Single- and Multi-Objective Game-Benchmark for Evolutionary Algorithms

Vanessa Volz
Queen Mary University of London, UK
v.volz@qmul.ac.uk

Boris Naujoks
TH Köln - University of Applied Sciences, Germany
boris.naujoks@th-koeln.de

Pascal Kerschke
University of Münster, Germany
kerschke@uni-muenster.de

Tea Tušar
Jožef Stefan Institute, Slovenia
tea.tusar@ijs.si

ABSTRACT

Despite a large interest in real-world problems from the research field of evolutionary optimisation, established benchmarks in the field are mostly artificial. We propose to use game optimisation problems in order to form a benchmark and implement function suites designed to work with the established COCO benchmarking framework. Game optimisation problems are real-world problems that are safe, reasonably complex and at the same time practical, as they are relatively fast to compute. We have created four function suites based on two optimisation problems previously published in the literature (*TopTrumps* and *MarioGAN*). For each of the applications, we implemented multiple instances of several scalable single- and multi-objective functions with different characteristics and fitness landscapes. Our results prove that game optimisation problems are interesting and challenging for evolutionary algorithms.

CCS CONCEPTS

• Theory of computation → Discrete optimization; **Continuous optimization**; **• Applied computing → Computer games**;

KEYWORDS

Benchmarking, evolutionary algorithms, games, single- and multi-objective optimisation

ACM Reference Format:
Vanessa Volz, Boris Naujoks, Pascal Kerschke, and Tea Tušar. 2019. Single- and Multi-Objective Game-Benchmark for Evolutionary Algorithms. In *Genetic and Evolutionary Computation Conference (GECCO '19), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3321707.3321805

1 INTRODUCTION

Artificial problems often exhibit vastly different characteristics than actual real-world problems. For example, artificially created functions usually have a discernible global structure, which is not necessarily true for real-world problems. Applying an evolutionary algorithm (or some alternative optimisation method) that has only been tested and evaluated on artificial test functions will thus probably lead to sub-optimal results on real-world applications.

The game-benchmark for evolutionary algorithms (GBEA) described in this paper was specifically designed to provide the means to analyse and compare the performance of evolutionary algorithms on non-artificial problems. The benchmark is comprised of several *game optimisation* problems, all of which are examples of previously published search-based procedural content generation approaches [14]. They are integrated as functions suites into the COCO (COmparing Continuous Optimisers) benchmarking framework [5]. GBEA is under active development and meant to be continuously extended with new problem suites from publications related to game optimisation[1].

The reason for developing a novel benchmark is the lack of real-world benchmarks for evolutionary algorithms that measure anytime performance and provide sufficient post-processing features. 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 very complex systems, but their true state is always fully observable. This is a contrast to problems that rely on real-world measurements such as described in [2].
(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[2] make large datasets available[3] 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 providing challenging benchmarks for evolutionary algorithms, this study is also important in the context of games research. The characteristics and complexity of game optimisation problems are rarely considered in research, and data from the benchmark can provide important insights into this type of problems (e.g., regarding the choice of search algorithm).

[1] http://norvig.eecs.qmul.ac.uk/gbea
[2] see recent successes of OpenAI's DotA AI https://openai.com/five/
[3] e.g. for StarCraft II [19] or League of Legends https://developer.riotgames.com/

GECCO '19, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-6111-8/19/07…$15.00
https://doi.org/10.1145/3321707.3321805

We first identify requirements for benchmarks at the end of Section 1. Then, some often used benchmarks in the field of evolutionary computation are described in Section 2, followed by a description of how the requirements are achieved, along with more technical details on our game-benchmark (Section 3). First insights into the presented functions are given in Section 4. Section 5 concludes our contribution with a summary and future work directions.

Requirements

We identified the following requirements for the benchmark:

R-I: Problem characteristics. Problems should not be artificial in nature. The benchmark should contain a diverse set of fitness functions which are expected to make sense within their real-world context. Fitness functions should be of considerable complexity and possibly involve different types of realistic uncertainties.

R-II: Practicality. Executing 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 times on standard machines.

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

R-IV: Investigation of scaling behaviour. Functions should be scalable in search space, so that scaling behaviour can be analysed.

2 RELATED WORK

The popular `bbob` test suite [6] from the COCO framework [5] includes 24 common test functions with diverse characteristics, including, e.g., sphere and linear, as well as Schwefel and Rosenbrock functions. The same functions are combined to form the bi-objective function suite `bbob-biobj` [18]. For multi-objective problems with larger dimensions, DTLZ [3] and ZDT [23] are popular test suites. Recent visualisation approaches revealed that popular multi-objective benchmarks mostly contain problems with very simple fitness landscapes [4]. 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 [4]. 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 evaluations, such as surrogate-assisted evolutionary algorithms. According to [2] and [16], these benchmarks are rare because real-world problems in relevant publications are mostly proprietary in nature. The shortage of benchmark problems could also not be solved by the Black Box Optimization Competition (BBComp)[4], which includes expensive as well as bi-objective problems. BBComp is set up as a competition rather than a benchmark, which means that it is impractical to analyse (R-II, R-III, R-IV).

