Multi-objective Adaptation of a Parameterized GVGAI Agent Towards Several Games

Ahmed Khalifa[1](✉), Mike Preuss[2], and Julian Togelius[1]

[1] Department of Computer Science and Engineering, New York University, Brooklyn, NY 11201, USA
ahmed.khalifa@nyu.edu, julian@togelius.com

[2] Department of Information Systems, Westfälische Wilhelms-Universität Münster, Münster, Germany
mike.preuss@uni-muenster.de

Abstract. This paper proposes a benchmark for multi-objective optimization based on video game playing. The challenge is to optimize an agent to perform well on several different games, where each objective score corresponds to the performance on a different game. The benchmark is inspired from the quest for general intelligence in the form of general game playing, and builds on the General Video Game AI (GVGAI) framework. As it is based on game-playing, this benchmark incorporates salient aspects of game-playing problems such as discontinuous feedback and a non-trivial amount of stochasticity. We argue that the proposed benchmark thus provides a different challenge from many other benchmarks for multi-objective optimization algorithms currently available. We also provide initial results on categorizing the space offered by this benchmark and applying a standard multi-objective optimization algorithm to it.

Keywords: Multi-objective optimization · GVGAI · MCTS

1 Introduction

Very many problems more or less naturally lend themselves to a multi-objective formulation, and thus to be solved or understood through multi-objective optimization. This creates a demand for better multi-objective optimization algorithms, which in turn creates a demand for fair, relevant and deep benchmarks to test those algorithms. It would therefore seem important for the furthering of research on multi-objective optimization to create such benchmarks, ideally through building on real, challenging tasks from important application domains.

In the research field of AI and games, there is a long-standing tradition of benchmarking algorithms through their performance on playing various games. For a long time beginning very early on in the history of computer science, Chess was one of the most important AI benchmarks, and advances in adversarial search were validated through their performance on this classic board game. Later on, as classic board games such as Chess, Checkers and eventually Go

H. Trautmann et al. (Eds.): EMO 2017, LNCS 10173, pp. 359–374, 2017.
DOI: 10.1007/978-3-319-54157-0_25

succumbed to the advances of algorithms and computational power, attention has increasingly turned to video games as AI benchmarks. The IEEE Conference on Computational Intelligence and Games now hosts a plethora of game-based AI competitions, based on games such as *Super Mario Bros* [23], StarCraft [14], *Unreal Tournament* [10], and *Angry Birds* [19]. These offer excellent opportunities to benchmark optimization and reinforcement learning algorithms on problems of real relevance.

For many—but not all—games, one can easily identify a single and relatively smooth success criterion, such as score or percent of levels cleared. This makes it possible to use single-objective optimization algorithms, such as evolutionary algorithms, to train agents to play games. For example, when evolving neural agents to drive simulated racing cars, one can simply (and effectively) use the progress around the track in a given time span and as the fitness/evaluation function [21]. However, in many cases the training process can benefit from multi-objectivization, i.e. splitting the single objective into several objectives, or constructing additional objectives in addition to the original one. For example, when using genetic programming to learn a car-driving agent, adding an extra objective that promotes the use of state in the evolved program can improve performance and generalization ability of the agent [1]. In another example, car-driving agents were evolved both to drive well and to mimic the driving style of a human; while these objectives are partially conflicting, progress towards one objective often helps progress towards the other as well [25]. Finally, when using optimization methods for procedural content generation—generating maps, levels, scenarios, items and such game content—the problem is often naturally multi-objective. For example, when evolving balanced maps for *StarCraft*, it is very hard to formulate a single objective that accurately captures the various aspects of map quality we are looking for [22].

So far we have talked about playing individual games. Recently, there has been a trend toward going beyond individual games and addressing the problem of *general game playing*. The idea here is that to be generally intelligent, it is not enough to be good at a single task: you need to have high performance over some distribution of tasks. Therefore, general video game playing is about creating agents that can play not just a single game, but any game adhering to a specific interface. The *General Video Game AI (GVGAI) Competition* was developed in order to provide a benchmark for general video game playing agents. The competition framework features several dozens of video games similar to early 80s arcade games, and competitors submit agents which are then tested on *unseen* games, i.e. games that the competitor did not know about when submitting the agent [15,16].

In other words, general video game playing agents should be optimized for playing not just a single game well, but a number of different games well. This immediately suggests a multi-objective optimization benchmark: select a number of games, and optimize an agent to perform well on all of them, using the performance on each game as an objective.

