Following initial model construction (and validation), the algorithm starts a loop where in each iteration, a new point is evaluated and added to the model. In [68], the next point to be sampled is the one that maximises the *expected improvement*, which is identified using a branch-and-bound algorithm. Expected improvement is the stochastic expected value of improvement over the current best solution found by the optimiser, computed using the uncertainty prediction of the surrogate model. It thus automatically introduces a compromise between improving the model (sampling in areas with higher uncertainty) and improving the optimisation solution (sampling in more promising areas).

The algorithm stops when the highest expected improvement value is small (less than 1%) relative to the function values. In many adaptations of EGO, an evolutionary algorithm is used to choose the next point to sample instead of the branch-and-bound algorithm. Additionally, other *infill criteria* for deciding which point to sample next have been proposed, such as lower confidence bounds [100], and probability of improvement (of the best solution found) [27].

The EGO algorithm has many strengths (see [23]), such as its flexibility due to the free choice of kernel function for Kriging [108] and the possibility to incorporate expert knowledge, for instance via Co-Kriging [40], or trend functions [4]. Furthermore, noisy data can also be considered in a statistically sound fashion [39]. Additionally, EGO is very well suited for settings where multiple optima have to be found in multi-modal functions [155]. However, there are also several issues with EGO (see [23]). Chief among them the are computational effort and number of evaluations required for training satisfactory non-local Kriging models [9], especially in case of high-dimensional search spaces [1]. Furthermore, the flexibility of the Kriging model also necessitates various decisions, which are difficult to make without domain expertise. Additionally, if a model in EGO is used without validity checks, the infill criterion used might mislead the search completely [124].

3.2.2 Single- and Multi-objective Evolutionary Optimization Assisted by Gaussian Random Field Metamodels (Pre-screening) [37]

Pre-screening is an approach where a local search is applied to the offspring before the most promising ones are evaluated (cf. figure 3.3). The steps added to the EA algorithmic skeleton are visualised in more detail in figure 3.4. Effectively, a bias is added to the offspring generation step based on knowledge gathered from previous function evaluations, similar to estimation of distribution algorithms, such as CMA-ES (see section 2.1.3.1). This approach has been analysed for both single- and multi-objective evolutionary optimisation in [37].

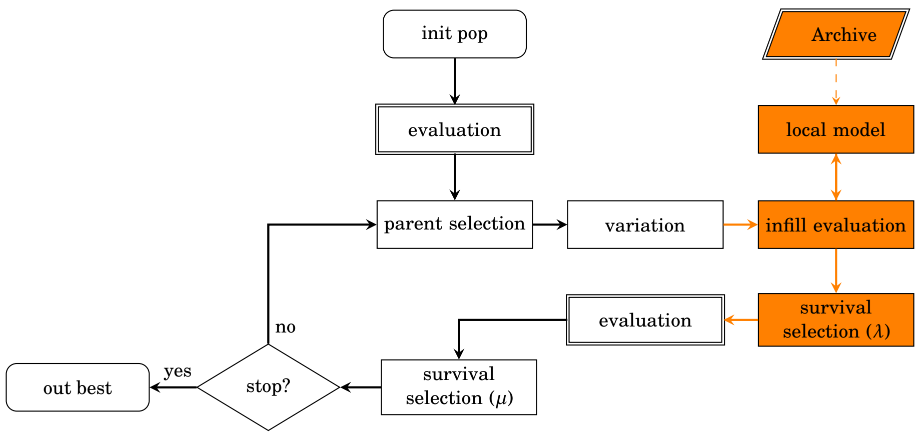


Figure 3.4: Pre-Screening. A local search is conducted on generated offspring in order to bias the search, i.e. (1) select λ individuals based on infill criterion, (2) select μ individuals based on fitness functions. Steps added to the EA skeleton are marked in orange. Evaluations indicated with double borders are added to an archive.

The algorithm can be applied to any evolutionary algorithm and type of surrogate model. Applying it only requires two changes to the EA:

- All fitness evaluations are recorded
- Only the most promising offspring are selected for fitness evaluation

In order to identify how promising the individuals are, several criteria popular in surrogate-assisted evolutionary optimisation are used, namely expected improvement [68], probability of improvement [142], lower confidence bounds [32, 140] and the mean predicted value. In [37], these criteria are also introduced for multi-objective problems, where the improvement is expressed in terms of some performance criterion for multi-objectives EAs. In the paper, hypervolume (see section 2.1.1.2) is used for that purpose.

The experiments in the paper were conducted using a $(\mu+\lambda)$-ES as an underlying algorithm with $\mu = 5, \lambda = 100$. From the λ individuals, at most $\upsilon = 20$ were selected for evaluation via pre-screening. Pre-screening as described in [37] uses local surrogate models with $2d$ samples, where d is the search space dimension. The samples considered for each model are the $2d$ closest ones w.r.t. the Euclidian distance in search space from the point of interest.

3.2.3 Differential Evolution for Multiobjective Optimization Based on Gaussian Process Models (GP-DEMO) [95]

The algorithm most similar to SAPEO (cf. section 4.2) is called GP-DEMO and was published in [95]. While the ideas were developed independently and the algorithms use different optimisers as a basis, some of the main concepts still share a remarkable similarity.

GP-DEMO extends a specific differential evolution algorithm called DEMO [109] for multi-objective problems. This algorithm is based on the differential evolution framework, but allows for the existence of incomparable individuals as well, as required for multi-objective optimisation (cf. section 2.1.1.2). For each parent and child combination, DEMO still adds the better individual to the population, now in terms of Pareto dominance. In case the individuals are incomparable, both are added to the population. At the end of an iteration, the population is truncated to the required size. This is done using a secondary selection criterion like in many other MOEAs (cf. section 2.1.1.2). DEMO uses the *crowding distance* as does the popular NSGA-II [30].

As described in section 2.1.2, a partial order can also be introduced to compare individuals in a noisy environment. This same principle can be applied to fitness predictions, if the surrogate model also contains an uncertainty estimate from which uncertainty intervals can be derived. Kriging models (see section 2.3) have this feature and are used in GP-DEMO to estimate the fitness of newly created individuals. It then follows the same selection process as introduced in DEMO, i.e. adds all incomparable individuals, and thus avoids unnecessary evaluations. However, this also means that non-dominated sorting and the computation of crowding distance are computed on estimates instead of the actual fitness values. To avoid misleading the search, GP-DEMO evaluates all individuals on the first front with the expensive fitness function.

In order to not slow down the algorithm by computing Kriging models from all exactly evaluated solutions (*full Kriging model*), GP-DEMO uses a sparse-approximation method called *Sparse Gaussian Processes using Pseudo-inputs* (SPGP) [123]. The *active set*, i.e. the m solutions used to compute the model from, are computed in each iteration of the main loop according to SPGP. In addition, GP-DEMO also restricts the solutions considered for the active set by only considering the k most recently evaluated search points. Assuming the search steadily moves towards the true Pareto front, this sliding window approach causes more samples to be chosen close to the approximation of the Pareto front in a given iteration. As a result, the Kriging model would be most accurate around the first front, which is also the area where the most critical selection decisions are made.

While the comparisons under uncertainty do consider estimates of the prediction errors, which increases their reliability, there are still two potential sources for incorrect decisions. The first is the fact that the comparisons are based on bounding boxes, whereas the predictions are expressed in the form of probability distributions with unbounded support. As a result, there is a chance, albeit small and controllable, of a true value outside the bounding box. The second, more critical source is the validity of the model and specifically the reliability of the uncertainty estimates. The failure to validate the Kriging model in GP-DEMO could thus lead to a large amount of incorrect decisions slowing down the algorithm, especially in high-dimensional problems where a large number of samples is required to build a reliable Kriging model.

In order to maintain consistency in the descriptions, we also provide a visualisation of GP-DEMO in context of the EA algorithmic skeleton in figure 3.5. To do this, we only visualise the main feature from GP-DEMO, i.e. the comparisons under uncertainty, as an abstract concept within an EA. The same concept was previously used in [85] to

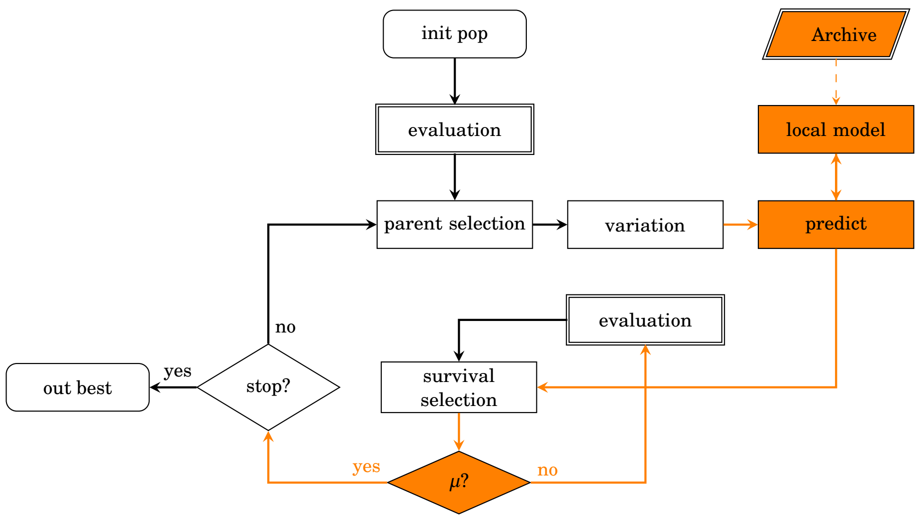


Figure 3.5: EA and comparisons under uncertainty. Generalised visualisation of comparisons under uncertainty when applied to EA algorithmic skeleton. Steps added to the EA skeleton are marked in orange. Evaluations indicated with double borders are added to an archive.

assist a combination of the multi-objective evolutionary algorithms NSGA-II [30] and SMS-EMOA [11].

3.3 Uncertainty Handling in Evolutionary Optimisation

Uncertainty handling by evolutionary algorithms for different types of uncertainties is surveyed in [67] and we give short overview in section 2.1.2. However, there are several applications, where several sources of uncertainty interact at the same time. Specifically for physical processes with a simulator and an emulator, history matching has been developed as a method to take into account different uncertainties [4] and create more trustworthy models. The authors depict a physical process with the flow chart showed in figure 3.6. A simulator in this case is the implementation of a theoretical model of a physical process. For example, when optimising the shape of a car to optimise flow conditions, a simulator might be based on models developed in research on computational fluid dynamics. However, evaluating this simulator at a specific search point might be expensive, which is where the emulator comes into play. The emulator is a data-driven model trained on data obtained from the simulator with some basic assumptions about the shape of the function.

The physical process y is observed through measurements z with a finite accuracy, thus introducing uncertainty (*observation uncertainty [OU]*). The simulator and emulator both produce a prediction $f(x)$ for y based on the input x. The emulator, however, additionally introduces *code uncertainty [CU]*, as the number of data points that it is

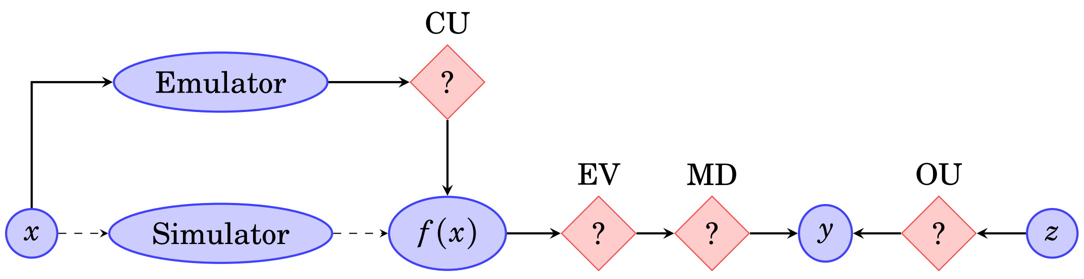


Figure 3.6: The physical process y is observed via z and described by the simulator output $f(x)$. The simulator is substituted by the emulator for computational efficiency. The question mark indicate the various sources of uncertainty present in the system. Plot from [4]

trained on. Furthermore, the stochasticity of the simulator introduces further uncertainty (*ensemble variability [EV]*). Finally, the *model discrepancy [MD]* is the uncertainty caused by inaccurate selection or tuning of the model. The uncertainties are all linked to the observation of the physical process z.

The paper goes ahead and suggests a workflow for estimating these different uncertainties and accounting for them during the optimisation process. This is done via an *implausibility* check that identifies areas in the search space where the model is insufficient. The authors apply history matching, i.e. they seek to minimise the implausible regions of the search space by iteratively reducing the models' input space. The result is then a more trustworthy simulator in the restricted space.

A similar approach as described above is also taken in [38], where its success is demonstrated on a realworld application optimising an airfoil shape.

3.4 Benchmarks for Expensive Continuous Optimisation

Despite a large interest in real-world problems from the research field of evolutionary optimisation (see for example Real-World Optimisation track at GECCO [2]), established benchmarks in the field are mostly artificial. The popular BBOB test suite (see section 2.2.4), for example, includes 24 popular test function with diverse characteristics, including e.g. sphere and linear functions, as well as Schwefel and Rosenbrock functions. The same functions are combined for a bi-objective function suite for the BBOB-BIOBJ function suite (see section 2.2.4). For multi-objective problems with larger dimensions, the DTLZ test suite is also popular [31].

A common trick to assess performance independent of the computational effort required is to record the number of function evaluations instead of execution time. *Pseudo-expensive* problems have been proposed as well, achieved by artificially delaying

[2] http://gecco-2018.sigevo.org/index.html/tiki-index.php?page=Program%20Tracks#id_RWA%20-%20Real%20World%20Applications)

the execution. However, either way, artificial problems often exhibit vastly different characteristics than actual real-world problems. For example, while single-objective multi-modal functions are considered regularly, this is less true for functions with plateaus. The existence of plateaus is a characteristic that we would expect many game optimisation problems exhibit (see section 3.1). Furthermore, recent visualisation approaches have determined that popular multi-objective benchmarks such as DENT, DTLZ2 and ED2 mostly contain problems with very simple fitness landscapes[45]. In contrast, if the multi-objective functions are constructed as a combination of multiple single-objective functions (as they are in BBOB-BIOBJ), the structures in the fitness landscapes are usually very complex[45]. It is however not clear, whether these functions are at all comparable to real-world functions, or whether they instead contain unnecessary complexity.

This results in a lack of appropriate benchmarks for algorithms specifically designed for expensive fitness function, such as surrogate-assisted evolutionary algorithms (see section 3.2). According to [28], these benchmarks are rare because real-world problems in relevant publications are mostly proprietary in nature. This claim is made despite the existence of the Black Box Optimization Competition (BBComp)[3], which includes expensive as well as bi-objective problems, but is set up as a competition rather than a benchmark. This means that the problems included in the competition are only seldom published or available for analysis, as the competition is intended to be on black box optimisation.

Recently, efforts have been made to tackle these issues. For example, in [28], three real-world problems involving *computational fluid dynamics* (CFD) are compiled into a benchmark for computationally expensive optimisation available on BitBucket[4]. Two of these problems are single-objective and one is bi-objective, and all rely on a CFD simulation for the computation of a fitness function. The problems are also scalable in search space dimension and offer multiple instances. However, the function suite lacks features that the established benchmarks have, such as the ability to estimate any-time performance as well as sophisticated post-processing. Therefore, it is likely that the widespread usage of this suite is going to be difficult to achieve.

Another recent effort was a workshop at PPSN 2018 [106] entitled *Investigating Optimization Problems from Machine Learning and Data Analysis*[5]. In this instance, the organisers suggest to use problems from the area of machine learning in order to compile a benchmark. The problems they propose include standard applications such as clustering and model training, as well as more specific ones such as one simulating buoy placement[6]. According to the organisers, they plan to extend the set of problems they proposed and eventually compile them into a benchmark. However, this benchmark is not publicly available at the time of writing.

Therefore, in this thesis, we propose to use game optimisation problems in order to form a benchmark and include it as function suites within the established COCO

[3]https://bbcomp.ini.rub.de/

[4]https://bitbucket.org/arahat/cfd-test-problem-suite/src

[5]https://sites.google.com/view/optml-ppsn18/home

[6]https://drive.google.com/file/d/1fc1sVwoLJ0LsQ5fzi4jo3rDJHQ6VGQ1h/view

benchmarking framework (see section 2.2). The resulting benchmark is described in section 4.3. As argued in section 1.1, game optimisation problems are real-world problems that are safe, reasonably complex and at the same time practicable, as they are relatively fast to compute. Additionally, benchmarks in general are also rare in the field of *computational intelligence in games*. This is often caused by licensing issues for games, as well as the effort required to set-up game-based problems. These issues are resolved for the popular AI and game-related competitions, however. There are a variety of popular competitions in this field[7], chief among them the *general video game AI* (GVGA) competition. Unfortunately, there exists no systematic analysis of the problems posed in these competitions and the comparison mechanics are difficult to interpret[8]. This makes them difficult to use as an independent benchmark. For this reason, we use the established benchmarking techniques implemented in the COCO framework as a baseline for the game-based benchmark.

[7] https://project.dke.maastrichtuniversity.nl/cig2018/competitions/

[8] https://ls11-www.cs.tu-dortmund.de/people/volz/gamesbench_events.html#cig18

CHAPTER 4

Approach

In this chapter, we describe our systematic approach towards developing and evaluating an algorithm to optimise games such as defined in 3.1. In this regard, we first develop a taxonomy (see section 4.1) that enables a more detailed understanding of the fitness landscapes and sources of uncertainty characteristics for game optimisation problems. Based on the taxonomy and a corresponding survey of game optimisation problems, we identify several interacting sources of uncertainty. We hypothesise that this uncertainty has a large effect on optimisation algorithms run on this type of problem. We support this claim with a case study in appendix B, which clearly demonstrates these effects.

Based on these findings, we propose SAPEO, an algorithm designed specifically for game optimisation due to the fact it takes into account and efficiently handles validated uncertainty information of fitness evaluations, even in the presence of systematic bias. A description and justification of SAPEO can be found in section 4.2.

We also develop a benchmark suitable to assess whether SAPEO or any other algorithm successfully handles the challenges characteristic to game optimisation. The benchmark is described in section 4.3. The related experiments are conducted through an additional experimental framework that allows for the easy comparison of various algorithm and includes specifically designed analysis features. The experimental framework is described in section 4.4.

4.1 Taxonomy of Automatic Game Evaluation

In the following, we first describe the taxonomy of automatic game evaluation we propose. In order to improve the readability of the following, we use the automatic evaluation of Mario levels as a usecase (cf. section 2.4.2). For example, we use the taxonomy to classify different approaches to evaluate Mario levels in order to provide a characterisation of each of the taxonomic categories. All explanation related to this usecase are printed in cursive.

Afterwards, we interpret our taxonomy in the context of existing publications that provide a characterisation of game evaluation methods to highlight similarities and differences in order to demonstrate the novelty of our approach.

4.1.1 Concept

The most straightforward way to evaluate a game or specific content is obviously conducting a survey and asking players to provide feedback. This is commonly done, for

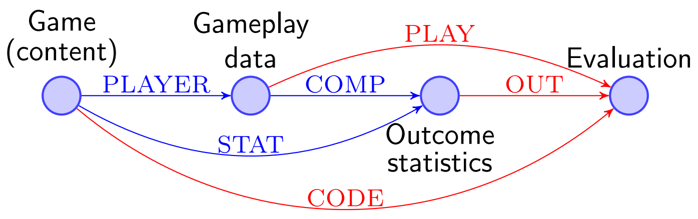


Figure 4.1: Visualisation of game evaluation AIs and their interactions. AIs are paths between data visualised as red and blue arrows. Paths that produce game evaluation are highlighted in red. Used acronyms are COMP: compute statistics, CODE: encoding, OUT: outcome statistics, PLAY: gameplay data and STAT: statistics.

example using think-aloud testing [42]. In this case, it is important to not only listen to verbal feedback, but also interpret social cues in order to obtain a reliable evaluation. *For example, the play testers might say the Mario level that they tested is very difficult as they were unable to complete it, but while playing they were talking to their neighbour or were distracted on their phone. In this case, the failure in terms of game progression would stem from a lack of engagement instead of a lack of skill, which a human hosting the play-test could observe and interpret.*

In contrast to many existing approaches, we focus on fully automatic evaluation, where the data collection and interpretation has to be executed by an AI agent. All approaches thus require a model defined and trained before runtime. This fact enables us to formalise all methods as statistical models that can be classified based on the type of data processed (for example tournament results or level encodings). As a result, we propose a taxonomy for game evaluation methods based on two dimensions. The first dimension is the type of **input** data that is processed in order to arrive at an evaluation of the game (content). The second characterising dimension is what type of **feedback** is used to train the model. In the following, we first describe each dimension separately, followed by a description of the complete taxonomy. The taxonomy can be applied to both the evaluation of complete games (i.e. rules) and specific game content (e.g. game levels) by simply changing the frame of reference and comparing the resulting evaluations.

4.1.1.1 Input Dimension

In figure 4.1, we illustrate different existing and possible approaches to automatic game evaluation by visualising the flow of information, i.e. what data is collected or generated. In the figure, the blue circles describe information or data. The arrows (both colours) visualise how this information is transformed and interpreted via different types of models / AI. It thus depicts different paths from a game prototype or specific game content (leftmost) to a qualitative or quantitative evaluation (rightmost). The red arrows all describe an evaluation method, while the blue arrows describe data generation methods.

The most direct path from a game to an evaluation is depicted as CODE in Fig 4.1, short for encoding. The distinguishing feature of this path is that the evaluation does not rely on simulating or analysing gameplay, but is solely informed by the ENCODING of the game (content). *For example, the difficulty of a Mario level could be expressed based on the weighted sum of the number of opponents and other obstacles in the level.* The aesthetics of game (content) can also often be directly evaluated based on their representation (i.e. visualisation) by human judges.

Instead of the direct approach, game evaluation can also be based on directly observable statistics of the final game state (OUTCOME STATISTICS) such as the final score as depicted with path OUT. *An estimate of Mario level difficulty could therefore be the average score of a large number of playthroughs by human players.* In this case, the game (content) is only represented by selected statistics that do not take into account the actual gameplay, i.e. how the game ran its course.

As depicted in the figure, these statistics can be obtained in several ways. A very popular approach is to generate playtraces using AI PLAYERS (path PLAYER) and then COMPUTING the appropriate statistics on the outcome, e.g. the mean score of multiple tries (path COMP). Of course, gameplay data can also be collected from human players. Alternatively, sometimes gameplay STATISTICS are available via APIs, but the gameplay data is not fully or easily accessible to the public[1], which is depicted as path STAT. A model following path STAT could also be learned and used to directly predict gameplay statistics without playtraces, e.g. in order to speed up the process of data collection (cf. [148]).

Finally, instead of just relying on outcome statistics, game evaluation can also be based directly on GAMEPLAY DATA (cf. path PLAY), i.e. data that changes in every game tick. Gameplay data can either be the complete observable state to a player (AI) (e.g. *all pixels on the screen*) at a given time, or statistics describing the same (e.g. *number of visible enemies*). These statistics are dependent on the game tick and thus describe a trend. However, if the order is not of importance, sometimes aggregated statistics, such as entropy, are used instead. If gameplay data from human players is available, a method that follows this path is affective computing [71]. *A Mario level could for example be evaluated based on the facial expressions of human players recorded during a playthrough. Another option could be to base an estimation of the difficulty (curve) directly on the frequency of inputs received from an AI or human player at a given point in the game.*

4.1.1.2 Feedback Dimension

With regard to feedback, the approaches can be distinguished based on whether they (1) employ an independently defined model (i.e. non-supervised learning (NONE) using expert knowledge or theories from related fields such as psychology and neurology), or (2) are trained from labelled data (using e.g. a machine learning approach). The second category can be further divided in terms of how the labelled data is obtained; some algorithms call for explicit feedback from users (EXP) and some base the data labels on

[1]for example RiotGames API: https://developer.riotgames.com/

Table 4.1: Taxonomy of game evaluation approaches along input and feedback dimension.

feedback / input	**none** NONE	**implicit** IMP	**explicit** EXP
encoding CODE	CODE-NONE	CODE-IMP	CODE-EXP
outcome statistics OUT	OUT-NONE	OUT-IMP	OUT-EXP
gameplay data PLAY	PLAY-NONE	PLAY-IMP	PLAY-EXP

observations of human behaviour instead (implicit feedback IMP). The key difference here is that in EXP, the targeted question is asked directly to the player, whereas in IMP, the answer is sought from unconscious responses. *If we were targeting the difficulty of a Mario level, for example, if we ask the player to play the game and rate its difficulty, it would be explicit feedback (*EXP*). Alternatively, we could base our measure of difficulty on the player's score instead, which is implicit feedback (*IMP*).* The resulting categories are:

- NONE: non-supervised model based on designer experience or scientific theories
- IMP: trained from data labelled by implicit feedback
- EXP: trained from data labelled by explicit feedback

4.1.1.3 Taxonomy Description

We can thus classify game (content) evaluation methods based on the taxonomy described in table 4.1. Naturally, the identified approaches can potentially be combined across both dimensions. Because some approaches complement each other, this has great potential for improving the evaluation model.

Additionally, different sources of information can also be combined as input for the model. This is especially suitable for deep learning approaches or when using algorithms that are capable of handling multiple models with varying fidelity.

4.1.2 Application to MarioAI Usecase

In the following, we will demonstrate the practicability of our taxonomy by providing different methods of evaluating the difficulty of MarioAI levels that fit into each of the resulting categories.

4.1.2.1 Encoding-based methods (CODE)

Since levels with more enemies and more obstacles to conquer are arguably more difficult than ones with less, a weighted sum of the number of enemies and gaps can potentially be a reasonable estimate of the level difficulty if the levels are of similar lengths. As this approach operates without data from playthroughs and is based on assumptions formulated by a designer, it would fit into category CODE-NONE.

Asking human players for feedback directly on a level representation would likely be most suitable when evaluating aesthetics. However, it is of course also possible to train a model based on the level encoding, but using labelled data obtained through different means. For example, to obtain a CODE-EXP model, one could ask players to play several levels and rank them in terms of difficulty. Afterwards, a correlation between a level representation (e.g. the number of enemies and obstacles as described above) and the obtained feedback can be computed and employed as a model. To keep with the previous example, this approach could be used in order to compute the weights for the sum.

Instead of labelling the data explicitly via a survey, one could record video footage of human players and collect data on physiological responses. By interpreting this information in terms of the level of engagement and frustration of the players, one could obtain the labels for the data. A model based on level encoding and this data would be categorised as CODE-IMP.

4.1.2.2 Outcome-based methods (OUT)

A possible outcome-based model could rely on the assumption that winrates are negatively correlated with level difficulty. Any unknown level would then be played by human and/or AI players and the winrate would be computed. This method is based purely on designer experience and statistics of the game outcome, and thus is considered OUT-NONE.

As humans would probably use a similar assumption if asked to directly interpret winrates, to train OUT-IMP and OUT-EXP models, one could again obtain labelled data using different means as described in the previous subsection. Afterwards, models can be computed using e.g. winrates as an input.

4.1.2.3 Gameplay-based methods (PLAY)

Playtraces contain a large amount of information that could be used as an input for a model that estimates level difficulty. One possibility would be to compute the distance of the optimal path through the level from the actual player trajectory. The assumption for a PLAY-NONE model would be that, the more the player diverges from optimal gameplay, the more difficult the level. As described in the previous paragraphs, labelled data can be obtained from human players in order to train a model with supervised learning methods on this specific measure for models categorised as PLAY-IMP and PLAY-EXP.

4.1.3 Context

In several survey papers and books on procedural content generation (PCG) and game research in general, such as [116], some categorisation of game evaluation methods is applied. In the following, we motivate our taxonomy by relating it to existing characterisations of game (content) evaluation and demonstrate its novelty, as well as its suitability to find strengths and weaknesses in the different modelling approaches.

In [158], the authors describe a general framework of experience-driven procedural content generation (EDPCG), which consequently also includes automatic evaluation of game content. The framework divides the general PCG process into four components: *Player Experience Model*, *Content Quality*, *Content Representation*, and *Content Generator*. It provides categorisations for each of these components. We discuss which categories in our taxonomy have counterparts in [158] in the following.

The categories for the component *Content Quality* in [158] correspond most to our first taxonomy dimension: input. *Direct* evaluation functions are the counterpart of our encoding-based category CODE. In [158], the methods are further distinguished between *theory-driven* (i.e. CODE-NONE) and *data-driven* (i.e. CODE-IMP and CODE-EXP). The category *simulation-based* encompasses all approaches that require data obtained from playthroughs, while we distinguish between methods that use the final game state OUT and approaches that include additional gameplay information PLAY. In [158], the *simulation-based* category is instead split based on whether the player AI (PLAYER) dynamically adapts to the game. The third category of the *Content Quality* component in [158] is *interactive*. This is not included in our taxonomy as it is only semi-automatic, but all online-learning algorithms fall into this category (*mixed-initiative*).

While this categorisation seems similar at first glance, the distinction in the EDPCG framework is actually not made based on the input to the game evaluation model, but based on the content representation used for the content generator. This becomes obvious when considering the example provided in [158]. *In their example, Mario levels are evaluated based on the playing style expressed by features such as the number of enemies killed by stomping on them and the time spent moving left.* In our taxonomy, this would be considered PLAY as the evaluation would be based on data that describes playthroughs and requires a *simulation*. However, in [158], the example is categorised as *direct (data-driven)* because *the search space for the content generator is a representation of the level*. While this is certainly the most helpful distinction when looking to describe different PCG methods, it is not the most suitable categorisation when intended for a discussion on strengths and limitations. This is because it does not include the mapping from the content representation to the model input and thus ignores a potential source of otherwise explicable bias.

The *Player Experience Model* in [158] describes different approaches to modelling the experience of a player. They distinguish between *subjective*, *objective*, and *gameplay-based* models based on what type of data they are trained on. The category *subjective* corresponds here to the mapping from explicit feedback to an evaluation, whereas *objective* is the mapping between implicit feedback and an evaluation. *GamePlay-Based* methods, according to [158], model a link between gameplay and a player's experience. The proposed categories seem to roughly correspond to the distinctions we make regarding the feedback dimension (EXP, IMP, NONE). However, in their example the model is built using *explicit feedback and based on measures expressing the playing style* and is thus categorised as both *subjective* and *gameplay-based*. The *Player Experience Model* *component* thus also contains elements of both dimensions of our taxonomy.

However, it could not describe a model that takes just the encoding of a level as an input. This would require the combination of the *Player Experience Model* and *Content Quality* components, which the authors also provide. But as both of the components contain aspects describing input and feedback, the resulting taxonomy is rather complex and contains many hybrid cases. In combination with the fact that the representation mapping is not considered, it is difficult to interpret this taxonomy in terms of which modelling biases are introduced where. This is corroborated by the fact that no distinction is made regarding what type of data is collected from the simulations.

We thus conclude that the taxonomy in [158], while very similar, does not meet the requirements for our analysis. It was of course also intended to provide a thorough characterisation of EDPCG instead. These statements also hold true for the categorisations described in [116] chapter 2 and 10, which correspond roughly to the *content quality* and *player experience model* components, respectively. It is still interesting to note the considerable similarities between our proposed taxonomy and the EDPCG framework, even though we took a completely different more technical approach to identifying taxonomic categories.

Besides EDPCG, there are other related taxonomies, that, however, only focus on one dimension of our proposed taxonomy. The categorisation along the first dimension, input, aligns with the one that is described to characterise the 57 aesthetic measurements for board games in [18]. *Intrinsic* criteria are directly based on the rule set, i.e. the encoding, which corresponds to CODE in our model. The *playability* criteria in [18] are based on self-play outcomes, i.e. outcome statistics, corresponding to OUT. Finally, like PLAY, *quality* measurements in [18] take trends of the gameplay into consideration. A characterisation in terms of our second taxonomy dimension, feedback, is described in [116, ch. 12], where model-based approaches are called *top-down*, while approaches based on player feedback are *bottom-up*. By providing a more detailed distinction of the methods, we hope to be able to characterise the survey game (content) evaluation method in a more thorough fashion.

Gameplay evaluation measures were also discussed in a 2017 Dagstuhl workshop [151]. In the report, the measures are categorised based on what information and computation is required when implemented inside a logging framework. For example, measures on agent decisiveness would need an AI-specific implementation and likely information on previously considered decisions. The Dagstuhl taxonomy thus focuses more on the technical aspects of logging, not the type of information needed or assumed.

In their chapter on Game Data Mining [34], Drachen et al. also provide a characterisation of related methods, but focus more on data mining and corresponding technical aspects. Additionally, it provides an overview of methods that can be used to model the data. Thus, methods of retrieving labelled data as well as algorithms that can train based on it are categorised in the chapter. This is an entirely different focus than what we hope to provide here.

In [161, ch. 5], a taxonomy for player modelling is discussed which is based on the different possible types of input and output of the model. However, the output dimension does not align with our feedback dimension. Furthermore, the taxonomy does not consider encoding as an input as it is intended for player modelling, not content evaluation. In chapter 4 of the same book, different aspects of PCG are discussed, among them the content evaluation component from EDPCG. In addition, a model of four roles of PCG is defined. According to this model, we address autonomous PCG here, both experience-driven and experience-agnostic.

4.1.4 Sources of Uncertainty

Based on the described taxonomy, we can identify several different sources and types of uncertainties that all interact in a typical game optimisation problem (see sections 4.1.4.1 and 4.1.4.2). These also depend on the choice of evaluation model (see section 4.1.4.3).

4.1.4.1 Feedback Dimension

The process of obtaining appropriate feedback for game content can be relatively complex, and there are many caveats to consider, depending on what type of data is acquired.

Feedback Survey For example, the most common way to obtain feedback from human players, either explicitly or implicitly (EXP, IMP) is via a survey. While using this type of feedback reduces the amount of assumptions required to build an evaluation model, there are also several possible problems due to the nature of real-world experiments. For example, there could be external issues with the experiments, potentially causing loss of immersion for the participants. Additionally, human bias might affect the creation and answering of a survey questionnaire. For instance, the questions might not be neutral or a participant might be influenced by factors unrelated to a given question, e.g. by the entertainment value of a game. Furthermore, it is difficult to even mobilise enough survey participants to obtain enough data to train an evaluation model. Nevertheless, survey participants should be selected with care and considering the target audience in order to avoid introducing a sampling bias.

Interpretation of feedback data No matter how and what feedback is obtained, the collected data needs to be interpreted and translated to a fitness function. This can introduce additional errors and biases. While it is of course true that the risk of errors is reduced the more detailed and the more explicitly human players are asked (EXP), there is always the possibility of miscommunication. Additionally, when qualitative feedback is obtained, it is interpreted through the perspective of a designer, opening up this process to potential subjectivity and issues such as confirmation bias.

Obtaining implicit feedback (IMP) does alleviate these issues, as no human is consciously involved in either the interpretation or response process. IMP approaches, however, have to rely on a model that translates unconscious behaviour such as physical signals to some form of qualitative feedback for the game. These models are still heavily researched at the time of writing [41, 78, 132, 159], and it is not entirely clear how reliable they are. They therefore probably introduce an unobservable bias if applied exclusively. Additionally, the interpretation of physical signals in the context of games is complicated by the fact that survey participants react less expressively when confronted with a virtual reality [33].

Finally, if the evaluation model does not consider feedback data at all (NONE), the evaluation is of course entirely reliant on the accuracy of the modelling assumptions made by the designer. Designers might be able to define a fitness function that expresses their design goals, and it seems to be common practice in industry to consider some statistics such as win rates in design considerations. Still, the definition of these functions undoubtedly requires design experience and their results are usually only used in conjunction with playtests.

Non-determinism in Games As most games are non-deterministic, depending on a specific playthrough, the feedback might vary as well. Especially if no human players are involved (NONE), this issue is mostly addressed by aggregating quantitative feedback. In surveys (EXP, IMP), questions can be instead formulated in a way that does not (significantly) change with the playthrough.

Types of Errors In order to handle the uncertainties introduced in an optimisation problem, it is important to consider what type of errors might occur. Errors resulting from a survey or the interpretation of its result will probably be non-symmetric. The same is true for modelling errors as potentially introduced by approaches in category NONE. In contrast, most non-determinism in games will cause symmetric noise.

4.1.4.2 Input Dimension

Uncertainties can of course also result from the decisions made on the input dimension. Depending of what type of data is used for a game evaluation model, for instance, additional issues might occur.

Data Selection As in any data-driven modelling approach, the selection of input data is important. Omitting relevant data will not produce accurate models, whereas adding too much data will result in overfitting and uninterpretable results. Thus, especially if the evaluation model is only trained from outcome statistics (OUT) or measures based on the encoding (CODE), many intricacies that affect the gameplay and the resulting feedback might be completely missed and thus not modelled. The resulting evaluation model would thus not express the intended fitness function.

A related problem also occurs in methods that use playthrough data (PLAY). Here, all relevant data theoretically should be available, but usually raw data cannot be parsed as input. The data is therefore selected and aggregated, thus allowing for potential biases introduced by the conscious decisions and intentions of the designer.

It is furthermore important with all approaches that the input space is sampled adequately. Sparsely sampled regions can result in extrapolation issues for the model trained on the data. Bias in the choice of samples can affect the model as well.

Data Generation As the data used in evaluation models for game optimisation is generated from a process that is consciously designed as well, additional issues can occur in methods that use data from playthroughs (OUT, PLAY). One of the main issues in this regard are the AIs required to automatically obtain this data. Despite the continuous efforts in the field of player modelling [56, 94, 127], there is no reliable general approach to developing AIs that behaves human-like. In fact, there is not even an appropriate measure that expresses behavioural differences in players on a strategic level [146]. As a result, in most game optimisation problems, AIs are used to generate input data despite the fact that AIs might behave entirely differently than human players. The effect this has one the evaluation models is rarely investigated, but it might be more striking the more information is used (playthrough data PLAY vs. outcome statistics OUT).

In order to combat the issues described above, some approaches (e.g. *restricted play* [64]) choose to analyse specific usecases instead of the whole game. This reduces the complexity of the problem and the restricted setting might improve an AI's ability to imitate human behaviour. At the same time, the reduced complexity might not produce data that can be translated in order to evaluate the complete game. Additionally, the need to select usecases to analyse of course introduces an additional source of bias and important aspects might be missed entirely.

Types of Errors As explained in the previous section, it is important to consider the types of errors that need to be handled. Both data selection and data generation issues will likely result in non-symmetric error distributions, as these issues are modelling problems.

4.1.4.3 Evaluation Model

In addition to potential issues and uncertainties in the data obtained, the choice and implementation of the model to train can introduce problems. This is in addition to issues resulting from insufficient data, for example caused by the lack of data in either dimension (see above sections 4.1.4.1 and 4.1.4.2).

Model Choice Many machine learning models do make assumptions about the problem and data they are trained on. The Kriging model used in this thesis, for example, assumes a specific form of correlation between search space and objective space by the choice of kernel (see section 2.3). The assumptions of a specific model in question should therefore be carefully tested before using it as an evaluation model.

This is especially important for we are dealing with black-box problems, where it is often difficult to decide which assumptions are safe to make. Additionally, in game optimisation due to the many interacting mechanisms in a game, fitness landscapes are likely rarely continuous. This is because even when only making small changes to game parameters, the balance between different characters or mechanisms might flip, causing entirely different behaviour. This is a caveat that should be considered as most models do assume some form of continuity of the fitness function.

Types of Errors The errors discussed above are all modelling errors and most likely non-symmetric.

4.2 SAPEO

In the following, we describe the SAPEO (Surrogate-Assisted Partial order-based Evolutionary Optimisation) algorithm, which we suggest for game optimisation problems. As explained in section 3.1, game optimisation problems usually have an expensive black-box fitness function, which is why we suggest a surrogate-assisted evolutionary algorithm. The SAPEO algorithm [145, 149] is aimed at finding an optimal balance between the number of function evaluations and the uncertainty introduced by fitness estimates, which is then propagated throughout the runtime of the algorithm .

Additionally, SAPEO was developed considering the many non-symmetric uncertainties it would have to handle as identified in section 4.1.4. This achieved via uncertainty aware comparisons for solutions. We hope to address Data Generation bias (see section 4.1.4.2) specifically, as it is a very common issue that does affect automatic game evaluation significantly as demonstrated in appendix B. To tackle these issues, we propose to use *comparisons under uncertainty* that are able to take non-symmetric error estimates into account as well. Furthermore, in the analysis of sources of uncertainty, it also becomes apparent that appropriate model validation is critical, as common modelling assumptions might fail in the context of game optimisation.

Finally, SAPEO is essentially a framework, where any type of surrogate model with uncertainty estimate and evolutionary algorithm can be combined. This allows, for example, an easy adaptation from single- to multi-objective problems. This is important due to the fact that many game optimisation problems differ in terms of their search and objective spaces (e.g. mixed-integer vs. continuous). What is more, the game evaluation function might also be adapted based on feedback obtained during the development process of the game. By modifying the evolutionary algorithm and/or surrogate model, SAPEO can easily be adapted to changing circumstances.

We describe the SAPEO framework in detail in the following. We first discuss the most important underlying concepts, namely *comparisons under uncertainty* and *runtime model validation*. Following that, we present some deliberations on the probabilities of introducing errors into an evolutionary algorithm using those concepts. Finally, we present the SAPEO framework.

Please note that SAPEO as introduced in this thesis uses the same concepts as previous publications [145, 149], but is heavily modified. Previous versions used a self-adapting threshold for allowable uncertainty estimates. This was intended as a form of model validation, but proved to be ineffective and added complexity and parameters to the algorithm. Additionally, in contrast to the uncertainty threshold, the cross-validation approach used in this thesis is justifiable from a statistical perspective. This also removed the need to enforce the generation of a space-filling design of experiments as an initialisation of the algorithm. More details on the model validation approach are described in section 4.2.2.