Recently, efforts have been made to tackle these issues. For example, in [2], three real-world problems involving *computational fluid dynamics* (CFD) are compiled into a benchmark for computationally expensive optimisation[5]. 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 offer scalability in search space dimension and multiple instances. However, the function suite lacks features of the established benchmarks, such as the ability to estimate any-time performance as well as sophisticated post-processing (R-III). In addition, the computational effort required for the benchmark is also impractically large (R-II).

Another recent effort were workshops hosted at PPSN '18[6] and GECCO '19[7], in which the organisers suggest to use problems from the areas of machine learning and data analysis in order to compile a benchmark [11]. 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. The problems have recently been integrated into the benchmarking framework nevergrad[8]. In the future, it would be interesting to investigate similarities and differences between the contained problems, because they all come from different areas of applications.

Benchmarks 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. There are a variety of popular competitions in this field[9], among them the *general video game AI* competition. Unfortunately, there exists no systematic analysis of the problems posed in these competitions and the comparison mechanics are difficult to interpret[10]. This makes them difficult to use as an independent benchmark.

3 GAMES AND BENCHMARK FUNCTIONS

The two problems we will focus on have been published previously [21, 22] along with their respective results. According to the framework proposed in [9], both problems can be classified as *level generation methods* with *embedded input*. Both problems use automatic evaluation, but are intended to also allow for an *interactive process* with input from *human-based computation*. They were selected to stand in for the two main approaches to representing solutions (direct encoding and genotype-phenotype mapping), and because a sizeable amount of previous works exists to source the fitness functions from.

In the following, we first detail how our benchmark set-up based on the COCO framework fulfils the requirements detailed above. Following that, we give a detailed description of the implemented function suites.

3.1 Experimental Framework

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

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

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

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

[7]http://www.erc.is/go/gecco2019

[8]https://github.com/facebookresearch/nevergrad/

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

[10]http://norvig.eecs.qmul.ac.uk/gbea/gamesbench_events.html#cig18

Table 1: Overview and characterisation of functions in `rw-top-trumps`. Column [min] indicates how the fitness measure x is transformed into a minimisation problem.

fid	Fitness Measure (x)	min
t_1	deck hypervolume	$-x$
t_2	standard deviation of category averages	$-x$
t_3	winrate of better player	$-x$
t_4	switches of trick winner	$-x$
t_5	trick difference at end of game	x

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 (R-IV). Similarly, COCO expects the existence of multiple instances of any given function. COCO further automatically computes performance assessment measures based on the algorithm's aggregated performance across these instances. The resulting values are interpretable and suitable for statistical analysis (R-III).

COCO does provide some rudimentary features that allow the creation of new functions and corresponding suites. We 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 hinges on the ability to define suitable benchmarking problems. Similarly, the problem characteristics (R-I) and execution speed (R-II) also rely on the included benchmarking problems. We therefore discuss the function suites developed for GBEA below, specifically with regards to the requirements.

3.2 TopTrumps Suites

This problem is based on the card game TopTrumps and the task of generating a deck for the game.

Game Description. TopTrumps is a themed card game, where popular themes include cars, motorcycles, and aircrafts. Each card in the deck corresponds to a specific member of the theme and displays several of its characteristics. During gameplay, the deck is shuffled first and then 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.

TopTrumps Suites Details. 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 to create scalable problems (R-IV). The original publication [21] already contains diverse functions with differing numbers of objectives (R-I). Furthermore, AIs of different skill levels are already implemented (R-I).

As expected in game optimisation problems, the included functions are noisy. However, the fitness for each solution is reported as the average of 2 000 simulations, which has been shown in [21] to produce an appropriate balance between computational effort (R-II) and resulting standard deviations.

Table 2: Function suite details.

	TopTrumps		MarioGAN	
	Single-Obj.	Bi-Obj.	Single-Obj.	Bi-Obj.
dimensions	(88, 128, 168, 208) = 4 · (22, 32, 42, 52)		10, 20, 30, 40	
functions	5	3	28	10
instances	15	15	7	7
simul./point	2 000 [21]	2 000 [21]	30	30

The only remaining issue is to create suitable instances of the functions (R-III), which on 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. 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. 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. This way we create 15 different instances.

Based on the TopTrumps functions described in the original publication [21], we design a single- (`rw-top-trumps`) and a bi-objective (`rw-top-trumps-biobj`) suite. The functions are denoted t_i and T_i for the single- and bi-objective suite, respectively.

`rw-top-trumps`. Contains 5 different functions, where t_1 and t_2 are based on encoding, whereas the others are based on a simulation. An overview of the functions can be found in Table 1.

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 t_3, for example, is the winrate of the better player. The winrate is set to be maximised so that higher skill levels lead to higher winrates. In contrast, t_4 and t_5 target the tension of the game instead, as they both reach their optimum if the game was close, independent of the skill levels of the players. Function t_5 looks at the final outcome, while t_4 also considers how dramatic the playthrough was.

Functions t_1 and t_2 are computed without simulations, but still target similar concepts. If the deck hypervolume[11] (t_1) is maximised, each card is not dominated by any other one[12]. This ensures that the choice of the category matters, as there is always at least one way for each card to beat another. Maximising this function allows for tension in the game, just like expressed in t_4 and t_5. However, it also decreases the tolerance of errors and thus punishes players that are unfamiliar with the deck. Similarly, t_2, if decreased, requires more detailed knowledge of the deck in order to make these choices.