This paper describes a multi-objective optimization benchmark constructed on top of the GVGAI framework. We first describe the underlying technologies this builds on, including the GVGAI framework (Sect. 2), the Monte Carlo tree search algorithm (MCTS, Sect. 3), and the specific parameterizable agent representation developed for the benchmark (Sect. 4). We then quantitatively characterize the benchmark through sampling in the space of agent parameters, and by applying a standard multi-objective optimization algorithm on several versions of the benchmark. More precisely, our experimental analysis (Sect. 5) shall answer the following questions:

– Which games enable learning via proper feedback and are different enough from each other so that multi-objective optimization makes sense?
– What is the experienced noise level and how much noise can be tolerated while doing optimization?
– How do we aggregate scores, winning, and set the run length in a meaningful way? What are the properties of the resulting optimization problem?

Finally, we summarize the features of the obtained optimization problem and compare the setting to the one of other recent multi-objective benchmarks/competitions in Sect. 6 before concluding in Sect. 7.

The main contribution of this paper is a new multi-objective optimization benchmark based on general video game playing, which we hope can help the development of better such optimization algorithms. However, we also envision that this tool leads to an improved understanding of the MCTS algorithm, which our parameterizable agent builds on. Employing multi-objective optimization algorithms could further help elucidating the relations between different games based on what agent configurations play them well. A first contribution in this direction is some insight about the conflicts between adapting the agents towards different games, based on an experimental analysis. We assume that very wide and large Pareto front approximations stand for strongly conflicting game requirements (towards the agent), whereas small and narrow Pareto front approximations mean that the gameplay is rather similar and can be accomplished well by only one agent (or that the problem is so difficult that getting near to the real Pareto front is very hard).

2 Extending the General Video Game AI Framework

The General Video Game AI (GVGAI) framework is a Java framework that allows running different games. Most of these games are ports of old Arcade of home computer games or some newer indie games. The GVG-AI framework has been continuously extended and now contains more than 80 implemented games. Figure 1 shows 4 different games implemented in the framework where they range from puzzle to action/arcade games. The framework also enables users to easily design new games by describing them in the Video Game Description Language (VGDL) [7]. VGDL is a declarative description language that enables defining the game itself and the used levels (usually 5 levels are defined for each game) in

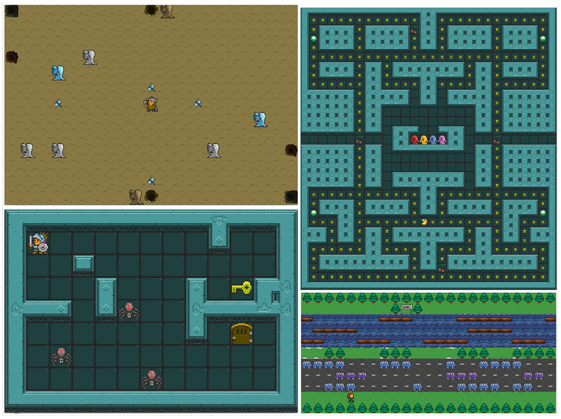


Fig. 1. Different games in the GVGAI framework in order from top left to bottom right: Defem, Pacman, Zelda, and Frogs.

a human-readable form. The game description contains information about game objects, interactions, goals, etc., While the level description contains information about how game objects are spatially arranged.

The GVGAI framework was initially introduced to allow people to implement their own AI agents that can play multiple of unseen games. The best way to encourage people to use a new framework is by organizing competitions. Multiple different competition were organized at different conferences starting at Computational Intelligence and Games (CIG) 2014 [16]. The best agents so far can win only 50% of 50 different games in the framework [3]. After the success of this competition, multiple different tracks were introduced in 2016, such as the level generation track [12] and the 2 player planning track [9]. In the level generation competition, the participants try to design a general level generator that generates game levels for any game provided its description. The two-player planning track is similar to the original track but for games that have two players. Games can be cooperative or competitive based on the game description. In all competition tracks, agents don't know what game they are playing and have to figure it out based on interactions or simulations.

In this work, we are introducing a new track for the GVGAI competitions, intended to attract researchers from a different area, namely multi-objective optimization. The goal for this competition is to optimize a parameterized Monte-Carlo Tree Search (MCTS) algorithm to perform well on different games. This agent and the underlying main algorithm will be described in Sects. 3 and 4.

This new competition track (and its associated benchmark software) aims to obtain a deeper understanding of the induced landscape of different MCTS Upper Confidence Bound equations (explained in Sect. 3). This knowledge will then enable creating better hyper-heuristic agents [13]. Hyper-Heuristic agents are AI agents that can change the current algorithm or heuristic (parameters of the basic equation in our current context), based on the current game. By means of our approach, we also provide a new benchmark for multi-objective optimization, with a fair number of parameters (14 in our current version), a scalable number of objectives (we use 3 here but this only depends on the number of games taken into account), also adding limited noise to problem features, as could be expected in many real-world scenarios. Thus we obtain a new tool for measuring the performance of different multi-objective algorithms under near-realistic conditions, with the ability to explain (after revealing the games and the exact encoding) the MCTS agent behavior that results from the best found solutions.