Furthermore, SAPEO as introduced in the following is more modularised and can thus incorporate ideas from related algorithms more easily. For example, the creation of the surrogate model can be modified, so that instead of the k nearest neighbours, the k most recently evaluated samples are used. This thesis also focuses on the dominance relation based on confidence intervals exclusively (cf. GP-DEMO, section 3.2.3). In [145], alternatives to this relation were introduced, which we do not discuss in this thesis to streamline the experiments and maintain comparability to related work. However, these dominance relations could still be used within the framework as introduced in this thesis.

4.2.1 Comparisons under Uncertainty

Comparison under uncertainty is a concept that allows the comparison of solutions based on fitness predictions up to a controllable degree of certainty. When used for instance in context of an evolutionary algorithm, these comparisons can be used in order to save exact function evaluations. The comparisons rely on a dominance relation defined on confidence intervals computed for the predictions. This dominance relation and the resulting partial orders where first analysed in the context of evolutionary algorithms in [110]. Comparisons under uncertainty are used in GP-DEMO [95] (see also section 3.2.3) as well as SAPEO [145, 149]. For this thesis, we have extended the definitions to also allow arbitrary surrogate models (such as AI player simulations) with non-symmetric error predictions.

Let $f:\mathbb{R}^n \to \mathbb{R}$ be a single-objective fitness function. Now assume there is a surrogate model that predicts the fitness for each individual x_i as $\hat{f}(x_i) \in \mathbb{R}$ with a prediction uncertainty $\hat{\sigma}_i$ as

$$\hat{f}(x_i) = f(x_i) + e_i, \quad e_i \sim F(\hat{\sigma}_i, \ldots), \tag{4.1}$$

where F is an arbitrary probability distribution controlled by $\hat{\sigma}_i$ and potentially additional parameters. Assuming that the predicted uncertainty and error distribution is correct (Assumption **A1**), it follows that

$$\begin{aligned} &\mathbb{P}\Big(f(x_i) \in [\hat{f}(x_i) + l_i, \hat{f}(x_i) + u_i] \Big) = 1 - \alpha \quad \text{with} \\ &l_i = F^{-1}\Big(\frac{\alpha}{2}\,\Big|\,\hat{\sigma}_i, \ldots\Big),\ u_i = F^{-1}\Big(1 - \frac{\alpha}{2}\,\Big|\,\hat{\sigma}_i, \ldots\Big), \end{aligned} \tag{4.2}$$

since $P(e_i < l_i \vee e_i > u_i) \leq \frac{\alpha}{2} + \frac{\alpha}{2} = \alpha$.

In case the model in question is actually a Kriging model, the previous definitions can be further specified to

$$\hat{f}(x_i) = f(x_i) + e_i, \quad e_i \sim \mathcal{N}(0, \hat{\sigma}_i^2). \tag{4.3}$$

With assumption **A1**, and considering that $l_i = -u_i$ due to zero mean and symmetry of normal distribution, we obtain

$$\begin{aligned} &\mathbb{P}\Big(f(x_i) \in [\hat{f}(x_i) - u_i, \hat{f}(x_i) + u_i] \Big) = 1 - \alpha \quad \text{with} \\ &u_i = \hat{\sigma}_i z\Big(1 - \frac{\alpha}{2}\Big). \end{aligned} \tag{4.4}$$

Here, z denotes the quantile function of the standard normal distribution.

Based on the partial orders introduced for noisy optimisation in [110] (cf. section 2.1.2), we can then define the following dominance relation $\preceq_c$ to compare solutions under uncertainty (c for confidence intervals):

$$x_i \preceq_c x_j := \hat{f}(x_i) + u_i < \hat{f}(x_j) + l_j \tag{4.5}$$

and specified for Kriging models

$$x_i \preceq_c x_j := \hat{f}(x_i) + u_i < \hat{f}(x_j) - u_j. \tag{4.6}$$

The confidence intervals as described above can of course also be computed separately for multiple dimensions. The dominance relations can thus be extended to also apply to multi-objective optimisation. Let $f : \mathbb{R}^n \to \mathbb{R}^d$ be a multi-objective fitness function. Let further $f_k : \mathbb{R}^n \to \mathbb{R}, k \in \{1 \ldots d\}$ be the function that describes objective k. Now assume there is a separate surrogate model for each objective k that predicts the fitness for each individual x_i as $\hat{f}_k(x_i) \in \mathbb{R}$ with a prediction uncertainty $\hat{\sigma}_{k,i}$. As above, we can then define confidence intervals as

$$\begin{aligned} &\mathbb{P}\Big(f_k(x_i) \in [\hat{f}_k(x_i) + l_{k,i}, \hat{f}_k(x_i) + u_{k,i}] \Big) = 1 - \alpha \quad \text{with} \\ &l_{k,i} = F_k^{-1}\Big(\frac{\alpha}{2} \,\Big|\, \hat{\sigma}_{k,i}, \ldots\Big),\; u_{k,i} = F_k^{-1}\Big(1 - \frac{\alpha}{2} \,\Big|\, \hat{\sigma}_{k,i}, \ldots\Big). \end{aligned} \tag{4.7}$$

The corresponding dominance relation can then be defined as:

$$x_i \preceq_c x_j := \bigwedge_{k \in \{1 \ldots d\}} \hat{f}_k(x_i) + u_{k,i} < \hat{f}_k(x_j) + l_{k,j} \tag{4.8}$$

Again, if the surrogate models used are Kriging models, these definitions can be specified further. In the following, we also assume that the models are trained on the same set of points. In Kriging models, the error prediction only depends on the distances of the samples and the point to predict in search space as defined by the kernel (see section 2.3). The prediction uncertainty $\hat{\sigma}_{k,i}$ is thus equal for all k. We therefore only use $\hat{\sigma}_i = \hat{\sigma}_{k,i}, k \in \{1 \ldots d\}$ in the following. We can then specify

$$\begin{aligned} &\mathbb{P}\Big(f_k(x_i) \in [\hat{f}_k(x_i) - u_i, \hat{f}_k(x_i) + u_i] \Big) = 1 - \alpha \quad \text{with} \\ &u_i = \hat{\sigma}_i \, z\Big(1 - \frac{\alpha}{2}\Big) \end{aligned} \tag{4.9}$$

and

$$x_i \preceq_c x_j := \bigwedge_{k \in \{1 \ldots d\}} \hat{f}_k(x_i) + u_i < \hat{f}_k(x_j) - u_j \tag{4.10}$$

4.2.2 Runtime Model Validation

In state-of-the-art surrogate-assisted evolutionary algorithms as described in our section on related work, the models used are rarely validated (see section 3.2). However, in iterative sampling methods, the surrogate model is only used to guide the search for new samples. The model will thus improve over time, even if the mode would fail a validation check. Similarly, in most evolution control methods, the predicted function values are not actually propagated throughout the algorithm. As a result, an invalid model might slow down the optimisation process, but not effectively derail it.

This is different in case of GP-DEMO and SAPEO, where solutions are only evaluated if they are incomparable with dominance relation $\preceq_c$ as defined in section 4.2.1. Here, an invalid model would violate assumption **A1** and thus have a probability larger than α of resulting in incorrect orders and selections. Additionally, the risk is now not actually controllable, as this probability is only limited to 1.

We therefore introduce runtime model validation in SAPEO. We use *leave-one-out cross-validation* as suggested in the original publication of the EGO framework [68]. This method allows a validation of the model without adding new samples, thus making it applicable during the runtime of the algorithm without causing disruptions.

Cross-validation is regularly applied to any type of prediction models. Assume there is a sample X with k points evaluated using fitness function $f : \mathbb{R}^n \to \mathbb{R}$. The idea is to pick a previously evaluated sample $x_i \in X$ and compute a model $\hat{f}_{-i} : \mathbb{R}^n \to \mathbb{R}$ only with the remaining $k-1$ points. As x_i was evaluated previously, we are then able to compute the prediction error as $\|f(x_i) - \hat{f}_{-i}(x_i)\|$. Statistics such as the *mean squared error* can then be determined from the prediction errors for all $x \in X$, computed as above.

In SAPEO, the models provide an error estimate in addition to a prediction. Such a model can still be considered valid, even if the prediction error is high, provided that the corresponding predicted error is as well. Therefore, instead of computing the prediction error, model validation procedures in this case determine whether the true value is within a confidence interval predicted by the model. The confidence intervals we use are the same as defined in section 4.2.1. We thus effectively test whether assumption **A1** from that section holds.

For Kriging models, this process can again be simplified, as the predicted errors follow a known symmetric distribution. To do this, instead of prediction errors, the standard errors are computed. Assume as above a sample X with k points, but now let the model also predict the uncertainties $\hat{\sigma}_{-i} : \mathbb{R}^n \to \mathbb{R}$. The standard error for each sample $x_i \in X$ can then be computed as

$$\frac{\|f(x_i) - \hat{f}_{-i}(x_i)\|}{\hat{\sigma}_{-i}} \tag{4.11}$$

However, to make a binary decision whether a model is valid or not, we need to define a cut-off value regarding average standard errors of the model. A popular method is to use the value 3 [4], which is motivated by *Pukelsheim's* 3σ *rule* [105]. The reasoning here is, that with this setting, we only incorrectly denote a model as invalid in 5% of the cases.

We can thus give an estimate of model validity by computing the average standard errors using leave-one-out cross-validation and checking the result against the value 3. Thus, to determine the validity of a model to predict the value and corresponding uncertainty for a point $x' \in \mathbb{R}^n$, the k nearest neighbours for x' are determined. The resulting sample we call X, with each $x_i \in X$ being one of the neighbours. The model is accepted if it holds that

$$\frac{1}{k}\sum_{i=1}^{k}\frac{\|f(x_i)-\hat{f}_{-i}(x_i)\|}{\hat{\sigma}_{-i}} < 3.$$

4.2.3 Probability of Ranking and Selection Errors

Even when the model is validated and the comparisons consider uncertainty, errors can still be introduced when using these concepts. In the following, we analyse their theoretical limits in single-objective functions in more detail. In the analysis, we only consider Kriging models in order to simplify the computation. Kriging models are also the only type of surrogate model used in this thesis.

4.2.3.1 Ranking Errors

We define *ranking errors* as the pairwise differences between the correct order according to fitness values and the one induced by the proposed relations. We present the relations in the following along with the probability of ranking errors e.

We can compare two individuals in a population in terms of their fitness just based on the predicted intervals, provided we allow the occurrence of errors in the comparison. Given **A1**, these errors only occur if the actual function value $f(x_i)$ does not fall inside the defined interval. The probabilities of a ranking error $e_{i,j}$ occurring, i.e. two individuals x_i and x_j being sorted incorrectly, is then bounded by α (cf. [145]).

Let x_i, x_j be individuals in a population with predicted values $\hat{f}^{(i)} := \hat{f}(x_i)$, $\hat{f}^{(j)} := \hat{f}(x_j)$ and uncertainties $\hat{\sigma}_i, \hat{\sigma}_j$, respectively. Please note that the notation $\hat{f}^{(i)}$ is chosen in order to avoid confusion with f_i, which often denotes an objective function within a multi-objective problem. We now want to compute the probability that ranking solutions with relation $\preceq_c$ sorts given pairs of individuals correctly. More formally, we compute

$$P(f(x_i) \le f(x_j) \,|\, x_i \preceq_c x_j) =: P(x_i \preceq_f x_j \,|\, x_i \preceq_c x_j).$$

For the analysis, we assume in the following that the computed Kriging model is valid, i.e. assumption **A1** holds.

Now, let Y_i, Y_j be random variables distributed according to the prediction of a surrogate model with $Y_i \sim \mathcal{N}(\hat{f}^{(i)}, \hat{\sigma}_i^2)$ and $Y_j \sim \mathcal{N}(\hat{f}^{(j)}, \hat{\sigma}_j^2)$. We can now express the probability we are looking for in terms of the random variables independent of the condition as

$$P(f(x_i) \le f(x_j)) = P(Y_i \le Y_j) = P(Y_i - Y_j \le 0).$$

Generally, for two random variables A and B, it holds for the expected value that $\mathbb{E}(A+B) = \mathbb{E}(A) + \mathbb{E}(B)$ and $\mathrm{Var}(A+B) = \mathrm{Var}(A) + \mathrm{Var}(B) + 2\,\mathrm{Cov}(A,B)$. Since the Kriging model assumes that Y_i and Y_j follow a joint distribution, it follows that $Y_i - Y_j \sim \mathcal{N}(\hat{f}^{(i)} - \hat{f}^{(j)}, \hat{\sigma}_i^2 + \hat{\sigma}_j^2 - 2\hat{\sigma}_{ij})$. Given the definition of the normal distribution, we know that

$$P(Y_i \leq Y_j) = \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{-\hat{f}^{(i)} + \hat{f}^{(j)}}{\sqrt{2(\hat{\sigma}_i^2 + \hat{\sigma}_j^2 - 2\hat{\sigma}_{ij})}}\right)\right),$$

with erf being the Gauss error function defined as

$$\operatorname{erf}(x) = \frac{2}{\pi}\int_0^x e^{-t^2}\,dt.$$

In the following, we look at $P(Y_i \leq Y_j)$ given a ranking by the dominance relation $\leq_c$. First, however, we compute the same probability if the solutions were ranked just based on their predicted values. We denote this relation $\leq_p$ and use it as a baseline comparison.

Relation $\leq_p$ The relation $\leq_p$ ranks individuals according to their predicted value, i.e.

$$x_i \leq_p x_j \Leftrightarrow \hat{f}^{(i)} \leq \hat{f}^{(j)}.$$

Thus, $x_i \leq_p x_j \Rightarrow \hat{f}^{(i)} + d = \hat{f}^{(j)}$ with $d \geq 0$. It follows that:

$$\begin{aligned} P(Y_i \leq Y_j | x_i \leq_p x_j) &= \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{-\hat{f}^{(i)} + (\hat{f}^{(i)} + d)}{\sqrt{2(\hat{\sigma}_i^2 + \hat{\sigma}_j^2 - 2\hat{\sigma}_{ij})}}\right)\right) \\ &= \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{d}{\sqrt{2(\hat{\sigma}_i^2 + \hat{\sigma}_j^2 - 2\hat{\sigma}_{ij})}}\right)\right) \end{aligned}$$

Because $d \geq 0$ and the error function erf is monotonous increasing, we can estimate

$$\frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{d}{\sqrt{2(\hat{\sigma}_i^2 + \hat{\sigma}_j^2 - 2\hat{\sigma}_{ij})}}\right)\right) \geq \frac{1}{2}(1 + \operatorname{erf}(0)) = \frac{1}{2}.$$

Therefore:

$$P(Y_i \leq Y_j | x_i \leq_p x_j) \geq \frac{1}{2}$$

The relation $\leq_p$ therefore performs at least as well in terms of ranking errors as random sorting would. The performance improves the further apart the two individuals in question are. This relationship is visualised in figure 4.2. The figure also shows that, while clearly always better than random sorting, the values of $\hat{\sigma}_i, \hat{\sigma}_{ij}, \beta = \frac{\hat{\sigma}_j}{\hat{\sigma}_i}$ clearly influence the probability of ranking errors.

As can be observed in the first row of plots, the smaller the absolute values for $\hat{\sigma}_i$ and $\hat{\sigma}_j$, the less likely are ranking errors. This is of course expected, as this situation results in less overlap of the confidence intervals for the same d. This overlap is used for a second dominance relation described in the following section.

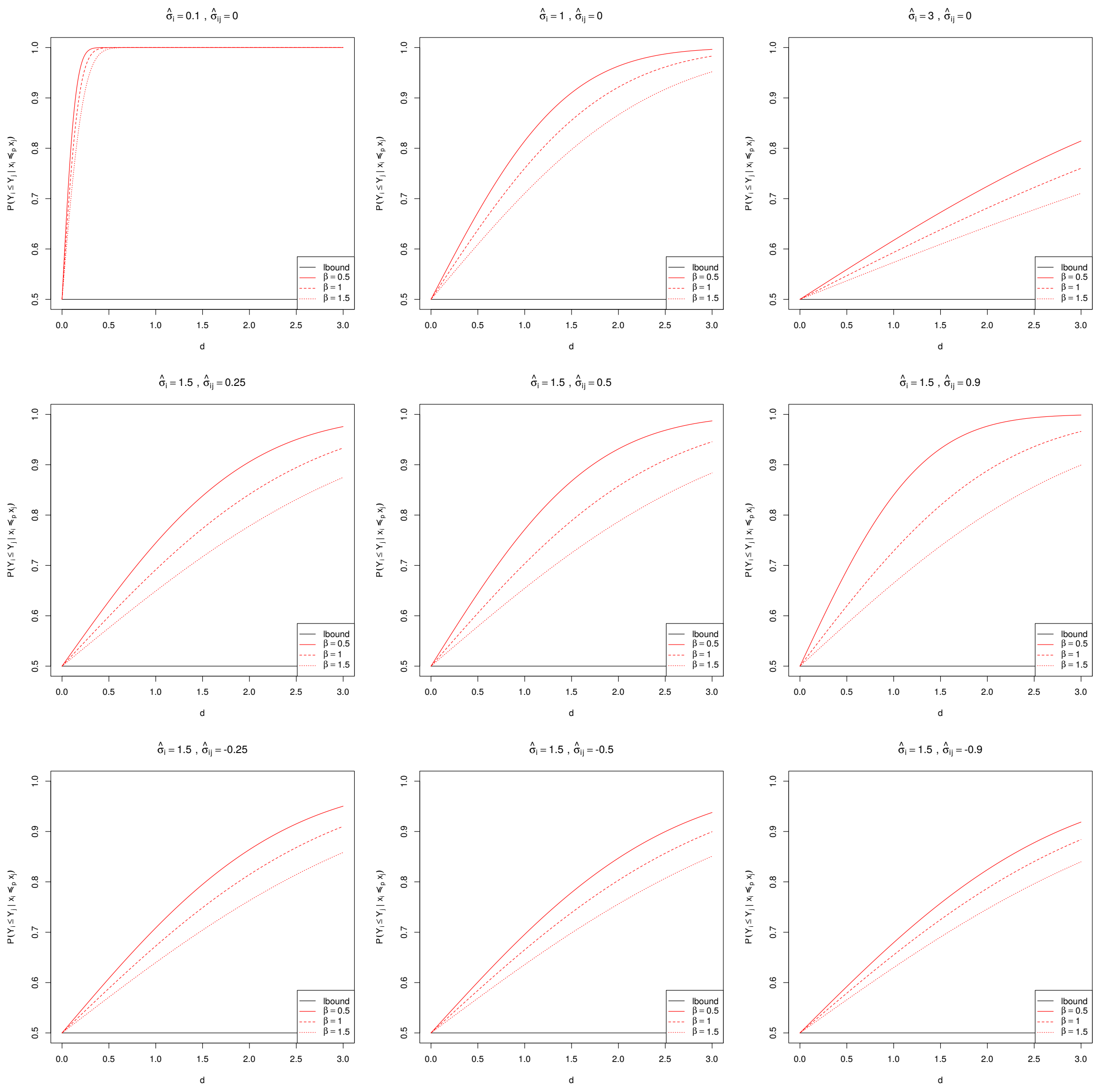


Figure 4.2: Lower bounds (solid black) and computed probabilities for correct ranking with dominance relation $\preceq_p$ plotted against d for different values of $\hat{\sigma}_i, \hat{\sigma}_{ij}, \beta = \frac{\hat{\sigma}_j}{\hat{\sigma}_i}$, where line type corresponds to β.

This same issue relating to the width and overlap of the confidence intervals can also be observed when comparing the different lines in each plots, signifying different values of $\beta = \frac{\hat{\sigma}_j}{\hat{\sigma}_i}$. Here, when one confidence interval is made smaller (e.g. $\beta = 0.5$ vs. $\beta = 1$), the probability of ranking errors decreases.

Comparing the second to the third row of plots, we observe generally lower probabilities of correct ranking with negative correlation. This is probably because with positive correlation, errors are mainly made in the same direction.

Relation $\preceq_c$ Dominance relation $\preceq_c$ only identifies individuals as dominant if the confidence intervals do not overlap. Thus, if $x_i \preceq_c x_j$, the upper confidence interval bound of Y_i has to be smaller than the lower bound for Y_j.

More formally:

$$x_i \preceq_c x_j \Rightarrow \hat{f}^{(i)} + \hat{\sigma}_i z_{(1-\frac{\alpha}{2})} \leq \hat{f}^{(j)} - \hat{\sigma}_j z_{(1-\frac{\alpha}{2})}.$$

with $\Phi^{-1}(1-\frac{\alpha}{2})$ denoted as $z_{(1-\frac{\alpha}{2})}$. Let $d \geq 0$, so that

$$\begin{aligned} \hat{f}^{(i)} + \hat{\sigma}_i z_{(1-\frac{\alpha}{2})} + d &= \hat{f}^{(j)} - \hat{\sigma}_j z_{(1-\frac{\alpha}{2})} \\ \Leftrightarrow \hat{f}^{(j)} &= \hat{f}^{(i)} + \hat{\sigma}_i z_{(1-\frac{\alpha}{2})} + \hat{\sigma}_j z_{(1-\frac{\alpha}{2})} + d \\ \Leftrightarrow \hat{f}^{(j)} &= \hat{f}^{(i)} + z_{(1-\frac{\alpha}{2})}(\hat{\sigma}_i + \hat{\sigma}_j + d') \end{aligned}$$

with $d' = \frac{d}{z_{(1-\frac{\alpha}{2})}}$. Thus:

$$\begin{aligned} P(Y_i \leq Y_j | x_i \preceq_c x_j) &= \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{-\hat{f}^{(i)} + (\hat{f}^{(i)} + z_{(1-\frac{\alpha}{2})}(\hat{\sigma}_i + \hat{\sigma}_j + d'))}{\sqrt{2(\hat{\sigma}_i^2 + \hat{\sigma}_j^2 - 2\hat{\sigma}_{ij})}}\right)\right) \\ &= \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{z_{(1-\frac{\alpha}{2})}(\hat{\sigma}_i + \hat{\sigma}_j + d')}{\sqrt{2(\hat{\sigma}_i^2 + \hat{\sigma}_j^2 - 2\hat{\sigma}_{ij})}}\right)\right) \\ &= \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{z_{(1-\frac{\alpha}{2})}}{\sqrt{2}}\sqrt{\frac{(\hat{\sigma}_i + \hat{\sigma}_j + d')^2}{\hat{\sigma}_i^2 + \hat{\sigma}_j^2 - 2\hat{\sigma}_{ij}}}\right)\right) \end{aligned} \tag{4.12}$$

In the following, we determine the limits of two parts of the equation in order to obtain the lower bound of its value. We can state that

$$z_{(1-\frac{\alpha}{2})} := \sqrt{2}\operatorname{erf}^{-1}\left(2\left(1-\frac{\alpha}{2}\right)-1\right) = \sqrt{2}\operatorname{erf}^{-1}(1-\alpha).$$

Further, it holds that

$$\hat{\sigma}_i^2 + \hat{\sigma}_j^2 - 2\hat{\sigma}_{ij} > 0 \tag{4.13}$$

since it the variance of $Y_i - Y_j$ and thus per definition larger than 0. As we show in the following, we determine that

$$\frac{(\hat{\sigma}_i + \hat{\sigma}_j + d')^2}{\hat{\sigma}_i^2 + \hat{\sigma}_j^2 - 2\hat{\sigma}_{ij}} \geq \frac{(\hat{\sigma}_i + \hat{\sigma}_j)^2}{\hat{\sigma}_i^2 + \hat{\sigma}_j^2 - 2\hat{\sigma}_{ij}} \geq 1. \qquad \text{(shown below)}$$

$$\begin{aligned} (\sigma_i + \sigma_j)^2 &\geq \sigma_i^2 + \sigma_j^2 - 2\sigma_{ij} && \text{(with 4.13)} \\ \Leftrightarrow \sigma_i^2 + 2\sigma_i\sigma_j + \sigma_j^2 &\geq \sigma_i^2 + \sigma_j^2 - 2\sigma_{ij} \\ \Leftrightarrow 2\sigma_i\sigma_j &\geq -2\sigma_{ij} \\ \Leftrightarrow 1 &\geq -\frac{\sigma_{ij}}{\sigma_i\sigma_j} \\ \Leftrightarrow -1 &\leq \frac{\sigma_{ij}}{\sigma_i\sigma_j} && =: \rho_{ij} \in [-1, 1] \end{aligned}$$

The last inequality holds per definition, as the correlation coefficient ρ_{ij} is always in the interval $[-1,1]$. With the obtained inequalities, we can then state that

$$\begin{aligned} P(Y_i \leq Y_j \,|\, x_i \preceq_c x_j) &= \frac{1}{2}\left(1+\operatorname{erf}\left(\frac{z_{(1-\frac{\alpha}{2})}}{\sqrt{2}}\sqrt{\frac{(\hat{\sigma}_i+\hat{\sigma}_j+d')^2}{\hat{\sigma}_i^2+\hat{\sigma}_j^2-2\hat{\sigma}_{ij}}}\right)\right) \\ &\geq \frac{1}{2}\left(1+\operatorname{erf}\left(\frac{\sqrt{2}\operatorname{erf}^{-1}(1-\alpha)}{\sqrt{2}}\sqrt{1}\right)\right) \\ &= \frac{1}{2}\left(1+\operatorname{erf}\left(\operatorname{erf}^{-1}(1-\alpha)\right)\right) = 1-\frac{\alpha}{2}. \end{aligned}$$

The computed lower bound for correct ranking with relation $\preceq_c$, i.e. $P(Y_i \leq Y_j \,|\, x_i \preceq_c x_j)$, is depicted in figure 4.3 as a solid black line for values of $\alpha \in [0,1]$. Of course, for large α, the probability of incorrect ranking approaches $\frac{1}{2}$, i.e. random sorting performance. We also include the probabilities according to equation 4.12 for different values of $d', \hat{\sigma}_i, \hat{\sigma}_{ij}$ and $\beta = \frac{\hat{\sigma}_j}{\hat{\sigma}_i}$. The colours of the lines identify the value of $d' \in \{0,1,2\}$ and the type of the line (solid, dashed or dotted) corresponds to the value of $\beta \in \{0.5, 1, 1.5\}$ in each plot. Each plot is generated from different combinations of $\hat{\sigma}_i$ and $\hat{\sigma}_{ij}$.

As is obvious from equation 4.12, the higher d', the higher is the probability of correct ranking. Therefore, it is expected that the green lines are generally above the blue ones which are above the red ones. What we also observe from the comparison of the solid, dashed and dotted lines per colour is that, if the absolute values of uncertainties $\hat{\sigma}_i$ and $\hat{\sigma}_j$ are higher, the correct ranking probability is higher in most cases (except for small $\hat{\sigma}_i$). However, depending on $\hat{\sigma}_i$ and $\hat{\sigma}_{ij}$, the overall order of the compared combinations does vary.

By comparing the different plots in figure 4.3, we can see that, if the uncertainties are uncorrelated ($\hat{\sigma}_{ij}$, first row), small differences in the distance of the predictions d' make a large difference in terms of the resulting correct ranking probabilities, especially for smaller d'. For higher d', the probabilities remain very high, even for large α. With increasing $\hat{\sigma}_i$ (first row, left to right), however, d' is a much less distinguishing factor as the probabilities decrease overall. This is of course expected, since for large prediction uncertainties, the actual predictions contains less certain information for ranking them.

In the second row of the plots, we can observe that for constant $\hat{\sigma}_i$, increasing the correlation of the prediction uncertainties $\hat{\sigma}_{ij}$ also increases the probability of correct ranking overall. In this case, β, i.e. the ratio between $\hat{\sigma}_i$ and $\hat{\sigma}_j$, is the main distinguishing factor. This is probably because the relation can handle larger absolute uncertainties if they are positively correlated, as errors are mainly made in the same direction.

In case of negative correlation, as depicted in the third row of the plot, we do not observe the same increase of correct ranking probability with increasing absolute correlation. Consequently, β, i.e. the ratio between $\hat{\sigma}_i$ and $\hat{\sigma}_j$, seems to make less of a difference.

We have reviewed our theoretical results for relation $\preceq_c$ by simulating ranking with $\preceq_c$ on data generated according to the various combinations of the parameters. We were able to observe very similar results as depicted in figure 4.3. In general, for the default

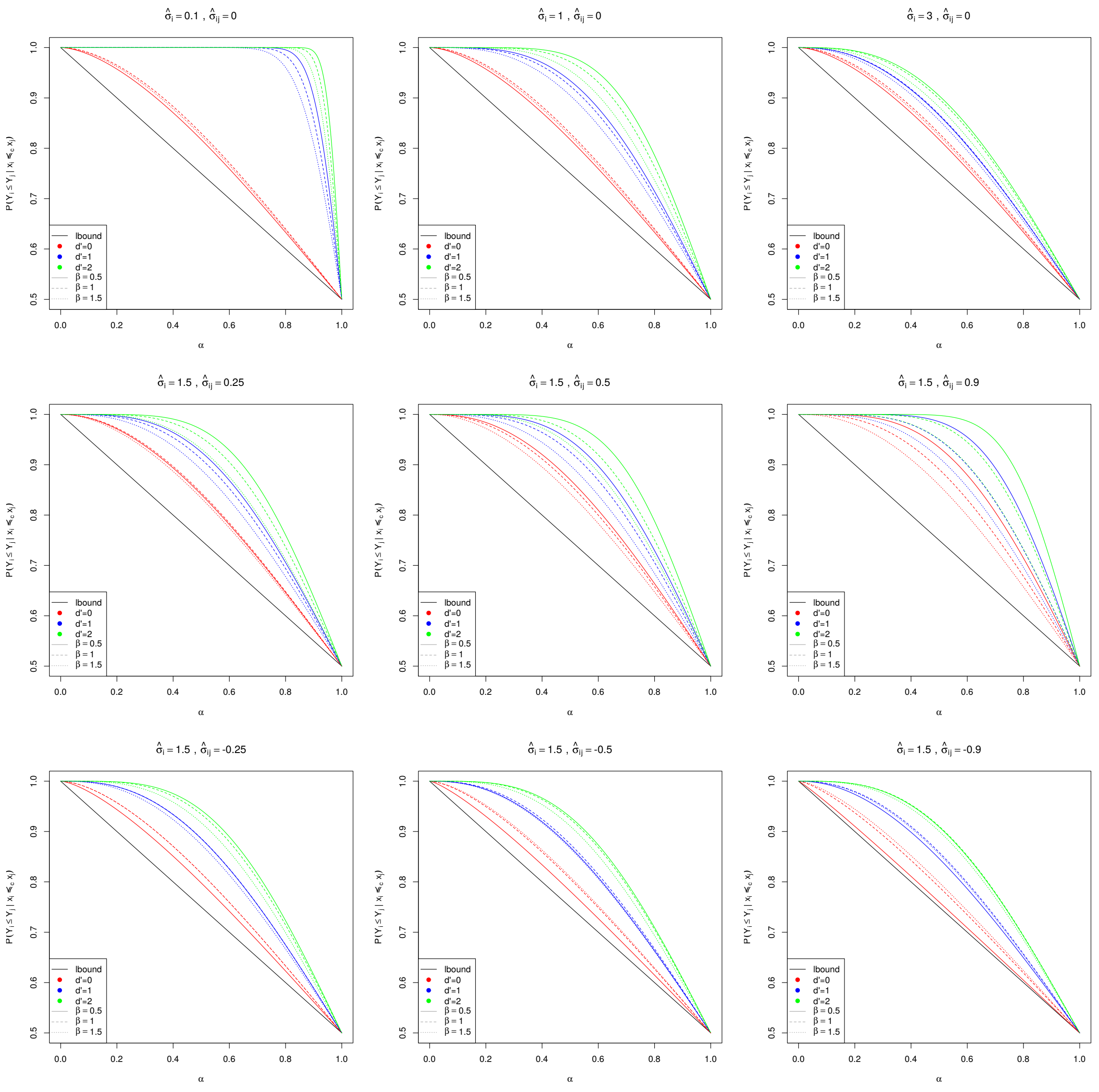


Figure 4.3: Lower bounds (solid black) and computed probabilities for correct ranking with dominance relation $\preceq_c$ plotted against α for different values of d',$\hat{\sigma}_i$,$\hat{\sigma}_{ij}$,$\beta = \frac{\hat{\sigma}_j}{\hat{\sigma}_i}$, where colours correspond to d' and line type corresponds to β.

value of $\alpha = 0.05$, the observed probability / frequency of correct ranking is very high. Even the lower bound of $(1 - \frac{\alpha}{2}) = 0.975$ results in a acceptable error probability.

Conclusions As expected, we were able to significantly improve the probability of achieving correct rankings using the concept of comparison under uncertainty when contrasted against comparing individuals just using their predicted values. However, for the previous analysis, we assumed that the model was actually valid (assumption **A1**

holds). The lower bound for ranking errors with $\preceq_c$ was determined to be $1 - \frac{\alpha}{2}$ with this assumption.

However, if we consider that in SAPEO, all models are validated using cross-validation as described in section 4.2.2, we can relax this assumption further. Given a model that is valid in SAPEO, there is a 5% chance that the model is not reliable. In the following, we assume the worst case, i.e. all rankings based on these invalid models are incorrect. We then get an updated lower bound for correct rankings as

$$0.95 \cdot \left(1 - \frac{\alpha}{2}\right) + 0.05 \cdot 0 = 0.92625 \tag{4.14}$$

for $\alpha = 0.05$, which is still very promising.

4.2.3.2 Selection Errors

We define selection errors as the number of incorrectly selected parents. More formally, the number of selection errors is the set difference between the correct selection according to the actual fitness function and the population selected based on the dominance relation. In the following, we analyse the expected number of selection errors.

Let R_r be the set of individuals of rank r according to $\preceq_c$ and r_h be the highest rank. The ranking does not cause an incorrect selection iff all pairwise comparisons between individuals of different ranks are correct, i.e.

$$\forall r \in \{1, \ldots, r_h - 1\}, \forall x_i \in R_r, \forall x_j \in R_{r+1} : x_i \preceq_f x_j.$$

The necessary comparisons can be encoded in a similar form as above, i.e. as the difference of two random variables. Let $x_i^{(r)} \in R_r$ be the i-th individual in rank r. $Y_i^{(r)}$ then denotes the random variable describing the fitness value prediction for individual $x_i^{(r)}$. Also, let

$$Z_{i,j}^{(r)} = Y_i^{(r)} - Y_j^{(r+1)},$$

with $r \in \{1, \ldots, r_h - 1\}, i \in \{1, \ldots, |R_r|\}$ and $j \in \{1, \ldots, |R_{r+1}|\}$. Now, let E_r be a random variable that describes the number of ranking errors. Thus, the probability of no ranking errors can be described as

$$P(E = 0) \Leftrightarrow P\left(Z_{i,j}^{(r)} \leq 0 \text{ for all } r \in \{1, \ldots, r_h - 1\}, i \in \{1, \ldots, |R_r|\}, j \in \{1, \ldots, |R_{r+1}|\}\right).$$

In order to express this probability, we can use an affine transformation with $y = c + Bx$ and $c = \vec{0}$ where

$$x \sim N(m, \Sigma),\ m = (\hat{f}(x_1), \ldots, \hat{f}(x_\lambda))^T, \Sigma = \begin{pmatrix} \sigma_1^2 & \sigma_{1,2} & \cdots & \sigma_{1,\lambda} \\ \vdots & & & \\ \sigma_{\lambda,1} & \cdots & \cdots & \sigma_\lambda^2 \end{pmatrix}$$

$$y \sim N(Bm, B\Sigma B^{-1})$$

according to [139, p. 32]. In this case, B encodes the comparisons, i.e. B contains λ columns and a row for each $Z_{i,j}^{(r)}$ where the i-th element is 1, the j-th is -1 and the rest

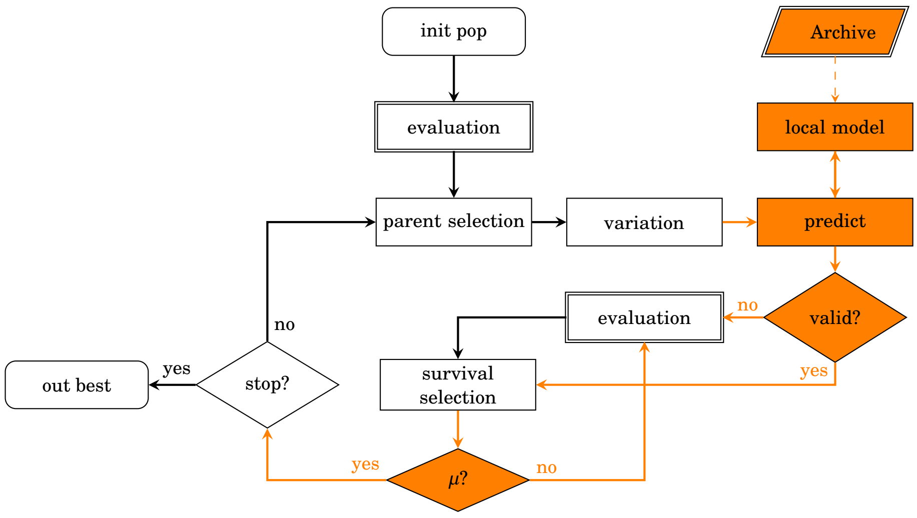


Figure 4.4: SAPEO. A framework to integrate uncertain evaluations into an evolutionary algorithm. Indviduals are only evaluated when necessary: (1) the model is unreliable, or (2) no safe selection can be made. Evaluations indicated with double borders are added to an archive. Additions to the EA skeleton from figure 2.1 are marked in orange.

0. We can then compute the probability of no ranking errors $p = P(E_r = 0)$ using the cumulative distribution function of the multivariate normal distribution for y.

Let E_s be a random variable that describes the number of selection errors. It is clear that $E_r = 0 \Rightarrow E_s = 0$, which is the case with a probability of p. For a non-strict upper bound on the expected value of E_s, we assume that in cases where $E_r \neq 0$ (and thus $E_r > 0$, as the number of ranking errors is always positive), we observe the worst-case selection error. This means that none of the μ individuals in the parent population would have been selected based on the actual fitness function, thus resulting in μ selection errors. We thus assume

$$P(E_s = 0) = p \text{ and}$$
$$P(E_s = \mu) = 1 - p.$$

It follows that for the expected value of E_s it holds that $\mathbb{E}(E_s) \leq \mu(1-p)$.

4.2.4 SAPEO Framework

In the previous section, we have shown that we can reasonably assume that comparisons under uncertainty are correct, provided the models are validated. We use these findings to define the SAPEO framework. We thus describe how we integrate comparisons under uncertainty and model validation into the standard EA algorithmic skeleton. A visualisation of SAPEO can be found in figure 4.4.

In general, an evolutionary algorithm (cf. white elements in figure 4.4) creates an initial population, which it then evaluates and sorts according to the obtained fitness values. Based on the order, it then selects a subset as parent population to generate a new population from via variation. This loop continues until a stopping criterion is reached, for example because the algorithm converged or the budget is exhausted. Fundamentally, SAPEO follows the same steps, but adds a small number of mechanisms in order to incorporate a surrogate model and the dominance relations.

The most important change is that the exact evaluation is replaced with a prediction computed by a local surrogate model. The model is trained on k individuals that have been evaluated exactly in previous iterations. For that reason, SAPEO keeps an archive of all exact function evaluations.

The main characteristic of SAPEO is its lazy evaluation, which is possible because as long as the individuals can be selected according to their predicted fitness value, the true objective value is not needed in an evolutionary algorithm. Thus, in SAPEO, exact evaluations are replaced with predictions computed by a validated local surrogate model.

Provided that a partial order already clearly identifies the μ best individuals, the evolutionary algorithm can continue with these individuals as parents for the next generation without diverging from the evolutionary path. If this is not the case, SAPEO refines the partial order by ranking the individuals that can not be selected confidently (*critical individuals*) with another dominance relation[145]. In this thesis, we only consider Pareto dominance based on the exact function evaluations as a secondary dominance relation. Critical individuals are therefore evaluated with the expensive fitness function until all relevant ties are broken and μ individuals can be selected confidently.

4.3 Game-Benchmark for Evolutionary Algorithms

The game-benchmark for evolutionary algorithms (GBEA)[2] was specifically designed to provide the means to analyse and compare the performance of evolutionary algorithms on game optimisation problems. The first version of the benchmark was presented to the research community at a workshop at GECCO 2018 (Kyoto, Japan)[3]. It is under active development and meant to be continuously extended with new problem suites from publications related to game optimisation[4].

The reasoning for developing a novel benchmark for the purposes of this thesis, and research beyond that, are two-fold. As discussed in section 3.4, at the time of writing …

1. … there exists no benchmark for game-related algorithm based on established benchmarking principles.

2. … there exists no real-world benchmark for evolutionary algorithms that measures anytime performance and provides comparable post-processing features.

A major advantage of creating a new benchmark is of course that it enables us to tailor the problems specifically to the questions we want to tackle in this thesis. As

[2]https://ls11-www.cs.tu-dortmund.de/people/volz/gamesbench.html
[3]https://ls11-www.cs.tu-dortmund.de/people/volz/gamesbench_events.html#gecco18
[4]https://ls11-www.cs.tu-dortmund.de/people/volz/gamesbench_part.html

we want to specifically tackle uncertainty handling in game optimisation, we take into account the insights developed in section 4.1.4 in order to identify requirements for the benchmark in section 4.3.1. Following that, we describe how these requirements are achieved along with more implementation and technical details in section 4.3.2.

4.3.1 Requirements Analysis

For the game-based benchmark used in this thesis, we identify the requirements described in the following.

I: Problem characteristics Problems contained in the benchmark should not be artificial in nature. As justified in sections 1.1 and 3.1, we focus on game optimisation problems. The problems and their fitness functions therefore should make sense within the context of the game and the corresponding design goals. The benchmark should contain a diverse set of fitness functions and reflect numerous state-of-the-art approaches to game evaluation. To follow this requirement, several fitness functions should be included for the same game that use different principles according to the taxonomy (see section 4.1). In order to simulate the playthroughs, AI players should be available for the game that perform at a reasonable standard.