The problems are relatively large in terms of search dimension. This is because we intended to create realistic problems with a typical number of cards in the deck (i.e. 22, 32, 42 and 52) and number of categories on each card (4), see Table 2.

`rw-top-trumps-biobj`. The bi-objective function suite combines functions from the single-objective suite that are seemingly conflicting. We made this selection based on the meaning of the functions and an estimation of their conflict using linear regression on the

[11] The hypervolume generated by the deck, if each card is interpreted as a point in n-dimensional space, where n is the number of categories. $n = 4$ in our implementation.
[12] Except in unlikely edge cases.

Table 3: Construction of the bi-objective functions from the `rw-top-trumps-biobj` suite (T_i) as pairs of single-objective functions from the `rw-top-trumps` suite (t_j).

$$T_1 = (t_1, t_2) \qquad T_2 = (t_3, t_4) \qquad T_3 = (t_3, t_5)$$

function values along three diagonal walks through the decision space (see Section 4.1 for more details on these walks). We computed the correlation coefficient for each pair of functions and have chosen pairs with a negative correlation. An overview of the functions is presented in Table 3.

Function T_1 is based on t_1 and t_2, so computed directly from the encoding. It is thus significantly faster to compute than the others. However, the functions are only partly conflicting, as t_1 expresses both tension and the impact of decisions. The conflict of objectives is more obvious for functions T_2 and T_3, where the first function (t_3) targets fairness, while the second function (t_4 or t_5) targets some expression of tension in the game.

3.3 MarioGAN Suites

Game description. The objective of the game is to progress through the fictional Mushroom Kingdom to save Princess Toadstool (later called Princess Peach). This is performed by Mario, the main player character, racing through different levels while defeating enemies, collecting items and solving puzzles without dying. As a side-scrolling platformer, the player moves from the left side of the screen to the right side in order to reach exit objectives within a given time limit, and thereby continue to the next level.

MarioGAN Suites Details. The Mario game has been heavily researched in past years [15], the levels are relatively short, and there is a publicly available framework, *MarioAI*, containing various state-of-the-art AI players. The function suites are based on a recently published level generation method using Generative Adversarial Networks (GANs) [22]. 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 TopTrumps.

Another benefit of the latent vector encoding is that it allows for easily scalable functions (R-IV), 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 (R-III), as they represent different level generation models that exhibit similar characteristics. This was verified visually, as well as based on *Exploratory Landscape Analysis* [7, 10]. Hence, in order to create instances, GAN models are trained from different seeds, resulting in neural networks with different weight configurations. Furthermore, a simulation of a playthrough with a player AI in the *MarioAI* framework is capped at 20 seconds, thus also enabling practical benchmarking speeds (R-II).

Based on the MarioGAN approach and the included optimisation problems, we created a single- (`rw-gan-mario`) and a bi-objective (`rw-gan-mario-biobj`) function suite, and denoted the functions contained therein m_i and M_i, respectively.

`rw-gan-mario`. We implemented a set of diverse functions, cf. Table 4. More details on these and how they are computed are given below. We have added two types of variations for each of the fitness functions. The first variation are the various models trained for the respective sets of samples (underground and overworld). An example of the different characteristics of underground and overworld levels can be seen in Figure 1. 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. This 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 [20].

Table 4: Overview of functions in `rw-gan-mario`. Function ids in the benchmark are indicated in the last four columns, divided by overworld [o], underground [u], overworld concatenated [oc] and underground concatenated [uc]. Column [min] indicates how the fitness measure x is transformed into a minimisation problem.

Fitness Measure (x)	AI	min	o	u	oc	uc
enemyDistribution	-	$-x$	1	2		
positionDistribution	-	$-x$	3	4		
decorationFrequency	-	$1 - x$	5	6		
negativeSpace	-	$1 - x$	7	8		
leniency	-	x	9	10		
basicFitness	A*	x	11	12	13	14
basicFitness	Scared	x	15	16		
airTime	A*	$1 - x$	17	18	19	20
airTime	Scared	$1 - x$	21	22		
timeTaken	A*	$1 - x$	23	24	25	26
timeTaken	Scared	$1 - x$	27	28		

enemyDistribution: standard dev. (std.) of enemy tiles (x-axis)
positionDistribution: std. of tiles you can stand on (y-axis)
decorationFrequency: percentage of *pretty tiles*[13] [13]
negativeSpace: percentage of tiles you can stand on
leniency: weighted sum of subjective *leniency* of tiles
basicFitness: MarioAI championship score[14] for AI [15]
airTime: ratio between ticks in air vs. total ticks
(if level completed, otherwise 1)
timeTaken: ratio between time taken and total time allowed
(if level completed, otherwise 1)

enemyDistribution and positionDistribution are based on statistics suggested in [13] with no directly assumed meaning. decorationFrequency is proposed as an aesthetic measure in [13]. Leniency is a weighted sum designed to express the leniency of the level design, as suggested in [12]. negativeSpace is intended to capture how much of the space in the level is traversable, as proposed in [1].