3 Monte-Carlo Tree Search

Monte-Carlo Tree Search (MCTS) [6] is a stochastic game tree search algorithm that balances between exploiting the best nodes and exploring new nodes. As usual, every node of the tree is associated with one possible action that may be played next. Every specific path through the tree resembles the moves of one game (or at least the first n moves of the game if the game has n levels). MCTS is divided into four main steps:

1. **Selection:** the algorithm selects a suitable node to investigate further. As mentioned before, it tries to balance between exploiting the best nodes and exploring new nodes. It uses an equation called Upper Confidence Bound (UCB) to achieve this balance.
2. **Expansion:** the algorithm selects which child node to expand at the current node. If there is no more nodes to expand, the algorithm returns to step 1. In vanilla MCTS, it picks a random node to be expanded.
3. **Simulation:** the algorithm simulates what will happen in future by applying a forward model. It plays out random actions until a terminal state is reached (win or loss), time is up, or a predefined depth is reached.
4. **Backpropagation:** the algorithm evaluates the simulated state (by means of a heuristic if it is not a terminal state), then it updates all parent nodes up to the root. This means that for every node, the information stored in the children is aggregated. After this step, the algorithm repeats all these steps starting from the root, until it finishes (usually after a specified time or number of iterations).

After the whole algorithm has been terminated (there is no natural ending), we have to choose one move that is actually performed, and usually the one with the highest win rate is chosen. The nodes that reside directly under the root aggregate all known information about the consequences of what happens after the action associated with this node is played next.

The most well-known UCB equation is UCB1 (1), and it is divided into two terms. The first term pushes search to exploiting the best node so far while the second part controls the effort spent on exploring new nodes.

$$UCB1 = \overline{X_j} + C\sqrt{\frac{2 \cdot \ln n}{n_j}} \tag{1}$$

In addition to many other extensions (such as integration of heuristics that represent human expertise) to classic MCTS agents [20], researchers have explored modifying the UCB equation in order to obtain different behaviors of MCTS agents. Some of these modification are done manually, such as using MixMax instead of the average value as the exploitation term [8,11,17]. Other researchers used genetic programming to find alternatives for the current UCB equation. These approaches have been applied to different games such as Go [5] by Cazenave or games in the GVG-AI framework [4] by Bravi et al.

4 A Parameterizable Agent as Optimization Problem

The findings from Bravi's work provide us with multiple different UCB equations for different games. Each equation is evolved over a certain game and tested against all other games in the framework. The evolved equations have terms that explain the nature of the game. For example; if the game goal is to collect all the moving butterflies that move randomly everywhere. The evolved equation will have a term to explore different map locations to find all of them. In this work, we combined features from all of the evolved equations from Bravi's work into one big, parameterized equation, depicted in (2). This equations can be seen as a very general formula for node selection, which can be specified into a very large number of different strategies depending on its parameterization.

$$\begin{aligned} UCB_{opt} = P_0\overline{X_j} + P_1 max(X_j) + P_2(\frac{\ln n}{n_j})^{P_3} + P_4 E_{xy}^{P_5} + P_6 min(D_{npc})^{P_7} \\ + P_8 min(D_{port})^{P_9} + P_{10} min(D_{mov})^{P_{11}} + P_{12} min(D_{res})^{P_{13}} \end{aligned} \quad (2)$$

Here, X_j is the total reward of the current node, n is the number of visits for the parent node, n_j is the number of visits for the current node, E_{xy} is the number of times a certain tile is visited providing its x,y position, D_{npc} is the distance to a NPC[1] object in the game, D_{port} is the distance to a portal object in the game, D_{mov} is the distance to a movable object in the game, and D_{res} is the distance to a resource object in the game.

The first two terms are the most frequently ones representing exploitation in the evolved equations. If their coefficients equal q and $1-q$, we have the MixMax term. The rest of the terms stand for different ways of exploration, either based on the tree information (3^{rd} term), space information (4^{th} term), or different game sprite information (the rest of the terms).

To summarize, we have 14 different parameters to set (or optimize), where each parameter can take values between -1 to 1. Each term in Eq. 2 has a different meaning, based on the sign of the parameter. However, we can expect that the first two terms (exploitation terms) most likely will have positive values. The other terms (exploration terms) have different meanings based on the sign. Zero value means this term doesn't have any effect on the agent behavior. Negative values mean that this node/tile/sprite should be avoided while playing the game,

[1] Non-player character, usually but not necessarily adversarial.

positive values mean that getting closer to a sprite or accessing the same visited tile/node will be rewarded.

In the remainder of this section, we provide some information concerning implementation and interface, being well aware that the actual competition setting may differ from this in details.