II: Practicality Despite the fact that the benchmark is intended to contain (expensive) real-world problems, the execution of the benchmark should still be possible within a reasonable time frame. Therefore, it should be easy to parallelise the benchmark and the evaluation of a single solution should result in practical execution speeds on standard machines.

Ideally, it should also be possible to export and import benchmarking results. This enables a comparison of algorithms without the need for re-implementation and re-running the algorithm.

III: Analysis of Uncertainties The benchmark should allow an analysis of all or some of the uncertainties that occur in game optimisation problems as identified in section 4.1.4. It should also include features that allow an analysis of non-symmetric uncertainties, such as *Data Generation bias*. Data Generation bias is specifically relevant, as it occurs in game optimisation (and other simulation-based optimisation problems) and significantly affects algorithms (cf. case study in appendix B), but is rarely addressed systematically (see section 3.1).

In the context of an automatic benchmark, it is naturally not possible to include feedback from human players, unless pre-recorded. However, this would be impracticable for even small game optimisation problems. An alternative is therefore to simulate human feedback with an AI or model different from the one used for automatic evaluation.

IV: Statistical significance As evolutionary algorithms are stochastic, the statistics obtained via the benchmark should be statistically justified and thus interpretable.

V: Investigation of scaling behaviour Functions should be scalable in search space dimension, so that scaling behaviour can be analysed. Plots that visualise scaling behaviour should be generated.

Bonus: Comparability of SAEA The benchmark is intended for evolutionary algorithms targeted at expensive optimisation problems. Most of these algorithms are surrogate-assisted EAs, therefore it would be a useful feature to provide a modularised framework with implementations of commonly employed models and techniques, such as Kriging models and infill functions. This allows for a fair comparison independent of specific implementations for the algorithms and enables an analysis of the underlying approaches instead.

4.3.2 Implementation of Requirements

For many of the requirements, the COCO framework provides suitable features already. For example, COCO already includes a batch mechanism, allowing for the independent execution of subsets of the benchmark functions. This provides an easy opportunity to parallelise the the benchmark as required (II). Export and import features are also provided by the COCO framework (II).

Furthermore, the COCO framework is also designed to include the same functions in multiple search-space dimensions and provides post-processing features that contain plots that visualise an algorithm's behaviour in that regard (V). Similarly, COCO expects the existence of multiple instances of any given function. The average runtime measures that are automatically computed based on an algorithm's aggregated performance across these instances are easily interpretable and justified in terms of statistical significance (IV) (see section 2.2.1).

COCO also provides features that allow the creation of new functions and corresponding suites. However, this is only intended for artificial functions defined within c. For the GBEA workshop at GECCO 18, we (mainly Tea Tušar) created an interface to allow the interaction with external applications, called either via c or python.

Given this interface, fulfilling the requirements listed above only relies on the ability to define benchmarking problems for multiple search dimensions and with suitable instantiation methods. Similarly, the problem characteristics (I) and execution speed (II) also relies on the included benchmarking problems. We therefore discuss the function suites developed for GBEA, specifically with regards to the requirements. We have created function suites (*TopTrumps* and *MarioGAN*) based on two optimisation problems previously published in literature. Both of these problems concern the generation of game content and allow for its automatic evaluation. They are described in more detail in sections 2.4.1 and 2.4.2, respectively. Unfortunately, we were not able to find feedback data appropriate to evaluate the targeted problems. We therefore are not able to represent functions in feedback categories EXP and IMP, but plan to do so in the future. However, we were able to define functions spanning all categories of the input dimension.

TopTrumps Suite The TopTrumps problem could be used without major modifications. Specifications of the function suite and more details can be found in in section 4.3.3.1. As the problem requires the generation of a deck, the predetermined number of cards in a deck and/or the number of values on each card can be modified in order to create scalable problems (V). The original publication [148] already contains diverse functions with differing numbers of objectives (I). For instance, all taxonomic categories across the input dimension (i.e. NONE, IMP, EXP) are represented. Furthermore, AIs of different skill levels are already implemented (I) as well as surrogate functions suitable to simulate AI bias (III).

As expected in game optimisation problems, the included functions are noisy. However, the fitness for each solution is reported as the average of 2000 simulations, which has been shown in [148] to produce an appropriate balance between computational effort (II) and resulting standard deviations.

The only remaining issue is to create suitable instances of the functions (IV), that one the one hand create fitness landscapes of similar type and structure, but on the other hand do not share the locations of e.g. optima (cf. section 2.2.1 for the intentions and requirements on COCO instances). We therefore decide to interpret instances as themes for the created decks. These themes along with the chosen categories dictate the value ranges that are expected for each of the categories on the cards (see figure 2.5). We therefore represent the different themes by introducing lower and upper bounds for each category on the cards. The bounds are created via seeded pseudo-random generation, and each configuration of constraints is considered a separate instance.

MarioGAN Suite In order to increase the variety of game optimisation problems in the benchmark (I), we wanted to include a second suite that addresses a game optimisation problem significantly different than TopTrumps deck generation, as described in section 2.4.2. We were thus looking for a more complex and popular game with existing well-performing AI (I), that still allow for reasonably fast simulations of a playthrough (II). We settled on Mario, as this game has been heavily researched before [138], the levels are relatively short, and there is a publicly available framework *MarioAI* containing various state-of-the-art AI players. Specifically, we base the function suite on the recently published level generation method using GANs [150] as discussed in section 2.4.2. The solutions of a problem are in this case represented as continuous latent vectors, thus making them suitable for state-of-the-art evolutionary computation methods. Additionally, this differs from the near direct encoding in the TopTrumps Suite.

Another benefit of the latent vector encoding is that it allows for easily scalable functions (V), as the dimension of this latent vector is chosen arbitrarily when training the GAN. Therefore, fitness functions with different search space dimensions can be created by simply basing them on the results of GANs trained to have appropriately sized latent vectors. Similarly, different GANs can also be used to create instances (IV), as they represent different level generation models that assumably exhibit similar characteristics. Therefore, to create instances, GAN models are trained from different seeds, resulting in neural networks with different weight configurations. As the best latent vector is dependent on the weight configuration, the location optima for the instances will likely be significantly different across the instances. Furthermore, a simulation of a playthrough with a player AI in the *MarioAI* framework is capped at 20 seconds, thus also enabling practicable benchmarking speeds (II).

A major issue with the MarioGAN suite, however, were the characteristics of the fitness functions (I) in the original publication [150]. Only two functions were proposed, and it was not entirely clear whether these functions were chosen to be meaningful from a design perspective. Additionally, the GAN was trained on a single level in order to test the limits of the level generation method proposed in the paper. However, in practice, the GAN would need to be trained on a sizeable sample of representative levels. The optimisation problem as described in section 2.4.2 with only a single level sample, is therefore still somewhat artificial. Another issue making this implementation not ideally suitable as a real-world problem is that the available mechanics, enemies and tiles for the Mario levels were restricted. This reduces the complexity of the actual level generation problem and constraints the search space significantly.

In order to create a better real-world representation in context of MarioGAN, we therefore made several changes:

- The set of samples was extended significantly (from a single level to 12, 2, or 19 levels, depending on which function is used)
- Additional fitness functions were added, inspired by a survey of automatic evaluation methods for Mario [130]
- Where possible, restrictions on the search space are lifted (added 4 more types of enemies, including variations)

Details on these changes and the specifications of the problems contained in the MarioGAN function suite can be found in section 4.3.3.2.

Experimental Framework Unfortunately, the COCO framework does not provide a way to track uncertainties during the runtime of the algorithm. Therefore, even if the function suites allow for usecases to investigate Data Generation bias and other errors, the actual tracking is not supported. To combat this issue, we provide a framework that includes an interface to COCO. To allow for easy integration, it is written in C++. More details can be found in section 4.4.

The framework provides a modularised approach to surrogate-assisted EAs using an object-oriented implementation of commonly required components, such as Kriging models, evolutionary algorithms and infill criteria. This satisfies the bonus requirement by facilitating fair comparisons between SAEA algorithms. Additionally, if the algorithms are implemented using the framework, predicted values, uncertainties, and correct values are tracked and reported automatically. The data can then be visualised with the included plot scripts for an analysis of uncertainties (III).

fid	description	input	feedback	goal
1	deck hypervolume	CODE	NONE	max
2	standard deviation of category averages	CODE	NONE	max
3	winrate of better player	OUT	NONE	max
4	switches of trick winner	PLAY	NONE	max
5	trick difference at the end of the game	OUT	NONE	min

Table 4.2: Overview and characterisation of functions in rw-top-trumps

4.3.3 Technical Details

In the following, we provide technical details on the GBEA and its implementation. We only describe information directly relevant to this thesis, such as for the interpretation of evaluation results. Any additional details can be found in the documentation of the framework, available from the GBEA website[5].

4.3.3.1 TopTrumps Suite Details

We created a single- and a bi-objective suite for TopTrumps based on the functions described in the original publication [148]. A detailed explanation of the functions can be found in section 2.4.1. However, the original publication explicitly addressed multi-objective optimisation, making their single-objective versions less interesting. In this thesis, we will thus only address the bi-objective function suite (rw-top-trumps-biobj), which contains functions combinations from the single-objective suite which are likely conflicting.

rw-top-trumps Contains 5 different functions, where function 1 and 2 are based on encoding, whereas the others are based on a simulation as described in section 2.4.1. An overview of the functions along with a characterisation according to the taxonomy described in section 4.1 can be found in table 4.2. It can be seen that, while the proposed functions are diverse in terms of their input, none of them are based on any feedback. However, this imbalance is reflected in our survey on automatic game evaluation methods (see appendix A), as well as in the original publication [148]. As this benchmark is intended as a reflection of the game optimisation problems presently targeted in research, and not a discussion on game evaluation itself, we deem this selection of functions suitable. However, more functions can of course be added in the future.

While the functions are not based on feedback, they are motivated by a model of the intended gameplay achieved with a generated deck of cards for TopTrumps. Function 3, for example, is the winrate of the better player. This function is set to be maximised so that higher skill levels lead to higher winrates. This could appeal to a player's sense of fairness, while it might be frustrating for weaker players. In contrast, functions 4 and 5 target the tension of the game instead, as they both reach their optimum if the game was

[5]http://norvig.eecs.qmul.ac.uk/gbea/gamesbench_doc.html

rw-top-trumps	
objectives	1
dimensions	$(88, 128, 168, 208) = 4 \cdot (22, 32, 42, 52)$
functions	5
instances	15
simulations per point	2 000 [148]
rw-top-trumps-biobj	
objectives	2
dimensions	$(88, 128, 168, 208) = 4 \cdot (22, 32, 42, 52)$
functions	6
instances	15
simulations per point	2 000 [148]
rw-gan-mario	
objectives	1
dimensions	10, 20, 30, 40
functions	84
instances	7
simulations per point	30

Table 4.3: Function suite details

close, independent of the skill levels of the players. Function 5 looks at the final outcome, while function 4 also considers how dramatic the playthrough was.

Functions 1 and 2 are computed without simulations, but still target similar concepts. If the deck hypervolume (function 1) is maximised, each card is not dominated by another one. This makes it possible for anyone to come back from a losing streak, as there is always at least one category for each card with which it could beat another. Maximising this function allows for tension in the game, just like expressed in functions 4 and 5. However, having only non-dominated cards in a deck also reduces the randomness in the game, which would aid weaker players. Function 2 uses some implementation intricacies of the weaker and stronger AI player and, if maximised, should result in higher winrates of the stronger AI.

More details on the function suite are specified in table 4.3. As described in section 4.3.2, instances are interpreted as different deck themes, signified by setting different lower and upper bounds for the card values. In order to achieve compatibility to the BBOB function suite, we created 15 instances. The problems are relatively large in terms of search dimension (see table 4.3) when compared to most existing benchmark suites. This because we intended to create realistic problems with a typical number of cards in the deck (i.e. 22, 32, 42 or 52) and number of categories on each card (4). For the simulated functions, each solution is simulated 2 000 times as suggested in [148].

rw-top-trumps-biobj The bi-objective function suites combines functions from the single-objective suite that are seemingly conflicting. An overview of the functions is

	function 1			function 2		
fid	fid	input	feedback	fid	input	feedback
I	1	CODE	NONE	2	CODE	NONE
II	3	OUT	NONE	4	PLAY	NONE
III	3	OUT	NONE	5	OUT	NONE
IV	1	CODE	NONE	3	OUT	NONE
V	1	CODE	NONE	4	PLAY	NONE
VI	1	CODE	NONE	5	OUT	NONE

Table 4.4: Overview and characterisation of functions in rw-top-trumps-biobj

presented in table 4.4. Details on the suite can again be found in table 4.3.

Function I is just based on functions 1 and 2, so only computed based on the encoding. It is thus significantly faster to compute than the others. However, the functions are only partly conflicting, as function 1 expresses both tension as well as fairness (with regard to better winrates for the better player). However, previous work also showed that just optimising function 1 did not create satisfying results when compared to a Pareto front based on functions 3-5 [148]. To investigate this further, functions IV-VI test the case where functions 3-5 are added as a second objective to function 1.

The conflict of objectives is more obvious for functions II and III, where the first function (function 3) targets fairness, while the second function (function 4 or 5) targets some expression of tension in the game.

4.3.3.2 MarioGAN Suite Details

Based on the MarioGAN problem, we created an extensive single-objective function suite using the problem described in section 2.4.2 as a baseline. As stated in section 4.3.2, the existing work was modified in terms of the level encoding, the training samples as well as the evaluation functions. The details for all are described in the following.

The GBEA also contains a bi-objective function suite created by combining the single-objective functions. However, the conflicts between the objectives are not always clear, as these functions were created to be stand-alone. In this thesis, we will therefore only be targeting the single-objective suite.

Level Representation The encoding is detailed in table 4.5. Note that the tubes (both with piranha plant and without) as well as the bullet bill stands are only represented by a single symbol. When processing the generated level, if one of these symbols is detected, the corresponding obstacle is automatically extended downwards until it meets the ground or a solid tile. This reduces the complexity of the encoding and at the same time will ensure that tubes and bullet bill stands are always generated correctly (i.e. without missing chunks), which was an issue with the previous encoding [150]. Additional enemy characters are added when compared to the original publication. All regular enemy characters (identity 9 and upwards) are rendered as winged if they are on top of a passable tile (i.e. in the air).

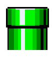
Tile type	Symbol	Identity	Visualization
Stone	X	0	
Breakable	x	1	
Empty (passable)	-	2	
Question Block with coin	q	3	
Question Block with power up	Q	4	
Coin	o	5	
Tube	t	6	
Piranha Plant Tube	p	7	
Bullet Bill	b	8	
Goomba	g	9	
Green Koopa	k	10	
Red Koopa	r	11	
Spiny	s	12	

Table 4.5: Tile types used in generated Mario levels.

original	replacement
empty question block	brick
surprise brick	question block
star	fire flower
1-up mushroom	regular mushroom
hammer bro	red koopa
lakitu	
lakitu egg	winged spiny
buzzy beetle	goomba

Table 4.6: Original tiles and their replacements

Modifications to the Training Data The VGLC encoding of the original levels was modified slightly, as not all tile behaviours in Super Mario Bros. are implemented in the MarioAI framework. A table with the original tiles and their replacements can be found in table 4.6.

Additionally, some of the levels had to be modified. Areas, where usage of moving platforms was required, were removed. As MarioAI does not implement the option to traverse through pipes, Mario always starts on the left of the level. Thus, vertical walls as visualised in figure 4.5, where Mario would have dropped down from the top, were also removed. In order to keep the height of the levels constant, empty rows also had to be removed for some levels.

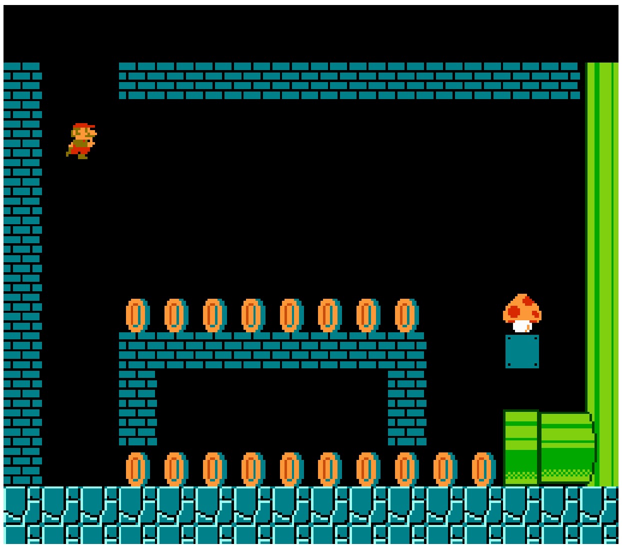


Figure 4.5: Example of Mario level with vertical walls (Super Mario Bros. Bonus Area D)

Training Samples In the following, we list the levels that were transferable to the MarioAI framework without large modifications an are thus used as training samples.

I Super Mario Bros [overworld]: 1-1, 2-1, 3-1, 3-2, 4-1, 5-1, 5-2, 6-1, 6-2, 7-1, 8-1, 8-2

II Super Mario Bros [underground]: 1-2, 4-2

III Super Mario Bros 2 (Japan) [overworld]: 1-1, 2-1, 2-2, 3-1, 4-1, 4-2, B-1

Separate GANs were trained for 3 sets of training samples, namely

- overworld (small): I
- underground: II
- overworld (large): I + III

For each of these sets, GANs were trained for an input vector of dimension $d \in \{5, 10, 20, 30, 40\}$, thus resulting in 15 distinct genotype-phenotype mappings for Mario levels. In order to create different instances, i.e. similar mappings, we started the respective training processes with varying random seeds. We thus created 7 instances per mapping.

Fitness Functions In the following, we discuss the objective functions included in the benchmark suite along with their original publications. We implemented a set of diverse functions, ranging through all types of inputs (see table 4.7). Like for the TopTrumps suite and with the same arguments (cf. section 4.3.3.1 and specifically the paragraph on rw-top-trumps), we only included model-based objective functions without feedback. All of these functions were proposed in previous literature (see source column in table 4.7).

fid	description	input	feedback	AI	goal	source
A	enemyDistribution	CODE	NONE	-	max	[130] (tile position)
B	positionDistribution	CODE	NONE	-	max	[130] (tile position)
C	decorationFrequency	CODE	NONE	-	max	[130]
D	negativeSpace	CODE	NONE	-	max	[20]
E	leniency	CODE	NONE	-	min	[118]
F	density	CODE	NONE	-	max	[130] (tile freq)
G	progress	OUT	NONE	A*	max	[150]
H	basicFitness	PLAY	NONE	A*	min	competition [138]
I	airTime	PLAY	NONE	A*	max	[130, 150]
J	timeTaken	PLAY	NONE	A*	max	[150], cf. [130]
K	progress	OUT	NONE	Scared	max	see G
L	basicFitness	PLAY	NONE	Scared	min	see H
M	airTime	PLAY	NONE	Scared	max	see I
N	timeTaken	PLAY	NONE	Scared	max	see J

Table 4.7: Overview and characterisation of functions in rw-gan-mario

More details on the functions and how they are computed can be found in the following list:

A	standard deviation of enemy tiles (x-axis)
B	standard deviation of tiles you can stand on (y-axis)
C	percentage of *pretty tiles*:= {Tube, Enemy, Destructible Block, Question Mark Block, or Bullet Bill Shooter Column} [130]
D	percentage of tiles you can stand on
E	weighted sum of subjective *leniency* of tiles
F	percentage of ground and breakable tiles
G;K	percentage of progress on x-Axis
H;L	MarioAI championship score for AI: $(\text{lengthOfLevelPassedPhys} - \text{timeSpentOnLevel} + \text{numberOfGainedCoins} + \text{marioStatus} * 5000)/5000$
I;M	ratio between ticks in air vs. total ticks
J;N	ratio between time taken and total time allowed

Function A, B and F are based on statistics suggested in [130] with no directly assumed meaning. Function B is proposed as an aesthetic measure in [130]. Function E is designed to express the leniency of the level design, as suggested in [118]. Function D is intended to capture how much of the space in the level is traversable, as proposed in [20].

The remaining functions are all based on simulations with two different types of AIs. One of them is a particularly successful agent from the MarioAI Championship based on the A* algorithm by Robin Baumgarten, which exhibits super-human performance on levels that do not require backtracking[6]. The other AI, ScaredAgent, is one of the default agents in the MarioAI source code, which works by avoiding any sort of obstacles, including enemies. It does not do any forward planning, however, and thus does not perform well in comparison to the A* agent. In contrast, it plays more like a novice human player.

Both of the AI agents are technically not stochastic, however, due to small variations during runtime in the game engine implemented in Java, their behaviour does vary in some cases. In instances where this variation occurs, if a given AI is run repeatedly on the same level, it will typically switch between a small number of (similar) behavioural patterns. However, in order to not rely on outliers to evaluate a given solution, we execute the simulation 30 times per solution and average the results given by the fitness function in question.

Function G is the progress of the AI, similar to the playability used in many publications, including [150]. Functions I and J are based on functions implemented for [150] as well (# jump actions and total actions taken, respectively), they are however modified slightly to optimise their expressiveness. Function I was modified to target the number of ticks in the air instead of the number of times the jump action was selected, because the jump action can be triggered without observable effect (while Mario is already in the air), and can result in different lengths of jumps. Airtime is thus more indicative of whether the level is easily traversable without jumping or not. For function J, we replaced the number of total actions with the ratio of time taken to total time allowed. Both functions, when optimised, result in levels that take longer / more actions to complete. However, timeTaken has a clear optimum and nadir value, which the number of total actions does not. In the original publication [150], for function I and J, the values were heavily penalised if the level was not playable, i.e. function G resulted in a value less than 1. For our implementation of the benchmark, this is still true, but as penalty, we are now able to set the nadir value instead of a large arbitrary value that might upset the optimisation algorithm significantly. Finally, function H is based on the MarioAI competition score for the AI agents.

The above also holds for functions K-N, which are the same respective objectives, but measured using the ScaredAgent instead of the A* agent.

Unfortunately, there is no clear target value or optimisation direction for any of these functions. For example, when trying to generate easy levels, one might intend to minimise leniency (E) and / or maximise how well the AI players do (basicFitness, H/L). As the benchmark is not directly related to a specific task, however, this decision is arbitrary. We fixed the optimisation goals as specified in table 4.7, aiming for interesting benchmark functions.

We have added two types of variations for each of the fitness functions described above:

[6]http://www.youtube.com/watch?v=DlkMs4ZHHr8

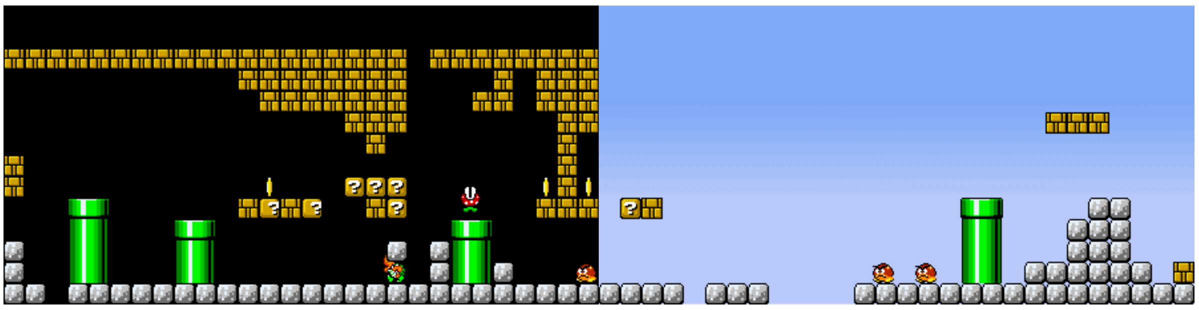


Figure 4.6: Examples of generated level segments. Left: underground level. Right: overworld level.

- training samples (overworld, underground, overworlds)
- level segment concatenation

The first variation are the various models trained for the respective sets of samples. Since, for example, overworld and underground levels exhibit different characteristics that are imitated by the generated levels, the same fitness function will produce different challenges in this case. An example of the different characteristics of underground and overworld levels can be seen in figure 4.6. The underground levels are designed to look like a dungeon which is visualised by the existence of a ceiling. This characteristic is picked up by the level generator and clearly reflected in the generated levels, as seen in the example in the figure. The ceiling adds an additional challenge to the level, as jumps might not be executed as planned when Mario bumps into the ceiling.

We further introduced a concatenation mode to make the problems more realistic and challenging. In the original publication [150], level segments of a fixed size are created by a generator. The levels were split into segments in order to assure a sufficient number of level samples. However, in practice, these segments would need to be played in direct sequence in order to allow for a reasonably long playing experience. We therefore also trained GANs with a latent vector dimension of 5. Since the dimensions this suite is available for are all multiples of 5, the latent vector inputs of a given problem can be split into multiple 5-dimensional vectors. These are then fed to the corresponding generator for 5-dimensional inputs and the resulting levels are concatenated. The concatenation mode adds an additional realistic challenge, as the intersections between different segments still need to be playable, which is not considered in the training phase of the generator.

The variations naturally extend the number of problems in the suite significantly. The resulting functions are listed below. The function ids can be computed as follows:

$$id = g + f \cdot G + c \cdot F \cdot G + 1,$$

where

- G is the number of different sample sets (3: overworld + underground + overworlds)
- F is the number of fitness functions (14)

- g is the id for sample sets (i.e. 0 for overworld, 1 for underground, 2 for overworlds)
- f is the id for fitness functions (i.e 0 for enemyDistribution, 1 for positionDistribution, etc.)
- c is an indicator for concatenation (i.e. 0 if not concatenated, 1 if concatenated)

The resulting 84 functions are listed in the following tables, with the letters A-N specifying the type of objective function as listed in table 4.7.
Without concatenation:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	f/g
1	4	7	10	13	16	19	22	25	28	31	34	37	40	I: overworld
2	5	8	11	14	17	20	23	26	29	32	35	38	41	II: underground
3	6	9	12	15	18	21	24	27	30	33	36	39	42	I + III: overworld

With concatenation:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	f/g
43	46	49	52	55	58	61	64	67	70	73	76	79	82	I: overworld
44	47	50	53	56	59	62	65	68	71	74	77	80	83	II: underground
45	48	51	54	57	60	63	66	69	72	75	78	81	84	I + III: overworld

rw-gan-mario The resulting single objective suite thus contains 84 different functions for 4 different search space dimensions and 7 instances. The dimension of the search space is solely determined by the size of the random vector that is fed into the neural network and can thus be set arbitrarily. We chose dimensions 10, 20, 30 and 40 based on the similarity to the BBOB search space dimensions. We also considered that in the original publication [150], the random vector was 32-dimensional, which resulted in several dimensions of the vector with only minor effects on the generated levels. It is important to note, however, that the corresponding GAN was trained on only a single level. The details of the function suite are summarised in table 4.3.

4.4 Experimental Framework

As stated in section 4.3.2, in order to fulfil the requirements developed in section 4.3.1, we have implemented an experimental framework in C++ that interfaces to the COCO benchmarking framework. The main reason for not using the COCO framework directly is that for our analysis, additional features were necessary that were difficult to add within the existing software. This approach also ensures the validity of the results, as no programming bugs can be added to the evaluation. Furthermore, this decision allows COCO, which is in continuous development, to be updated easily. The experimental framework is made available on GitHub[7].

[7]https://github.com/TheHedgeify/uncertaincoco

In section 4.4.1 we thus present an overview of the features implemented in the framework. Section 4.4.2 gives further details on features that relate to the usage of surrogate models and their predictions. Afterwards in section 4.4.3, the algorithms that are already included in the framework are described. Finally, we detail the post-processing features implemented in the framework in section 4.4.4.

4.4.1 Features

Below is an overview of features included in the experimental framework along with a short description of their purpose.

- COCO interface: Direct interface to the COCO framework, in order to make use of all of COCO's features, including logging and post-processing
- Shark: The Shark Machine Learning library [8] is used, which allows access to stable, fast and tested implementations of several state-of-the-art optimisation algorithms for both single- and multi-objective problems.
- Features for Surrogate-assisted Optimisation:
  - Improved Hypercube Sampling (IHS): Hypercube Sampling strategy using the ihs library[9].
  - Archive: Function evaluations can be automatically archived to collect the data required for training a surrogate model.
  - Kriging: As no suitable Kriging implementation was available in Shark, we added one using the libgp library[10].
  - "Cheater" Function: With each prediction made by the Kriging model, the true value of the search point in question is also evaluated and logged. This way, the prediction errors of the surrogate model can be computed and used for further analysis. Of course, the true value is not made available to the optimisation algorithm and does not add to the number of function evaluations counted within COCO.
  - Uncertainty Logging: In addition to the true values, all predictions along with the estimated uncertainties are logged automatically.
- Surrogate function: The experimental framework allows to specify another function, dubbed *surrogate function*, within the same function suite that is made available to the optimisation algorithm in addition to the actual function. The surrogate function can be evaluated independently from the actual function, and its evaluations do not count as function evaluations. This feature enables imitating the common practice of using models with different fidelities and associated computation costs

[8] http://image.diku.dk/shark/
[9] https://people.sc.fsu.edu/~jburkardt/cpp_src/ihs/ihs.html
[10] https://github.com/mblum/libgp

when optimising expensive problems. The experimental framework thus also allows benchmarking model management strategies for multi-fidelity models. In addition, it is possible to gather information on the reliability and systematic errors of the lower-fidelity model.

- Online convergence detection (OCD): In order to make the algorithms within the framework as comparable as possible, and also to increase the accuracy of the aRT measurements in COCO by ensuring multiple restarts (see section 2.2.1), we implemented online convergence detection based on the performance statistics used within COCO (i.e. fitness value for single-objective suites, and dominated hypervolume for multi-objective suites). An algorithm will be deemed as converged according to a χ^2 statistical test proposed in [153], which checks the most recent improvements over a given window of evaluations.

- Batch mode: As benchmarking a given algorithm on a given problem is independent to its performance on other problems, running the same algorithm on different problems at the same time is a natural and easy avenue to reduce the runtime of the benchmark through parallelisation. We implemented a batch mode within the framework that supports this type of parallelisation. The number of batches can be set arbitrarily, and the different problems are spread evenly between the batches. When the desired number of batches is set to the number of functions multiplied by the number of available instances for a given suite, the problems are distributed in a way, where one batch consists of a single problem (i.e. unique function and instance) in all available dimensions. This is an attempt at evening out the runtime for each batch, as problems with a larger search space take longer to solve, and are allocated a larger budget of function evaluations.

A schematic depiction of the framework can be found in figure 4.7. When starting an experiment, the optimiser is set up, as well as a function suite. The suite will then automatically handle iterating through the functions contained and restarting the optimiser as needed. The optimiser is implemented as an abstract class that aligns with the Shark implementation. Any optimiser in the Shark library can thus be set up as an optimiser within the framework. Several additional algorithms are also implemented, as visualised in figure 4.8 and described in section 4.4.3. All optimisation algorithms have access to their evaluation archive, a Kriging model and several sorting strategies, including comparisons under uncertainty (see section 3.2.3). In addition to the convenience added by the features listed above, they also improve the comparability of the implemented algorithms, as all use the same baseline implementations. For the surrogate-assisted evolutionary algorithms targeted in this thesis, this is especially relevant regarding the implementations of sampling and modelling, as well as the underlying optimisation algorithms.

The suite and the optimiser interact through the objective function, which is a direct interface to the COCO framework. COCO then handles the function evaluations as intended, either internally for the BBOB function suites, or by calling the respective applications for Mario and TopTrumps. The implementation is however also compatible

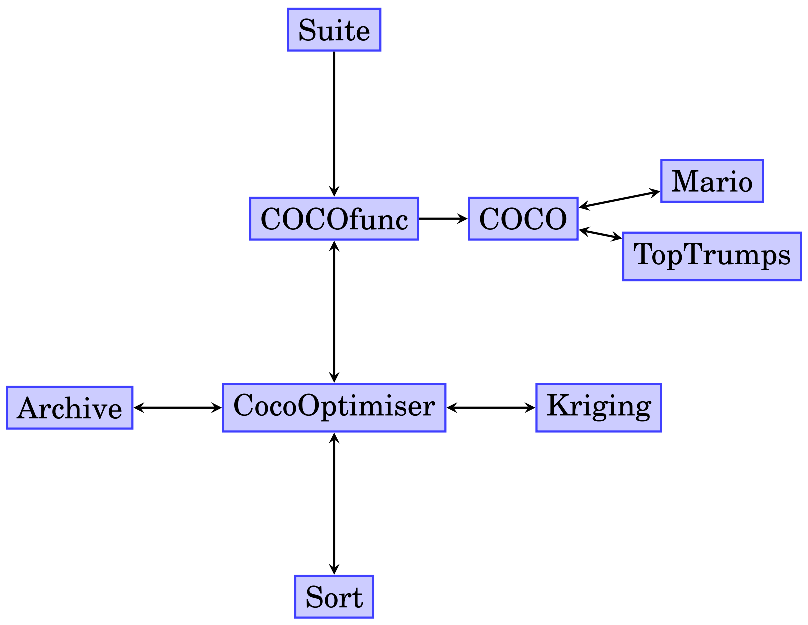


Figure 4.7: Schematic depiction of experimental framework.

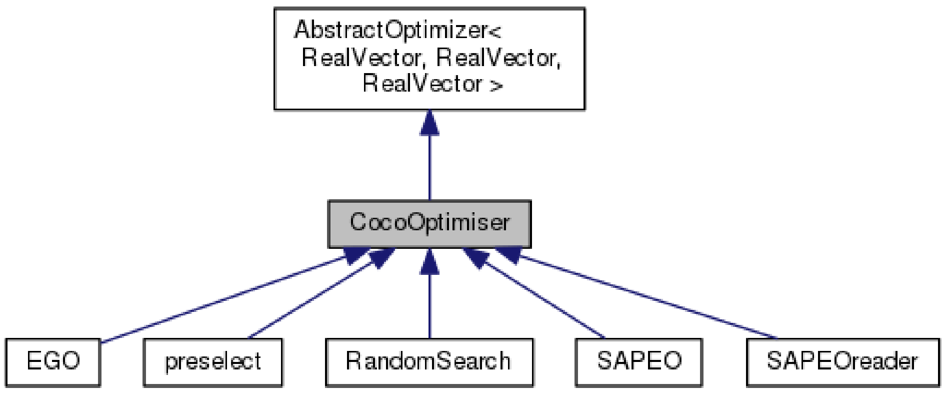


Figure 4.8: Inheritance graph for CocoOptimiser

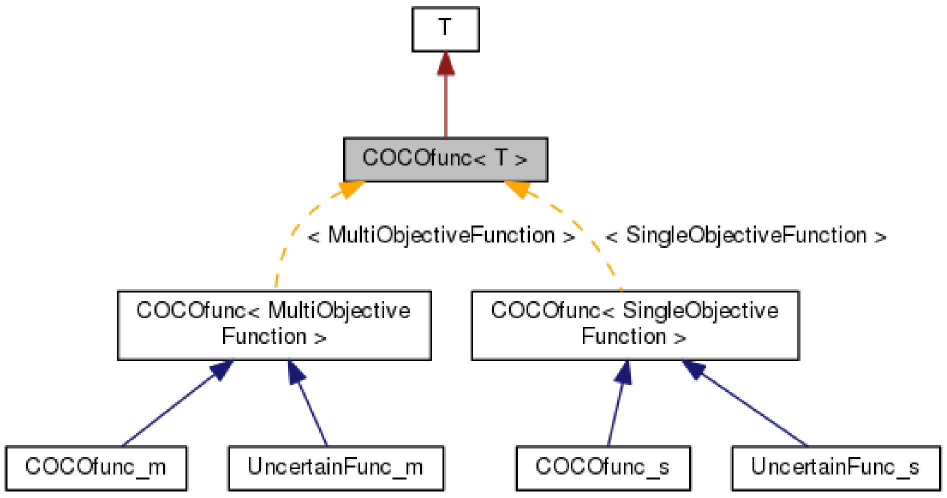

T
COCOfunc< T >
< MultiObjectiveFunction >
< SingleObjectiveFunction >
COCOfunc< MultiObjective Function >
COCOfunc< SingleObjective Function >
COCOfunc_m
UncertainFunc_m
COCOfunc_s
UncertainFunc_s

Figure 4.9: Inheritance graph for COCOfunc

with the implementation of objective functions in Shark. This allows running any o the existing algorithm in Shark on COCO benchmarks. See the inheritance graph in figure 4.9 for more details. In the graph, COCOfunc_m and COCOfunc_s are direct interfaces to COCO. On the other hand, UncertainFunc_m and UncertainFunc_s implement the model prediction, surrogate functions and several logging features. More details can be found in section 4.4.2.

In addition to the usual COCO postprocessing of the logs, we have added additional features described in section 4.4.4.

4.4.2 Uncertain Functions

As depicted in figure 4.7, evaluations of the fitness functions are handled by the COCO framework, which processes them and returns the result (COCO function). These evaluations are also automatically logged by COCO, as two COCO observers are tied to them. At the end of a run, the recorded data is written to files in a format suitable for the COCO post-processing features (see section 2.2.2). However, there are specific cases where calls are made to the COCO framework, but the evaluations are not tracked by a COCO observer. These cases are detailed below.

Uncertain Functions generally use a surrogate model and previously archived evaluations to compute an output value. They therefore are not counted towards the performance of the algorithm on the true fitness function. However, if configured accordingly, they can return true evaluations in certain cases, for example if there are not enough evaluations to train a model from. In these cases, evaluations are delegated to the corresponding COCO function, and therefore counted via the COCO framework. These evaluations are of course also added to the internal archive.

However, in order to be able to assess the errors throughout the runtime of an algorithm with the framework, the true values need to be computed and logged alongside the predicted values and their uncertainties. Therefore, for each evaluation, Uncertain Functions make a call to a second instance of COCO Function, which is not tracked by an observer. The resulting values are then logged, but not accessible to the algorithm.

Uncertain functions furthermore have a feature intended to enable the analysis of Data Generation bias. Data Generation bias can occur when AI playthrough behaviour is used as a basis for the evaluation of game content for human players. The intent of this approach is to include additional information on the problem, in order to reduce the number of expensive function evaluations required. However, this information can also be misleading, which is why it is important to analyse these specific errors within the experimental framework.

As it is impractical to include live human feedback in a benchmark, this behaviour can be simulated by indicating a specific function from the same function suite that can be used as a surrogate. Based on this surrogate function, a prediction model can then be computed that is not just based on the location of the sample, as is the default for surrogate-assisted EAs. Instead, a prediction model is computed that takes the surrogate function value (or both, surrogate value and location) as an input. In practice, the surrogate function would be chosen to be appropriately cheap to compute. Therefore, evaluations of the surrogate function are not recorded via the COCO framework.

The prediction model implemented is a Kriging model and trained during runtime, similar to the regular location-based model. However, the only requirement for the model is that it provides both a prediction and an error estimate to enable comparisons under uncertainty. Therefore, other modelling techniques can also be used, as well as pre-computed models.

4.4.3 Algorithms

Despite the efforts made in the implementation, the algorithms mentioned in the related work section 3.2 are still not comparable directly, as the original papers and corresponding implementations all contain specificities that the others do not. The most obvious difference is the usage of different underlying optimisation algorithms: The original EGO uses a branch-and-bound algorithm, while pre-screening uses the SMS-EMOA and GP-DEMO uses the DEMO algorithm. There are a number of additional differences, such as whether the algorithm was originally intended for single- or multi-objective optimisation. Even more differences become apparent when going into more detail, such as the fact that ParEGO uses scalarisation to handle multi-objective problems, where pre-screening uses an SMS-EMOA and therefore non-dominated sorting and hypervolume to guide the selection. The algorithms also differ in terms of their strategies to select samples to train the surrogate models.

As a result, a reliable comparison of the algorithms would require an extensive hyperparameter optimisation, potentially even specific to each function suite. Such a comparison could however also only rely on the performance exclusively and not take into account more in-depth statistics e.g. on the uncertainty predictions made or the percentage of offspring that improve the previous solution. Furthermore, this thesis is targeted at uncertainty handling, which should thus be evaluated independently of which surrogate model or which evolutionary algorithm is used.

Therefore, instead of exact implementations of the algorithm, the concepts introduced in the respective papers are implemented. Based on our analysis described in section 3.2, this results in three basic concepts that are available in the experimental framework:

- Iterative Sampling: Implemented according to figure 3.2. Covers publications mentioned in section 3.2.1, among them EGO and ParEGO.
- Pre-Screening: Implemented according to figure 3.4. Covers publications mentioned in section 3.2.2.
- Lazy Evaluation: Implemented according to figure 3.5. Covers publications mentioned in section 3.2.3, among them GP-DEMO, as well as SAPEO, the algorithm proposed in section 4.2.

Because of the required abstraction, however, some details of the algorithms are not captured fully. This is critical when it comes to functionalities that all algorithms require, such as a strategy to select samples for training the surrogate model. This leads to a conflict of interest, as choosing one of the proposed solutions (e.g. using the best individuals observed as a sample), would unfairly favour the corresponding algorithm (GP-DEMO in this case). However, implementing different sampling strategies for each algorithm again reduces the comparability of the algorithms. A similar problem occurs for common parameters, such as the number of samples selected for training the model. To solve this problem, we implemented all of the proposed solutions within the experimental framework, but without a tie to the original algorithm. This way, all variations can be used independent of which one of the implemented concepts (see list above) is selected. This resulted in the following list of configurable parameters, applicable to all, or a subset, of the concepts.