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 by R. Baumgarten, which is based on the A* algorithm [15]. 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 order to not rely on outliers when evaluating a

[13] *pretty tiles*:= {Tube, Enemy, Destructible Block, Question Block, or Bullet Bill}
[14] (lengthOfLevelPassedPhys − timeSpentOnLevel + numberOfGainedCoins + marioStatus ∗ 5000)/5000

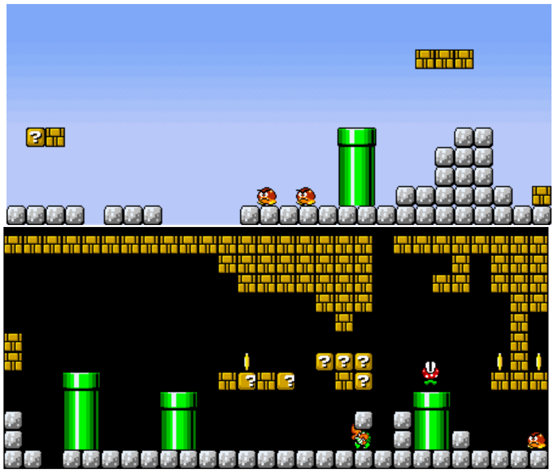


Figure 1: Examples of generated level segments. Top: overworld level. Bottom: underground level.

given solution, we executed the simulation 30 times per solution and averaged the results given by the respective fitness measure.

Fitness measures airTime and timeTaken are modified from their implementation in [22] as described in [20]. Both functions, when optimised, result in levels that take longer / more actions to complete. Both measures also evaluate to 1 (the maximum value) if the AI in question fails to complete the corresponding level. Finally, basicFitness is the MarioAI competition score for the AI agents.

The variations extend the number of problems in the suite significantly. We ran preliminary experiments on the 11 functions from Table 4 for each available combination. Based on the results, we removed functions with similar fitness landscapes and kept 28 functions as indicated in Table 4. Many of the functions with the ScaredAgent were removed as they were not interesting due to the AI failing on all levels.

The resulting single-objective suite thus contains 28 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. These specifications were motivated by observations on the original publication [22], where the 32-dimensional random vector produced a fitness landscape with large plateaus. 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 2.

`rw-gan-mario-biobj`. For the bi-objective function suite, we chose functions that are likely to conflict. Again, we base our decision on our intuition as well as the correlation coefficients computed on the diagonal walks (see Section 4.1 for more details). For example, functions that result in early failure of a Mario agent (such as basicFitness) will contradict functions that try to extend the duration and complexity of gameplay (such as airTime and timeTaken). All of the functions selected based on context knowledge were then verified to have a large negative linear correlation. The functions resulting from these combinations are listed in Table 5.

Table 5: Construction of the bi-objective functions from the `rw-gan-mario-biobj` suite (M_i) as pairs of single-objective functions from the `rw-gan-mario` suite (m_j).

$M_1 = (m_4, m_6)$	$M_6 = (m_{12}, m_{24})$
$M_2 = (m_4, m_8)$	$M_7 = (m_{13}, m_{19})$
$M_3 = (m_{11}, m_{17})$	$M_8 = (m_{13}, m_{25})$
$M_4 = (m_{11}, m_{23})$	$M_9 = (m_{14}, m_{20})$
$M_5 = (m_{12}, m_{18})$	$M_{10} = (m_{14}, m_{26})$

4 EVALUATION

As mentioned in Section 3, all functions in the four suites are justified in terms of the context of the real-world application, as all functions have been used in previous research (R-I). However, the functions in the corresponding game optimisation problems are rarely analysed and usually treated as black boxes. We seek to determine some characteristics of the functions to evaluate their complexity and novelty. These results can also be used to help the interpretation of the GBEA results. We further provide a brief overview of computational efforts required for the different functions, in order to assess the practicality of the benchmark.

The four GBEA function suites contain problems scalable in the number of variables (R-IV). However, to keep the analysis concise, we will in the following mostly conduct the experiments for a single dimension per function suite. In case of the MarioGAN suites, 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 the TopTrumps suites, we chose dimension 128, as this results in 32 cards, which is a common deck size for card games (R-I).

4.1 Diagonal Walks

In order to gain a first impression of the fitness landscape of the various functions contained in the benchmarks, we conducted what we call diagonal walks through a random point. This means we generate a random point, which represents a valid solution, and "walk" in equidistant steps along a straight line (in the search space) through the random point. These walks are repeated three times for the same random point, but using three different directions. Performing diagonal line walks rather than axis-aligned ones enables to investigate the effects of changing the values of all variables simultaneously. Of course, the observations only correspond to the selected random point, and can not offer any insights in terms of global optima. Yet, this approach nonetheless offers a simple way to (visually) inspect some properties (e.g. sensitivity) in a high dimensional search space.

Examples of resulting plots can be found in Figure 2. Labels on the x-axes are neglected as all lines feature different lengths, which are normalised to fit the axes and, thus, are meaningless. The depicted function values only represent a single instance.