We wanted our interface to be as accessible as possible and the problem be treated as any other (noisy) black box optimization problem. Therefore, we designed the interface to be language independent. A solution (14 numbers) is evaluated by providing it as set of command line parameters to an executable Java archive (JAR) file. After running the parameterized MCTS on all three games a predefined number of times, the average objective value per game is written to a file, together with a control flag that indicates if the execution ended correctly or not. The objective values have been encoded according to (3), with win being 1 if the game is won, and 0 else. This weights winning the game much higher than achieving a high score, and for most GVGAI games the difference is huge. However, there is no defined maximum score that can be obtained for one game, so that the ratio can vary.

$$f_{\text{game}} = win + \frac{score}{1000} \tag{3}$$

Whenever we perform test runs with a multi-objective optimization method (we employ the SMS-EMOA [2] as it is one of the most popular ones, but it could also be another one) in the following, we switch to minimization by means of the transformation $f_{\text{sms}} = 3 - f_{\text{game}}$, resulting in a Nadir point at $(3, 3, 3)$. We take this also as reference point, such that the maximal (theoretical) hypervolume between the Pareto front and the origin is 27. However, this would mean that we would win every single game with a score of 2000, which is not possible in most games. The real maximum is therefore unknown, but considerably smaller. E.g., if the achieved Pareto front approximation contains points that realiably win every one of the 3 games (but do not get a high score), the hypervolume would be in the range of $3^3 - 2^3 = 19$.

5 Experimental Analysis

By means of our experimental analysis, we generally investigate the suitability of the previously described setup as benchmark problem. We do this in several steps. At first, we select appropriate games by looking at their adaptability value [18]. We then consider the noise level in order to find out how stable single measurements are and how they must be aggregated. Finally, we perform some test optimization runs, relying on the SMS-EMOA [2] in order to see how competition results would look like.

5.1 Game Selection

From the currently available 80 games, we picked 12 that seemed to be interesting and suitable for tackling them with the parametrizable MCTS-based agent described in Sect. 4. These games were selected from different clusters based on Bontrager's work [3]. They were selected equally from the top 3 hardest clusters (all clusters except the easiest one). Additionally, we take into consideration that the games shall either be solvable easily (i.e. Frogs) or provide feedback often (i.e. Pacman). We avoided long rewarding puzzle games because optimization methods would not get any improvement feedback except from *win* and *loose*. This is the list of the 12 considered games (Fig. 1 shows some screenshots):

– **Defem:** a port of an indie-game with the same name. The goal of the game is to survive. The player should avoid/kill enemies that are spawning in the level. The player shoots automatically in any random direction.
– **Eggomania:** a port of a mini-game in Pokemon stadium 2. The goal is to catch all the falling eggs before they reach the ground.
– **Frogs:** a port of Frogger, a famous Atari game. The goal is to cross the street and the lake to the other side without getting hit by cars or drowning in water.
– **Modality:** a puzzle game about two different dimensions. The goal is to push a tree to its correct spot. The level is divided into two colors, each color represent a separate dimension. The player/tree can only pass from one dimension to the other through certain spots in the map.
– **Pacman:** a port of Pacman game. The goal is to eat all the pills without being caught by the chasing ghosts.
– **Painter:** a puzzle game where the player plays as a painter. The goal is to change all the level tiles to be colorful. The tile color toggles between plain and colorful when the player pass over it.
– **Plants:** a port of the indie-game Plants vs Zombies. The goal is to prevent waves of zombie from reaching your house. The player can grow plants that shoot the zombies to kill them and the zombies attack back when they are close enough.
– **Zelda:** a port of the dungeon system in The Legend of Zelda. The goal is to get the key and go to the exit without getting killed. The player can kill the enemies using his sword.
– **Boulderdash:** a port of Boulder Dash a famous Atari game. The goal is to collect 10 gems and go towards the exit without dying. The player should avoid falling boulders and monsters while searching for the gems.
– **Sokoban:** a variant of Sokoban, a famous Japanese puzzle game. The goal is to destroy all the boxes in the level by pushing them into holes.
– **Solarfox:** a port of Solarfox a famous Atari game. The goal is to collect all the jewels and avoid hitting the borders of the screen or getting hit by enemy missiles.
– **Roguelike:** an action game. The goal is to collect as much treasure as you can and reach the exist. The path toward the goal is filled with moving monsters and doors. The player needs to get a sword to protect itself from the monsters, and collect keys to open the locked doors in its way.