- Uncertainty Method: Specifies what value is returned when an evaluation is based on the surrogate model. Available options are:
  - Mean: The predicted value by the Kriging model, i.e. the mean prediction.
  - PoI: Probability of improvement, see [37] for more details
  - ExI: Expected improvement, see [37] for more details
  - LB: Lower bound, i.e. the predicted value minus 2 time the predicted standard deviation, see [37] for more details and justification
- Sampling Method: Specifies the method used to select k training samples for the surrogate model
  - fit: Select the k fittest individuals from the archive. For multi-objective problems, these are the k individuals that would be selected from the archive by non-dominated sorting combined with hypervolume contribution as a secondary criterion

  - recent: The k individuals added to the archive most recently. If the algorithms are able to make continuous improvements, this strategy selects a similar sample as "fit", but is a significantly faster to compute.
  - close: The k-nearest individuals in the archive, computed based on which point in search space is to be predicted
  - random: k random points from the archive
  - all: All point in the archive

- Sample Size: Configurable as required. Largest sample size was used for ParEGO, with $11d-1$ samples for the initial design of experiments, where d is the search space dimension.

- Transformation: In some cases, transforming the fitness function can improve the accuracy of the surrogate model
  - None: No transformation done - $t(f(x)) = f(x)$
  - Log: Natural logarithm - $t(f(x)) = \ln(f(x))$
  - Inverse: Inverse function value - $t(f(x)) = \frac{1}{-f(x)}$

- Surrogate Data: Usually, the input for the surrogate model is the location of the point in search space. When a surrogate function is used (see section 4.4.2), this evaluation result of the surrogate function for that point is available as well:
  - Loc: The location of the point in search space (Default case)
  - Surr: The objective value(s) of the point according to the surrogate function
  - Both: Concatenation of both of the above options

- Optimisation Algorithm: We are using three popular algorithms that are implemented in Shark. Other algorithms available in Shark could easily be added.
  - CMA-ES `http://image.diku.dk/shark/doxygen_pages/html/classshark_1_1_c_m_a.html`
  - MO-CMA-ES (with Hypervolume Indicator) `http://image.diku.dk/shark/doxygen_pages/html/classshark_1_1_indicator_based_m_o_c_m_a.html`
  - SMS-EMOA `http://image.diku.dk/shark/doxygen_pages/html/classshark_1_1_s_m_s_e_m_o_a.html`

- Algorithm Budget: Most papers included a fixed budget for evaluations of the evolutionary algorithm. However, as the optimal parameter here has a strong dependency to the problem, we decided to enable online convergence detection (see feature list in section 4.4.1) for all evolutionary algorithms used. This has the added benefit of allowing automatic restarts of the complete algorithm in case of early convergence.

4.4.4 Post-Processing

For the most part, the experimental framework relies on the post-processing capabilities of the COCO framework, which provides statistical tests and plots regarding the performance of the tested algorithms. However, the features added for the experimental framework are naturally not included. We therefore developed scripts that are able to process the information logged from evaluations of Uncertain Functions (see above).

If Uncertain Functions functions are used, there are three types of logs available:

- fevals: Tracks all evaluations of the objective function (x and y values)
- krig_pred: Tracks all evaluations of both the objective function and the Uncertain Function, along with predicted value, predicted uncertainty and cross-validation value
- EAlog: Tracks all individuals present in the population after each iteration, along with additional information on each individual, such as x and y values, predicted value and uncertainty, as well as cross-validation value. It also records the individual's rank according to the algorithm in questions, and whether it is selected for the survival. In addition, it also computes the true rank of the individual within the population based on the correct fitness value, as well as whether it would have been selected based on it.

4.4.4.1 Selection Error Plots

While there are numerous interesting plots that can be generated based on the data collected, we were especially interested on how many correct decisions are made during the runtime of an algorithm using lazy evaluation. This question was investigated before with small-scale simulations in [149], as well as theoretically in section 4.2.3.2. We implemented two types of plots that visualise the collected information on selection errors. The first one is a bar plot that depicts the frequency correct and incorrect selections for a given function (averaged over all instances). In order to be able to put these frequencies into context, the number of rankings that were made based on predictions are added in the last bar as well. An example can be found in figure 4.10 on the left. The second plots the number of iterations against the number of correctly selected individuals. An example plot is included in figure 4.10 on the right. Each black line represents an optimisation run on a separate instance. The additional blue line marks 1 correctly selected individual per iteration, while the red line represents behaviour without any selection errors.

We also implemented an overview plot, which plots the frequencies in the bar plot for each function.

4.4.4.2 SAPEOreader

An additional post-processing feature is targeted at assessing further the behaviour of algorithms that use lazy evaluation approaches based around comparisons under uncertainty (see section 4.2.1), namely GP-DEMO and SAPEO. This is because the

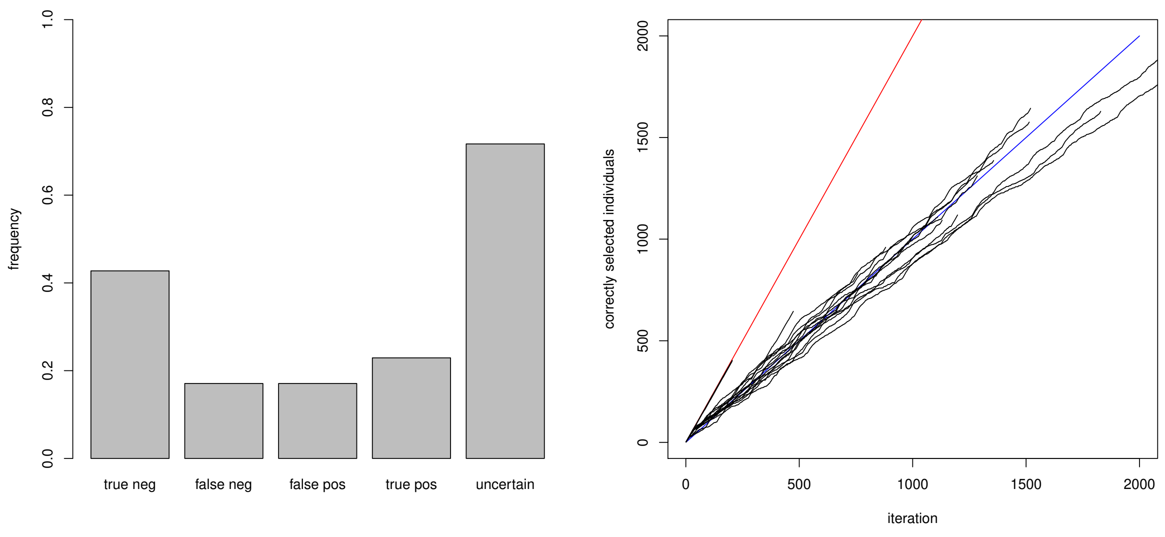


Figure 4.10: Example for selection error plots. Left: Barplot of selection error frequencies. Right: Number of correctly selected individuals over runtime.

report on anytime performance via the COCO framework only considers actual function evaluations. However, both of the algorithms mentioned will delay evaluating very promising solutions for as long as individuals can be selected confidently using the comparisons under uncertainty. At the same time, both algorithms evaluate the final population as their last step before being stopped. Therefore, to accurately record the performance after a given amount of function evaluations, all remaining uncertain individuals would need to be evaluated.

Thus, assume one of these algorithms is stopped after k evaluations and has three remaining uncertain individuals. These evaluations would then be executed, thus recording their values for evaluations $k+1, k+2$ and $k+3$. However, if the algorithm is stopped after $k+1$ evaluations instead, whichever individual is evaluated first in the next iteration would be recorded. Therefore, in this case, there are 2 equally valid values that could be recorded for $k+1$ function evaluations. The number of potential candidates of course is not limited to two, but can be significantly higher depending on the number of individuals in a population that are not evaluated with the actual fitness function.

In order to assess how much this issue affects performance, we implemented another post-processing feature within the experimental framework called SAPEOreader. It records all potential evaluations for each number of function evaluations by *reading* the EAlog files, and then selects the solution with the best fitness value. We then simulate an algorithm run with COCO, where for each evaluation, the previously selected solutions are evaluated. This allows us to capture the performance of an algorithm with lazy evaluation without the issues potentially introduced by the COCO logging approach.

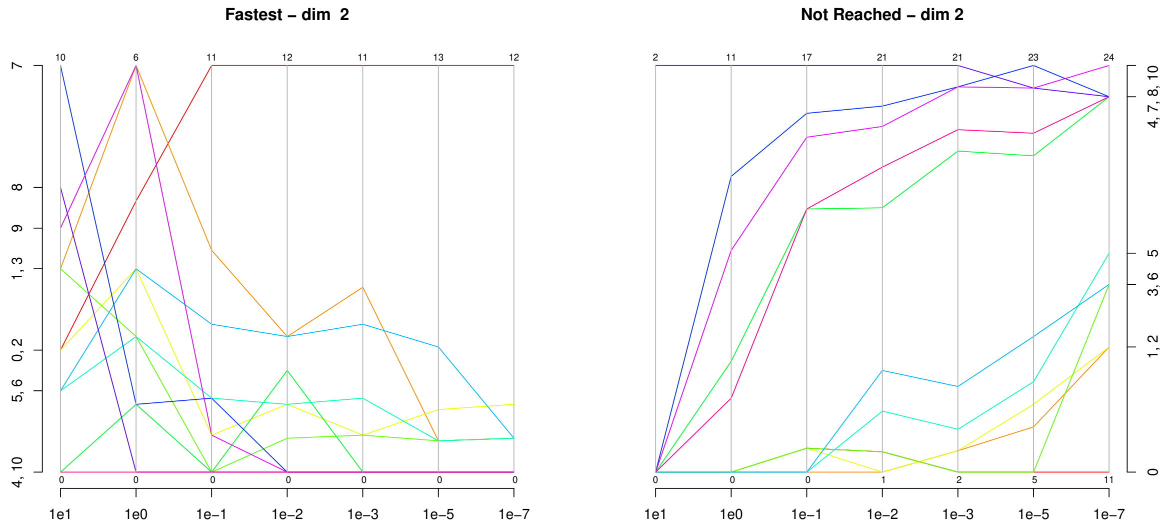


Figure 4.11: Example for aRT table plots. Algorithms are depicted as colour-coded lines. Left: Number of functions where algorithm was fastest to reach target as indicated on x-axis. Right: Number of functions where algorithm did not reach indicated target.

4.4.4.3 aRT Table Plots

In addition to the uncertainty-related post-processing features, we also added visualisations for the statistical tests included in COCO post-processing. This enables a digestible and interpretable representation of all the information produced (cf. section 2.2.2). As mentioned in this section, the tables list the aRT values for all algorithms and all functions and selected targets. This of course allows a more precise assessment of algorithm performance when compared with the ECDF plots.

However, the information does not allow evaluating overall performance over multiple functions. For this reason, we have introduced plots that depict an aggregation of the information recorded in the tables. Two of the most interesting points of information is (1) whether a target was reached at all, and (2), which algorithm was fastest to reach it (i.e. the algorithm with the lowest aRT). We process this information from the tables and depict it with parallel plots (example in figure 4.11). Each coloured each line represents one algorithm recorded with COCO. The plot on the left depicts the number of functions where the algorithm in question was able to reach the respective targets the fastest. The plot on the right visualises the number of functions where the algorithm was not able to reach the respective target at all.

CHAPTER 5

Evaluation

In section 1.2, we developed hypotheses regarding three main topics. In order to investigate all of these claims, we conducted a series of experiments which are described in section 5.1. We present a detailed analysis of the results in the following sections, sorted based on the three hypotheses. The claims translate to the following points of investigation, which are targeted in separate sections as indicated below.

H1 General performance of SAPEO on artificial functions (see section 5.2)

H2 Suitability of the game benchmark GBEA (see section 5.3)

H3 Performance of SAPEO on GBEA (see section 5.4)

5.1 Experiments

We ran several sets of experiments using the experimental framework as described in section 4.4. We ran 5 sets of experiments, where we tested several optimisation algorithms on the following testbeds.

E1 bbob suite (single-objective, see section 2.2.4)

E2 bbob-biobj suite (bi-objective, see section 2.2.5)

E3 rw-mario-gan suite (single-objective, see section 4.3.3.2)

E4 rw-top-trumps-biobj suite (bi-objective, see section 4.3.3.1)

E5 rw-mario-gan-offset suite (single-objective)

The function suites have all been described previously in the sections referenced, except for the last one. This function suite was created using the *Surrogate Function* feature as described in section 4.4.1. This means that, during the optimisation process, additional information is available as computed by a surrogate function. The pairs of objective functions and their respective surrogate functions are listed in table 5.1. Please refer to section 4.3.3.2 and specifically table 4.7 for more details on the functions.

The objective functions in these pairings are all based on AI simulations, as these can be assumed to be more computationally expensive. The first 6 functions all provide information based on an encoding-based surrogate function. Function K, i.e. the progress the AI is able to make, is paired with leniency (E), which is intended as a difficulty

	surrogate function				objective function			
fid	id	description	sample	AI	id	description	sample	AI
19	13	leniency (E)	I	-	19	progress (K)	I	A*
20	14	leniency (E)	II	-	20	progress (K)	II	A*
21	15	leniency (E)	I+III	-	21	progress (K)	I+III	A*
25	16	density (F)	I	-	25	airTime (I)	I	A*
26	17	density (F)	II	-	26	airTime (I)	II	A*
27	18	density (F)	I+III	-	27	airTime (I)	I+III	A*
31	19	progress (G)	I	A*	31	progress (K)	I	Scared
32	20	progress (G)	II	A*	32	progress (K)	II	Scared
33	21	progress (G)	I+III	A*	33	progress (K)	I+III	Scared
34	22	basicFitness (H)	I	A*	34	basicFitness (L)	I	Scared
35	23	basicFitness (H)	II	A*	35	basicFitness (L)	II	Scared
36	24	basicFitness (H)	I+III	A*	36	basicFitness (L)	I+III	Scared
37	25	airTime (I)	I	A*	37	airTime (M)	I	Scared
38	26	airTime (I)	II	A*	38	airTime (M)	II	Scared
39	27	airTime (I)	I+III	A*	39	airTime (M)	I+III	Scared
40	28	timeTaken (J)	I	A*	40	timeTaken (N)	I	Scared
41	29	timeTaken (J)	II	A*	41	timeTaken (N)	II	Scared
42	30	timeTaken (J)	I+III	A*	42	timeTaken (N)	I+III	Scared

Table 5.1: Surrogate and objective function pairings in suite rw-mario-gan-offset. Refer to section 4.3.3.2 and table 4.7 for more details on the functions.

measure. In case of airTime (function I), density (F) is used as a surrogate in order to express the frequency of obstacles in the level.

The remaining functions all correspond to the same fitness measure, but are computed based on a simulation with different AIs. The intention in this case is to imitate the case where information from AI playthroughs is used as a surrogate for human behaviour. Of course, this is not a completely accurate model of the issue, as the objective function is still based on an AI simulation. However, as the behaviour of ScaredAgent is much more human-like than that of the A*, we believe that these experiments will still provide interesting insights.

All of the experiments described in the following sections use some common parameters, which are listed in table 5.2. For each of the experiments, there are also several modifiable settings, which are explained in table 5.3.

The results for all experiments are presented using a series of plots, which have been described previously in this thesis. For more details, please refer to the sections and example figures as indicated below.

- Runtime distribution (ECDF) plots (see section 2.2.2 and example figure 2.2)
- Selection error plots (see section 4.4.4.1 and example figure 4.10)
- aRT table plots (see section 4.4.4.3 and example figure 4.11)

name	value	description	comment
krigSize	$11d-1$	sample size for local Kriging models	largest value observed in related work
histSize	1	k in k-fold cross-validation of model	instead of general fit, we are only trying to estimate this value locally
i	1 000	duplication factor	see section 4.4.1
α	0.05	significance level for confidence intervals	
rs	10 000	granularity for random search algorithms	number of different options in discrete distribution per dimension
cs	16	window considered for OCD (single-objective)	see section 4.4.1
cm	160	window considered for OCD (multi-objective)	see section 4.4.1

Table 5.2: Common parameters for all experiments

name	description
id	unique experiment id for reference
EA	underlying evolutionary algorithms for surrogate-assisted EAs (see section 4.4.3)
ht	cut-off value for model validation (see section 4.2.2)
um	Uncertainty Method: Value returned when evaluation is based on surrogate model (see section 4.4.3)
sm	Sampling Method: Method used to select training samples (see section 4.4.3)

Table 5.3: Modifiable parameters in experiments

The COCO post-processing application performs automatic consistency checks, which remove any questionable results from the generated plots. Incomplete optimisation runs are removed, for example. This feature thus explains why in some rare cases, specific algorithms on specific instances are not depicted in the corresponding generated plots.

5.2 Experiments on Artificial Functions

In the first set of experiments, we investigate the performance of several algorithms on existing benchmarks consisting of artificial functions. We test the single-objective versions as well as the multi-objective ones of these algorithms using the bbob and bbob-biobj suites. An overview of the experiments run can be found in tables 5.4 and 5.5.

Both sets of experiments contain random search in order to obtain a baseline performance (experiment ids 9 and 20). Additionally, each of the evolutionary algorithms selected for the experimental framework are run in their stand-alone versions in order to

id	algorithm	EA	ht	um	sm	note
0	CMA-ES	-	-	-	-	-
1	SAPEO	CMA-ES	3	mean	close	-
2	SAPEO	CMA-ES	5	mean	close	-
3	SAPEO	CMA-ES	3	mean	fit	GP-DEMO
4	SAPEO	CMA-ES	∞	mean	close	without model validation
5	prescreening	CMA-ES	∞	mean	close	-
6	prescreening	CMA-ES	∞	LB	close	-
7	EGO	CMA-ES	∞	PoI	fit	-
8	EGO	CMA-ES	∞	ExI	fit	-
9	RS	-	-	-	-	random search
10	SAPEO	CMA-ES	3	mean	fit	transformation: inverse

Table 5.4: Experiments in set E1 (bbob suite)

id	algorithm	EA	ht	um	sm	note
11	SMS-EMOA	-	-	-	-	-
12	SAPEO	SMS-EMOA	3	mean	close	-
13	SAPEO	SMS-EMOA	5	mean	close	-
14	SAPEO	SMS-EMOA	3	mean	fit	GP-DEMO
15	SAPEO	SMS-EMOA	∞	mean	close	without model validation
16	prescreening	SMS-EMOA	∞	mean	close	-
17	prescreening	SMS-EMOA	∞	LB	close	-
18	EGO	SMS-EMOA	∞	PoI	fit	-
19	EGO	SMS-EMOA	∞	ExI	fit	-
20	RS	-	-	-	-	random search
21	SAPEO	SMS-EMOA	3	mean	fit	transformation: inverse
22	MOCMA	-	-	-	-	-
23	SAPEO	MOCMA	3	mean	close	-
24	SAPEO	MOCMA	5	mean	close	-
25	SAPEO	MOCMA	3	mean	fit	GP-DEMO
26	SAPEO	MOCMA	∞	mean	close	without model validation
27	prescreening	MOCMA	∞	mean	close	-
28	prescreening	MOCMA	∞	LB	close	-
29	EGO	MOCMA	∞	PoI	fit	-
30	EGO	MOCMA	∞	ExI	fit	-
31	SAPEO	MOCMA	3	mean	fit	transformation: inverse

Table 5.5: Experiments in set E2 (bbob-biobj suite)

be able to analyse the effects of adding a surrogate model. This means we ran CMA-ES in set E1 (experiment id 0) and SMS-EMOA and MOCMA in set E2 (experiment ids 11 and 22).

The experiments on the BBOB functions in this case serve as a baseline performance comparison and allow selecting the most viable algorithms for further investigation with the GBEA. We thus also ran 9 other types of experiments on surrogate management strategies for each of the evolutionary algorithms listed above. They are described in more detail in the following list:

1;12;23 Default SAPEO as described in section 4.2

2;13;24 SAPEO, but with more lenient model validation (ht=5 instead of ht=3)

3;14;25 SAPEO, but sampling the k fittest individuals instead of the k closest, as suggested in GP-DEMO 3.2.3

4;15;26 SAPEO, but without any model validation

5;16;27 Default pre-screening as proposed in [37] and described in section 3.2.2

6;17;28 Pre-screening version with lower bounds instead of mean predictions (performed well in [37])

7;18;29 Popular EGO version with probability of improvement instead of expected improvement (see section 3.2.1) and sampling of the k fittest individuals.[1]

8;19,30 Most common EGO version with expected improvement and sampling of the k fittest individuals.

10;21;31 Default SAPEO, but with transformation of fitness functions (see section 4.4.3)

5.2.1 Single-Objective Results (bbob)

In the following, we present a detailed analysis of the performance results on the bbob function suite. We first investigate aggregated performance measures in section 5.2.1.1 and then investigate identified observed patterns in more detail by considering selected functions separately in section 5.2.1.2. Following this, in an attempt to explain the observed behaviour patterns, we perform an in-depth analysis of SAPEO in section 5.2.1.3, using some of the additional features introduced in the experimental framework (see section 4.4).

[1]While this is not true for the original publication of EGO, many popular versions of EGO use global models, i.e. all available samples are used to train the model. This has a noticeable effect on the computational costs, especially with higher budgets. Additionally, it makes the comparison with algorithms that only use local models difficult. As the function evaluations in the benchmarks are relatively cheap, even for the GBEA, using excessively many resources for model computation seemed unreasonable. We thus choose to use local models within EGO as well. However, we opt for relatively large sample sizes as suggested in [73] in order to reflect EGO behaviour as closely as possible.

5.2.1.1 Aggregated overall performance

Runtime distribution plots A summary of all experiments in set E1, i.e. the aggregated aRT from all 24 functions, can be found in figure 5.1. As can be seen in the plots, the stand-alone CMA-ES (experiment id 0) slightly outperforms all other algorithms for higher budgets. However, especially for the lower-dimensional problems, we see other algorithms perform better or on par until approximately 100 function evaluations. This observation is very obvious for both versions of EGO (experiment ids 7 and 8). Based on the bootstrapped values, it seems that the EGO algorithm reaches precision targets comparably fast, but seldom reaches higher targets. However, the performance of random search (experiment 9) is very similar to that of EGO across all tested dimensions. EGO should probably be rerun with full global models in order to assess how the decision to only allow relatively small models for each of the algorithms affected the performance. The SAPEO with fitness transformations (experiment id 10) and the SAPEO without model validation (experiment id 4) also perform consistently worse than the remaining algorithms, but also slightly better than EGO and random search. This suggests that a well-fitted model is crucial for SAPEO.

Both versions of pre-screening (experiment ids 5 and 6) perform similar to each other across all dimensions. For dimension 2, pre-screening reaches slightly more targets after 10 function evaluations than most other algorithms, but still less than random search and both EGO versions. The algorithm starts performing obviously better than EGO, random search and the weaker SAPEO versions mentioned above starting after about 100 function evaluations. At 500 function evaluations, it does however not perform as well as the stronger SAPEO versions (experiment ids 1 – 3) and CMA-ES (experiment id 0). A similar behaviour pattern can be observed for all other dimensions. However, for dimensions 3 and 5, pre-screening seems to have been able to reach a relatively higher number of high-precision targets, resulting in a higher aRT than some SAPEO runs for large budgets.

The default SAPEO version, along with SAPEO with less strict model validation and the SAPEO version inspired by GP-DEMO are consistently performing well across all dimensions, although mostly slightly worse than CMA-ES.

Based on the aggregated results it thus seems that surrogate-assisted algorithms can not really improve the overall performance of CMA-ES, independent on which model management strategy is chosen. For low budgets, where EGO is performing comparably well, random search offers comparable results. SAPEO with verified models is likely performing very similarly to the CMA-ES for small budgets, as at this point, not enough points have been sampled to train a verified model. While pre-screening is not performing model validation, the less fit models after only a few function evaluations will create a less effective bias for selection, which evidently does affect performance. It also has to be noted that none of the algorithms is close to the aggregated best performances observed in 2009. Due to the no free lunch theorem, this observation is not surprising when analysing aggregated results over all functions in the benchmark.

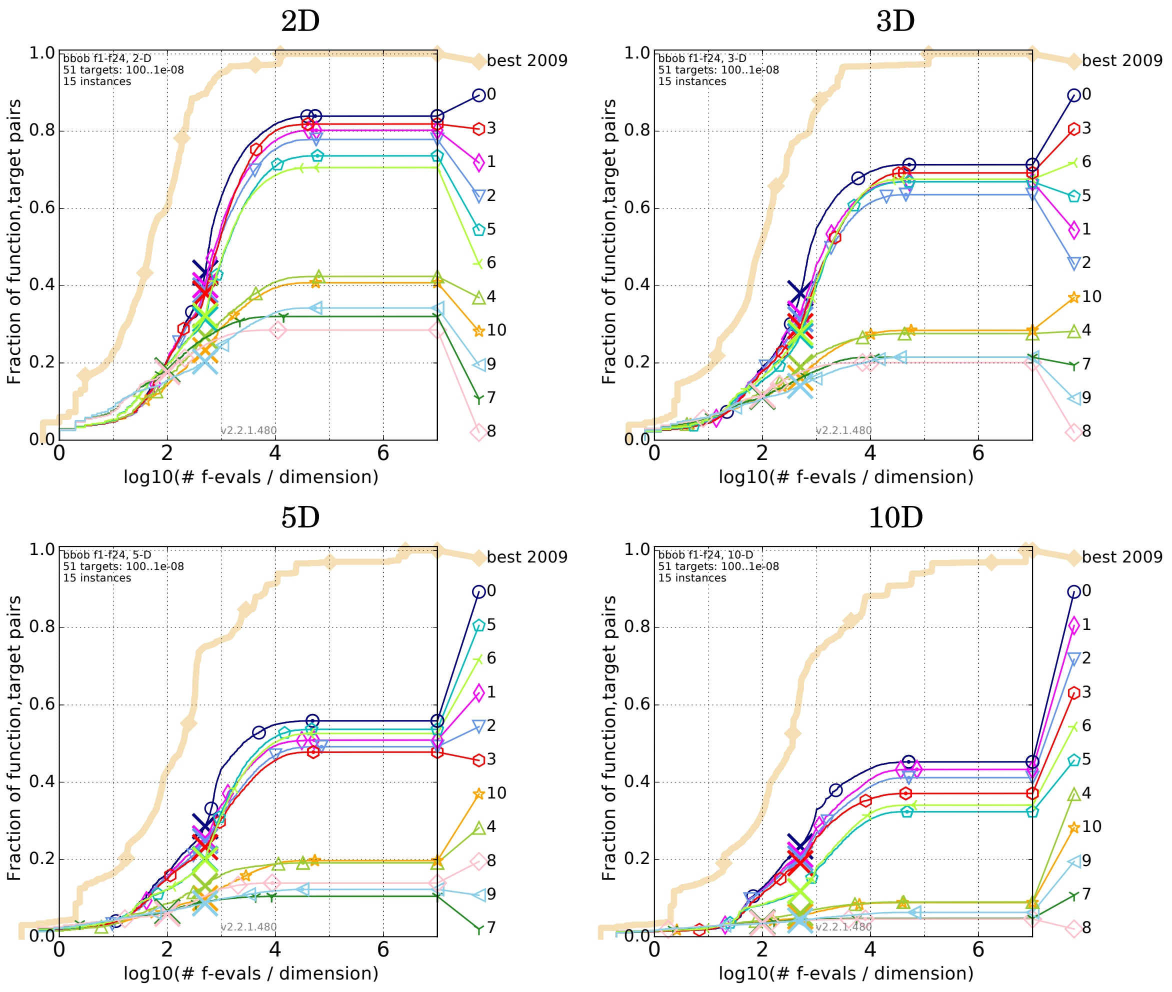


Figure 5.1: E1 runtime distribution plots aggregated over all bbob functions ○ CMA-ES (0), ◇ SAPEO (1), ▽ SAPEO - less validation (2), ⬡ SAPEO - GP-DEMO (3), △ SAPEO - no validation (4), ⬠ pre-screening (5), ≺ pre-screening - lower bound (6), ⋎ EGO - PoI (7), ◇ EGO - ExI (8), ◁ Random Search (9), ⋆ SAPEO - transformation (10)

aRT Table Plots As the previous observations were solely based on the ECDF plots obtained using the COCO post-processing features, in the following, we want to verify whether these observations are also reflected in the aRT values computed by COCO (see section 2.2.2). In order to do that, we use parallel plots as described in section 4.4.4.3. The resulting plots can be found in figure 5.2.

The *Fastest* aRT table plot for dimension 2 depicts for each algorithm for how many functions they were able to reach a given target the fastest. In the plot, we see a similar pattern reflected as described above. The EGO algorithms (experiment ids 7 and 8) reach the lowest target $1e1$ quickest in most of the experiments. We also clearly see the good performance of the CMA-ES (experiment id 0), especially starting from target $1e-1$. CMA-ES reaches target $1e-7$ the quickest for 12 functions, i.e. half the time. However, we also see that pre-screening and some versions of SAPEO do achieve higher targets

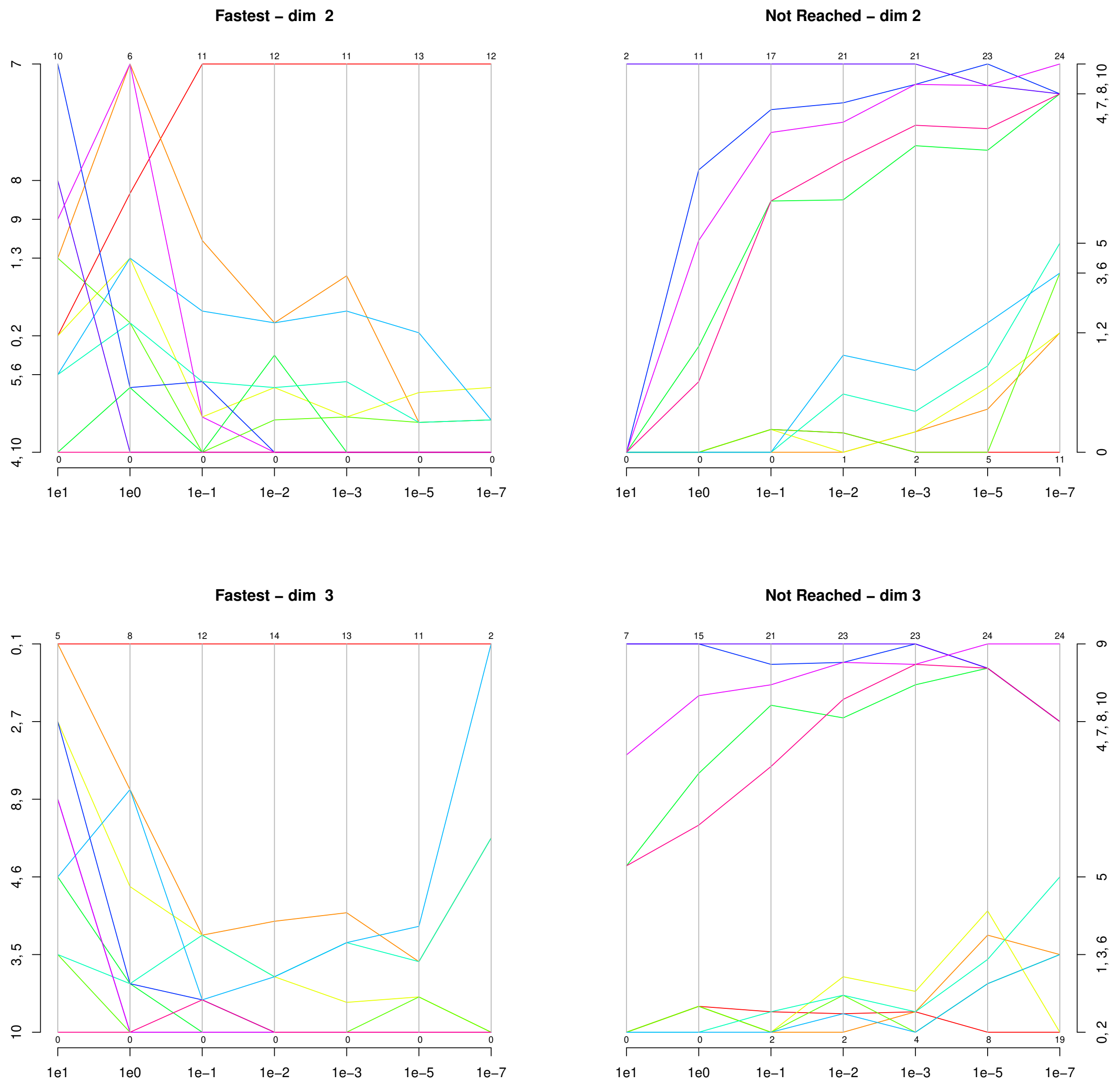


Figure 5.2: aRT table Plots for E1, dimensions 2 and 3. ● CMA-ES (0), ● SAPEO (1), ● SAPEO - less validation (2), ● SAPEO - GP-DEMO (3), ● SAPEO - no validation (4), ● pre-screening (5), ● pre-screening - lower bound (6), ● EGO - PoI (7), ● EGO - ExI (8), ● Random Search (9), ● SAPEO - transformation (10)

the quickest for some functions. Especially interesting is target $1e0$, which is reached the quickest 6 times by random search (experiment id 9), as well as the default SAPEO (experiment id 1).

From the plot on how many times target were not reached, we can also clearly see the lack of robustness of some of the algorithms. As expected, the number of missed targets for random search (experiment id 9) increases rapidly the higher the target. Both EGO algorithms (experiment ids 7 and 8) also rarely reach higher targets, as do the SAPEO with function transformation (experiment id 10) and the SAPEO without model validation (experiment id 4). As assumed based on the ECDF plots, we also observe that the two pre-screening versions (experiments 5 and 6) do not reach higher targets as often as the more successful SAPEO versions (experiment ids $1-3$) and CMA-ES (experiment id 0).

We thus see similar behaviour based on the parallel plots for dimension 2 as described above. For dimension 3 we see comparable behaviour in the aRT table plots with only a few exceptions. Target $1-e7$ is rarely reached by any algorithm, making it seem that the EGO algorithms are suddenly more successful than they are (in the plot on bottom left). After the CMA-ES, the default SAPEO again achieves good performance for lower targets (until $1e-1$), but there is no pronounced spike (cf. plots for dimension 2). Random search also performs worse than in dimension 2, which is expected. Both EGO versions perform better for lower targets, but outperform other functions less often in dimension 3. For dimensions 5 and 10, we observe smaller absolute differences in the aRT values. However, the parallel plots make them appear larger due to solely focusing on which algorithm reached the target first, and additionally because of the rescaling along the y-axis. As this is misleading, we do not include the corresponding plots here.

5.2.1.2 Performance on Selected Functions

In order to further analyse the results, we look at separate functions in the following. We discuss interesting patterns and observations, especially ones that depart from the overall behaviour described in the previous section. Looking at the single function plots, we see the overall trends as described above confirmed in terms of general performance differences. We also see reflected that there are some functions where different algorithms perform best, although the stand-alone CMA-ES (experiment id 0) seems generally most successful. In many functions, as was suggested based on the aggregated runtime distribution plots in figure 5.1, we see two groups of algorithms. EGO (experiment ids 7 and 8), random search (experiment id 9), SAPEO with transformation (experiment id 10) and SAPEO without model validation (experiment id 4) form a group with generally lower performance than the remaining algorithms. The performance within these groups tends to not differ much. An example with slightly larger performance differences can be found in the top row in figure 5.3, which plots the performance on the sphere function in 2D and 5D.

There are, however, some functions where only a small number of algorithms was significantly more successful than the remaining ones. Examples can be found in the second and third row of plots in figure 5.3. The last row shows instances where the

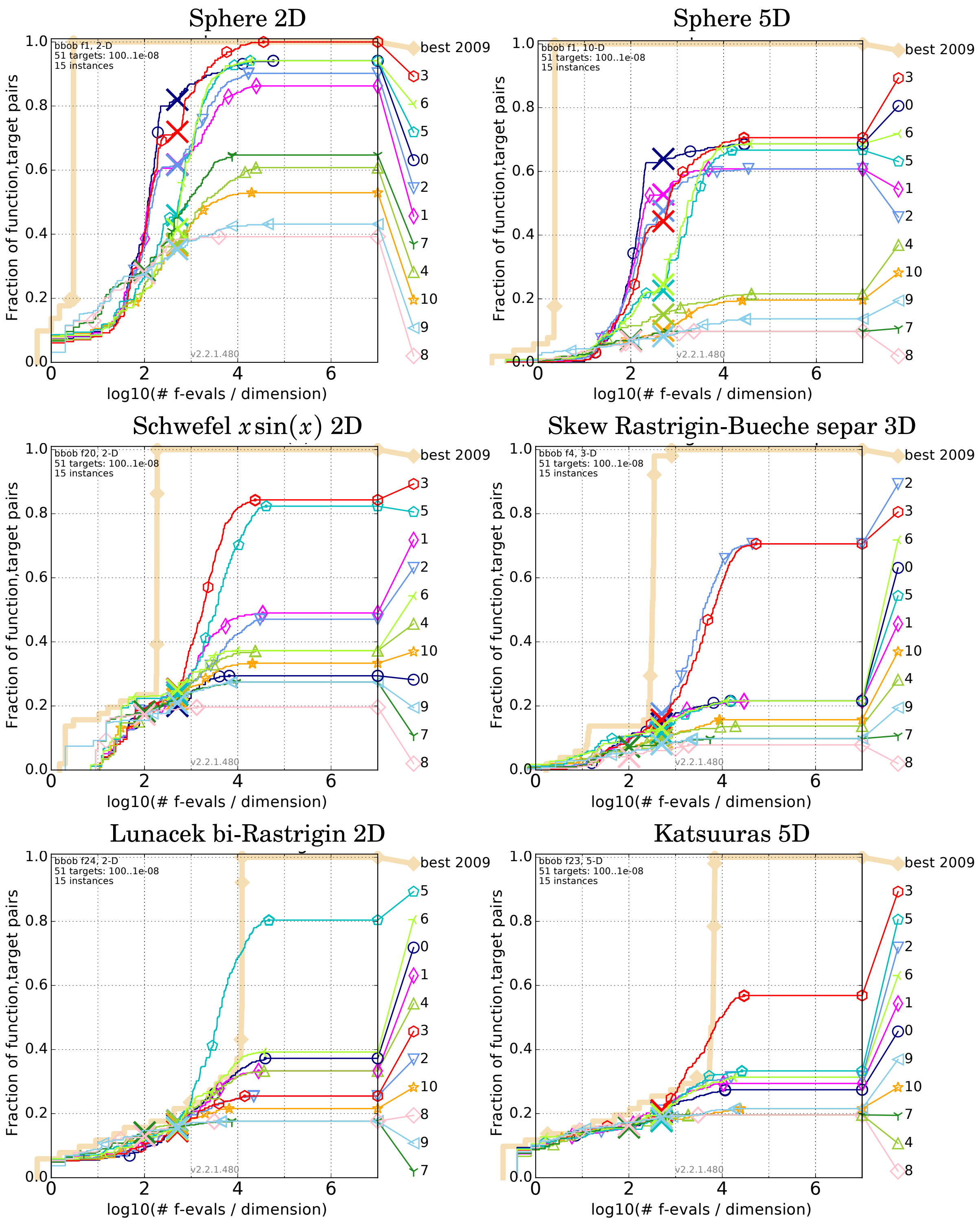


Figure 5.3: E1 runtime distribution plots on selected functions ○ CMA-ES (0), ◇ SAPEO (1), ▽ SAPEO - less validation (2), ⬡ SAPEO - GP-DEMO (3), △ SAPEO - no validation (4), ⬠ pre-screening (5), ≺ pre-screening - lower bound (6), Y EGO - PoI (7), ◇ EGO - ExI (8), ◁ Random Search (9), ★ SAPEO - transformation (10)

best2009 performance was improved notably. No obvious patterns in scaling behaviour were detected.

Unfortunately, there did not seem to be any obvious pattern of performances regarding the different function groups defined in COCO, as described in section 2.2.4. In order to look for potentially existing patterns, we take a closer look at a few functions that are contained in the benchmark in multiple versions. It would be reasonable to assume that functions with more complex landscapes could benefit more from surrogate models in cases where the complexity is due to deceptive characteristics. However, it is also true that highly multi-modal landscapes are difficult to model with local models and small sample sizes. In the following, we thus investigate the influence of some of the characteristics considered in the benchmark, namely separability and condition number (i.e. the sensitivity to small changes).

To investigate the influence of separability, we compare two sets of separable functions (Ellipsoid and Rastrigin) with their non-separable counterparts. See figure 5.4, rows 1 and 2 for the corresponding plots. When comparing the results on the Ellipsoid functions, we see similar overall patterns. However, there are some striking differences. The performance of pre-screening with lower bounds (experiment id 6) drops significantly for the non-separable Ellipsoid and exhibits a similar performance as pre-screening with mean predictions (experiment id 5). This seems to indicate that the uncertainty predictions for the separable Ellipsoid were more reliable and thus rewarded the more optimistic behaviour of selecting for individuals with minimal lower bounds of their fitness predictions. This same behaviour can not be seen when comparing Rastrigin with its separable version, as both pre-screening algorithms consistently perform similarly.

SAPEO performance seems to be better in general for non-separable functions. This might be due to the choice in the Kriging kernel, which does not assume separability. We also observe interesting behaviour patterns when comparing the successful SAPEO (experiment ids $1-3$) versions with each other. The performance of the version with less strict model validation actually increases when the function is not separable for the Ellipsoid function. Meanwhile, the ranking between the default SAPEO (experiment id 1) and the SAPEO inspired by GP-DEMO (experiment id 3) seems to be similar in both versions. For Rastrigin, als three versions of SAPEO exhibit similar performances.

It appears that for the separable Rastrigin function, it pays off to select the samples for Kriging according to fitness instead of locality, as pre-screening (experiment ids 7 and 8), which has fitness-based selection, and SAPEO inspired by GP-DEMO (experiment id 3) perform well. This pattern is less pronounced for the non-separable version of Rastrigin.