Plots (a) and (b) in Figure 2 are good representatives of the encoding-based functions. They have numerous steps in the fitness function, as well as a discernible global structure. The steps are likely a result of the genotype-phenotype mapping. If values are varied along a line, for a specific cut-off value, the encoding in the neural network will flip and produce a tile (for more details on the encoding, see [20]). This is a result of using GANs on Mario levels in a discrete encoding, as opposed to images with pixels encoded

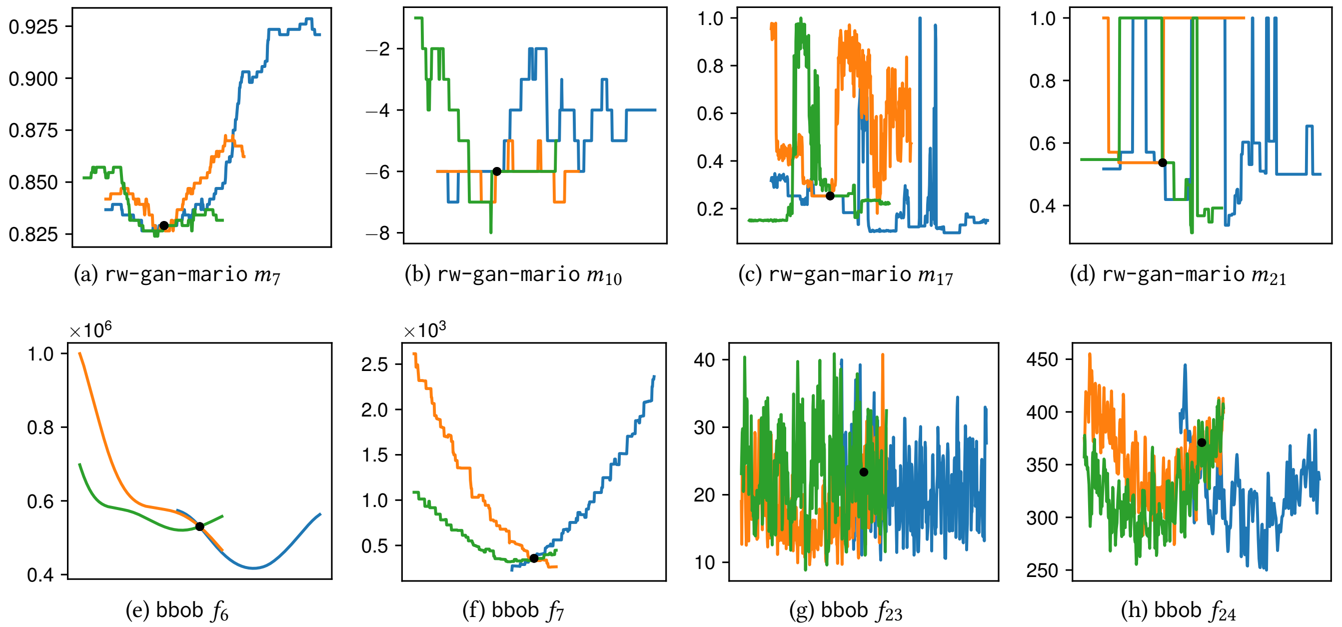


Figure 2: Diagonal walks for instance 1 of `rw-gan-mario` functions m_7, m_{10}, m_{17} and m_{21} (top) and `bbob` functions f_6, f_7, f_{23} and f_{24} (bottom). Colours indicate separate walks and the black point denotes the common random point.

as continuous values. Because of this discrete encoding, the steps in the tile-based fitness functions will always occur.

In contrast, plots (c) and (d) in Figure 2 are representative of what most simulation-based fitness functions look like, with plateaus (d), very high spikes (c) 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 diagonal walks for functions from `rw-gan-mario` with those on selected functions from the `bbob` suite (cf. bottom row of Figure 2). Function f_6 is representative for a lot of `bbob` functions, as it is continuous and has a global structure without any major local irregularities. Function f_7 could be considered similar to the encoding-based fitness function m_7, as both have pronounced steps (cf. Figure 2 (b)). The `bbob` suite however also contains functions with high local irregularities. While most functions do possess an obvious global structure, such as f_{24}, there are evidently also functions where (at least for the diagonal line walk) no structure is discernible, for example, f_{23}.

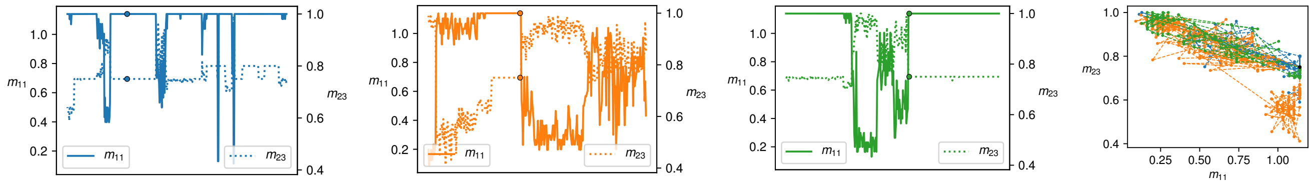
Due to the described similarities between the functions in the `rw-gan-mario` and the `bbob` suites, we conclude that these functions are not degenerate. This means we can expect meaningful results from the benchmark. However, the functions in `rw-gan-mario` also contain some very distinctive features, such as realistic noise levels, as well as extreme function changes, very small valleys of attraction and large plateaus (see Figure 2, plots (b-d)).