We now analyze how well suited to optimization the different games are by performing a well distributed sample over the search space $[0, 1]^{14}$, do repeated measuring and then look at the empirical tuning potential (adaptability, Eq. (4)) value as introduced in [18]. The ETP was originally considered to measure how easy it is to obtain a better algorithm configuration by tuning, but the situation here is quite similar (repeated runs due to noise, unknown peak performance, vast parameter space). It takes the performance of the best ($\mathbf{y}_p$) and an average configuration ($\mathbf{y}_a$) and the semi-quartile range (non-parametric alternative to standard deviation) at these points into account. The higher the computed value, the easier it is to improve the performance, starting with an average configuration. In case one semi-quartile range is zero, we set the ETP value to zero as well.

$$\mathrm{ETP}(\mathbf{y}_p, \mathbf{y}_a) := \frac{\mathrm{median}(\mathbf{y}_a) - \mathrm{median}(\mathbf{y}_p)}{\mathrm{sq}(\mathbf{y}_a)} \cdot \frac{\mathrm{median}(\mathbf{y}_a) - \mathrm{median}(\mathbf{y}_p)}{\mathrm{sq}(\mathbf{y}_p)} \tag{4}$$

We perform the following experiment in order to remove unsuitable games - games with very low ETP values - from this set.

Experiment: which games are well suited for a multi-objective benchmark?

Pre-experimental planning. After some preliminary tests, we found that 50 samples per game with 10 repeats each is a good compromise between runtime and result quality. This combination requires about 30 h of computation time for the 12 games[2].

Task. What ETP value do we require in order to keep a game? As concrete values are difficult to estimate, we will order the games according to their ETP and then look for a gap between "near zero" and "clearly larger than zero".

Setup. We generate a common sample of 50 well distributed 14 parameter controller configurations by means of the MaxiMin reconstruction (MMR) method of Wessing [26], and run the agent 10 times for each sample point and each game, recording the objective values.

Results/Visualization. Table 1 holds the measured ETP values for each game. Note that the value gets zero as soon as one of the semi-quartile ranges get zero.

Table 1. Expected tuning potential (ETP) for the 12 considered games.

Game	Defem	Sokoban	Roguelike	Frogs	Zelda	Boulderdash
ETP	0.72	0.0	0.0	0.0	144.65	0.0
Game	Modality	Pacman	Eggomania	Painter	Solarfox	Plants
ETP	0.0	16.49	0.0	891.37	46.25	11.54

Observations. 6 of the games have a zero value, whereas the non-zero values for the other games vary quit a lot. It seems that Painter is the game with the best adaptation/optimization potential, and Defem the one with the worst.

[2] Using a single core of an Intel i7-3770 CPU @ 3.4 GHz.

Discussion. Although there is some variance in the observed point samples for Frogs, Eggomania, and Modality, the ETP values clearly recommend only using Defem, Zelda, Pacman, Painter, Solarfox, and Plants. However, Solarfox is a special case: one cannot win, not even have a positive score, but only loose with different negative scores. In consequence, the ETP is positive, but nearly all objective function values for the game would lead to a point beyond the Nadir point (such that the gradual differences in the signal would be cut out). Frogs and Modality provide basically binary feedback (won or lost), and for Eggomania more than half of the samples end up with 10 losses without any point in a row. It seems reasonable that these are not really well suited for optimization; if they are used, an additional difficulty is added (fitness cliffs). We thus end up with a game selection of 8 games: Defem, Zelda, Pacman, Painter, Plants, Frogs, Eggomania, and Modality.

We can also state that the ETP value provides valuable information, but the necessities of the optimization process may lead to a positive ETP value when the game is still unsuitable (in the case of Solarfox). Adding 3 more games with (almost or completely) binary signals may make the problem more interesting and shall be considered.

5.2 Analyzing Variance and Noise

In this section, we analyze the parameterized agent and check its response to noise and variance. We ran two experiments, the first one is running each game using random parameters for 2000 times (without repetitions). Table 2 shows the result of this experiment. As you can see, the mean is very low in all games except for Modality and Painter. The reason is that both these games have a very small map with a size of 4×4 tiles which make it easy for an agent to win it. Also, we can notice there is no agent has won pacman. The main reason is that the pacman map is very huge which takes a lot of time to finish it. Only Modality, Painter, and Frogs have a very high standard deviation compared to the other games which shows that these games are more sensitive to the parameter change than others (or, as described above, have few very extreme attained objective values).

Table 2. Average statistics over 2000 random configurations.