To investigate the influence of the condition number, we compare Schaffer F7 with condition 10 to Schaffer F7 with condition 1000, which are both contained in bbob. The results are depicted in figure 5.4, row 3. For the Schaffer F7 function with condition 1000, we see a stark performance drop for many algorithms that use fitness-based sample selection methods, namely pre-screening (experiment ids 7 and 8) and SAPEO inspired by GP-DEMO (experiment id 3). The higher sensitivity resulting from the higher condition number seems require a model with higher accuracy in order to perform well. The performances of the default SAPEO with stricter model validation (experiment id 1)

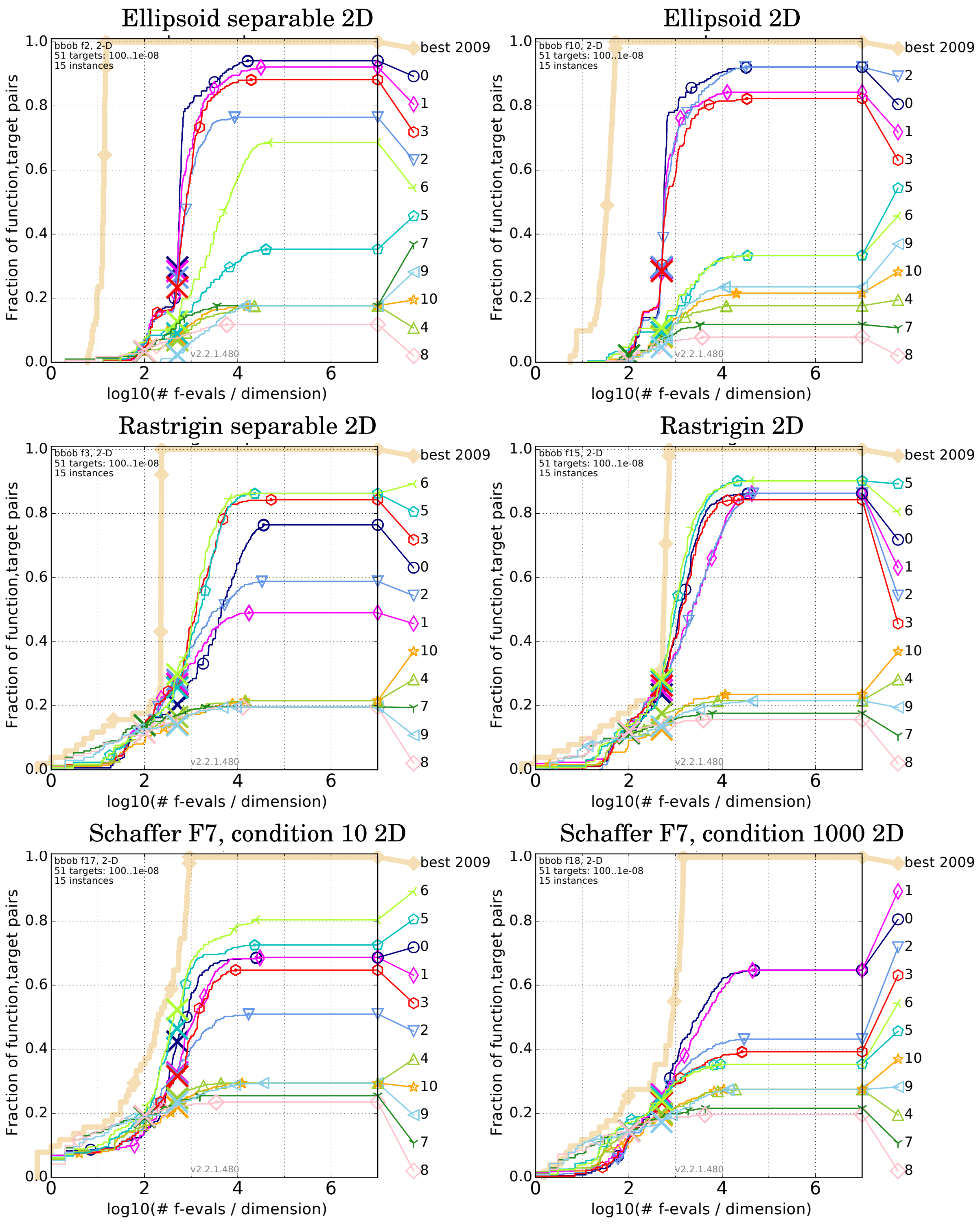


Figure 5.4: E1 runtime distribution plots on selected functions ○ CMA-ES (0), ◇ SAPEO (1), ▽ SAPEO - less validation (2), ⬡ SAPEO - GP-DEMO (3), △ SAPEO - no validation (4), ⬠ pre-screening (5), ⤙ pre-screening - lower bound (6), Y EGO - PoI (7), ◇ EGO - ExI (8), ◁ Random Search (9), ★ SAPEO - transformation (10)

as well as the stand-alone CMA-ES (experiment id 0) are barely affected by the higher condition number, whereas we see another drop for the SAPEO with less strict model validation (experiment id 2). However, this algorithm did also perform significantly worse for the Schaffer F7 with lower condition number.

5.2.1.3 Analysis of SAPEO Variants

In order to be able to assess the success of SAPEO as proposed in this thesis (see section 4.2), and especially regarding the modifications made when compared to previous publications in [145, 149], we compare different SAPEO variants in the following.

SAPEOreader However, before we conduct a more in-depth analysis, we check how influential the COCO logging behaviour is on the performance of SAPEO. In other words, would SAPEO performance be improved if the COCO framework allowed recommending solutions that would capture the best predicted solution, instead of only recording evaluated solutions. In order to do this, we use the SAPEOreader feature as described in section 4.4.4.2. The resulting simulated runs are named like the original experiments, but with an "a" appended to the experiment id.

For most functions, there are no or only minor differences between the original data collected from SAPEO and the version based on the recommendations. A depiction of the most common behaviour can be found in figure 5.5 in the top left plot. However, there are also a number of functions where there are considerable differences between potential performance and the performance that was logged. For examples, see the last three plots in figure 5.5. Especially the default SAPEO seems to be affected by the lack of recommendation ability. The observed differences occur mostly in problems with lower dimensional search spaces, which might be because the SAPEO is more likely to fall back to the underlying EA behaviour if the Kriging predictions are less certain. This case would be more likely in higher dimensions. Other patterns were not observed.

While these results are certainly interesting and could be an argument in favour of implementing recommendation capabilities in COCO, it does not suggest that the aRT values as computed by COCO are misleading in general. As the differences between the SAPEOreader and original SAPEO algorithms are mostly minor, we consider the experiment results reliable nevertheless.

Selection Errors Next, we investigate the behaviour of the default SAPEO version, which was approached from a theoretical angle in section 4.2.1 in more detail. This is intended, on the one hand, to put the theoretical results into context. On the other hand, we seek to explain why SAPEO is outperformed by the underlying CMA-ES for many functions. This is insofar remarkable as SAPEO should make only a very small number of ranking errors if the model predictions can indeed be trusted. For our analysis, we therefore use the default SAPEO which has the strictest model validation approach.

We first plot an overview of the frequencies of selection errors, which are depicted as solid lines in figure 5.6 and colour coded by search space dimension as indicated in the legend. All functions available in bbob are listed with their id on the x-axis. The

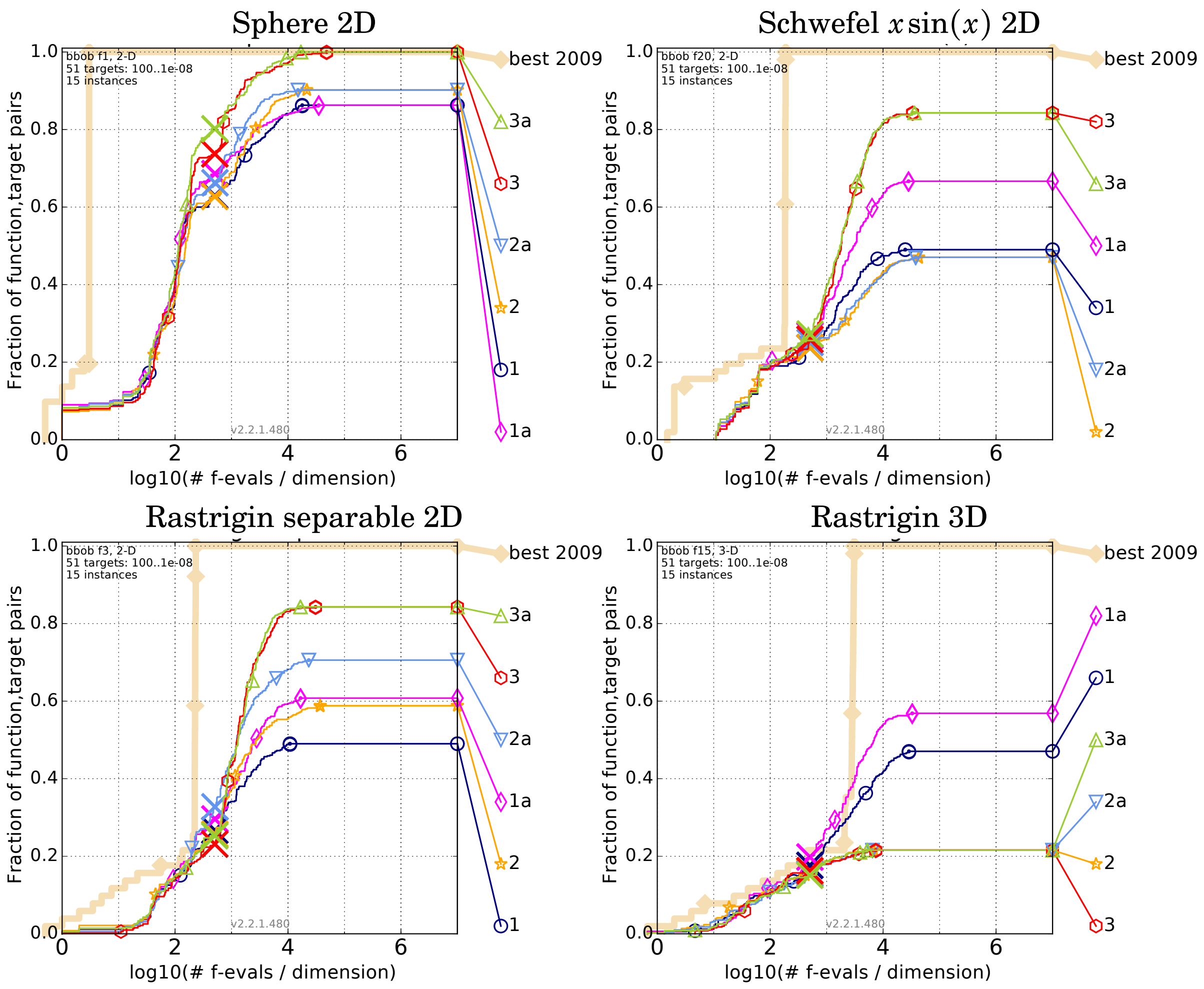


Figure 5.5: E1 runtime distribution plots with recommendations ○ SAPEO (1), ◇ SAPEOreader (1a), ⋆ SAPEO - less validation (2), ▽ SAPEOreader - less validation (2a), ⬡ SAPEO - GP-DEMO (3), △ SAPEOreader - GP-DEMO (3a)

dashed lines in the plot represent the percentage of rankings made based on predicted values for each of the functions. This value is added for context in order to determine how successful the comparisons under uncertainty were.

A first observation from the figure is that, while both the percentage of incorrect selections and the percentage of rankings under uncertainty vary depending on the function, there seems to be only minor variations between different search space dimensions. Only the numbers for dimension 10 seem to vary more, but this could be explained by the fact that for dimension 10 both λ and μ as determined by CMA-ES are different from the remaining dimensions.

As could be expected, in functions where only few rankings are computed under uncertainty, the resulting number of incorrect selections is also low. For some functions (2 - Ellipsoid separable, 5 - Linear slope, 19 - Griewank-Rosenbrock F8F2), it seems that barely any rankings are based on predicted values. In these cases, the SAPEO

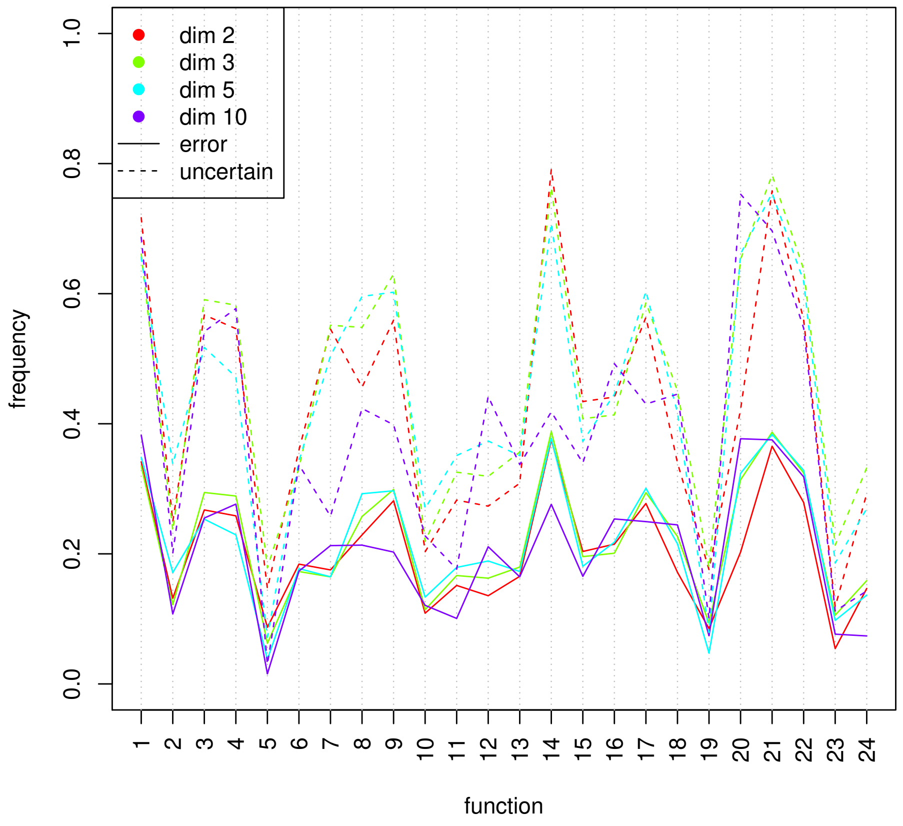


Figure 5.6: Selection errors for default SAPEO (experiment id 1) on E1

performance should very closely resemble that of the CMA-ES. This behaviour can be considered a fallback for functions where comparisons cannot be made under uncertainty, because either the confidence intervals overlap, or because the model is not considered valid. For an ECDF plot, see the top left plot in figure 5.4.

However, the default SAPEO performs similar to the CMA-ES in many more than the aforementioned functions. Examples are both Schaffer F7 functions (function ids 17 and 18, see bottom row in figure 5.4). This is despite the fact that around 60% (50%) of the rankings are based on predicted values for function 17 (18). For both of these functions, we observe a medium amount of ranking errors around 20%. Of course, the impact of the errors also depends on the specific individuals in question, and on whether their selection still results in an overall improvement of the populations. For more details and an analysis of the impact of selection errors on the evolution path of the CMA-ES, see [149].

Functions 1 (Sphere, see figure 5.7, top left), 14 (Sum of different powers, see figure 5.7, bottom left) and 21 (Gallagher 101 peaks) see particularly frequent uncertain rankings according to the plot. We also see a relatively high frequency of selection errors in these functions. Interestingly, this still results in very similar behaviour of the CMA-ES and SAPEO on function 21. In both other functions, this results in a detriment to the performance of SAPEO.

In order to investigate this correlation further, we analyse functions 1 and 14 in more detail in the following, using separate plots of ranking errors as described in section 4.4.4.1. To contrast the observations, we conduct the same experiments for function 16 (Weierstrass), where the default SAPEO shows a particularly good performance after around 1000 function evaluations, especially when compared to the CMA-ES. In figure 5.7 we thus plot the runtime distribution plots of the aforementioned functions next to graphs depicting the number of correctly selected individuals over the runtime of the algorithm. The exact frequencies of selection errors and uncertain decisions (as depicted in figure 5.6) are indicated above the plots in the second column.

From the selection error plots we can see that for all functions, there are instances (black lines, detailed description of the plot in section 4.4.4.1) where SAPEO only selects correct individuals in almost all iterations, i.e. that follow the red line. These are probably instances where the predictions are only very rarely used for the comparisons. From the comparison with the blue line, which marks 1 correctly selected individual per iteration, we can see that there are instances for all functions where the number of correctly selected individuals averages less than 1 per instance. However, we see decidedly more instances to the left of the blue line for the Weierstrass function in the middle row, when compared to the sphere function in the top row and the sum of different powers in the bottom row. Interestingly, even for the Weierstrass function where SAPEO performed remarkably well, the average number of correctly selected individuals seems to be closer to 1 than to 2 (red line, 2 individuals are selected by CMA-ES 2-dimensional search spaces).

Additionally, from the plots it seems that on instances where SAPEO made fewer selection errors (i.e. left of the blue line) the algorithm stopped earlier than on instances where the number of selection errors was larger (right of blue line). In addition, the runs on the Weierstrass function were stopped earlier than on the other functions. SAPEO stops either when the final precision target ($1e-8$) was hit for a given instance, or because the algorithm detected convergence. According to the tables generated by COCO (see section 2.2.2), SAPEO was not able to hit the final target for any of the functions in any of the instances. That means that the observed behaviour is entirely explained by convergence. This observation might be used to compute an indicator based on convergence rate that determines whether the number of decision based on predictions should be reduced in order to improve the performance of SAPEO.

Another interesting observation is that SAPEO outperforms CMA-ES on the Weierstrass function, even though the number of decisions that rely on the prediction model and where function evaluations are thus avoided, is relatively low. This might indicate that the predictions of the surrogate model should be used more selectively as already done in the default SAPEO. We investigate this deliberation more in the following by analysing how the model validation approach used within SAPEO affects performance.

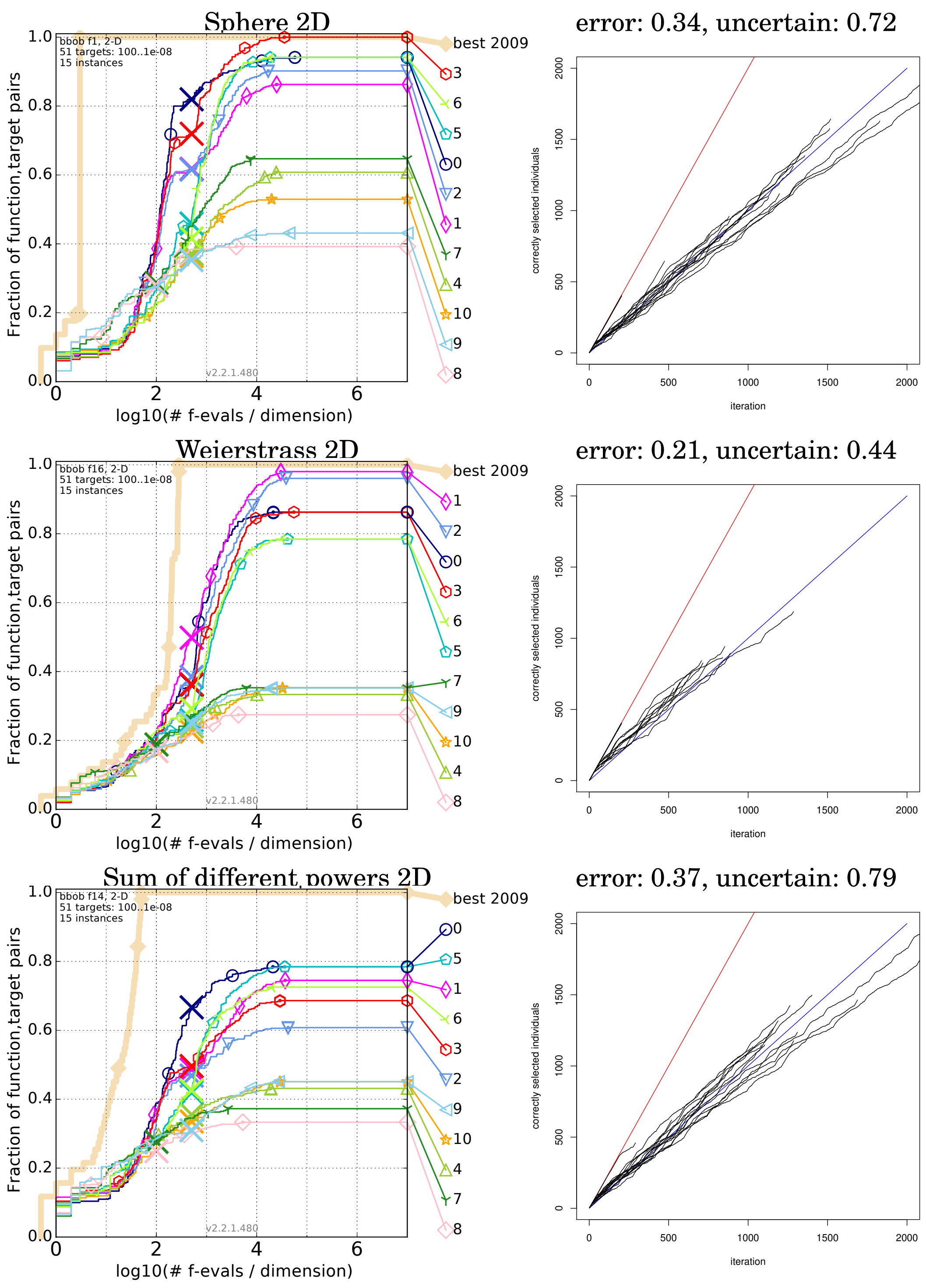


Figure 5.7: Left: Runtime distribution plots for E1 and dimension 2. Right: Selection error plots for default SAPEO (experiment id 1) and dimension 2. Top: Sphere (fid 1). Middle: Weierstrass (fid 16). Bottom: Sum of different powers (fid 14). ○ CMA-ES (0), ◇ SAPEO (1), ▽ SAPEO - less validation (2), ⬡ SAPEO - GP-DEMO (3), △ SAPEO - no validation (4), ⬠ pre-screening (5), ≺ pre-screening - lower bound (6), ⋎ EGO - PoI (7), ◇ EGO - ExI (8), ◁ Random Search (9), ⋆ SAPEO - transformation (10)

Model Validation In order to investigate the effects of the model validation approach as proposed in section 4.2.2, we included three different versions of SAPEO, the default one (experiment id 1), one with more lenient model validation (ht=5 instead of ht=3, see section 5.2) and one without any model validation (experiment id 4). See list of experiments in table 5.4 for details. It is important to note here that none of the other algorithms surveyed for the related work use model validation, except for the original publication of EGO. As pre-screening and EGO both do not actually introduce the predicted values into the algorithm, using a model with bad fit likely does not affect their performance as much as it would for SAPEO or GP-DEMO. We seek to verify this hypothesis in the following.

From the results presented in section 5.2.1.1 and 5.2.1.2, we can already clearly see that for SAPEO, using models without additional validations hinders the performance of the underlying CMA-ES significantly. The corresponding SAPEO variant (experiment id 4) is consistently in the weaker group of algorithms together with random search, whereas both of the other SAPEO versions belong to the top group most of the time. Some plots showing a direct comparison of the three aforementioned SAPEO versions can be found in figure 5.8.

In our post-processing results, we see the same pattern of overall performance regardless of dimension. The default SAPEO with stricter model validation reaches targets faster than both other variants for more functions (see figure 5.8, right column). However, the aggregated ECDF plots in figure 5.8 (left column) also show that the resulting absolute differences in terms of aggregated aRT values are only minor.

While there are some functions where the SAPEO without model validation performs better than both other variants (only for lower budgets, see figure 5.9), it seems worthwhile to validate the model in order to achieve a robust performance. This is true even though there are still numerous selection errors made if the model is validated, as discussed in the previous paragraph on selection errors. In order to obtain a more detailed picture on the effect of stricter model validation on the number of selection errors, we generated a side-by-side comparison of selection errors for the default SAPEO (experiment id 1) and SAPEO with weaker validation (experiment id 2).

The corresponding plots can be found in figure 5.10. While we can clearly see that both SAPEO variants react similarly to the different functions in terms of the frequency of using rankings under uncertainty, the SAPEO variant with less model validation does so more often. The corresponding numbers of selection errors are also slightly higher when compared to the default SAPEO. This leads to the conclusion that model validation is indeed important and might need to be even stricter than using a cut-off value of 3 for the cross-validation, as suggested in literature.

Sample Selection Method Another interesting question is how selecting the samples for the local Kriging models affects the performance of SAPEO. Obviously, increasing the sample size reduces the uncertainty of the model and also often results in better performance [149]. However, for the experiments in the thesis, we decided to investigate the method of sampling instead. In GP-DEMO, the k fittest individuals are selected

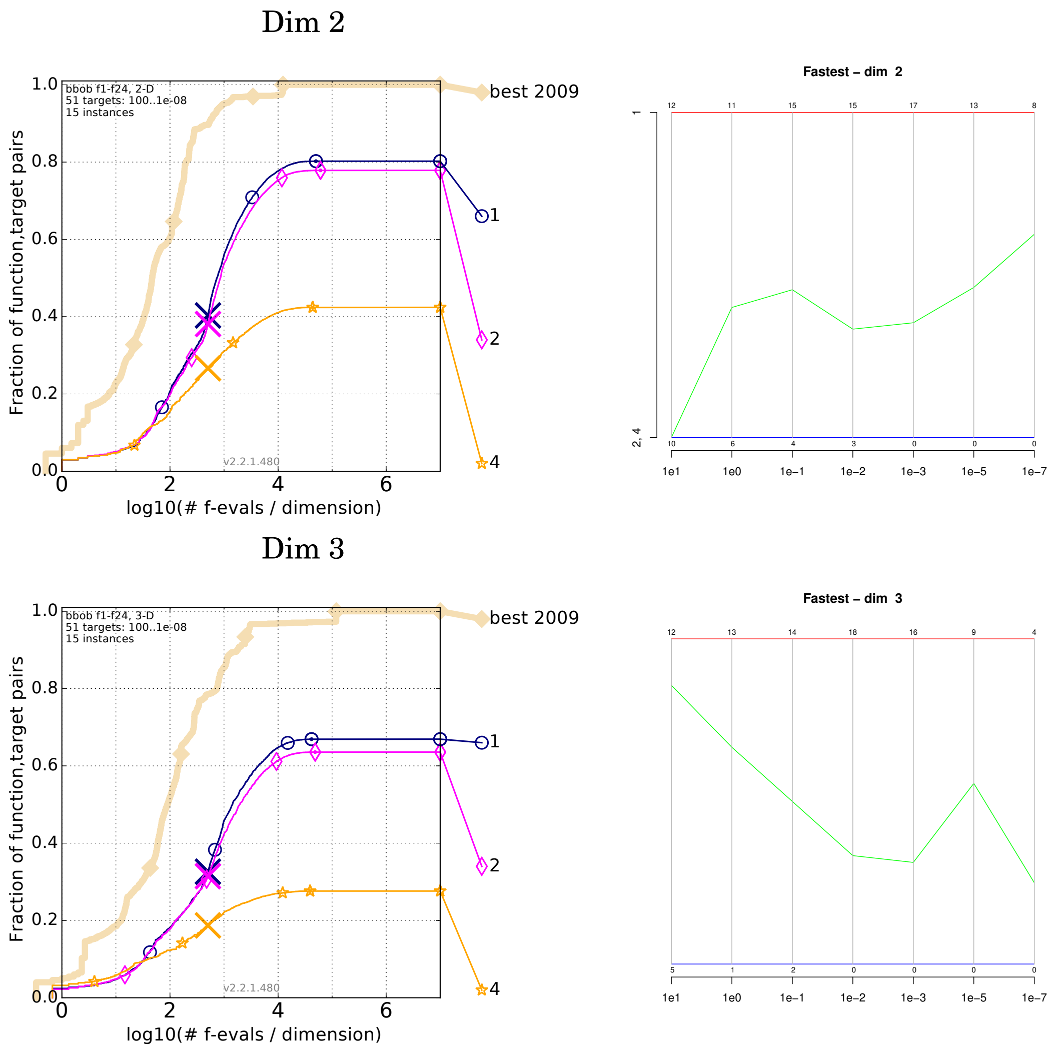


Figure 5.8: Left: Runtime distribution plots for E1, dimensions 2 and 3, aggregated over all functions ○ SAPEO (1), ◇ SAPEO - less validation (2), ☆ SAPEO - no validation (4). Right: aRT Table plots for E1, dimensions 2 and 3. ● SAPEO (1), ● SAPEO - less validation (2), ● SAPEO - no validation (4).

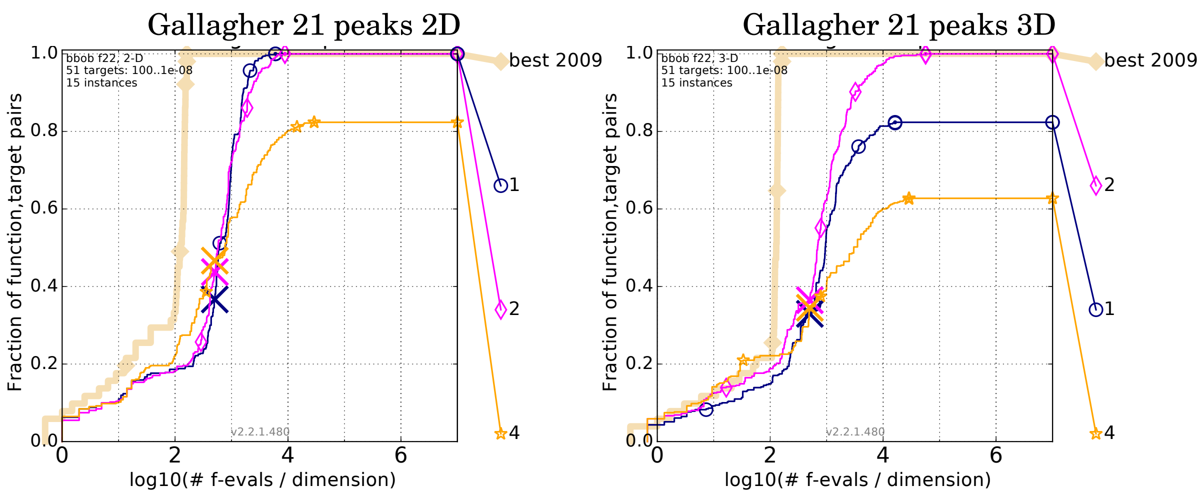


Figure 5.9: Runtime distribution plots for Gallagher 21 peaks function (fid 22) in E1, dimensions 2 (left) and 3 (right) ○ SAPEO (1), ◇ SAPEO - less validation (2), ★ SAPEO - no validation (4).

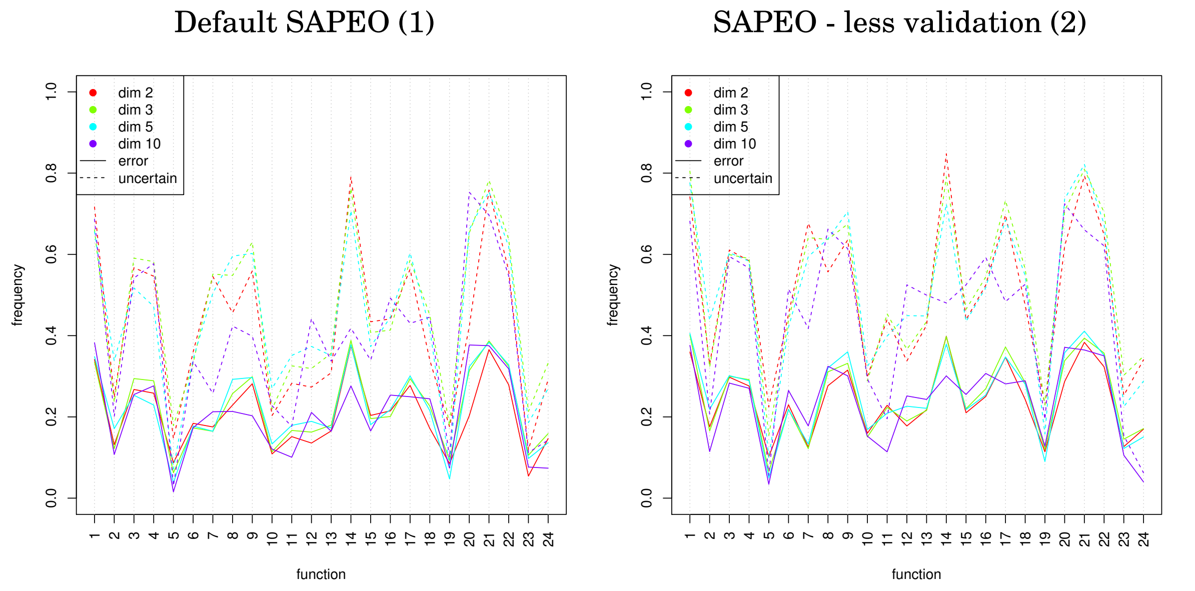


Figure 5.10: Selection errors on E1. Left: Default SAPEO (experiment id 1). Right: SAPEO with weaker model validation (experiment id 2)

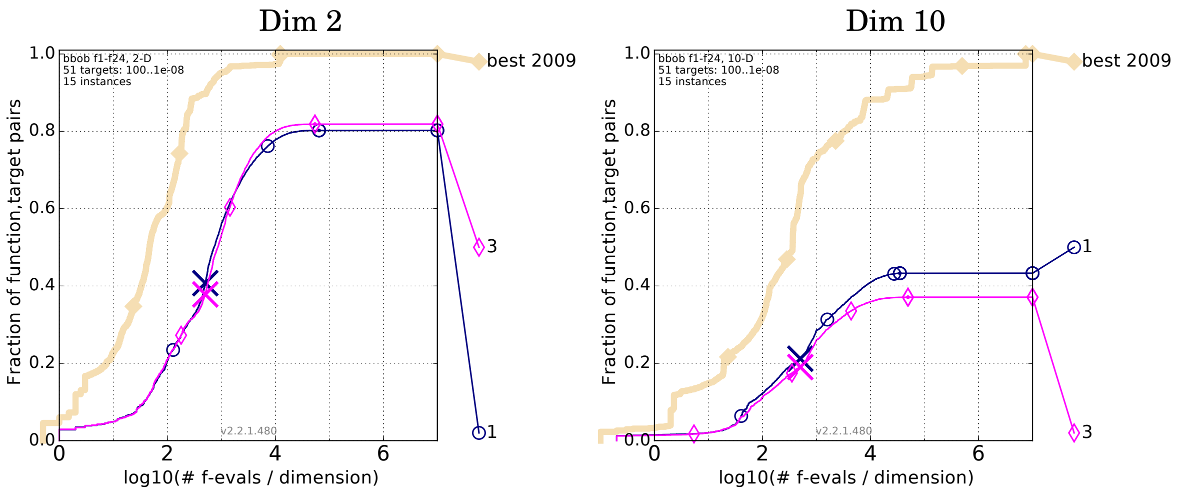


Figure 5.11: Runtime distribution plots for E1, dimensions 2 and 10, aggregated over all functions ○ SAPEO (1), ◇ SAPEO - GP-DEMO (3)

instead of the k closest ones.

Based on the runtime distribution plots in figure 5.11, we see only very slight performance differences. In terms of aggregated aRT values, default SAPEO is performing slightly better on lower budgets, but is overtaken by SAPEO with fitness-based selection at around 10^3 function evaluations on dimensions 2 and 3. The default SAPEO has a consistently better overall performance on dimensions 5 and 10.

The ECDF plots suggest very similar behaviour, even when looking at functions separately. However, this observation is not at all true, which we see in the selection error plots in figure 5.12. The SAPEO variant with fitness-based selection for sampling relies on predictions far less than the default SAPEO variant. This is most likely due to the fact that with fitness-based sampling, the predicted uncertainties for any given point are likely higher than those predicted by a local model. The resulting overlap of the confidence intervals then forces evaluation.

As comparisons under uncertainty are used less often in SAPEO - GP-DEMO, we also tend to see less selection errors overall (with a few exceptions). However, this SAPEO variant is also not able to save as many fitness evaluations as the default SAPEO, as confidence intervals are overlapping more often. These two aspects seem to even out in terms of overall aRT values, resulting in the behaviour observed in figure 5.11. However, based on the analysis above, it seems that a similar effect in terms of a reduction of selection errors could also be achieved by improving the models by either increasing sample size or by introducing stricter model validation. Using fitness-based selection does not seem to produce the intended effect of better predictions in relevant regions.

5.2.2 Multi-Objective Results (bbob-biobj)

A summary of all experiments in set E2, i.e. the aggregated aRT from all 55 functions, can be found in figures 5.13 and 5.15. The former depicts all algorithms that are based around

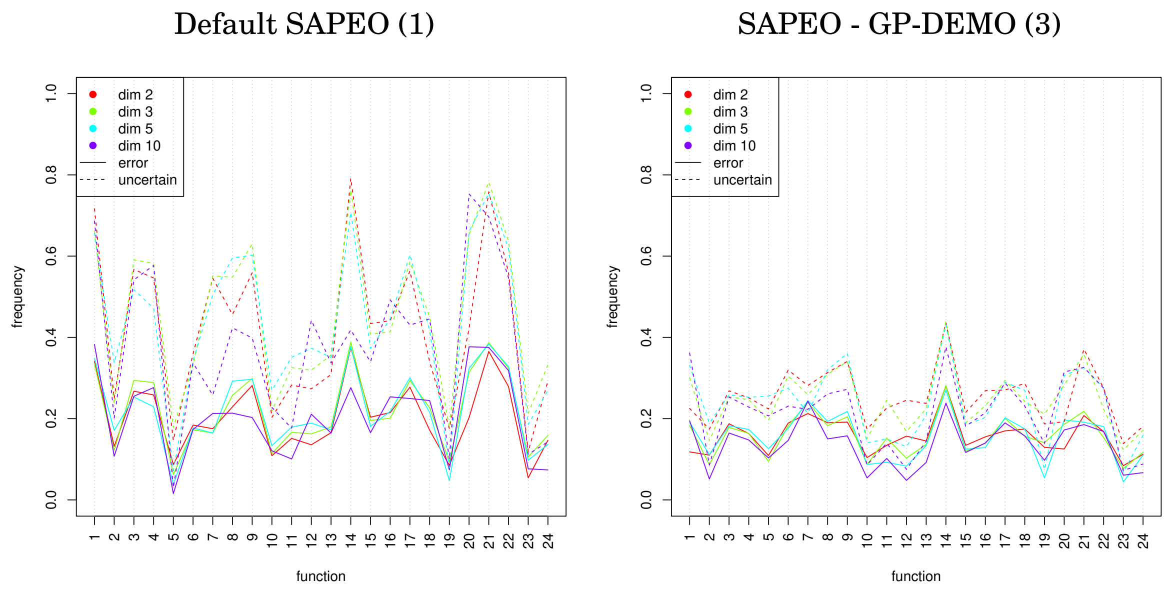


Figure 5.12: Selection errors on E1. Left: Default SAPEO (experiment id 1). Right: SAPEO with fitness-based selection (experiment id 3)

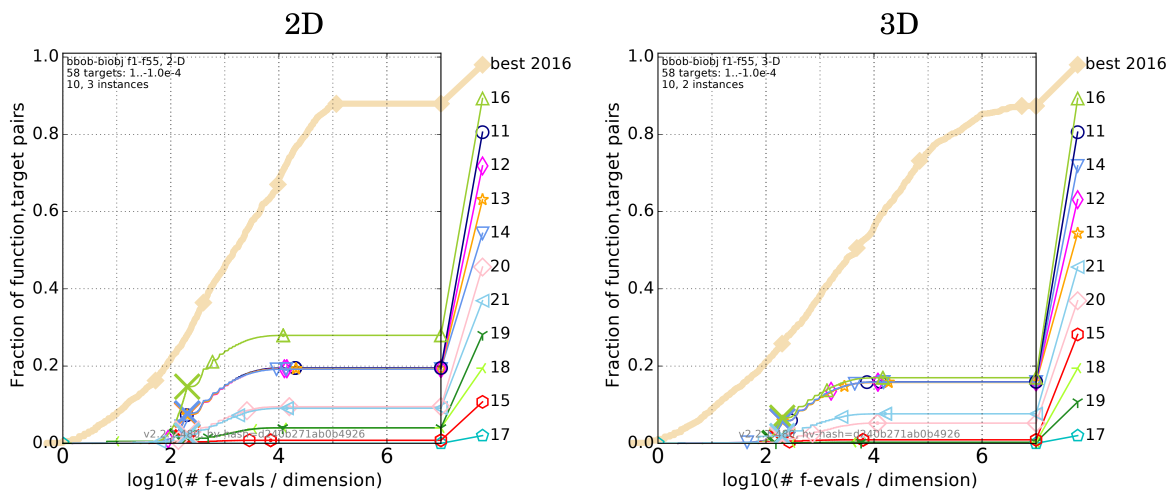


Figure 5.13: E2 runtime distribution plots aggregated over all bbob-biobj functions with algorithms based on SMS-EMOA. Dimensions 2 (left) and 3 (right) ○ SMS-EMOA (11), ◇ SAPEO (12), ★ SAPEO - less validation (13), ▽ SAPEO - GP-DEMO (14), ⬡ SAPEO - no validation (15), △ pre-screening (16), ⬠ pre-screening - lower bound (17), ≺ EGO - PoI (18), Y EGO - ExI (19), ◇ Random Search (20), ◁ SAPEO - transformation (21)

the SMS-EMOA, while the latter depicts the algorithms using the MOCMA. As can be seen in the plots (figure 5.13, left), pre-screening with the SMS-EMOA (experiment id 16) outperforms all other algorithms for dimension 2. This observation is also reflected when considering functions separately. For larger dimensions, the difference in performance is less pronounced, but still existent (see figure 5.13, right). In a stark contrast, pre-screening with lower bounds (experiment id 17) is the worst-performing algorithm in this set. As in the single-objective version, both EGO versions (experiment ids 18 and 19) as well as the SAPEO with transformation (experiment id 21) do not perform well. They produce aRT values similar (or even worse) than random search (experiment id 20).