In order to also be able to visualise the bi-objective landscapes, we produce diagonal walks where we show the values for both objective functions. Examples can be found in Figures 3, 4, 5 and 6. Each of the three walks is shown separately for the specific functions selected, so that conflicts in the functions are observable. In addition, we plot the walks in objective space for each of the selected bi-objective functions.

For function M_4, we see clear conflicts in all diagonal walks as visualised in Figure 3. This is expected, because if a given agent receives a low competition score (basicFitness, m_{11}), this likely means that it failed early in the simulation. This is of course in conflict with timeTaken (m_{23}), which seeks to maximise the time that the agent spends on the level. In addition, the basicFitness measure penalises agents for taking longer. The same observations are true for function M_8 (see Figure 4), which combines the same type of fitness measures for underground levels instead of overworld.

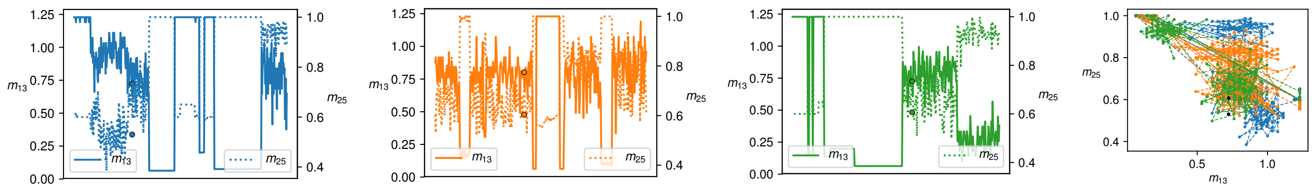
Finally, we show diagonal walk results for functions from the bi-objective `rw-top-trumps-biobj` suite. For function T_1 (see Figure 5), it seems that, while in one direction (blue) both objectives can be improved at the same time, other directions show the conflict between them. This is expected behaviour, as in some cases, increasing the hypervolume t_1 by increasing the absolute values of a card will also increase the standard deviation between card averages t_2. However, if these values are increased too far, this might result in fewer non-dominated cards, resulting in abrupt jumps in the function value as seen in the plot. The value for t_2 will however still decrease.

We also show diagonal walks for function T_3 with objectives t_3 and t_5 in Figure 6. Here, due to the high irregularities of both functions, the plots are harder to interpret. Based mainly on the walks in the objective space, we can infer that the two objectives are mostly in conflict.



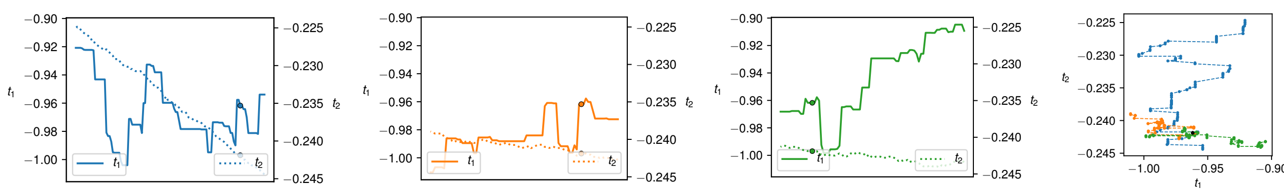
(a) Walk 1 (b) Walk 2 (c) Walk 3 (d) Walks in the obj. space

Figure 3: Diagonal walks for `rw-gan-mario-biobj` function M_4 with objectives m_{11} and m_{23} (instance 1)



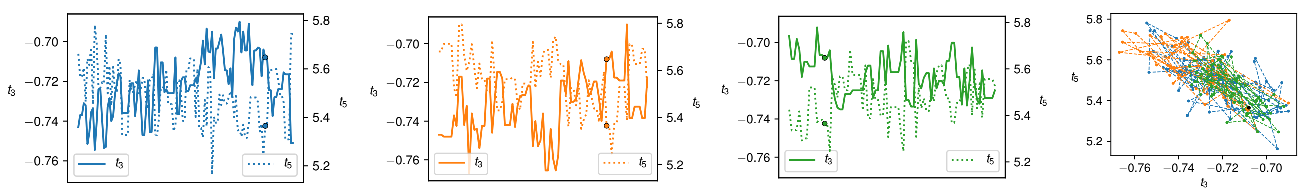
(a) Walk 1 (b) Walk 2 (c) Walk 3 (d) Walks in the obj. space

Figure 4: Diagonal walks for `rw-gan-mario-biobj` function M_8 with objectives m_{13} and m_{25} (instance 1)



(a) Walk 1 (b) Walk 2 (c) Walk 3 (d) Walks in the obj. space

Figure 5: Diagonal walks for `rw-top-trumps-biobj` function T_1 with objectives t_1 and t_2 (instance 1)



(a) Walk 1 (b) Walk 2 (c) Walk 3 (d) Walks in the obj. space

Figure 6: Diagonal walks for `rw-top-trumps-biobj` function T_3 with objectives t_3 and t_5 (instance 1)

4.2 Practicality

The practicality in terms of computational effort is an important consideration in real-world benchmarks (R-II). 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. CFD model). In cases where the functions are only moderately expensive, they can still not be easily compiled into a benchmark, as multiple scalable (R-IV) instances of the functions should exist (R-III). 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 of algorithms.