Game	Mean	Std	Min	25%	50%	75%	Max
Defem	0.033	0.099	0.0	0.001	0.032	0.046	1.05
Frogs	0.105	0.306	0.0	0.0	0.0	0.0	1.001
Modality	0.583	0.494	0.0	0.0	1.001	1.001	1.001
Painter	0.561	0.442	0.039	0.118	0.395	1.042	1.482
Zelda	0.011	0.102	−0.001	−0.001	0.0	0.002	1.008
Pacman	0.025	0.036	0.0	0.002	0.009	0.03	0.244
Eggomania	0.01	0.095	0.0	0.0	0.0	0.001	1.116
Plants	0.045	0.154	0.003	0.016	0.021	0.026	1.109

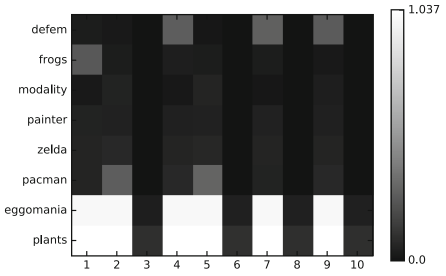


Fig. 2. Average values for 1000 runs over all the games vs. 10 different random parameter configurations. (Color figure online)

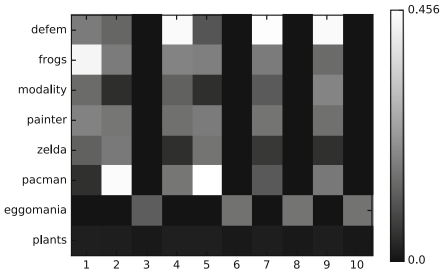


Fig. 3. Standard deviation values for 1000 runs over all the games vs. 10 different random parameter configurations. (Color figure online)

In the second experiment, we picked 10 random configurations for the parameters, then we ran the configured agent on each game repeatedly 1000 times. Figure 2 shows a heat map representing the average values of the 1000 runs for all 8 games and all 10 configurations. Darker colors mean the agent neither wins nor obtains a descent score, brighter colors otherwise. Figure 3 shows the standard deviation of the previous experiment. Brighter colors shows more variance in their results than darker colors. From analyzing the heatmaps, we can notice that agent configuration numbers 3, 6, 8, and 10 loose almost all games and therefore have small variance. Also, we can notice that both on Eggomania and Plants repeated measurements of single agent configurations have a very low standard deviation. This means that these games behave more deterministic than the others. On the other hand, some configurations result in very high standard deviations on Defem. One could argue that this is the noisiest game in the set. Pacman is somewhere in between, most likely due to the huge amount of pills and the presence of the chasing ghosts, factors that surely increase variance. Eggomania and Plants are very sensitive to the agent configuration as almost 50% of all the configurations lose the game.

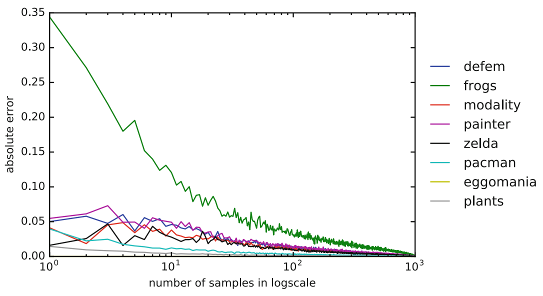


Fig. 4. Absolute error of a point averaged from repeated samples (resampled 50 times).

Table 3. Number of iterations until the average error goes below 0.01.

Game	Defem	Frogs	Modality	Painter	Zelda	Pacman	Eggomania	Plants
Iterations	181	586	148	181	114	15	1	2

Figure 4 shows the absolute error of the first configuration using different number of samples for all 8 games. From the graph, we can see that all games have error values around 0.05 from the start except for Frogs. Frogs is a very noisy game, it takes 150 resamples to obtain an error of less than 0.05. Table 3 shows the number of resamples necessary to get an error of less than 0.01. We notice that Pacman, Eggomania, and Plants need quite few resamples to reach an error of less than 0.01 (at most 15 iterations). Therefore, these games are more suitable for the optimization setting than others.

5.3 Test Optimization Runs

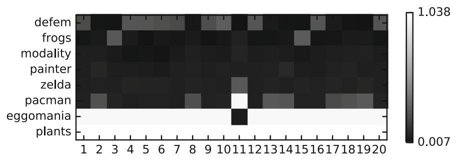
We perform some example runs with a standard multi-objective optimization algorithm in order to find out how well the established optimization problems are suited to be used as basis for a competition.

Experiment: Is the problem difficulty adequate? Do we obtain interesting fronts?

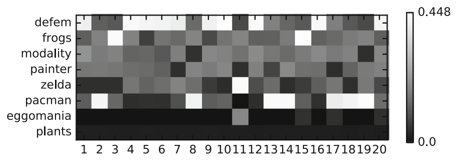
Pre-experimental planning. The run length of 500 evaluations (x3) with 2 repeats each (averaged) is designed to add up to a wallclock time of around 12–15 hours per run (single core).

Task. Our expectation is rather fuzzy: we strive for some variance in the runs, but also some similarity (the problem should be difficult enough but not too difficult). Also, we expect to find large fronts (more than 20 is not possible in this setup), especially not a single dominating optimum.