However, all SAPEO variants and the SMS-EMOA exhibit extremely similar performances. As this is true even for the SAPEO without any model validation (experiment id 15), this seems to suggest that comparisons on predicted values cannot be used, because there is always an overlap in the predicted confidence intervals. However, when looking at the functions, we see that this patterns is actually not always true. But it is nearly always the case that the successful SAPEO variants with model validation achieve the same or similar aRT values. As is confirmed in the logs, this is because the models can be validated only very rarely. The result is that SAPEO falls back onto the underlying SMS-EMOA and thus shows the same performance. This is insofar a positive, as SAPEO is not misled by an unreliable model. However, this obviously also does not result in any significant improvements of the performance.

In comparison to the *best2016* benchmark, overall performance of all algorithms seems not to be up to par. There are a few functions where relatively good values are reached, though. For examples, see the first row in figure 5.14, as well as the plot in the middle row on the right.

There are only a few functions where the patterns described above are not true. In a few cases, both EGO versions perform comparably well (see figure 5.14, middle row, left). For some functions, the successful SAPEO variants along with the SMS-EMOA outperform pre-screening or perform at least on par (see figure 5.14, bottom row).

Overall, the experiments based on the SMS-EMOA perform better than the ones based on the MOCMA (see comparison between figures 5.13 and 5.15). Interestingly, it seems that pre-screening, which worked well in conjunction with both the CMA-ES and the SMS-EMOA, has a very weak performance when paired with the MOCMA (experiment ids 27 and 28). Instead, EGO with expected improvement (experiment id 30) performs best on 2-dimensional problems overall, and the SAPEO with transformation (experiment id 31) is successful for 3D problems. However, due to the generally low performance, we are not going to look into the results in more detail, as they likely do not offer meaningful interpretations.

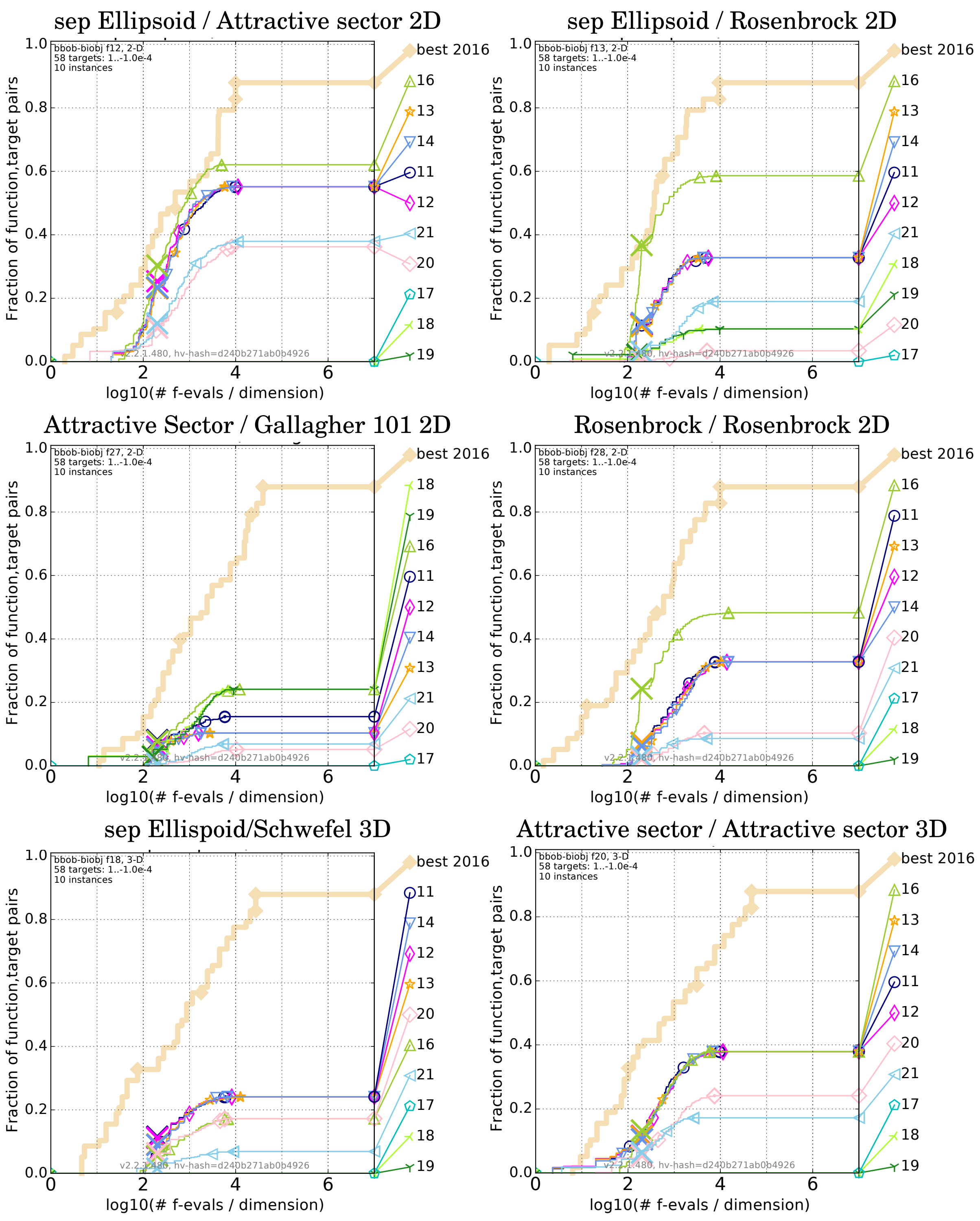
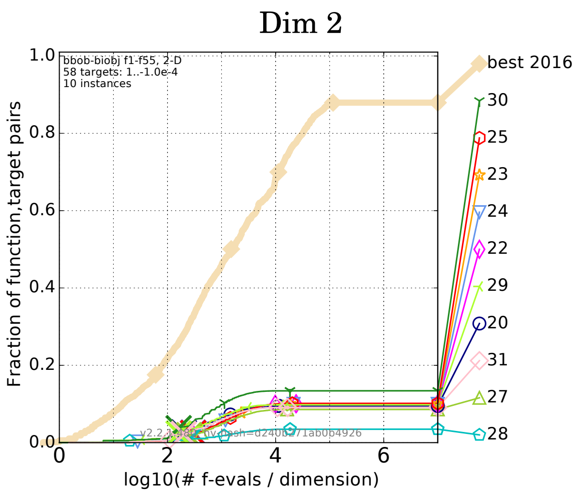


Figure 5.14: E2 runtime distribution plots for algorithms based on SMS-EMOA on selected functions ○ SMS-EMOA (11), ◇ SAPEO (12), ⋆ SAPEO - less validation (13), ▽ SAPEO - GP-DEMO (14), ⬡ SAPEO - no validation (15), △ pre-screening (16), ⬠ pre-screening - lower bound (17), ≺ EGO - PoI (18), Y EGO - ExI (19), ◇ Random Search (20), ◁ SAPEO - transformation (21)



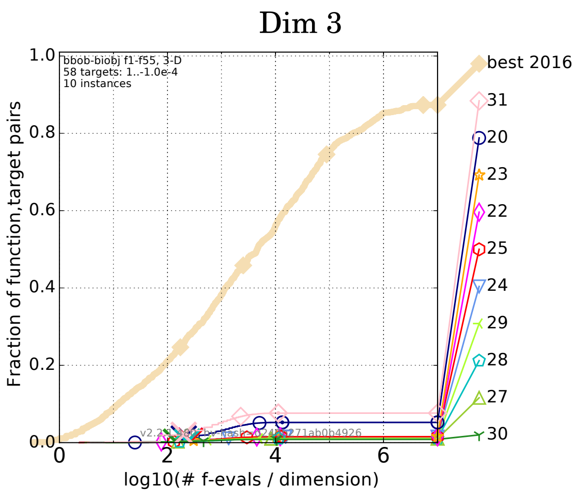


Figure 5.15: E2 runtime distribution plots aggregated over all bbob-biobj functions with algorithms based on MOCMA. Dimensions 2 (left) and 3 (right) ◇ MOCMA (22), ☆ SAPEO (23), ▽ SAPEO - less validation (24), ⬡ SAPEO - GP-DEMO (25), △ pre-screening (27), ⬠ pre-screening - lower bound (28), ≺ EGO - PoI (29), ⋎ EGO - ExI (30), ○ Random Search (20), ◇ SAPEO - transformation (31)

5.2.3 Summary of Results

In the following, we summarise the results presented previously in this section, which are based on results from experiment suites E1 and E2, i.e. benchmarks with artificial functions. We make the following observations:

- While there are functions where an improvement can be achieved using surrogate-assisted algorithms, the underlying evolutionary algorithms tend to perform on par or even better. This is especially true for the single-objective versions and for higher budgets.
- While performance is lower for higher dimensional problems overall, the Kriging-based algorithms were able to handle 10 dimensional problems. Kriging is thus suitable for more than just very low-dimensional problems.
- SAPEO and pre-screening tend to perform best, while EGO does not work well in our implementation.
- Model validation is required for SAPEO and improves performance. An even stricter validation could potentially improve performance further.
- Fitness-based selection in SAPEO as suggested in GP-DEMO is not particularly successful.
- For SAPEO, algorithm convergence and the number of selection errors might be correlated.

Regarding hypothesis H1 from section 1.2, we can thus confirm that SAPEO does exhibit comparable performance to state-of-the-art optimisation algorithms on established benchmarks.

5.3 Suitability of GBEA

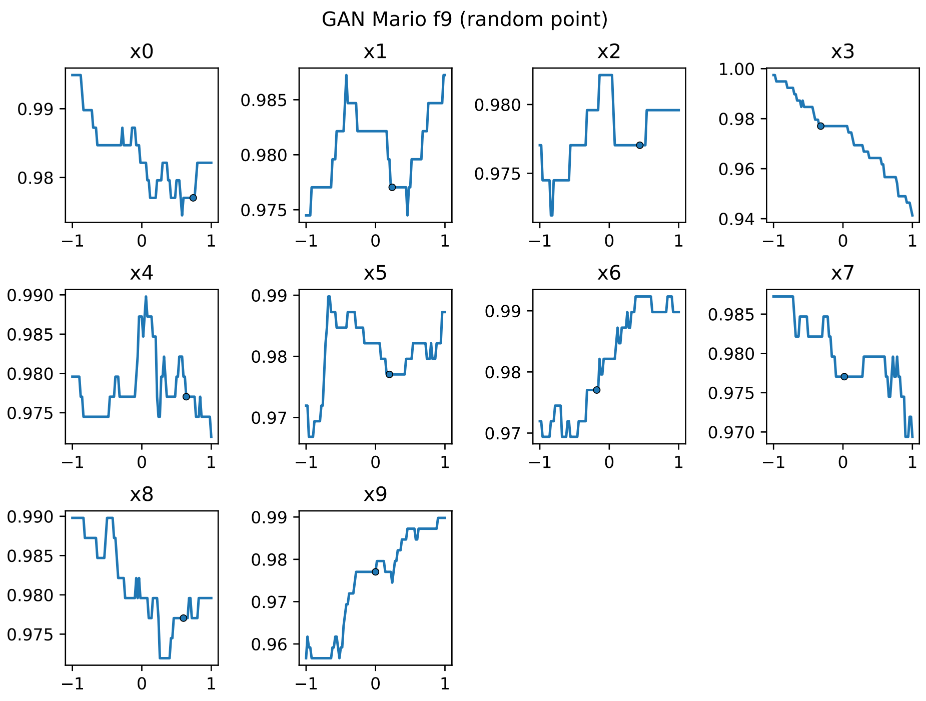
Before we run the algorithms that proved successful on the artificial functions (see section 5.2), we conduct a brief analysis of the functions contained in the benchmark. As mentioned in section 4.3, all functions in both suites can be justified in terms of the context of the real-world application, as all functions have been used in previous research. However, the functions in the corresponding applications are rarely analysed and usually treated as black boxes. In order to help our interpretation of the GBEA results, we thus seek to determine some characteristics of the functions.

Both of the GBEA function suites contain scalable problems. However, to keep the analysis concise, we will mostly conduct the experiments in the following for only one dimension for each function suite. In case of the rw-mario-gan suite, we selected the smallest dimension 10 in order to speed up the experiments, but also to achieve comparability with the artificial single-objective function suite bbob. For rw-top-trumps-biobj, we chose dimension 128, as this results in 32 cards, which is a common deck size for card games.

5.3.1 Line Walks

In order to gain a first impression of the fitness landscape of the various functions contained in the benchmarks, we conducted so called line walks through a random point. This means we generate a random point that represents a valid solution. We then "walk" on a line parallel to the axis corresponding to the first dimension that goes through the random point, evaluating the fitness function at equidistantly spaced points. We do this for each dimension separately. Two examples of resulting plots can be found in figure 5.16. The function values depicted only represent a single instance. The observations made are of course only true for the specific random point picked, and can not offer any insights in terms of global optima. However, this approach does offer a simple way to investigate locality, for example.

The first plot in figure 5.16 is very representative of the encoding-based functions. It has numerous steps in the fitness function, as well as a discernible global structure for most, if not all, dimensions. The steps are likely a result of the genotype-phenotype mapping. If values are varied along a continuous axis in the random vector, for a specific cut-off value, the one-hot encoding in the ANN will flip and produce a tile (for more details on the encoding, see section 2.4.2.4). This is a result of using GANs on Mario levels in a discrete encoding, as opposed to images with pixels encoded as continuous values. In the latter case, it is possible to produce smooth transitions [12]. For Mario, the levels created in this manner are still similar visually, as well as in terms of distance



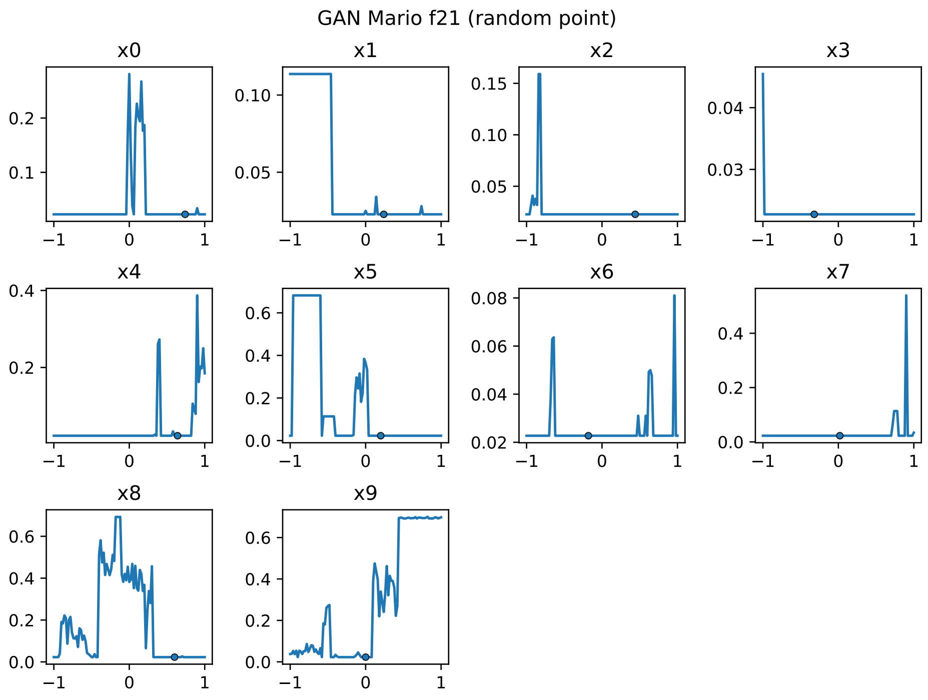


Figure 5.16: Line walks for rw-mario-gan functions 9 (before inversion, top) and 21 (bottom)

measures such as Hamming distance. However, because of the discrete encoding, the steps in the tile-based fitness functions will always occur.

In contrast, the second plot on the bottom is what most simulation-based fitness function look like, with extremely large and flat plateaus, very high spikes and almost no structure at all. The steps are significantly less pronounced, because the addition or removal of a single tile can influence the gameplay significantly. This is then captured by simulation-based fitness functions, and we therefore do not see the distinctive steps.

In the following, we compare the line walks from rw-mario-gan with selected function from the bbob suite. The results can be found in figures 5.17 and 5.18. Function 6 (figure 5.17, top) is representative of a lot of bbob functions, as it is continuous and has a global structure without any major local irregularities. Function 7 (figure 5.17, bottom) could be considered similar to the encoding-based fitness function, as both have pronounced steps (cf. figure 5.16, top). The bbob suite however also contains functions with high local irregularities, as depicted in figure 5.18. While most functions do possess an obvious global structure, such as function 24 (figure 5.18, bottom), there are evidently also functions where (at least for the line walk) no structure is discernible (function 23, figure 5.18, top).

Unfortunately, line walks cannot easily be created for multi-objective functions, which is why we refrain from generating them for suites rw-top-trumps-biobj and bbob-biobj.

5.3.2 Practicality

The practicality in terms of computational effort is an important consideration in real-world benchmarks. The optimisation problems inspired by real-world applications are usually expensive, which makes compiling these functions into a benchmark difficult. The functions then either need to be simplified (e.g. in terms of search space dimension) or represented by a simulation instead of an actual evaluation (e.g. computational fluid dynamics model). In cases where the functions are only moderately expensive, they can still not be easily compiled into a benchmark, as multiple instances of the functions should exist, and they need to be scalable. Even if these two requirements are fulfilled, a full benchmark with a diverse set of functions is likely still impractical to compute for a multitude algorithms.

In order to assess the practicality of the GBEA with regards to computation time, we thus measure the time it takes to compute one function evaluation. The experiments were run on a regular quad-core laptop. We obtain the following results:

- rw-mario-gan, without simulation: 0-1 seconds
- rw-mario-gan, with poorly performing AI: 1-3 seconds
- rw-mario-gan, with A*: 1-350 seconds, majority under 100 seconds. Theoretical maximum would be 600 seconds due to limit on simulation time
- rw-top-trumps, without simulation: 0-1 seconds
- rw-top-trumps, with simulation: 0-2 seconds

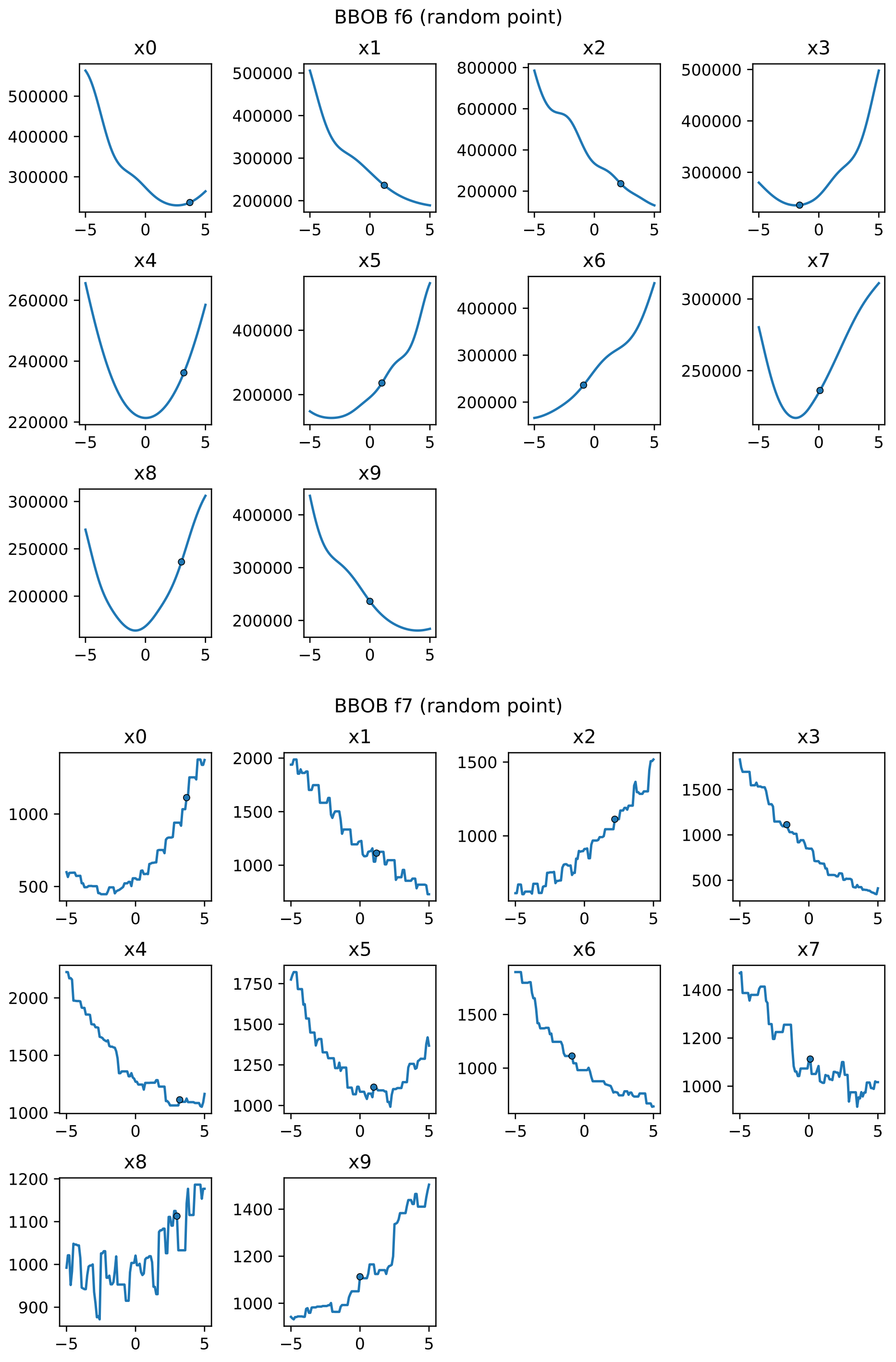


Figure 5.17: Line walks for bbob functions 6 (top) and 7 (bottom)

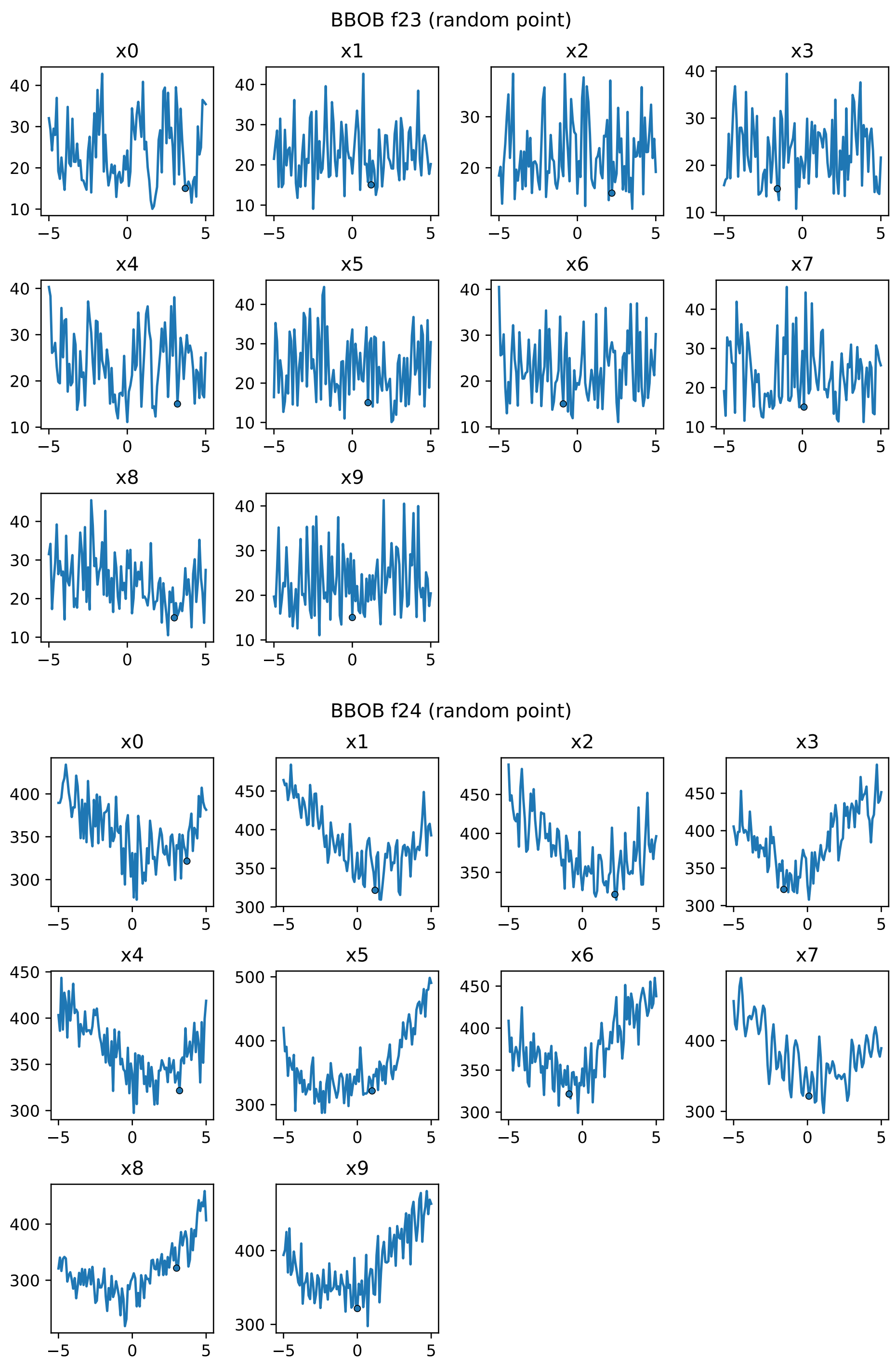


Figure 5.18: Line walks for bbob functions 23 (top) and 24 (bottom)

We observe that, as expected, functions that do not rely on simulations are fast to compute. For TopTrumps, even the simulated functions are very fast despite, the fact that 2000 simulations are executed for each point. For Mario, if the AI agent performs well and does not fail at the start of the levels, the simulations do take longer. Although a majority of the simulations finish within less than 100 seconds, there are a considerable number that take longer and finish after up to 350 seconds. We have not observed any evaluations that took as long as 600 seconds, which is the maximally allotted time.

The execution times were calculated on dimension 10 for rw-mario-gan and 128 for rw-top-trumps. However, the runtimes for Mario are independent of the size of the search space, as the solution vector is always transformed into a level snippet of constant size. The time to simulate TopTrumps playthroughs will increase in larger dimensions. But, as the simulation is very fast, increasing the dimension further is likely still going to result in reasonable runtimes.

We would consider these results sufficient to claim that the benchmark is indeed practical in terms of computational resources required. This is based on the average execution times reported for a comparable benchmark [28]. For their CFD-based benchmark, the authors report average execution times of 40.35, 947.37 and 34.44, respectively, for the three functions included in the benchmark. The observed execution times for the functions in GBEA are significantly lower for a majority of the functions included. The only exception are simulated functions in rw-mario-gan, which take longer in comparison. However, the resulting average computation time is still in the same ballpark as in the CFD benchmark, and definitely below the second function.

Additionally, the batchmode included in the experimental framework (see section 4.4.1) also provides an easy way to parallelise the experiments and run them on a cluster. Given that a cluster is available, this can dramatically improve the practicability of running the complete benchmark.

5.3.3 Baseline Results

There is one major issue that arises when integrating real-world problems into the COCO framework. For real-world problems, the global optimum value is usually unknown, even when a theoretical optimum can be computed. This becomes an issue in conjunction with the COCO post-processing and logging, as it is based on pre-defined target values. If the optimal value for a given function is set to a theoretical optimum which can not be reached in reality, no algorithm can ever reach the higher precision targets. Due to the way the targets are distributed, this might make algorithms with widely different performance appear similar in terms of when they reach the targets.

A solutions is to compile a set of baseline results and to then define the best observed result as the global optimum. This was also done for the bbob-biobj suite, as the globally optimal hypervolume is not computable analytically. This issue is especially problematic for the rw-top-trumps-biobj suite, as in this case, even the optima for the single objective functions are unknown. The globally optimal hypervolume is therefore even more difficult to estimate. As the results presented for this thesis are the first results obtained, there is

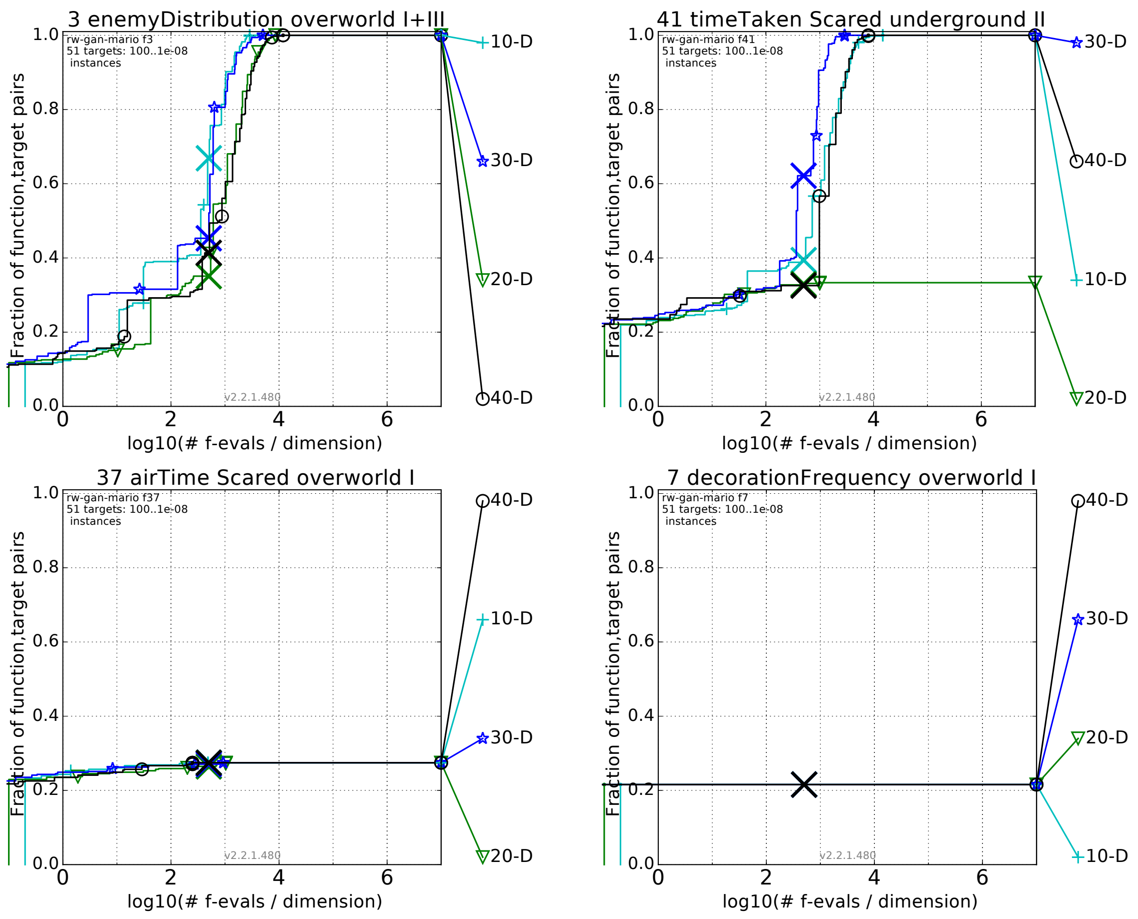


Figure 5.19: Runtime distribution plots for CMA-ES (experiment id 32) on selected functions in rw-mario-gan suite).

no baseline available. The plots generated from COCO therefore have to be interpreted with this issue in mind.

As a very first baseline for rw-mario-gan, we ran CMA-ES on the first half of the suite (non-concatenated functions, see section 4.3.3.2). Runtime distribution plots for selected functions may be found in figure 5.19. While there are some functions where the targets seem to have been suitably distributed (top row), there are also plots where this is decidedly not the case (bottom row). For both of the functions plotted in the bottom row, we can explain why the specified optimum is never reached.

For airTime (fid 37) the optimum is reached if all simulations time out, i.e. the AI playing Mario is stuck. This is only possible if levels are generated with insurmountable obstacles or levels that require backtracking. Naturally, the former type of levels are not playable and there are no training examples of levels with such obstacles. They are therefore also not generated (as intended), independent of the input to the generator.

For decoration frequency (fid 7), the optimal percentage of *pretty tiles* (i.e. Tube, Enemy, Destructible Block, Question Mark Block, or Bullet Bill Shooter Column) is specified as 1. While this is a theoretical upper bound, reaching it would require generating levels

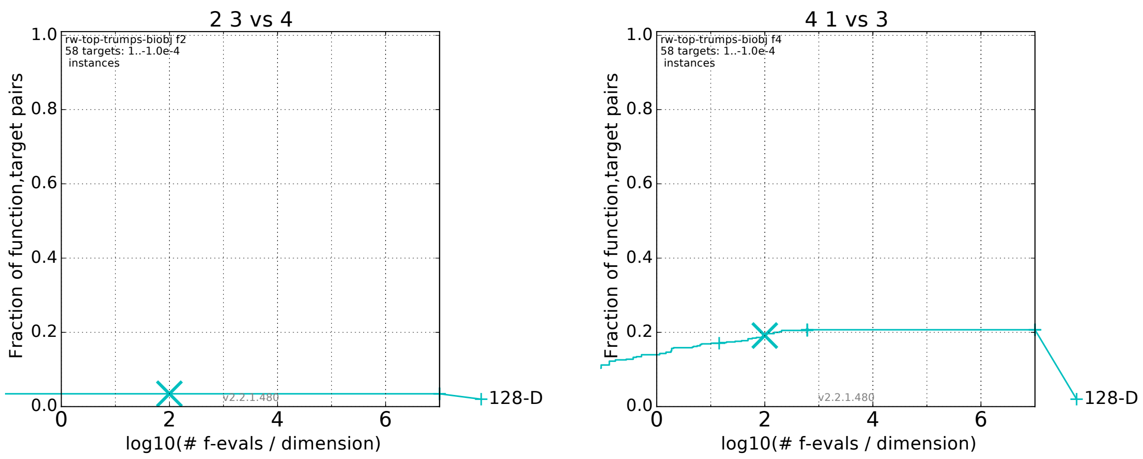


Figure 5.20: Runtime distribution plots for SMS-EMOA (experiment id 41) on selected functions in rw-top-trumps-biobj suite.

without any passable space or ground tiles at all. With the same explanation as above, the generator is very likely not able to produce such a level.

For a baseline for the rw-top-trumps-biobj suite, we first run the SMS-EMOA on the suite with dimension 128. As expected after the results on the rw-mario-gan suite, we obtain mostly flat graphs (see figure 5.20).

5.3.4 Summary of Results

Summarising the observations made in this section, we determine that:

- Based on the line walk plots, the game optimisation problems are interesting and challenging as optimisation problems (plateaus and steps, as well as existence / lack of global structure).
- Simulation-based and encoding-based functions possess different characteristics.
- Some functions resemble artificial functions, while other characteristics seem to be novel.
- The GBEA functions run with practical execution times.
- In order to use the COCO post-processing features fully, better estimates of the true global optima of each function need to be determined.

Regardless of the issues with plotting of the aRT values, we can confirm hypothesis H2. The functions contained in both suites seem to be challenging based on their fitness landscape. The examination of different uncertainties in game optimisation from section 4.1.4 also supports this claim.

id	algorithm	EA	ht	um	sm	note
32	CMA-ES	-	-	-	-	-
33	SAPEO	CMA-ES	3	mean	close	-
36	SAPEO	CMA-ES	3	mean	fit	GP-DEMO
39	prescreening	CMA-ES	∞	mean	close	-
40	RS	-	-	-	-	random search

Table 5.6: Experiments in set E3 (rw-mario-gan suite)

id	algorithm	EA	ht	um	sm	note
41	SMS-EMOA	-	-	-	-	-
42	SAPEO -	SMS-EMOA	3	mean	close	-
45	SAPEO -	SMS-EMOA	3	mean	fit	GP-DEMO
48	prescreening	SMS-EMOA	∞	mean	close	-
49	RS	-	-	-	-	random search

Table 5.7: Experiments in set E4 (rw-top-trumps-biobj suite)

5.4 Experiments on GBEA

After the considerations in the previous two sections, we can compute the results for promising algorithms (as per their performance on the artificial benchmarks, see section 5.2) on the GBEA benchmark, which is deemed interesting and practicable (see section 5.3). We thus set up experiment sets E3 and E4 on rw-mario-gan and rw-top-trumps-biobj, respectively. We reduced both sets of experiments in comparison to E1 and E2. As EGO was not successful on either of the benchmarks, the corresponding experiments were removed. The same holds for algorithms based on the MOCMA. Similarly, the SAPEO version without model validation and SAPEO with fitness transformation were removed for the same reasons. We also only kept 2 SAPEO versions and 1 pre-screening variant, as these algorithms seemed to perform similarly as each other.[2] Random search and the underlying evolutionary algorithms are of course kept as baselines. The resulting experiments with their ids can be found in tables 5.6 and 5.7.

However, due to the issues explored in section 5.3.3, the obtained results are barely interpretable. In order to visualise the issue, we plotted some results in figure 5.21. As determined previously based on the plots in figure 5.19, there are only a few functions in rw-mario-gan where different performances can be distinguished. For these functions, it seems that random search is among the worst-performing algorithms. However, even random search is able to reach the optimum for the enemyDistribution functions (see figure 5.21, top left). A majority of the remaining functions display very flat graphs,

[2]The selected pre-screening algorithm is the variant most similar to pre-screening as implemented for the analysis in [149], which already compares pre-screening with (a previous version of) SAPEO.

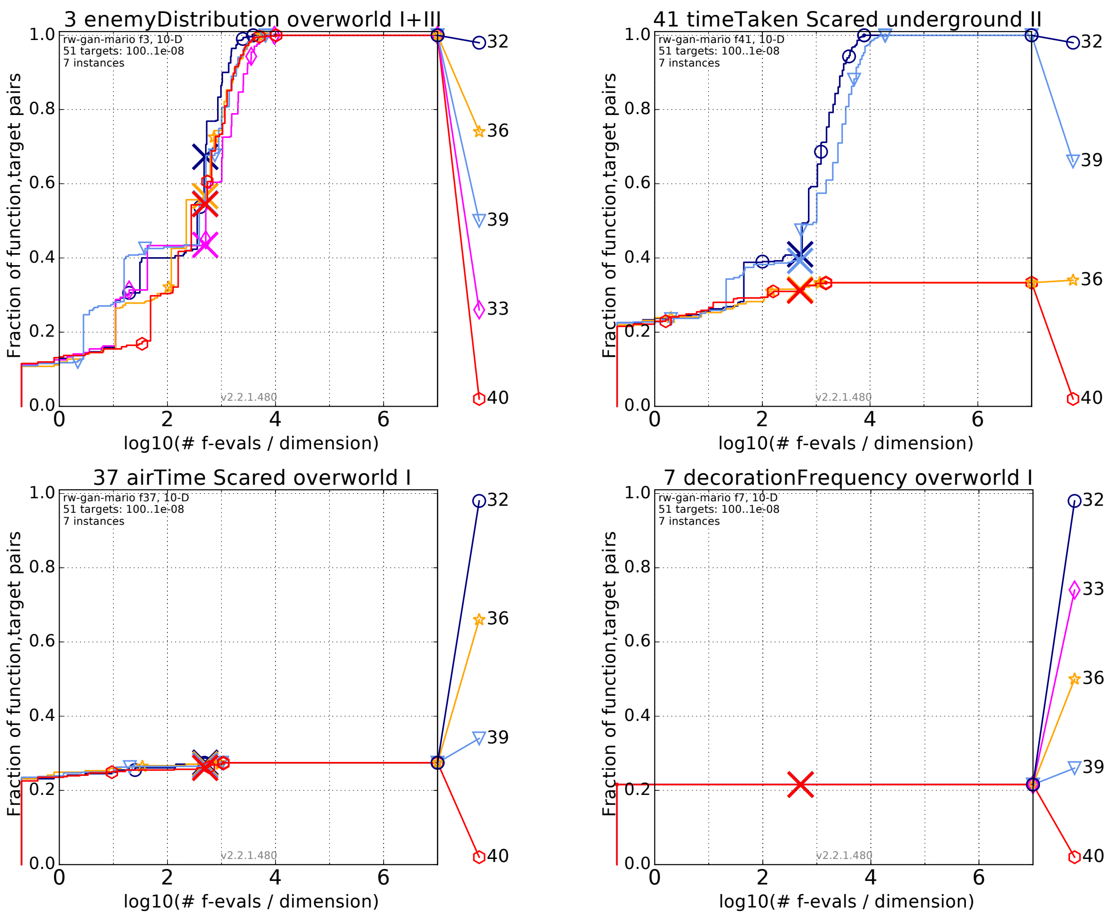


Figure 5.21: E3 runtime distribution plots on selected functions.

id	algorithm	EA	ht	um	sm	data	note
34	SAPEO	CMA-ES	3	mean	close	surr	-
35	SAPEO	CMA-ES	3	mean	close	both	-
37	SAPEO	CMA-ES	3	mean	fit	surr	GP-DEMO
38	SAPEO	CMA-ES	3	mean	fit	both	GP-DEMO

Table 5.8: Experiments in set E5 (rw-mario-gan-offset suite)

as expected (see figure 5.21, bottom row). The results are thus not useful to evaluate algorithm performances without a baseline comparison.