In order to assess the practicality of the GBEA with regards to computation time, we measured the time for computing a single function evaluation. The experiments were run on a regular computer and resulted in the following averaged runtimes:

- `rw-gan-mario`, without simulation: ca. 0.52 seconds
- `rw-gan-mario`, with ScaredAgent: ca. 1.52 seconds
- `rw-gan-mario`, with A*: ca. 35.16 seconds[15]
- `rw-top-trumps`, without simulation: ca. 0.002 seconds
- `rw-top-trumps`, with simulation: ca. 3.13 seconds[16]

We observe that, as expected, functions that do not rely on simulations are fast to compute. For TopTrumps, even the simulated

[15] average median 7.5 seconds, max observed 375 seconds
[16] average median 3 seconds, max observed 9 seconds

functions are very fast despite executing 2 000 simulations per 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 finishes within less than 10 seconds, there are a considerable number that take longer and finish after up to 375 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 for dimension 10 for the `rw-gan-mario` suite and 128 for the `rw-top-trumps` suite. However, the runtimes for MarioGAN 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 consider these results sufficient to claim that the benchmark is indeed practical in terms of computational resources required (R-II). This is based on the average execution times reported for a comparable benchmark [2]. For their CFD-based benchmark, the authors report average execution times of 40.35, 947.37 and 34.44 seconds, 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-gan-mario`, which takes only slightly longer than the fastest CFD function.

4.3 Interpretability of Results

There is one major issue that arises when integrating real-world problems into the COCO framework. For real-world problems, the value of the global optimum 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 target(s). Due to the way the targets are distributed, this might make algorithms with widely different performances appear similar in terms of when they reach the targets.

A solution 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, since the globally optimal hypervolume of its problems is not computable analytically. This issue is exacerbated for the `rw-gan-mario-biobj` and `rw-top-trumps-biobj` suites, as in both cases, even the optima for the single-objective functions are unknown. The globally optimal hypervolume is therefore even more difficult to estimate.

However, these issues will be automatically resolved with time, when more results become available. In addition, the progression of fitness values can still be plotted and interpreted independent of the COCO post-processing features.

5 SUMMARY AND FUTURE WORK

Summarising the observations made in this paper, we determine that, based on the line walk plots, the game optimisation problems incorporated in GBEA are interesting and challenging for evolutionary algorithms. They contain plateaus and steps, and vary in terms of existence of a global structure, just as artificial functions do. The GBEA functions, however, also possess novel characteristics due to the inherent noise and partial lack of locality. We further find that simulation-based and encoding-based functions possess different characteristics. We can also report that the benchmark runs with practical execution times, especially when run in batch mode. Thus, we conclude that GBEA can be used for benchmarking evolutionary algorithms designed for optimising real-world optimisation tasks.

In the future, we will continue improving the benchmark. One important step in this process is to collect data with numerous algorithms, so that the targets measured in the benchmark can be chosen more consciously. This will also automatically increase the interpretability of the benchmark and allow the identification of specific weaknesses for a given algorithm as argued in Section 4.3.

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 support noisy optimisation in this context. This would allow to leave noise handling up to the optimisation algorithm instead of reporting averages of multiple simulations. An extension towards benchmarking surrogate-assisted optimisation algorithms with features that include automatic logging of prediction errors is planned as well.

Furthermore, many of the game optimisation problems targeted in literature seem to have a mixed-integer search space [8]. In order to be able to represent these types of problems, appropriate function suites should be added to the GBEA. For example, we plan to add hyper-parameter optimisation problems to the benchmark. COCO, initially designed for continuous optimisation, has already been extended to support mixed-integer problems [17].

In addition to extending the GBEA with more problems, we will also conduct detailed analyses on the already implemented problems, e.g. using Exploratory Landscape Analysis. Based on the insights gained from these studies, we might observe (dis-)similarities to the established (artificial) benchmark problems. A better understanding of the problem characteristics also (a) helps to foster algorithm design, and (b) paves the way for more sophisticated studies, such as automated algorithm selection and configuration.

We hope that the presented GBEA benchmark will facilitate further research in real-world optimisation as well as help assess and improve surrogate-assisted algorithms intended for expensive optimisation. We further hope to produce data that helps to better understand game optimisation problems, specifically in terms of how hard the problems are for evolutionary algorithms.

ACKNOWLEDGMENTS

T. Tušar acknowledges the financial support from the Slovenian Research Agency (project No. Z2-8177), B. Naujoks acknowledges support by the European Commission's H2020 programme through the UTOPIAE Marie Curie Innovative Training Network, H2020-MSCA-ITN-2016, under Grant Agreement No. 722734 as well as through the Twinning project SYNERGY under Grant Agreement No. 692286, and P. Kerschke acknowledges the ERCIS network.

REFERENCES

[1] Alessandro Canossa and Gillian Smith. 2015. Towards a Procedural Evaluation Technique: Metrics for Level Design. In *Foundations of Digital Games (FDG)*. Society for the Advancement of the Science of Digital Games.