Setup. We employ the SMS-EMOA [2] with a population size of 20 individuals and otherwise standard parametrization. We choose the games Painter, Pacman, and Plants as objectives, based on our previous findings (others are possible).



(a)



(b)

Fig. 5. (a) shows the average fitness of 500 repetitions of the 20 solutions of the best Pareto front approximation of one run on {Painter, Pacman, Plants}, using the SMS-EMOA. (b) shows the standard deviations of the values measured.

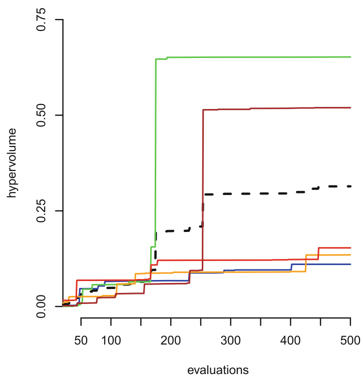


Fig. 6. Best dominated hypervolume of 5 SMS-EMOA runs on Painter, Pacman, and Plants. Average is dashed.

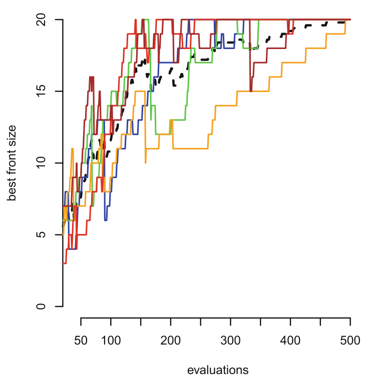


Fig. 7. Best front sizes over the same 5 runs. Average best front size is dashed. Population size is 20.

Results/Visualization. Figures 6 and 7 show the dominated hypervolume and number of solutions in the best front over 5 runs, respectively. Additionally, we take the resulting front of the first optimization run and measure how good the solutions do on all 8 games (average and standard deviation shown in Fig. 5).

Observations. We see gradual improvements in the hypervolume as well as huge jumps. There is a common tendency to improve, but with different speeds. The number of solutions in the best front starts around 5 and reaches the maximum of 20 around 250 evaluations (with one exception).

Discussion. Obviously, the optimization problem is neither trivial nor unfeasible, which corresponds well to our expectations. The amount of solutions in the best front supports that multi-objective optimization actually makes sense here: we have conflicts between the objectives. In future, this shall be investigated further, also broadening the scope to more games.

6 Optimization Problem Characterization

As performance measure for comparing different algorithms on this problem we want to employ the hypervolume dominated by the best Pareto front approximation. This is similar to what is used in the other recent multi-objective competitions/benchmarks, the Bi-objective BBOB [24] and the Black Box Optimization Competition BBComp[3] that features 2 and 3-objective tracks. However, unlike the other recent multi-objective competitions/benchmarks, we have noisy function evaluations here, with 2 types of noise in different strengths, depending on the actual game. If an agent can win often but not always, we have an almost binary noise, only the aggregation of several measurements gets us nearer to the

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

real function value. The other type of noise is more gradual and stems from the different rewards the agents get, this is especially visible for Pacman.

7 Conclusions

In this paper, we introduced a multi-objective optimization benchmark based on general video game playing, specifically the GVGAI framework. The task is to optimize the parameters of a general video game playing agent so that it plays several rather different games as well as possible. A parameterizable agent was developed for this purpose, where the parameters are coefficients in a complex formula for node selection in a Monte Carlo Tree Search Algorithm.

Testing was conducted to find a set of games on which the agent would have a large performance variance depending on its parameterization, and which would require so different playing styles that it would be likely that optimizing the agent's performance on them would yield partially conflicting objectives. A set of such games were found, and testing of random parameters as well as multiple samples of the same parameters showed that the variance in performance value was not only high between different configurations, but also relatively high between different samples of the same configuration. Attempts to optimize agents using the SMS-EMOA algorithm showed that it is indeed possible to find good solutions, that the objectives as expected are partially conflicting, and that there is relatively high noise in the fitness evaluation. We believe that these characteristics reflect many real-world problems, and that a benchmark with such characteristics provides a useful complement to existing multi-objective ones.

Finally, it should be noted that the framework and experiments described here are useful not only from the perspective of benchmarking multi-objective optimization algorithms, but also from the perspective of exploring the design space of games and generating new game-playing agents. It is for example an interesting idea to sample several agents from different positions on a Pareto front and then use hyper-heuristic methods to select the most appropriate agent parameterization for a particular game [3,13].