The same observations and conclusions unfortunately hold true for the results obtained from experiment suite E4 on rw-top-trumps-biobj (see figure 5.22). It therefore also does not make sense to run the experiments planned for suite E5, as the results will not be interpretable. However, for the sake of completeness, the planned experiments can be found in table 5.8.

Unfortunately, this also means that we are not able to answer hypothesis H3, as it

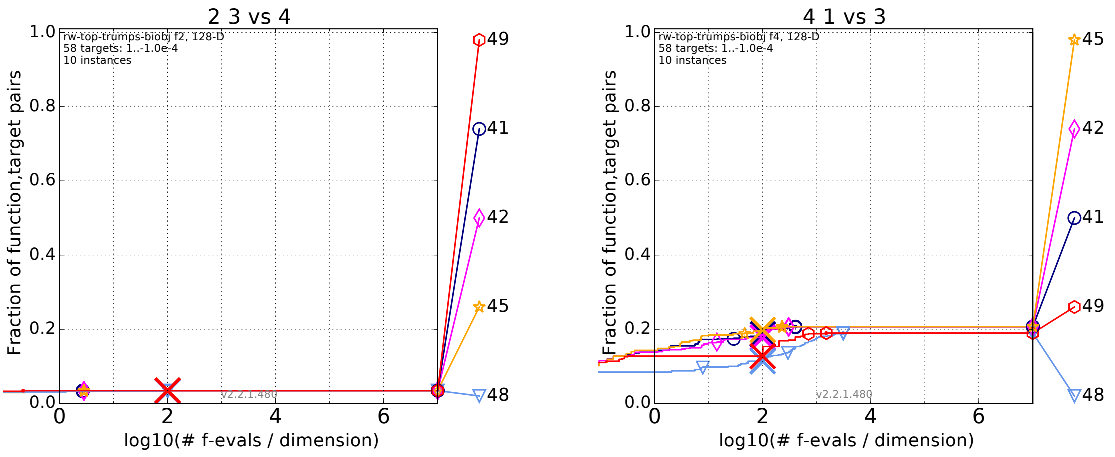


Figure 5.22: E4 runtime distribution plots for on selected functions.

relates to the performance comparisons of various algorithms on the GBEA benchmark.

CHAPTER 6

Conclusions and Future Work

In the following, we first summarise our conclusions in section 6.1. Following that, we detail various directions for future work in section 6.2.

6.1 Conclusions

In this thesis entitled *Uncertainty handling in surrogate assisted optimisation of games*, we started out with the goal to investigate the uncertainty in game optimisation problems, as well as to identify or develop suitable optimisation algorithms. In order to approach this problem systematically, we first created a benchmark consisting of suitable game optimisation functions (GBEA). The suitability of these functions was determined using a taxonomy that was created based on the results of a literature survey of automatic game evaluation approaches. In order to improve the interpretability of the results, we also implemented an experimental framework that adds several features aiding the analysis of the results, specifically for surrogate-assisted evolutionary algorithms.

After describing potentially suitable algorithms, we proposed a promising algorithm (SAPEO), to be tested on the benchmark alongside state-of-the-art optimisation algorithms. SAPEO is utilising the observation that most evolutionary algorithms only need fitness evaluations for survival selections. However, if the individuals in a population can be distinguished reliably based on predicted values, the number of function evaluations can be reduced. After a theoretical analysis of the performance limits of SAPEO, which produced very promising insights, we conducted several sets of experiments in order to answer the three central hypotheses guiding this thesis. We find that SAPEO performs comparably to state-of-the-art surrogate-assisted algorithms, but all are frequently outperformed by stand-alone evolutionary algorithms. From a more detailed analysis of the behaviour of SAPEO, we identify a few pointers that could help to further improve the performance.

Before running experiments on the developed benchmark, we first verify its suitability using a second set of experiments. We find that GBEA is practical and contains interesting and challenging functions. However, we also discover that, in order to produce interpretable result with the benchmark, a set of baseline results is required. Due to this issue, we are not able to produce meaningful results with the GBEA at the time of writing. However, after more experiments are conducted with the benchmark, we will be able to interpret our results in the future. The insights developed will most likely not only be able to provide an assessment of optimisation algorithms, but can also be used to gain a deeper understanding of the characteristics of game optimisation problems.

6.2 Future Work

In the following sections, we discuss several topics that constitute interesting directions of future work. We first address game optimisation problems in general in section 6.2.1, before considering possible improvements to the GBEA in section 6.2.2. We close with further thoughts on surrogate-assisted evolutionary optimisation in section 6.2.3.

6.2.1 Game Optimisation Problems

6.2.1.1 Validation of Evaluation Functions

The GBEA benchmark presented in this thesis and run for the GECCO 18 and 19 workshops[1] consciously utilises previously published game optimisation problems and the evaluation functions proposed in the corresponding papers. This is done in order to reflect the state-of-the-art in automatic game (content) generation and tuning.

This has the added benefit of not requiring a thorough validation of these functions in terms of how meaningful they are for human gameplay. However, as made apparent by our taxonomy (see section 4.1) and survey (see appendix A), validation functions in literature are of a very limited variety. Most evaluation approaches in the survey are based on model assumptions that are not validated for human players (category feedback NONE). However, we have also shown in our case study in appendix B that these assumptions can be very misleading for any optimisation algorithm or prediction model utilising the resulting evaluation functions.

For this reason, I would recommend that more effort is made towards validating evaluation functions in context of (semi-) automatic game design / tuning. Unfortunately, conducting comprehensive experiments with human players is not always possible, especially in cases where the evaluation approach is only a minor component of the project. In our planned task force on game evaluation, we are attempting to tackle these problems and facilitate the validation of evaluation functions.

One suggestion is to host websites where surveys based on popular research games (such as Mario and the GVGAI framework) can be set up for online participation. Reducing the effort to set up these surveys might lead to more researchers collecting data from human players. Making the survey available through a browser online should also increase the number of participants, and thus the significance of the results. Ideally, this set-up would also include several game playing agents, as well as extensive logging and visualisation capabilities. A description of what such a system could entail, in addition to a description of its potential can be found in a recent vision paper [47].

An even easier approach is to make researchers aware of previously published evaluation functions, especially if they have been validated. In context of the task force, a website that provides this information in an easily accessible way is planned. Further in the future, it would be worthwhile to investigate whether meaningful and validated evaluation functions exhibit specific patterns that can be generalised to multiple (similar) games. Such an investigation could be based on the data from both the website and

[1] http://norvig.eecs.qmul.ac.uk/gbea/gamesbench.html

online surveys as described above. From the GBEA results, we were already able to observe some characteristics that were consistent for certain types of functions, such as steps in fitness functions based on the encoding in MarioGAN. With more results, these observations could be extended and verified.

Another option is to depart from automatic evaluation and instead obtain the evaluations of game content by playtests directly. Of course, this is only practicable if a small number of solutions need to be evaluated. One potential approach could be surrogate-based algorithms that reduce the number of exact evaluations required to a minimum. A promising option here is to use surrogates based on a variety of information, possibly including multiple diverse fully automatic evaluation functions. Such an experimental set-up was proposed in section 5.1. Even though the corresponding experiments were conducted, they are unfortunately only interpretable after more baseline experiments have been completed for the GBEA. We will therefore definitely come back to these results in the future.

Further tools for similar scenarios, i.e. semi-automatic optimisation of game problems have also been proposed in [104] and tested using an real-world strategy (RTS) game. This thus shows that considering several types of data is a promising solution, even for complex games with large search spaces.

6.2.1.2 Analysis of Fitness Landscapes

Independent of the validity of the evaluation function, it should also be considered what type of fitness landscape is created with its usage. Information on the fitness landscapes are crucial for the choice of suitable optimisation algorithms as well as for putting their performance into context. The need for further analysis of existing evaluation functions has also been recognised in other publications as well, see for example recent surveys and vision papers [19, 130].

This is the reason we are investigating the function suites in GBEA in more detail. Part of this analysis are the line walks as described in section 5.3.1. In addition, we also plan to do further analysis using techniques from the field of **E**xploratory **L**andscape **A**nalysis (ELA). Corresponding approaches center around computable features intended to characterise functions in terms of a set of abstract concepts, such as their modularity and the existence of plateaus. The features have been commonly used as inputs for models that choose which one of a set of algorithms to run on a given problem [72]. We want to use them instead as a way to characterise the resulting fitness landscapes. This could be done by training models to recognise specific characteristics based on information from the ELA features.

The analysis is also going to include visualisations of the fitness functions, as ELA features do not necessarily provide interpretable information. Visualisations allow a more holistic overview for the human observer. However, corresponding plots would likely be done on smaller scale versions of the GBEA problems, due to the inherent dimensional limitations of visualisations.

Additionally, since the fitness functions seem to be flatter than expected due to the lack of baseline performance results (see section 5.3.3), properties that are usually assumed of a fitness landscape should be investigated further. One such property is *locality*, i.e. sensitivity to small modifications in search space. This is especially important with regard to the correlations between fitness and available mutation operators. This relationship is of course influenced by the representation of the search space, as game optimisation problems invariably include some form of phenotype-genotype mapping. For example, the steps in the encoding-based fitness functions were clearly a result of the one-hot encoding used in MarioGAN (see section 5.3.1). Based on these insights, common representation methods for levels and game parameters should also be analysed in terms of their influence on the properties of the resulting game optimisation problems.

6.2.1.3 Analysis of Uncertainty

The analysis of the uncertainties identified in the taxonomy described in section 4.1 have largely been quantitative in nature, see chapter 5. The only large exception is the case study in appendix B that verifies the existence of a specific type of bias. However, common sources of uncertainty should ideally be investigated qualitatively and in more detail. To discuss potential future work in this regard, we refer to the several sources of uncertainty identified in section 4.1.4.

For the feedback dimension, the main errors are based on survey design and the interpretation of the feedback. Both of these problems would be mitigated if the evaluations function could be validated as discussed in section 6.2.1.1. If enough data is available, even the non-determinism in games would not be an issue in terms of obtaining a meaningful signal for the evaluation.

Lacking a thorough validation of a given evaluation function, other approaches to analyse the uncertainty could be taken. If no feedback from human players is available, survey design becomes irrelevant. The issues caused by non-determinism in games are also alleviated, as evaluations that do not require playtests can usually be repeated often enough to obtain statistically significant results. What remains is the interpretability of the feedback, which is significantly harder without human feedback. Explainable AI as envisioned in [146] could be the key to translating AI behaviour into interpretable feedback. However, explainable and interpretable algorithms are still an active field of research, with sometimes counter-intuitive results.[2]

For the input dimension, data selection and data generation have been identified as the two main issues. The latter has been covered by the case study in appendix B. However, more of these studies should be conducted in different settings in order to identify how prevalent and noticeable data generation bias really is. Uncertainty from data selection can be approached by either avoiding a selection completely (by using enough computational resources and employing deep learning practices), or by using suitable dimensionality reduction techniques developed in machine learning, such as feature selection or principal component analysis.

The final source of uncertainty stemming from the choice of model can be investigated using model validation approaches, e.g. cross-validation as described in the context

[2]Such a result comes from a recent study that finds that increasing the transparency of models reduced the ability of human survey participants to detect erroneous model behaviour [101].

of SAPEO (see section 4.2.2). Because the experimental framework (see section 4.4) includes automatic logging of the prediction error, as well as the predicted uncertainty, model fit can be conveniently investigated. These features already produced interesting results in the experiments. An example is the conclusion from section 5.2.1.3 to increase the strictness of model validation in SAPEO.

6.2.2 Game Benchmark for Evolutionary Algorithms (GBEA)

The first and most important addition, as mentioned in the previous sections, are baseline performances. Based on these results, appropriate targets can be determined in order to increase the interpretability of results obtained with the benchmark. As the GBEA is part of a workshop at GECCO, this is an ongoing project and will be tackled in the future.

Furthermore, based on a more extensive analysis of the existing problems as described in section 4.3.2, the functions should be examined in terms of their contribution to the benchmark. Functions with high correlation or low meaningfulness might be dropped in order to reduce the computation time of the full benchmark.

An additional important aspect for further examination is whether the fitness functions can be further adapted so that evolutionary algorithms can be distinguished based on their performance. This seems to be a property that benchmarks aspire to, which is not the case in the GBEA at the moment. However, whether this is useful depends on what the reason behind the lack of differences in performance is. If it is solely because of the definition of targets, the issue will be solved by adding baseline results. If it is because some optimisation algorithms genuinely struggle on the problems, it still might be worthwhile to include them in the benchmark, as they did occur in previous research.

Besides potential modifications of the existing function suites, we also plan to add more suites based on different applications in the future. To do this, ideally, the COCO framework should be extended in order to fully supported noisy optimisation in this context. This would allow to leave noise handling up to the optimisation algorithm instead of setting a specific number of simulations in order to consistently produce similar values.

Furthermore, many of the game optimisation problems targeted in literature seem to have a non-continuous search space [75]. In order to be able to represent these types of problems, appropriate functions suites should be added to the GBEA. As COCO is designed for continuous optimisation, this aspect also requires further modification of the framework. For example, we plan to add the GVGAI parameter optimisation problems that were part of the BBComp competition at EMO in 2017 to the benchmark.

6.2.3 Surrogate-Assisted Evolutionary Algorithms

Based on the results for SAPEO in section 5.2.1.3, we have concluded that model validation is definitely one aspect to consider further. It seems that even stricter model validation might further improve the performance. In this context, it should be investigated where the behaviour switches and SAPEO always resorts to falling back on the underlying algorithms, as in these cases, no improvements are made either.

An additional result from the analysis is that potentially convergence detection mechanisms, especially in CMA-ES, could be able to detect whether the number of selection errors is large. If this hypothesis can be verified, this observation could be used to either restart SAPEO, or to adapt the strictness of model validation.

As many game optimisation problems also include mixed-integer search spaces (see section 6.2.2), it would also be interesting to investigate how SAPEO performs in conjunction with other surrogate models. One potential candidate are bandit models, which require only minor assumptions and work well for problems with small search spaces, but noisy fitness functions [75].

Furthermore, it should be investigated why the surrogate-assisted algorithm tested in our experiments performed mostly below our expectations. It is possible that these algorithms were only intended for a small subset of problems with very low budgets and specific properties. If that is the case, future work could be to find out whether performance can be improved overall. If this is not the case, additional implementations of these algorithms should be tested. One algorithm that should definitely be run is EGO with full global models.

APPENDIX A

Game Evaluation Survey

In the following, we first identify relevant areas of research in the field of *Artificial and Computational Intelligence in Games*. We then survey work from these areas and classify the described approaches according to our taxonomy as described in section 4.1. The presented work is grouped by type of game (content) that is evaluated. We hope to identify dominant and unexplored methods using this structure. This analysis will be visually supported by displaying the publications in tables based on our taxonomy (cf. Tab. 4.1).

A.1 Characterisation of Game Evaluation AIs

All the arrows in Fig. 4.1 describe an information processing step which can be executed by a human or an AI. In case of automatic processing as addressed in this survey, all steps need to be executed by an AI. The employed AIs can be classified using the taxonomy presented in [160]. This is done in the following, in order to identify areas with relevant literature.

So in terms of the *End User (Human) Perspective* according to the paper, the paths PLAYER, COMP and STAT (blue arrows) all **model player behaviour**. PLAYER is intended to predict the actions of a player within the game context, while STAT models the behaviour of a whole group of players in terms of gameplay statistics. In contrast, COMP models player behaviour in the sense that it aggregates gameplay data into statistics (e.g.*average final score*), thus potentially biasing it by selecting specific statistics.

The processes depicted in red, i.e. CODE, OUT and PLAY, all describe an **evaluation of content** in terms of the *End User (Human) Perspective*. In all cases, the game or content is evaluated in terms of a goal that is defined a priori. While CODE uses a direct evaluation based on an encoding of the content, PLAY and OUT evaluate the content based on further data that is generated from it.

The intended end user of game evaluation is mainly the game designer, but of course producers/publishers are indirectly affected as well. Depending on how the evaluation results are used, researchers have a stake in game evaluation as well.

In case of the red arrows, it is very clear that the methods employed here fall into the research area of *player modelling*. PLAYER, however, describes some form of player AI which relates to research in *nonplayer character (NPC) behaviour learning* and *search and planning*, depending on the game in question. *General game AI* also ties into this process, as the AI generating playtraces should be as general as possible in order to deal with different levels and rulesets equally well. Additionally, as the AI in case of PLAYER

Table A.1: Publications applying game (content) evaluation to grid-based games. Research on platformers is displayed in blue, on dungeons in green and on general arcade games in red.

input \ feedback	**none** NONE	**implicit** IMP	**explicit** EXP
encoding CODE	[58, 92, 121, 122, 129, 130] [82] [26, 88]	[5]	[117]
outcome statistics OUT	[63] [26, 59, 97, 98, 135]		
gameplay data PLAY	[46, 126] [125] [84]		[119]

serves as a stand-in for a human playtester, research in *believable agents* is relevant here as well.

COMP is the selection of statistics that characterise a playtrace appropriately and is thus most related to *player modelling*. Similarly, while there are not currently many examples of methods that follow STAT, they could be realised using machine learning methods and would then most likely fall under *player modelling* as well.

Finally, research on *AI-assisted game design* naturally includes various forms of game evaluation. The fields of *procedural content generation* and *computational narrative* do not relate directly to any of the AIs in the figure. However, publications using search-based algorithms in both fields often employ processes that are visualised in Fig. 4.1 in order to evaluate the generated content and are thus relevant as well [116, ch. 2].

A.2 Game Evaluation Methods

A.2.1 Grid-based Games

Grid-based games are a class of games that is very popular in general games research and also PCG research, probably since they are easily observable and usually have an obvious encoding. In this chapter, we survey game (content) evaluation methods that have been applied to platformer, dungeon and general arcade-like games that are based on a 2D-grid. The corresponding work is displayed in Tab. A.1, colour coded by the specific application.

A.2.2 Platformers

As is clearly visible from Tab. A.1, research on grid-based 2D-platformers focuses heavily on CODE-NONE models. [130] provides a summary of commonly used metrics to evaluate platformers based on their level encoding and adds more. The metrics range from expressing challenge (e.g. leniency) to measuring visual aesthetics (e.g. symmetry). They are either based directly on the positioning of different tiles on the grid or else on optimal paths that can be computed from the encoding. They all rely on models that are either defined using designer experience, or, especially in the case of visual aesthetics, based on design theories. Similar metrics have been used in [129] and [58].

While the measures and games are similar in these cases, the context in which they are used in is not. While [130] provides a survey and characterisation of different measures, in [58] they are used to describe the expressive range [116, ch. 12] of a level generator. In [129], they are used instead to guide a search-based level generator [116, ch. 2] employing Markov chains. In this case, the results are evaluated using explicit feedback from human players, information which the authors of [129] suggest to add to the evaluation function in future work.

Instead of grid-based encodings as described above, [122] and [121] use a specific geometry and rhythm-based encoding of platformer levels. In this case, the rhythm of required actions to traverse the level are represented. The encoding is thus abstracted from its visualisation and implementation and instead corresponds closer to the player experience. The evaluation of the level in [122] can be transformed into a constraint satisfaction problem because of the appropriate encoding and can thus be solved analytically. In [121], the measures on the rhythm-based representation are extended and used to analyse the expressive range of a level generator.

The authors of [92] take yet another approach by evaluating how similar two levels are based on a chunk-encoding. They use this measure to generate levels that replicate the style of existing ones designed by humans.

Other research operates on simulation-based evaluation instead. [63], for instance, takes a Monte-Carlo approach to evaluating diversity and playability by observing the scores of a large number of simple AI players. In contrast, [126] and [46] apply models of game experience that originate from related research, namely the concepts of *flow* and *empowerment*. In [46], the employed model is also verified using explicit quantitative and qualitative feedback from human players.

In contrast to the work presented above, which is all based on an independently defined model, [117] and [119] introduce methods that use models trained from explicit player feedback. Both publications use gameplay data with annotated experience based on a survey. While [117] learns a neural network that predicts the visual aesthetics of a MarioAI level based on different features that describe the level encoding, in [119], a model employing neuroevolutionary pairwise preference learning and automatic feature selection is trained to predict engagement, frustration and challenge from recorded gameplay data.

A.2.3 Dungeon Games

Grid-based dungeon games can be evaluated in a very similar fashion as the platformer games described above. An example of a similar-style evaluation of dungeon levels is [82], where measures for area control, exploration and balance are computed from coarse map sketches of either dungeons or strategy games.

However, a completely different approach is presented and applied to the game MiniDungeons[1] in [84]. In this work, the dungeon map is evaluated based on a set of AIs that all value different gameplay outcomes and actions differently (encoded in a utility function), thus exhibiting distinct playing styles. In this case, the AIs are used to provide critique of the maps based on their different utility functions. Ideally the AIs, dubbed *procedural personas* behave similar to different player types, thus modelling possible responses of the game audience.

A.2.4 Arcade Games

Many of the publications on arcade-style games that include game evaluation are related to the framework of the General Video Game AI competition (GVGAI)[2]. The framework allows arcade-style games to be defined using the video game description language (VGDL) and then automatically processed by an engine to be played by human and / or AI players. The player AIs that participate in the competition have to be general, i.e. play previously unseen games just based on the engine responses and a forward model. The evaluation methods for GVGAI games also tend to be generally applicable to all games that can possibly be defined within the framework.

Some of the evaluation methods are based directly on the VGDL representation, such as [5] and [88]. In [5], strategic depth is estimated based on the minimal complexity of a heuristics representation of human player behaviour, i.e. implicit feedback. The assumption here is that more complex behaviour patterns and irregular behaviour correspond to a higher perceived strategic depth. The model is evaluated based on a small case study with explicit feedback. In contrast, the methods in [88] are popular recommendation methods that rely on object- and user-similarity measures. The evaluation in [26] is also based on a ruleset representation (other than VGDL), but includes information on the map and outcome statistics as well.

Many evaluation approaches rely on a specific form of outcome statistics, namely *algorithm performance profiles* ([59, 97, 98, 135]). *Algorithm performance profiles* are a popular approach for automatic game (content) evaluation in general. The underlying principle is the *formal theory of fun* [112], that is based on the assumption that learning and progressing in a game over time is *fun*. Human players would thus enjoy learning patterns which can be mimicked by player AIs with *intrinsic rewards* for learning novel and surprising patterns [112]. *Algorithm performance profiles* take up this idea by measuring the performance of simple (random) agents vs. more sophisticated ones, or alternatively, algorithms with varying budget restrictions. The underlying idea is that, if the game distinguishes these agents based on performance, there are (1) a large set of learnable patterns with (2) differing difficulty levels. This would give a human player an opportunity for noticeable progress and increase enjoyment according to [112]. *Restricted play* proposed in [64] is also based a similar concept. In this case, however, restrictions

[1]http://minidungeons.com/

[2]http://gvgai.net/

Table A.2: Publications applying game (content) evaluation to parlour games. Research on board games is displayed in blue, on card games in green and on dice games in red.

feedback / input	none NONE	implicit IMP	explicit EXP
encoding CODE	[18] [99, 148]		
outcome statistics OUT	[18, 120] [99] [62]		
gameplay data PLAY	[18] [99, 148]		

are placed on the AI players and the resulting performances are compared. The influence of these different restrictions on the game outcome can thus be assessed.

In contrast, the work in [125] is on the enjoyability of Pacman ghost teams and not related to GVGAI. The presented method is a weighted sum of several measures that express challenge and spatial, as well as behavioural diversity, based on gameplay data. It is evaluated using explicit feedback.

A.2.5 Parlour Games

Parlour games are another topic of regular games research, popular examples are board games, card games and dice games. We survey research that falls into this category in the following section. Corresponding work is displayed in Tab. A.2, colour coded by the specific application.

One of the most prominent publications on board game evaluation is [18], where 57 different measures using varying types of inputs are defined to automatically evaluate board games. Many of the measures are based on scientific models describing aesthetics in different fields, while some are based on designer intuition. The measurements are integrated within a PCG framework called *Ludi* that has produced games that have successfully been published as board games[3].

[120] is another example of work on board games (Ticket to Ride[4]), this time based on outcome statistics from playthroughs of AIs that model player behaviour based on established strategies. The framework was able to detect situations that were not covered by the game rules, as well as dominant sub-strategies. The latter discovery would need to be verified with outcome statistics from human player data. The former, however, demonstrates that rulesets can successfully be tested for coverage automatically in a cost-efficient way.

[3]http://www.cameronius.com/games/yavalath/

[4]https://www.daysofwonder.com/tickettoride/en/

Table A.3: Publications applying game (content) evaluation to strategy, action, and narrative-based games. Research on strategy games is displayed in blue, on action games in green and on narrative-based games in red.

feedback / input	**none** NONE	**implicit** IMP	**explicit** EXP
encoding CODE	[81, 82, 103, 115, 136] [87]	[54, 154] [77]	[81] [79]
outcome statistics OUT	[22, 76, 115] [86, 134]		
gameplay data PLAY	[102] [3, 69] [43, 55]		[159]

Game evaluation of card games has also been used to detect issues in game specification, such as the work by Osborn et al. on Dominion[5] [99]. As in [18], the evaluation is also based on a combination of several measures that use different input data. In contrast, [148] demonstrates how gameplay-data based evaluation can be approximated with different models based only on the encoding for the game TopTrumps[6].

In case of [148], the evaluation is based on abstract concepts such as fairness, suspense and engagement that are modelled with gameplay statistics of different AI players. Similar concepts and implementations are used in [62] for dice games. The authors even suggest that the dice games can be a representation of battles in games in general, thus suggesting a generalisation of these concepts for a wide range of games beyond parlour games.

A.2.6 Strategy, Action and Narrative-Based Games

In the following section, we present work from several different game genres, namely strategy, action and narrative-based games. Strategy games include real-time strategy (RTS) games, physics-based games and puzzles as well as games with a purpose. Racing, fighting, sports and physical games are all considered examples of action games. Corresponding work is displayed in Tab. A.3, colour coded by the specific application.

Most of the strategy games identified as category CODE-NONE are targeted towards map-evaluation in (real-time) strategy games such as StarCraft[7]. Examples are [81, 82, 103, 136]. These publications all include model-based measures computed directly from grid-based simplifications of the maps which express properties such as enemy density and symmetry, that are expected to relate to difficulty and fairness, respectively. [81] also includes a model of designer preferences that is updated online based on explicit feedback during the design process.

[5]https://dominion.games/
[6]http://www.toptrumps.com/
[7]https://starcraft2.com/en-gb/

In [115], the levels of the physics-based puzzle game Cut the Rope[8] are represented as geometric constructs, thereby also encoding possible solutions of the puzzles. In the publication, the aesthetics and playability of generated Cut the Rope levels are evaluated mainly based on their representation, but outcome-based measures are added as well.

The measure for strategic depth proposed in [76] applies the concept of *algorithm performance profiles* that was described in section A.2.4 to strategy games in this case. This is true as well for the method to evolve car racing presented in [86]. The work in [22] is also based on outcome statistics, but in this case mainly evaluated in terms of system utility. This measure expresses the efficiency of computational resources (i.e. human player input), as the game in question is categorised as a game with a purpose (GWAP). [102] describes further genre-specific model-based evaluation measures for another physics-based game.

The measures applied to action games as defined above are used for very different purposes and thus differ significantly. One purpose is dynamic difficulty adaptation (DDA), i.e. adapting the difficulty in real-time such that the human player is challenged, but not frustrated. Work in this field naturally requires an evaluation of game (content) difficulty. DDA can of course be applied to a variety of genres, for example in car racing [134] and fighting games [3]. [134] adapts opponents in car races based on an aggregated statistics of the results. This makes for a slower adaptation when compared to [3], where the difficulty at each point in time is estimated based on player health.

A model based on implicit feedback from game (content) popularity is used in both [54] and [154]. While [54] gathers data on content popularity, [154] evaluates the game based on player retention. The models are used to either (1) evolve weapons in a space arms race[9] in [54] or (2) to predict player retention based on observed gameplay patterns in a sports game[10] [154]. The method described in [79] is applied to another space-themed game, but instead evaluates the aesthetics of spaceships using a model-based approach.

The evaluation methods described in [87] and [157] both measure emotional responses in their respective domains. In [87], the tension created by audio in the context of a horror game is evaluated. In contrast, the entertainment in physical children's games is measured in [157]. While [87] employs a model-based approach that evaluates different patterns of tension, the tension model is actually crowdsourced and learned from human feedback. The model for entertainment in [157] is based on human feedback as well. In this case, it is an artificial neural network (ANN) trained on explicit feedback.

Similar to the evaluation of dungeons based on so-called mission graphs that express the type and order of challenge in [69], all the evaluation methods for narrative in games we found [43, 55, 77] are based on a kindred representation of the temporal order of events. While a crowdsourced approach that includes implicit feedback is proposed in [77], [43, 55] add information collected from gameplay data instead.

[8]https://www.cuttherope.net/

[9]http://galacticarmsrace.blogspot.de/

[10]Madden NFL 11: https://www.easports.com/madden-nfl

A.3 Observations and Conclusion

Analysing the distribution of the surveyed work, it is very apparent that a majority of research that incorporates automatic evaluation of games and game content is based on models that express designer intuition and/or scientific theories. In contrast, data-driven approaches are much rarer, independent of the type of game that is addressed. Many publications use player feedback in order to evaluate their model, although not all do. A likely reason for this distribution is that there just is not enough data available to train a model successfully. It is also striking that we have found no published evaluation methods in categories OUT-IMP, OUT-EXP and PLAY-IMP at all. It is not obvious why that is and thus, these categories might be an interesting avenue of future research.

In terms of the distribution of work across the different categories, it is also noticeable that research based on the GVGAI framework is relying more heavily on outcome statistics when compared to other types of games. This can probably be explained by the fact that in case of a general evaluation function, it is not possible to introduce game-specific knowledge to the model. As a result, many of the models are based on algorithm performance profiles, which is a popular method for all types of games.

There are several other approaches that are used across multiple game genres, such as recommendation-based methods. These methods also allow for the incorporation of online feedback, resulting in mixed-initiative models. While there are a few successful approaches that rely on Monte-Carlo AI performance, many simulation-based approaches seem to include player AI that is supposed to behave similar to human players in regard to the property that is tested. One way of doing that is to use intrinsically motivated or utility-based agents, that also have been used to provide qualitative feedback on games. The downside of all these approaches is of course that it is difficult to control the error that is introduced by inaccurate models for player AI.

Automatic game evaluation often targets properties that can be determined objectively, such as playability or the exploitability of a ruleset. A significant subset of representation-based game evaluation methods also uses a graph- or heuristics-based representation that encodes possible solutions to the game. This obviously facilitates evaluation, especially when assessing playability.

Another common approach is to use concepts from other fields, especially when evaluating aesthetics. Weighted sums are very popular as well, and commonly used in methods based on encoding and gameplay data. The measures might in some cases appear arbitrary at first, but there are noticeable similarities between the abstract concepts they are intended to measure. Examples of these concepts are fairness, at what point a winner is determined and difficulty, which are also commonly addressed in game design literature. It is also apparent that many successful publications combine multiple approaches to game evaluation.

A disconcerting pattern, however, is that many approaches are not tested exhaustively in terms of potential errors inherent to the chosen method and how biases are propagated inside an application. We hope to see more work in this regard in the future. In the following section, we take a first step in this direction by discussing which different types of errors and biases can occur for different game evaluation methods.

APPENDIX B

Case Study on Data Generation bias

In the following, we specifically address the uncertainties addressed by basing the evaluation on data generated from AI instead of human playthroughs. We will be calling this type of uncertainty *Data Generation bias* in the following. We choose to discuss Data Generation bias further, as it is an issue that occurs in most game evaluation methods, but is rarely ever addressed (see survey in appendix A).

We therefore present a small case study on StarCraft II that demonstrates *Data Generation bias* and its effects on game optimisation. We choose StarCraft II for the example as it is a well-researched and at the same time popular game with an interesting complexity. Additionally, both AI and human player data is available for the game in sufficient quantity after the release of the the *StarCraft II Learning Environment (SC2LE)* [144]. The case study is based on the StarCraft II winner prediction problem described in section 2.4.3.

In the following, we first present the data this study is based on. Following that, we perform a descriptive analysis of the datasets in order to characterise them. Finally, we compute winner prediction models and assess the effect of Data Generation bias on their performance.

B.1 Acquired Data

In order to investigate Data Generation bias, we have obtained three different playthrough datasets:

- LADDER: 4955 1v1 ladder games human vs. human player randomly sampled from publicly available replay packs. Ladder games count towards a player's ranking, which one generally seeks to improve.
- WCS: 419 1v1 games human vs. human player, played during the World Championship Series (WCS) tournament in Leipzig, Germany, January 26th-28th 2018[1].
- AI: 651 1v1 games ai vs. ai player from the StarCraft II AI ladder[2]

It is important to note that, while the players whose games are contained in the LADDER dataset are not absolute beginners, their proficiency is expected to differ significantly from the (semi-)professional players in a WCS tournament. (Non-cheating) AI players

[1]https://wcs.starcraft2.com/en-us/tournament/3895/

[2]http://sc2ai.net/

are considered to be less proficient than most human players. This is one of the reason why the StarCraft AI competitions and the SC2LE continue to have traction.

After pre-processing as described in section 2.4.3, we are left with 4410 LADDER games, 419 WCS games and 651 AI games. For each of the games in the datasets, we collect several features that describe player progress. For the purposes of the following analysis, we only consider the values at the very last game tick. The features we were able to collect and use for further analysis are listed along with their interpretation in the following.

General features (metadata)

Map name	unique identifier of map
Race	Protoss, Terran, Zerg
Result	win, loss, tie, undecided
APM	actions per minute
Game duration	number of game ticks

The collected features contain some general information like the name of the map the match was played on, and how many game ticks it lasted. Additionally, the assigned race and result of both players are saved along with the *actions per minute* (APM) statistic.

Resource features (stats)

collection rate	resources collected per minute
current	number of unspent resources
used	number of spent resources (total)
killed	opponent units, buildings destroyed by player
lost	own units, buildings destroyed
friendly fire	own units, buildings destroyed by player

All resource features are available for both minerals and vespene gas separately and measured in these resource units. They describe the collection status of the respective resource, as well as building and units expressed in terms of their resource costs for an aggregated measure. The last four features are divided into three more categories (economy, army and technology) that indicate the type of building or unit the resource was spent on. Expert players are able to identify player strategy and progress based on these resource features.

B.2 Descriptive Analysis

All experiments are performed on data only and using a Kolmogorov-Smirnov test [25]. We are using the standard confidence level of $\alpha = 0.05$ for all tests. The results are supported visually by histograms and barplots similar to the one displayed in figure B.1. The figures plot the value distribution of a specific feature given in the title (e.g. vespene gas used for technology, i.e. used_vespene_technology, in figure B.1). Only relative frequencies are displayed to allow a comparison between datasets of different sizes. The

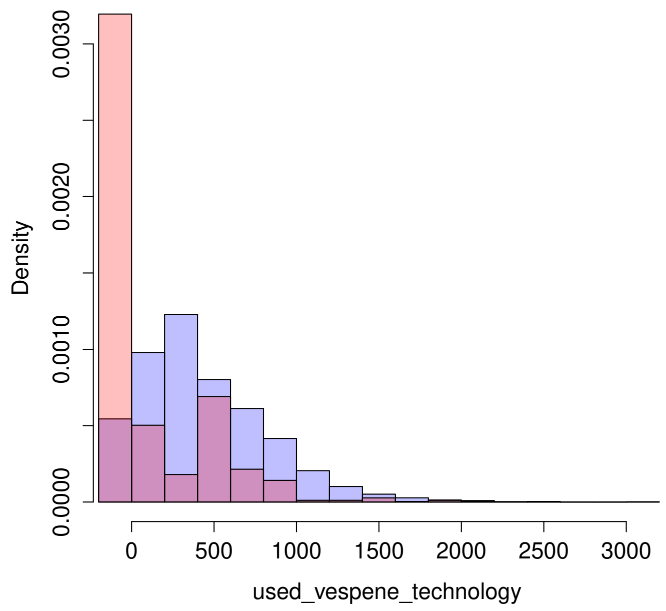


Figure B.1: Histogram of vespene gas used for technology. Values from the LADDER dataset are displayed in blue and AI in red.

different datasets visualised in the figures are colour coded (blue: LADDER, red: AI) and can overlap, resulting in a purple colouring.

While the plots show the complete datasets, i.e. feature values for both players in a game for the sake of completeness, the tests are done only on feature vectors where the corresponding player won the game. This ensures that the data is independently distributed as is assumed by the Kolmogorov-Smirnov Test. It should also make the values of the features more comparable, especially since there are considerably more undecided or tied games in the AI datasets that produce outliers, especially in terms of game duration.

The following experiment is intended to assess the influence of player modelling errors. Our usecase is that we want to use the feature measurement on the AI dataset to predict how human players play the game and find possible correlations between both data sets. Interesting features in this regard would be the APM, game duration and all resource statistics.The results are visualised in figures B.2 and B.3.

We observe that, while human players seem to play the races about equally (with slightly more Terran players, cf. figure B.2 upper left part), research seems to focus on Zerg players. There is also a striking difference in terms of game result, since 20% of AI games end without a winner (cf. figure B.2 upper right part), which is very unusual for games played by humans (LADDER as well as WCS). It is interesting to see that apm does not translate to proficiency in case of AI players, as they on average clearly perform

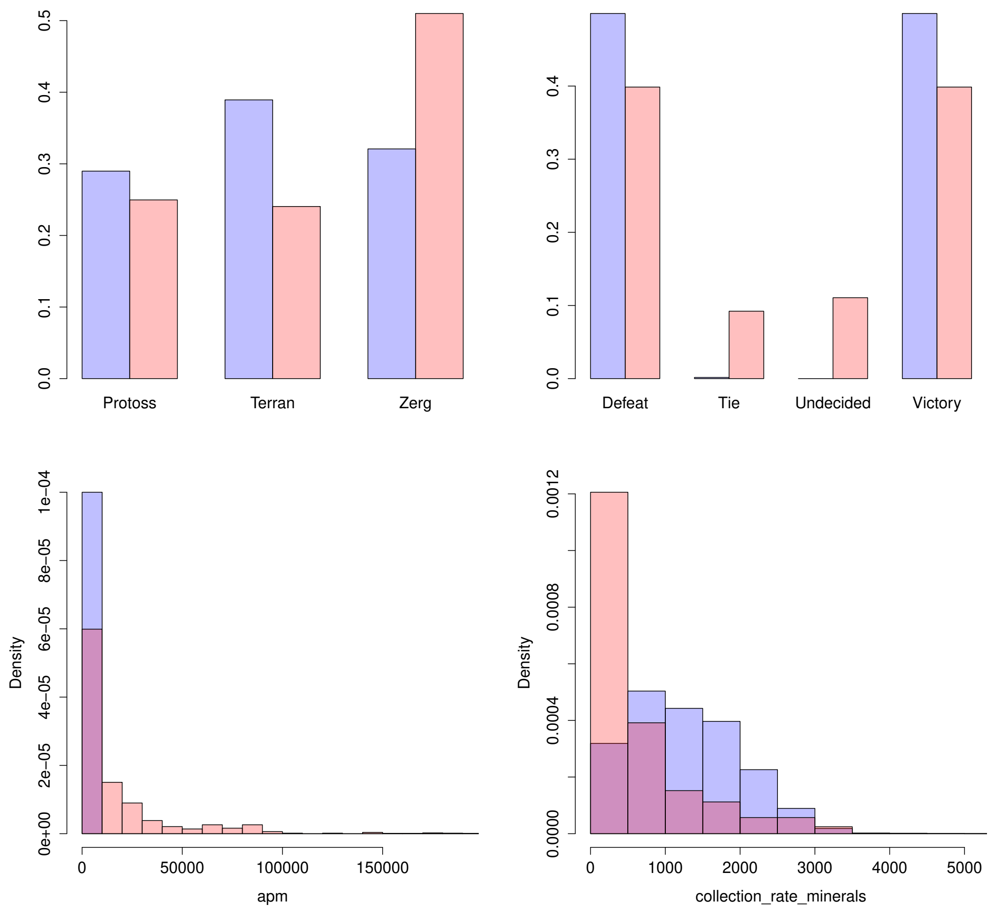


Figure B.2: Comparison of features in LADDER and AI. Dataset LADDER is displayed in blue, AI in red. Features displayed from left to right, top to bottom are assigned race, result, apm and mineral collection rate.

worse than human players from LADDER, i.e. less actions per minute as can be seen in figure B.2, left column, second row. This is most likely due to the fact that AIs will often execute actions that are not meaningful, especially if exploratory algorithms are used.

With only very few exceptions, the features from the AI and LADDER datasets are significantly differently distributed. We show some examples of differently distributed features, namely mineral collection rate (figure B.2, second row right), vespene gas used for technology and minerals used for economy (figure B.3, first row left and right, respectively) as well as minerals used for technology (figure B.3, second row left). As a counterexample, we also add vespene gas lost by economy (figure B.3, second row right).

The observations described above from the visual comparison of distributions presented in figures B.2 and B.3 are also strongly supported by the corresponding p-values received from the Kolmogorov-Smirnov test (see above). The hypothesis that both datasets share the same cumulative distribution function is rejected in almost all cases.