[2] Steven J. Daniels, Alma A. M. Rahat, Richard M. Everson, Gavin R. Tabor, and Jonathan E. Fieldsend. 2018. A Suite of Computationally Expensive Shape Optimisation Problems Using Computational Fluid Dynamics. In *Parallel Problem Solving from Nature (PPSN XV)*. Springer, Cham, Switzerland, 296–307.

[3] Kalyanmoy Deb, Lothar Thiele, Marco Laumanns, and Eckart Zitzler. 2005. Scalable Test Problems for Evolutionary Multiobjective Optimization. In *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications*. Springer, London, 105–145.

[4] Christian Grimme, Pascal Kerschke, and Heike Trautmann. 2019. Multimodality in Multi-Objective Optimization – More Boon than Bane?. In *Evolutionary Multi-Objective Optimization (EMO)*. Springer, 126–138.

[5] Nikolaus Hansen, Anne Auger, Olaf Mersmann, Tea Tušar, and Dimo Brockhoff. 2016. COCO: A platform for comparing continuous optimizers in a black-box setting. *ArXiv e-prints* arXiv:1603.08785 (2016).

[6] Nikolaus Hansen, Steffen Finck, Raymond Ros, and Anne Auger. 2009. *Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions*. Research Report RR-6829. INRIA, Paris, France.

[7] Pascal Kerschke and Heike Trautmann. 2019. Automated Algorithm Selection on Continuous Black-Box Problems By Combining Exploratory Landscape Analysis and Machine Learning. *Evolutionary Computation* 27, 1 (2019), 99–127.

[8] Kamolwan Kunanusont, Raluca D. Gaina, Jialin Liu, Diego Perez Liebana, and Simon Lucas. 2017. The N-Tuple bandit evolutionary algorithm for automatic game improvement. In *IEEE Congress on Evolutionary Computation (CEC)*. IEEE Press, Piscataway, NJ, 2201–2208.

[9] Antonios Liapis, Georgios N. Yannakakis, Mark J. Nelson, Mike Preuss, and Rafael Bidarra. 2019. Orchestrating Game Generation. *IEEE Transactions on Games* 11, 1 (2019), 48–68.

[10] Olaf Mersmann, Bernd Bischl, Heike Trautmann, Mike Preuss, Claus Weihs, and Günter Rudolph. 2011. Exploratory Landscape Analysis. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO)*. ACM, 829–836.

[11] Jeremy Rapin, Marcus Gallagher, Pascal Kerschke, Mike Preuss, and Olivier Teytaud. 2019. Exploring the MLDA Benchmark on the Nevergrad Platform. In *Companion of Genetic and Evolutionary Computation Conference (GECCO)*.

[12] Noor Shaker, Miguel Nicolau, Georgios N. Yannakakis, Julian Togelius, and Michael O'Neill. 2012. Evolving levels for Super Mario Bros using grammatical evolution. In *Conference on Computational Intelligence and Games (CIG)*. IEEE Press, Piscataway, NJ, 304–311.

[13] Adam Summerville, Julian R. H. Mariño, Sam Snodgrass, Santiago Ontañón, and Levi Lelis. 2017. Understanding Mario: An Evaluation of Design Metrics for Platformers. In *Foundations of Digital Games (FDG)*. ACM Press, New York.

[14] Julian Togelius and Noor Shaker. 2016. The search-based approach. In *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 17–30.

[15] Julian Togelius, Noor Shaker, Sergey Karakovskiy, and Georgios N. Yannakakis. 2013. The Mario AI Championship 2009–2012. *AI Magazine* 34, 3 (2013), 89–92.

[16] Tea Tušar. 2018. On using real-world problems for benchmarking multiobjective optimization algorithms. In *Proceedings of the International Conference on High-Performance Optimization in Industry, HPOI 2018, 21st International Multiconference Information Society, IS 2018*, Vol. D. Jožef Stefan Institute, 7–10.

[17] Tea Tušar, Dimo Brockhoff, and Nikolaus Hansen. 2019. Benchmark Problems for Single- and Bi-Objective Optimization. In *Genetic and Evolutionary Computation Conference (GECCO)*. ACM, New York, 9. https://doi.org/10.1145/3321707.3321868

[18] Tea Tušar, Dimo Brockhoff, Nikolaus Hansen, and Anne Auger. 2016. COCO: The Bi-objective Black Box Optimization Benchmarking (bbob-biobj) Test Suite. *ArXiv e-prints* arXiv:1604.00359v2 (2016).

[19] Oriol Vinyals et al. 2017. StarCraft II: A New Challenge for Reinforcement Learning. *ArXiv e-prints* arXiv:1708.04782 (2017).

[20] Vanessa Volz. 2019. *Uncertainty Handling in Surrogate Assisted Optimisation of Games*. Ph.D. Dissertation. TU Dortmund University, Germany.

[21] Vanessa Volz, Günter Rudolph, and Boris Naujoks. 2016. Demonstrating the Feasibility of Automatic Game Balancing. In *Genetic and Evolutionary Computation Conference (GECCO)*. ACM Press, New York, 269–276.

[22] Vanessa Volz, Jacob Schrum, Jialin Liu, Simon M. Lucas, Adam Smith, and Sebastian Risi. 2018. 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, 221–228.

[23] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. 2000. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation* 2 (2000), 173–195.