References

1. Agapitos, A., Togelius, J., Lucas, S.M.: Multiobjective techniques for the use of state in genetic programming applied to simulated car racing. In: IEEE Congress on Evolutionary Computation, IEEE 2007, pp. 1562–1569 (2007)
2. Beume, N., Naujoks, B., Emmerich, M.: SMS-EMOA: multiobjective selection based on dominated hypervolume. Eur. J. Oper. Res. **181**(3), 1653–1669 (2007). issn: 0377–2217
3. Bontrager, P., Khalifa, A., Mendes, A., Togelius, J.: Matching games, algorithms for general video game playing. In: AIIIDE (2016)
4. Bravi, I., Khalifa, A., Holmgård, C., Togelius, J.: Evolving UCT alternatives for general video game playing. In: The IJCAI-16 Workshop on General Game Playing, p. 63

5. Cazenave, T.: Evolving Monte Carlo tree search algorithms. Dept. Inf., Univ. Paris 8 (2007)
6. Chaslot, G.: Monte-Carlo tree search. Universiteit Maastricht (2010)
7. Ebner, M., Levine, J., Lucas, S.M., Schaul, T., Thompson, T., Togelius, J.: Towards a video game description language (2013)
8. Frydenberg, F., Andersen, K.R., Risi, S., Togelius, J.: Investigating MCTS modifications in general video game playing. In: Computational Intelligence and Games, pp. 107–113. IEEE (2015)
9. Gaina, R.D., Pérez-Liébana, D., Lucas, S.M.: General video game for 2 players: framework and competition. In: Proceedings of the IEEE Computer Science and Electronic Engineering Conference (CEEC) (2016)
10. Hingston, P.: Believable Bots. Springer, Heidelberg (2012)
11. Jacobsen, E.J., Greve, R., Togelius, J.: Monte Mario: platforming with MCTS. In: Proceedings of the Annual Conference on Genetic and Evolutionary Computation, pp. 293–300. ACM (2014)
12. Khalifa, A., Perez-Liebana, D., Lucas, S.M., Togelius, J.: General video game level generation
13. Mendes, A., Nealen, A., Togelius, J.: Hyperheuristic general video game playing. In: IEEE Computational Intelligence and Games (2016)
14. Ontanón, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D., Preuss, M.: A survey of real-time strategy game AI research and competition in starcraft. IEEE Trans. Comput. Intell. AI Games **5**(4), 293–311 (2013)
15. Perez, D., Samothrakis, S., Togelius, J., Schaul, T., Lucas, S., Couëtoux, A., Lee, J., Lim, C.-U., Thompson, T.: The 2014 general video game playing competition (2015)
16. Perez-Liebana, D., Samothrakis, S., Togelius, J., Lucas, S.M., Schaul, T.: General video game AI: competition, challenges and opportunities. In: Thirtieth AAAI Conference on Artificial Intelligence (2016)
17. Pettit, J., Helmbold, D.: Evolutionary learning of policies for MCTS simulations. In: Proceedings of the International Conference on the Foundations of Digital Games, pp. 212–219. ACM (2012)
18. Preuss, M.: Adaptability of algorithms for real-valued optimization. In: Giacobini, M., et al. (eds.) Applications of Evolutionary Computing, pp. 665–674. Springer, Heidelberg (2009). doi:10.1007/978-3-642-01129-0
19. Renz, J.: AIBIRDS: the angry birds artificial intelligence competition. In: AAAI, pp. 4326–4327 (2015)
20. Soemers, D., Sironi, C.F., Schuster, T., Winands, M.H.M.: Enhancements for real-time Monte-Carlo tree search in general video game playing. In: Proceedings of the IEEE Conference on Computational Intelligence and Games (2016)
21. Togelius, J., Lucas, S.M., De Nardi, R.: Computational intelligence in racing games. In: Baba, N., Jain, L.C., Handa, H. (eds.) Advanced Intelligent Paradigms in Computer Games. Studies in Computational Intelligence, pp. 39–69. Springer, Heidelberg (2007). doi:10.1007/978-3-540-72705-7_3
22. Togelius, J., Preuss, M., Beume, N., Wessing, S., Hagelbäck, J., Yannakakis, G.N., Grappiolo, C.: Controllable procedural map generation via multiobjective evolution. Genet. Program Evolvable Mach. **14**(2), 245–277 (2013)
23. Togelius, J., Shaker, N., Karakovskiy, S., Yannakakis, G.N.: The Mario AI championship 2009–2012. AI Mag. **34**(3), 89–92 (2013)

24. Tusar, T., Brockho, D., Hansen, N., Auger, A.: COCO: the bi-objective black box optimization benchmarking (bbob-biobj) test suite. In: CoRR abs/1604.00359 (2016)
25. Van Hoorn, N., Togelius, J., Wierstra, D., Schmidhuber, J.: Robust player imitation using multiobjective evolution. In: CEC, pp. 652–659. IEEE (2009)
26. Wessing, S.: Two-stage methods for multimodal optimization. Ph.D. thesis, TU Dortmund (2015).http://dx.doi.org/10.17877/DE290R-7804