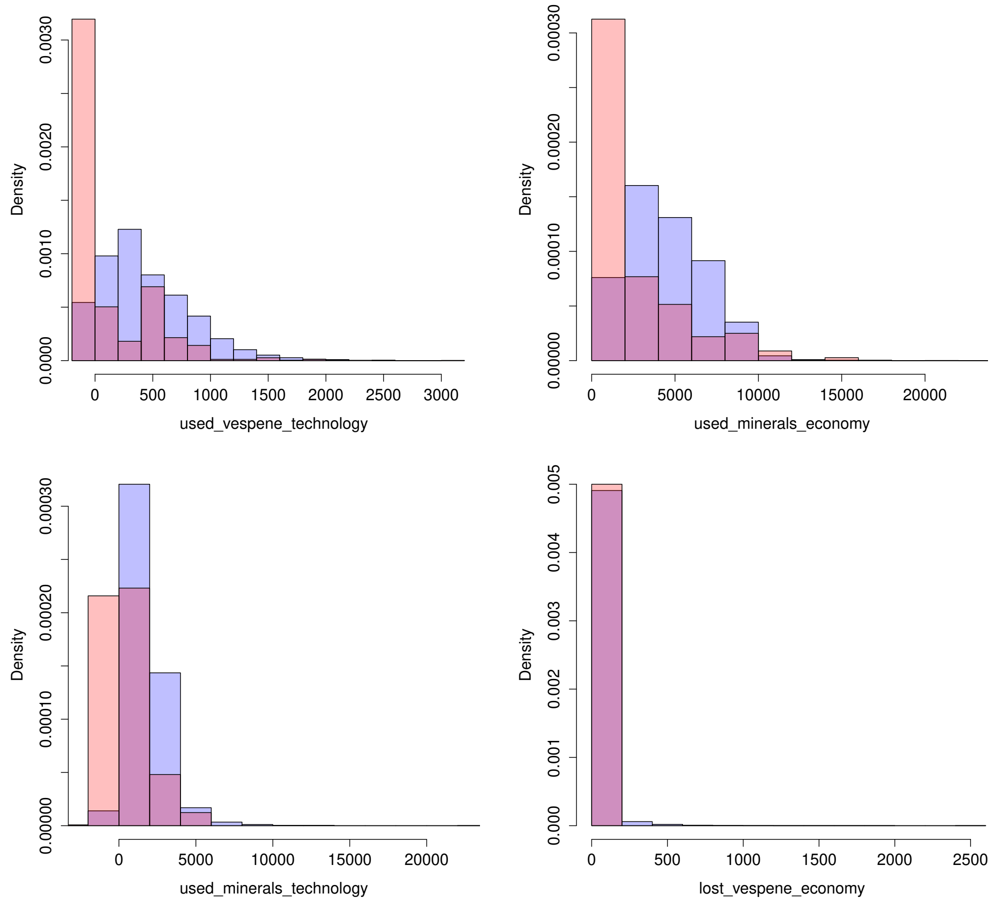


Figure B.3: Comparison of features in LADDER and AI. Dataset LADDER is displayed in blue, AI in red. Features displayed from left to right, top to bottom are minerals spent on army, economy and technology, respectively. The last graphic in the lower right corner shows economy lost measured in vespene gas.

Typical p-values received were $9.592 \cdot 10^{-14}$ for the chosen race, or less than $2.2 \cdot 10^{-16}$ for apm, minerals collection rate, vespene gas used for technology, or minerals used for technology. Otherwise, the hypothesis was accepted for economy lost measured in vespene gas (last row right in figure B.3) with a p-value of 0.7116. We thus conclude that our datasets follow different distributions.

B.3 Results

Having shown that the datasets in fact have different characteristics, suggesting different behaviours of human and AI players, in the following we investigate how these difference might affect game optimisation. We thus analyse the performance of an ANN trained on the datasets to predict the winner of a game. The ANN used is very simple (1 hidden layer, 10 neurons) and is given all collected features (see section B.1) except for the winner as an input. We report the mean and standard deviation of the prediction accuracies observed in 30 independent tests.

For a baseline, we first train a predictor separately on each of the datasets using cross-validation and a 90/10 split. Results are presented in table B.1. The obtained mean accuracies are very high, with a small standard deviation. This was expected, as the data describes the gamestate at the end of the game.

Table B.1: Winner prediction accuracy of the baseline experiment with a predictor trained on each data set separately. Mean values (`mean`) and standard deviations (`SD`) are provided.

dataset	mean	SD
ladder	0.939531	0.008019
AI	0.975128	0.014095
WCS	0.920238	0.035114

However, in automatic game evaluation, the model would be trained on artificially generated data, but applied to predict the winner of human vs. human matches, or vice versa. This would be necessary if e.g. a trained predictor is used to determine the frustration of a player. We thus conduct a second set of experiments, where we train the ANN on one data set and test it on a different one. In table B.2, we list the different combinations of training and test set in the first two columns as well as the obtained mean prediction accuracy and standard deviation in the last two columns.

Table B.2: Winner prediction accuracy received from second experiment where different datasets were considered for training and tests. Mean values (`mean`) and standard deviations (`SD`) are provided.

learned on	tested on	mean	SD
ladder	ai	0.529391	0.059723
ai	ladder	0.510401	0.011530
ladder	wcs	0.950040	0.006451
wcs	ladder	0.869531	0.019162

We observe that in the experiments involving AI playthroughs, the trained predictors achieve prediction accuracies of around 0.5. In case of the LADDER dataset, this is barely better than chance as there are almost no undecided or tied games. However, as a much more accurate predictor is possible for the datasets separately (cf. table B.1), we conjecture that different features are indicators for the outcome in the two datasets.

In order to investigate whether this observation is due to the artificially generated data, we repeat the experiment with the WCS dataset instead of AI. The players in the WCS are much more experienced and should behave significantly differently, including usage of different strategies. Despite this, the trained predictors still achieve relatively high accuracies. The prediction accuracy on the WCS dataset was even improved when compared to the baseline experiment. This might be due to the small number of games in WCS. This conjecture would also be supported by the fact that the WCS predictor has the highest standard deviation in table B.1. Even the predictor trained on WCS data is able to achieve 86% accuracy on the LADDER dataset.

These results indicate a substantial difference in behaviour between human and AI players. Approaches that rely exclusively on AI data (like self-play) should thus carefully be tested in terms of the error they induce. If it is forbiddingly large, as in our example, one way to reduce it would be to incorporate additional data that reflect human behaviour.

Another alternative would be to train a mapping function that is able to translate the features observed in the artificially generated dataset to the ones observed in real-world data. The mapping function (e.g. a transition matrix) could be specified using an EA that minimises the multivariate statistical distance between the mapped AI data and the target data (e.g. Energy distance). However, depending on the application, such a function does not necessarily exist and would introduce another, albeit measurable, source or error.

Since we have not tested different applications, we cannot generalise our findings to all instances where artificially generated data was the sole input to natural computing methods. However, we have provided a counter-example, thus demonstrating the need of a careful evaluation of the different errors introduced.

Bibliography

[1] C. C. Aggarwal, A. Hinneburg and D. A. Keim. 'On the Surprising Behavior of Distance Metrics in High Dimensional Space'. In: *Database Theory — (ICDT)*. Ed. by J. V. den Bussche and V. Vianu. Springer, Berlin, 2001, pp. 420–434.

[2] Y. Akimoto, A. Auger and N. Hansen. 'Continuous Optimization and CMA-ES'. In: *Companion of Genetic and Evolutionary Computation Conference (GECCO)*. ACM Press, New York, 2015, pp. 313–344.

[3] G. Andrade et al. 'Challenge-sensitive action selection: An application to game balancing'. In: *IEEE / WIC / ACM Intelligent Agent Technology*. IEEE Press, Piscataway, NJ, 2005, pp. 194–200.

[4] I. Andrianakis et al. 'Bayesian history matching of complex infectious disease models using emulation: A tutorial and a case study on HIV in Uganda'. In: *PLoS Computational Biology* 11.1 (2015), pp. 1–18.

[5] D. Apeldoorn and V. Volz. 'Measuring Strategic Depth in Games Using Hierarchical Knowledge Bases'. In: *IEEE Computational Intelligence and Games (CIG)*. IEEE Press, Piscataway, NJ, 2017, pp. 9–16.

[6] M. Arjovsky, S. Chintala and L. Bottou. 'Wasserstein Generative Adversarial Networks'. In: *International Conference on Machine Learning (ICML), Proceedings of Machine Learning Research (PLMR)*. Ed. by Y. W. T. Doina Precup. Vol. 70. http://proceedings.mlr.press/v70/, (accessed 12. Jan. 2019). 2017, pp. 214–223.

[7] P. Auer. 'Using confidence bounds for explotation-exploration trade-offs'. In: *Journal of Machine Learning Research* 3 (2002), pp. 397–422.

[8] P. Auer, N. Cesa-Bianchi and P. Fischer. 'Finite-time analysis of the multiarmed bandit problem'. In: *Machine Learning* 47.2-3 (2002), pp. 235–256.

[9] D. Barber. *Bayesian reasoning and machine learning*. Cambridge University Press, Cambridge, UK, 2012.

[10] T. Beielstein and S. Markon. 'Threshold selection, hypothesis tests, and DOE methods'. In: *IEEE Congress on Evolutionary Computation (CEC)*. IEEE Press, Piscataway, NJ, 2002, pp. 777–782.

[11] N. (Beume) Hochstrate, B. Naujoks and M. Emmerich. 'SMS-EMOA: Multiobjective Selection Based on Dominated Hypervolume'. In: *European Journal of Operational Research* 181.3 (2007), pp. 1653–1669.

[12] P. Bojanowski et al. 'Optimizing the Latent Space of Generative Networks'. In: *CoRR* abs/1707.05776 (2017). arXiv: 1707.05776.

[13] P. Bontrager, J. Togelius and N. D. Memon. 'DeepMasterPrint: Generating Fingerprints for Presentation Attacks'. In: *CoRR* abs/1705.07386 (2017). arXiv: 1705.07386.

[14] P. Bontrager et al. 'Deep Interactive Evolution'. In: *Computational Intelligence in Music, Sound, Art and Design (EvoMUSART)*. Ed. by A. Liapis, J. J. R. Cardalda and A. Ekárt. Springer, Cham, Switzerland, 2018, pp. 267–282.

[15] D. Brockhoff et al. 'Biobjective Performance Assessment with the COCO Platform'. In: *CoRR* abs/1605.01746 (2016). arXiv: 1605.01746.

[16] D. Browder. 'The Game Design of STARCRAFT II: Designing an E-Sport'. In: *Game Developers Conference (GDC)*. http://www.gdcvault.com/play/1014488/The-Game-Design-of-STARCRAFT (accessed 12. Jan. 2019). 2011.

[17] C. Browne et al. 'A Survey of Monte Carlo Tree Search Methods'. In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1 (2014), pp. 1–43.

[18] C. B. Browne. 'Automatic generation and evaluation of recombination games'. PhD Thesis. Queensland University of Technology, Brisbane, Australia, 2008.

[19] R. Canaan et al. 'Towards Game-based Metrics for Computational Co-creativity'. In: *IEEE Computational Intelligence and Games (CIG)*. IEEE Press, Piscataway, NJ, 2018.

[20] A. Canossa and G. Smith. 'Towards a Procedural Evaluation Technique: Metrics for Level Design'. In: *Foundations of Digital Games (FDG)*. http://www.fdg2015.org/proceedings.html (accessed 12. Jan. 2019). 2015.

[21] A. B. Cardona et al. 'Open Trumps, a Data Game'. In: *Foundations of Digital Games (FDG)*. http://www.fdg2014.org/proceedings.html (accessed 12. Jan. 2019). Society for the Advancement of the Science of Digital Games, Santa Cruz, CA, 2014.

[22] L. J. Chen, B. C. Wang and W. Y. Zhu. 'The design of puzzle selection strategies for ESP-like GWAP systems'. In: *IEEE Transactions on Computational Intelligence and AI in Games* 2.2 (2010), pp. 120–130.

[23] T. Chugh et al. 'Towards Better Integration of Surrogate Models in Optimizers'. In: *High-Performance Simulation Based Optimization*. Ed. by T. Bartz-Beielstein, B. Filipic and E. Korosec. Springer, 2019, To appear.

[24] C. C. Coello, G. Lamont and D. Veldhuizen. *Evolutionary Algorithms for Solving Multi-objective Problems*. 2nd ed. Springer, New York, 2007.

[25] W. J. Conover. *Practical nonparametric statistics*. 3rd ed. Wiley series in probability and statistics. Wiley, New York, 1999.

[26] M. Cook and S. Colton. 'Multi-Faceted Evolution of Simple Arcade Games'. In: *IEEE Computational Intelligence in Games (CIG)*. IEEE Press, Piscataway, NJ, 2011, pp. 289–296.

[27] I. Couckuyt, D. Deschrijver and T. Dhaene. 'Fast calculation of multiobjective probability of improvement and expected improvement criteria for Pareto optimization'. In: *Journal of Global Optimization* 60 (2014), pp. 575–594.

[28] S. J. Daniels et al. ‘A Suite of Computationally Expensive Shape Optimisation Problems Using Computational Fluid Dynamics’. In: *Parallel Problem Solving from Nature (PPSN XV)*. Ed. by A. Auger et al. Springer, Cham, Switzerland, 2018, pp. 296–307.

[29] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, Chichester, UK, 2001.

[30] K. Deb et al. ‘A fast and elitist multiobjective genetic algorithm: NSGA-II’. In: *IEEE Transactions on Evolutionary Computation* 6.2 (2002), pp. 182–197.

[31] K. Deb et al. ‘Scalable Test Problems for Evolutionary Multiobjective Optimization’. In: *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications*. Ed. by A. Abraham et al. Springer, London, 2005, pp. 105–145.

[32] J. Dennis and V. Torczon. *Managing approximation models in optimisation*. Tech. rep. CRPC-TR95550. Center for Research and Parallel Computation, Houston, TX, 1995.

[33] A. Drachen et al. ‘Correlation between Heart Rate, Electrodermal Activity and Player Experience in First-Person Shooter Games’. In: *5th ACM SIGGRAPH Symposium on Video Games*. Ed. by S. N. Spencer. ACM Press, New York, 2010.

[34] A. Drachen et al. ‘Game Data Mining’. In: *Game Analytics*. Ed. by M. S. El-Nasr, A. Drachen and A. Canossa. Springer, London, 2013, pp. 205–253.

[35] S. Droste and D. Wiesmann. ‘Metric Based Evolutionary Algorithms’. In: *Genetic Programming*. Ed. by R. Poli et al. Springer, Berlin, 2000, pp. 29–43.

[36] M. Emmerich, N. (Beume) Hochstrate and B. Naujoks. ‘An EMO Algorithm Using the Hypervolume Measure as Selection Criterion’. In: *Evolutionary Multi-Criterion Optimization (EMO)*. Ed. by C. A. Coello Coello, A. Hernández Aguirre and E. Zitzler. Springer, Berlin, 2005, pp. 62–76.

[37] M. Emmerich, K. Giannakoglou and B. Naujoks. ‘Single- and Multi-objective Evolutionary Optimization Assisted by Gaussian Random Field Metamodels’. In: *IEEE Transactions on Evolutionary Computation* 10.4 (2006), pp. 421–439.

[38] M. Emmerich et al. ‘Metamodel-Assisted Evolution Strategies’. In: *Parallel Problem Solving from Nature (PPSN VII)*. Ed. by J. J. M. Guervós et al. Springer, London, UK, 2002, pp. 361–370.

[39] A. Forrester, A. Sobester and A. Keane. *Engineering design via surrogate modelling*. Wiley, Chichester, UK, 2008.

[40] A. I. Forrester, A. Sóbester and A. J. Keane. ‘Multi-fidelity optimization via surrogate modelling’. In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 463.2088 (2007), pp. 3251–3269.

[41] T. Friedrichs et al. ‘Simple Games – Complex Emotions: Automated Affect Detection Using Physiological Signals’. In: *International Conference on Entertainment Computing (ICEC)*. Ed. by K. Chorianopoulos et al. Springer, Cham, Switzerland, 2015, pp. 375–382.

[42] T. Fullerton. *Game Design Workshop*. 3rd ed. CRC Press, Boca Raton, FL, 2014.

[43] S. Giannatos et al. 'Generating narrative action schemas for suspense'. In: *Workshop on Intelligent Narrative Technologies*. 2012, pp. 8–13.

[44] I. Goodfellow et al. 'Generative Adversarial Nets'. In: *Neural Information Processing Systems 27 (NIPS)*. Ed. by Z. Ghahramani et al. Curran Associates, Red Hook, NY, 2014, pp. 2672–2680.

[45] C. Grimme, P. Kerschke and H. Trautmann. 'Multimodality in Multi-Objective Optimization – More Boon than Bane?' In: *Evolutionary Multi-Objective Optimization (EMO)*. (accepted for publication). 2019.

[46] C. Guckelsberger et al. 'Predicting Player Experience without the Player: An Exploratory Study'. In: *Annual Symposium on Computer-Human Interaction in Play (CHI PLAY)*. ACM Press, New York, 2017, pp. 305–315.

[47] C. Guerrero-Romero, S. M. Lucas and D. Perez-Liebana. 'Using a Team of General AI Algorithms to Assist Game Design and Testing'. In: *IEEE Computational Intelligence and Games (CIG)*. IEEE Press, Piscataway, NJ, 2018.

[48] I. Gulrajani et al. 'Improved Training of Wasserstein GANs'. In: *Neural Information Processing Systems 30 (NIPS)*. Ed. by I. Guyon et al. Curran Associates, Red Hook, NY, 2017, pp. 5767–5777.

[49] N. Hansen and A. Ostermeier. 'Completely derandomized self-adaptation in evolution strategies'. In: *Evolutionary Computation* 9.2 (2001), pp. 159–195.

[50] N. Hansen. 'The CMA Evolution Strategy: A Tutorial'. In: *CoRR* abs/1604.00772 (2016). arXiv: 1604.00772.

[51] N. Hansen et al. 'COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting'. In: *CoRR* abs/1603.08785 (2016). arXiv: 1603.08785.

[52] N. Hansen et al. 'COCO: Performance Assessment'. In: *CoRR* abs/1605.03560 (2016). arXiv: 1605.03560.

[53] N. Hansen et al. *Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions*. Research Report RR-6829. INRIA, Paris, France, 2009.

[54] E. J. Hastings, R. K. Guha and K. O. Stanley. 'Automatic content generation in the Galactic Arms Race video game'. In: *IEEE Transactions on Computational Intelligence and AI in Games* 1.4 (2009), pp. 245–263.

[55] S. P. Hernandez, V. Bulitko and M. Spetch. 'Keeping the Player on an Emotional Trajectory in Interactive Storytelling'. In: *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*. AAAI Press, Palo Alto, CA, 2015, pp. 65–71.

[56] C. Holmgård et al. 'Automated Playtesting with Procedural Personas through MCTS with Evolved Heuristics'. In: *CoRR* abs/1802.06881 (2018). arXiv: 1802.06881.

[57] C. Holmgard et al. ‘Personas versus Clones for Player Decision Modeling’. In: *International Conference on Entertainment Computing (ICEC)*. Ed. by Y. Pisan, N. M. Sgouros and T. Marsh. Springer, Berlin, 2014, pp. 159–166.

[58] B. Horn et al. ‘A Comparative Evaluation of Procedural Level Generators in the Mario AI Framework’. In: *Foundations of Digital Games (FDG)*. http://www.fdg2014.org/proceedings.html (accessed 12. Jan. 2019). Society for the Advancement of the Science of Digital Games, Santa Cruz, CA, 2014.

[59] H. Horn et al. ‘MCTS/EA hybrid GVGAI players and game difficulty estimation’. In: *IEEE Computational Intelligence in Games (CIG)*. IEEE Press, Piscataway, NJ, 2016, pp. 278–285.

[60] F. C. Hsu and J.-S. Chen. ‘A study on multi criteria decision making model: interactive genetic algorithms approach’. In: *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. Vol. 3. IEEE Press, Piscataway, NJ, 1999, pp. 634–639.

[61] C. Igel, T. Suttorp and N. Hansen. ‘Steady-State Selection and Efficient Covariance Matrix Update in the Multi-objective CMA-ES’. In: *Evolutionary Multi-Criterion Optimization (EMO)*. Ed. by S. Obayashi et al. Springer, Berlin, 2007, pp. 171–185.

[62] A. Isaksen et al. ‘Characterising Score Distributions in Dice Games’. In: *Game & Puzzle Design* 2.1 (2016). http://game.engineering.nyu.edu/projects/exploring-game-space/ (accessed 12. Jan. 2019), pp. 24–37.

[63] A. Isaksen et al. ‘Discovering Unique Game Variants’. In: *Computational Creativity and Games Workshop at the International Computational Creativity*. http://game.engineering.nyu.edu/projects/exploring-game-space/, (accessed 12. Jan. 2019). 2015.

[64] A. Jaffe. ‘Understanding Game Balance with Quantitative Methods’. PhD Thesis. University of Washington, Seattle, WA, 2013.

[65] Y. Jin. ‘A Comprehensive Survey of Fitness Approximation in Evolutionary Computation’. In: *Soft Computing* 9.1 (2005), pp. 3–12.

[66] Y. Jin. ‘Surrogate-assisted evolutionary computation: Recent advances and future challenges’. In: *Swarm and Evolutionary Computation* 1.2 (2011), pp. 61–70.

[67] Y. Jin and J. Branke. ‘Evolutionary Optimization in Uncertain Environments - A Survey’. In: *IEEE Transactions on Evolutionary Computation* 9.3 (2005), pp. 303–317.

[68] D. Jones, M. Schonlau and W. Welch. ‘Efficient global optimization of expensive black-box functions’. In: *Journal of Global Optimization* 13.4 (1998), pp. 455–492.

[69] D. Karavolos, A. Liapis and G. N. Yannakakis. ‘Evolving Missions to Create Game Spaces’. In: *IEEE Computational Intelligence and Games (CIG)*. IEEE Press, Piscataway, NJ, 2016.

[70] D. Karavolos, A. Liapis and G. N. Yannakakis. 'Using a Surrogate Model of Gameplay for Automated Level Design'. In: *IEEE Computational Intelligence and Games (CIG)*. IEEE Press, Piscataway, NJ, 2018, pp. 277–284.

[71] K. Karpouzis and G. N. Yannakakis. *Emotion in Games: Theory and Praxis*. Springer, Cham, Switzerland, 2016.

[72] P. Kerschke. 'Automated and Feature-Based Problem Characterization and Algorithm Selection Through Machine Learning'. PhD thesis. WWU Münster, Germany, 2018.

[73] J. Knowles. 'ParEGO: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems'. In: *IEEE Transactions on Evolutionary Computation* 10.1 (2006), pp. 50–66.

[74] J. Knowles, L. Thiele and E. Zitzler. *A Tutorial on the Performance Assessment of Stochastic Multiobjective Optimizers*. TIK Report 214. Computer Engineering and Networks Laboratory (TIK), ETH Zurich, 2006.

[75] K. Kunanusont et al. 'The N-Tuple bandit evolutionary algorithm for automatic game improvement'. In: *IEEE Congress on Evolutionary Computation (CEC)*. IEEE Press, Piscataway, NJ, 2017, pp. 2201–2208.

[76] F. Lantz et al. 'Depth in Strategic Games'. In: *What's Next for AI in Games? Workshop of the Thirty-First AAAI Conference on Artificial Intelligence*. AAAI Press, Palo Alto, CA, 2017.

[77] B. Li et al. 'Cowdsourcing narrative intelligence'. In: *Advances in Cognitive Systems* 2 (2012), pp. 25–42.

[78] Y. Li et al. 'Using physiological signal analysis to design affective VR games'. In: *IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*. IEEE Press, Piscataway, NJ, 2015, pp. 57–62.

[79] A. Liapis, G. N. Yannakakis and J. Togelius. 'Adapting Models of Visual Aesthetics for Personalized Content Creation'. In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.3 (2012), pp. 213–228.

[80] A. Liapis, G. Smith and N. Shaker. 'Mixed-initiative Content Creation'. In: *Procedural Content Generation in Games*. Ed. by N. Shaker, J. Togelius and M. J. Nelson. Springer, Cham, Switzerland, 2016, pp. 195–214.

[81] A. Liapis, G. N. Yannakakis and J. Togelius. 'Designer modeling for Sentient Sketchbook'. In: *IEEE Computational Intelligence and Games (CIG)*. IEEE Press, Piscataway, NJ, 2014.

[82] A. Liapis, G. N. Yannakakis and J. Togelius. 'Towards a Generic Method of Evaluating Game Levels'. In: *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. AAAI Press, Palo Alto, CA, 2013, pp. 30–36.

[83] A. Liapis et al. 'Orchestrating Game Generation'. In: *IEEE Transactions on Games* (2018). (accepted for publication).

[84] A. Liapis et al. ‘Procedural personas as critics for dungeon generation’. In: *European Conference on Application of Evolutionary Computation (EvoApplication)*. Ed. by A. M. Mora and G. Squillero. Springer, Cham, Switzerland, 2015, pp. 331–343.

[85] P. Limbourg and D. E. Salazar Aponte. ‘An Optimization Algorithm for Imprecise Multi-Objective Problem Functions’. In: *IEEE Congress on Evolutionary Computation (CEC)*. IEEE Press, Piscataway, NJ, 2005, pp. 459–466.

[86] D. Loiacono, L. Cardamone and P. L. Lanzi. ‘Automatic Track Generation for High-End Racing games Using Evolutionary Computatation’. In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.3 (2011), pp. 245–259.

[87] P. Lopes, A. Liapis and G. N. Yannakakis. ‘Framing Tension for Game Generation’. In: *Proceedings of the Seventh International Conference on Computational Creativity (ICCC)*. Ed. by F. Pachet et al. Sony CSL, Paris, France, 2016, pp. 205–212.

[88] T. Machado et al. ‘Shopping for Game Mechanics’. In: *7th Workshop on Procedural Content Generation (PCG) within the 1st Joint International Conference of DiGRA and FDG*. http://game.engineering.nyu.edu/pcg-workshop-2016/ (accessed 12. Jan. 2019). 2016.

[89] A. Makhzani et al. ‘Adversarial Autoencoders’. In: *CoRR* abs/1511.05644 (2015). arXiv: 1511.05644.

[90] S. Markon et al. ‘Thresholding-a selection operator for noisy ES’. In: *IEEE Congress on Evolutionary Computation (CEC)*. Vol. 1. IEEE Press, Piscataway, NJ, 2001, pp. 465–472.

[91] G. Matheron. ‘Principles of Geostatistics’. In: *Economic Geology* 58 (1963), pp. 1246–1266.

[92] P. Mawhorter and M. Mateas. ‘Procedural level generation using occupancy-regulated extension’. In: *IEEE Computational Intelligence and Games (CIG)*. IEEE Press, Piscataway, NJ, 2010, pp. 351–358.

[93] M. Mckay, R. Beckman and W. Conover. ‘A comparison of three methods for selecting values of input variables in the analysis of output from a computer code’. In: *Technometrics* 42.1 (2000), pp. 55–61.

[94] B. Mikkelsen, C. Holmgård and J. Togelius. ‘Ethical Considerations for Player Modeling’. In: *What’s Next for AI in Games? Workshop of the Thirty-First AAAI Conference on Artificial Intelligence*. AAAI Press, Palo Alto, CA, 2017.

[95] M. Mlakar et al. ‘GP-DEMO: Differential Evolution for Multiobjective Optimization based on Gaussian Process models’. In: *European Journal of Operational Research* 243.2 (2015), pp. 347–361.

[96] A. Nguyen et al. ‘Plug & Play Generative Networks: Conditional Iterative Generation of Images in Latent Space’. In: *CoRR* abs/1612.00005 (2016). arXiv: 1612.00005.

[97] T. S. Nielsen et al. ‘Towards generating arcade game rules with VGDL’. In: *IEEE Computational Intelligence and Games (CIG)*. IEEE Press, Piscataway, NJ, 2015, pp. 185–192.

[98] T. S. Nielsen et al. ‘General video game evaluation using relative algorithm performance profiles’. In: *Applications of Evolutionary Computation (EvoApplications)*. Ed. by A. Mora and G. Squillero. Springer, Cham, Switzerland, 2015, pp. 369–380.

[99] J. C. Osborn, A. Grow and M. Mateas. ‘Modular Computational Critics for Games’. In: *Artificial Intelligence for Interactive Digital Entertainment Conference (AIIDE)*. AAAI Press, Palo Alto, CA, 2013, pp. 163–169.

[100] W. Ponweiser et al. ‘Multiobjective optimization on a limited budget of evaluations using model-assisted S-metric selection’. In: *Parallel Problem Solving from Nature (PPSN X)*. Ed. by G. Rudolph et al. Springer, Berlin, 2008, pp. 784–794.

[101] F. Poursabzi-Sangdeh et al. ‘Manipulating and Measuring Model Interpretability’. In: *CoRR* abs/1802.07810 (2018). arXiv: 1802.07810.

[102] E. J. Powley et al. ‘Semi-automated Level Design via Auto-Playtesting for Handheld Casual Game Creation’. In: *IEEE Computational Intelligence and Games (CIG)*. IEEE Press, Piscataway, NJ, 2016.

[103] M. Preuss, A. Liapis and J. Togelius. ‘Searching for good and diverse game levels’. In: *IEEE Computational Intelligence and Games (CIG)*. IEEE Press, Piscataway, NJ, 2014.

[104] M. Preuss et al. ‘Integrated Balancing of an RTS Game: Case Study and Toolbox Refinement’. In: *IEEE Computational Intelligence in Games (CIG)*. IEEE Press, Piscataway, NJ, 2018.

[105] F. Pukelsheim. ‘The Three Sigma Rule’. In: *The American Statistician* 48.2 (1994), pp. 88–91.

[106] R. Purshouse et al. ‘Workshops at PPSN 2018’. In: *Parallel Problem Solving from Nature (PPSN XV)*. Ed. by A. Auger et al. Springer, Cham, Switzerland, 2018, pp. 490–497.

[107] A. Radford, L. Metz and S. Chintala. ‘Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks’. In: *CoRR* abs/1511.06434 (2015). arXiv: 1511.06434.

[108] C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. MIT Press, Cambridge, MA, 2006.

[109] T. Robič and B. Filipič. ‘DEMO: Differential Evolution for Multiobjective Optimization’. In: *Evolutionary Multi-Criterion Optimization (EMO)*. Ed. by C. A. Coello Coello, A. Hernández Aguirre and E. Zitzler. Springer, Berlin, 2005, pp. 520–533.

[110] G. Rudolph. ‘A Partial Order Approach to Noisy Fitness Functions’. In: *IEEE Congress on Evolutionary Computation (CEC)*. IEEE Press, Piscataway, NJ, 2001, pp. 318–325.

[111] K. Sadowski et al. 'Exploring trade-offs between target coverage, healthy tissue sparing, and the placement of catheters in HDR brachytherapy for prostate cancer using a novel multi-objective model-based mixed-integer evolutionary algorithm'. In: *Genetic and Evolutionary Computation Conference (GECCO)*. ACM Press, New York, 2017, pp. 1224–1231.

[112] J. Schmidhuber. 'Formal theory of creativity, fun, and intrinsic motivation (1990-2010)'. In: *IEEE Transactions on Autonomous Mental Development* 2.3 (2010), pp. 230–247.

[113] J. Secretan et al. 'Picbreeder: Evolving pictures collaboratively online'. In: *SIGCHI Conference on Human factors in Computing Systems*. ACM Press, New York, 2008, pp. 1759–1768.

[114] B. Shahriari et al. 'Taking the human out of the loop: A review of Bayesian optimization'. In: *Proceedings of the IEEE* 104.1 (2016), pp. 148–175.

[115] M. Shaker et al. 'Automatic generation and analysis of physics-based puzzle games'. In: *IEEE Computational Intelligence and Games (CIG)*. IEEE Press, Piscataway, NJ, 2013, pp. 241–248.

[116] N. Shaker, J. Togelius and M. J. Nelson. *Procedural Content Generation in Games*. Springer, Cham, Switzerland, 2016.

[117] N. Shaker, G. N. Yannakakis and J. Togelius. 'Crowdsourcing the aesthetics of platform games'. In: *IEEE Transactions on Computational Intelligence and AI in Games* 5.3 (2013), pp. 276–290.

[118] N. Shaker et al. 'Evolving levels for Super Mario Bros using grammatical evolution'. In: *Conference on Computational Intelligence and Games (CIG)*. IEEE Press, Piscataway, NJ, 2012, pp. 304–311.

[119] N. Shaker et al. 'Fusing Visual and Behavioral Cues for Modeling User Experience in Games'. In: *IEEE Transactions on Cybernetics* 43.6 (2013), pp. 1519–1531.

[120] F. Silva et al. 'Search Driven Playtesting of Contemporary Board Games'. In: *Foundations of Digital Games (FDG)*. ACM Press, New York, 2017.

[121] G. Smith and J. Whitehead. 'Analyzing the expressive range of a level generator'. In: *Procedural Content Generation in Games (PCGames)*. ACM Press, New York, 2010.

[122] G. Smith, J. Whitehead and M. Mateas. 'Tanagra: Reactive planning and constraint solving for mixed-initiative level design'. In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.3 (2011), pp. 201–215.

[123] E. Snelson and Z. Ghahramani. 'Sparse Gaussian Processes Using Pseudo-inputs'. In: *Neural Information Processing Systems 18 (NIPS)*. MIT Press, Cambridge, MA, 2005, pp. 1257–1264.

[124] J. Snoek, H. Larochelle and R. P. Adams. 'Practical Bayesian optimization of machine learning algorithms'. In: *Neural Information Processing Systems 25 (NIPS)*. Vol. 2. Curran Associates, Red Hook, NY, 2012, pp. 2951–2959.

[125] W. Sombat, P. Rohlfshagen and S. M. Lucas. 'Evaluating the enjoyability of the ghosts in Ms Pac-Man'. In: *IEEE Computational Intelligence and Games (CIG)*. IEEE Press, Piscataway, NJ, 2012, pp. 379–387.

[126] N. Sorenson and P. Pasquier. 'The evolution of fun: Automatic level design through challenge modeling'. In: *Computational Creativity*. Ed. by D. Ventura et al. ACC Association for Computational Creativity. 2010, pp. 258–267.

[127] P. Spronck et al. 'Player Modeling'. In: *Artificial and Computational Intelligence in Games (Dagstuhl Seminar 12191)*. Ed. by S. M. Lucas et al. Vol. 2. 5. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2012, pp. 59–61.

[128] K. O. Stanley and R. Miikkulainen. 'Evolving Neural Networks Through Augmenting Topologies'. In: *Evolutionary Computation* 10.2 (2002), pp. 99–127.

[129] A. Summerville, S. Philip and M. Mateas. *MCMCTS PCG 4 SMB : Monte Carlo Tree Search to Guide Platformer Level Generation*. AAAI Technical Report WS-15-21, pages 68–74. AAAI Press, Palo Alto, CA, 2015, pp. 68–74.

[130] A. Summerville et al. 'Understanding Mario: An Evaluation of Design Metrics for Platformers'. In: *Foundations of Digital Games (FDG)*. ACM Press, New York, 2017.

[131] A. J. Summerville et al. 'The VGLC: The Video Game Level Corpus'. In: *7th Workshop on Procedural Content Generation (PCG) within the 1st Joint International Conference of DiGRA and FDG*. http://game.engineering.nyu.edu/pcg-workshop-2016/ (accessed 12. Jan. 2019). 2016.

[132] W. Szwoch. 'Emotion Recognition Using Physiological Signals'. In: *Mulitimedia, Interaction, Design and Innnovation (MIDI)*. ACM Press, New York, 2015, 15:1–15:8.

[133] H. Takagi. 'Active user intervention in an EC search'. In: *International Conference on Information Sciences*. http://hdl.handle.net/2324/1670068, (accessed 12. Jan. 2019). 2000, pp. 995–998.

[134] C. H. Tan, K. C. Tan and A. Tay. 'Dynamic game difficulty scaling using adaptive behavior-based AI'. In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.4 (2011), pp. 289–301.

[135] J. Togelius and J. Schmidhuber. 'An Experiment in Automatic Game Design'. In: *IEEE Symposium on Computational Intelligence and Games (CIG)*. IEEE Press, Piscataway, NJ, 2008, pp. 111–118.

[136] J. Togelius et al. 'Controllable procedural map generation via multiobjective evolution'. In: *Genetic Programming and Evolvable Machines* 14.2 (2013), pp. 245–277.

[137] J. Togelius et al. 'Search-based procedural content generation: A taxonomy and survery'. In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.3 (2011), pp. 172–186.

[138] J. Togelius et al. 'The Mario AI Championship 2009-2012'. In: *AI Magazine* 34.3 (2013), pp. 89–92.

[139] Y. Tong. *The Multivariate Normal Distribution*. Springer, New York, 1990.

[140] V. Torczon and M. Trosset. *Direct search methods: Then and now*. Tech. rep. 2000-26. NASA/CR-2000-210125. ICASE, Hamption, VA, 2000.

[141] T. Tušar et al. 'COCO: The Bi-objective Black Box Optimization Benchmarking (bbob-biobj) Test Suite'. In: *CoRR* abs/1604.00359 (2016). arXiv: arXiv:1604.00359v2.

[142] H. Ulmer, F. Streichert and A. Zell. 'Evolution Strategies assisted by gaussian processes with improved pre-selection criterion'. In: *Congress on Evolutionary Computation (CEC)*. IEEE Press, Piscataway, NJ, 2003, pp. 692–699.

[143] M. Čertický and D. Churchill. 'The Current State of StarCraft AI Competitions and Bots'. In: *Artificial Intelligence for Interactive Digital Entertainment Conference (AIIDE)*. AAAI Press, Palo Alto, CA, 2017.

[144] O. Vinyals et al. 'StarCraft II: A New Challenge for Reinforcement Learning'. In: *CoRR* abs/1708.04782 (2017). arXiv: 1708.04782.

[145] V. Volz, G. Rudolph and B. Naujoks. 'Surrogate-Assisted Partial Order-Based Evolutionary Optimisation'. In: *Evolutionary Multi-Criterion Optimization (EMO)*. Springer, Berlin, 2017, pp. 639–653.

[146] V. Volz, K. Majchrzak and M. Preuss. 'A Bottom-Up Approach to Explanations for (Game) AI'. In: *Computational Intelligence in Games (CIG)*. IEEE Press, Piscataway, NJ, 2018, pp. 474–481.

[147] V. Volz, M. Preuss and M. K. Bonde. 'Towards Embodied and Interpretable StarCraft II Winner Prediction'. In: *Computer Games Workshop at International Joint Conference on Artificial Intelligence (ICJACI)*. (in press). 2018.

[148] V. Volz, G. Rudolph and B. Naujoks. 'Demonstrating the Feasibility of Automatic Game Balancing'. In: *Genetic and Evolutionary Computation Conference (GECCO)*. ACM Press, New York, 2016, pp. 269–276.

[149] V. Volz, G. Rudolph and B. Naujoks. 'Investigating Uncertainty Propagation in Surrogate-Assisted Evolutionary Algorithms'. In: *Genetic and Evolutionary Computation Conference (GECCO)*. ACM Press, New York, 2017, pp. 881–888.

[150] V. Volz et al. 'Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network'. In: *Genetic and Evolutionary Computation Conference (GECCO)*. ACM Press, New York, 2018, pp. 221–228.

[151] V. Volz et al. 'Gameplay Evaluation Measures'. In: *Artificial and Computational Intelligence in Games: AI-Driven Game Design (Dagstuhl Seminar 17471)*. Ed. by P. Spronck et al. Vol. 7. 11. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2018, pp. 122–125.

[152] T. Voß, N. Hansen and C. Igel. 'Improved Step Size Adaptation for the MO-CMA-ES'. In: *Genetic and Evolutionary Computation Conference (GECCO)*. ACM Press, New York, 2010, pp. 487–494.

[153] T. Wagner and H. Trautmann. 'Online convergence detection for evolutionary multi-objective algorithms revisited'. In: *IEEE Congress on Evolutionary Computation (CEC)*. IEEE Press, Piscataway, NJ, 2010, pp. 1–8.

[154] B. G. Weber et al. 'Modeling Player Retention in Madden NFL 11'. In: *AAAI Innovative Applications of Artificial Intelligence*. AAAI Press, Palo Alto, CA, 2011.

[155] S. Wessing and M. Preuss. 'The true destination of EGO is multi-local optimization'. In: *IEEE Latin American Conference on Computational Intelligence (LA-CCI)*. IEEE Press, Piscataway, NJ, 2017, pp. 1–6.

[156] B. G. Woolley and K. O. Stanley. 'On the deleterious effects of a priori objectives on evolution and representation'. In: *Genetic and Evolutionary Computation Conference (GECCO)*. ACM Press, New York, 2011, pp. 957–964.

[157] G. N. Yannakakis. 'AI in computer games: generating interesting interactive opponents by the use of evolutionary computation'. PhD thesis. University of Edinburgh, Scotland, UK, 2005.

[158] G. N. Yannakakis and J. Togelius. 'Experience-Driven Procedural Content Generation'. In: *IEEE Transactions on Affective Computing* 2.3 (2011), pp. 147–161.

[159] G. N. Yannakakis and J. Hallam. 'Entertainment Modeling in Physical Play Through Physiology Beyond Heart-Rate'. In: *Affective Computing and Intelligent Interaction (ACII)*. Ed. by A. C. R. Paiva et al. Springer, Berlin, 2007, pp. 254–265.

[160] G. N. Yannakakis and J. Togelius. 'A Panorama of Artificial and Computational Intelligence in Games'. In: *IEEE Transactions on Computational Intelligence and AI in Games* 7.4 (2015), pp. 317–335.

[161] G. N. Yannakakis and J. Togelius. *Artificial Intelligence and Games*. Springer, Cham, Switzerland, 2018.

[162] E. Zitzler and L. Thiele. 'Multiobjective optimization using evolutionary algorithms — A comparative case study'. In: *Parallel Problem Solving from Nature (PPSN V)*. Ed. by A. E. Eiben et al. Springer, Berlin, 1998, pp. 292–301.

List of Tables

LIST OF FIGURES