Game AI Hyperparameter Tuning in Rinascimento

Ivan Bravi
Queen Mary University of London
London, UK
i.bravi@qmul.ac.uk

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

Simon Lucas
Queen Mary University of London
London, UK
simon.lucas@qmul.ac.uk

ABSTRACT

Hyperparameter tuning is an important mixed-integer optimisation problem, especially in the context of real-world applications such as games. In this paper, we propose a function suite around hyperparameter optimisation of game AI based on the card game Splendor and using the Rinascimento framework. We propose several different functions and demonstrate their complexity and diversity. We further suggest various possible extensions of the proposed suite.

CCS CONCEPTS

• **General and reference** → **Performance**; • **Theory of computation** → *Evolutionary algorithms*; • **Computing methodologies** → *Artificial intelligence*; • **Applied computing** → *Computer games*;

KEYWORDS

Benchmarking, evolutionary algorithms, hyperparameter tuning

ACM Reference Format:
Ivan Bravi, Vanessa Volz, and Simon Lucas. 2019. Game AI Hyperparameter Tuning in Rinascimento. In *Genetic and Evolutionary Computation Conference Companion (GECCO '19 Companion), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3319619.3326842

1 INTRODUCTION

At the time of writing, the Game-Benchmark for Evolutionary Algorithms (GBEA) contains 4 function suites based on two different games, namely *TopTrumps* and *Mario* [11]. Since the suites in the GBEA are intended to represent real-world problems related to games [11], we suggest to add a function suite aimed at hyperparameter tuning of game Artificial Intelligence (AI) algorithms. The performance of game AI is usually very sensitive to hyperparameters [7]. It is thus important to find a suitable configuration for game AI hyperparameters that optimises its performance in a game, usually measured as a score or winrate. Tuning problems are difficult to tackle for several reasons. Hyperparameters are usually a mixture of continuous and discrete variables, making tuning them a mixed-integer problem. These types of problems have not received much attention in research on benchmarking evolutionary algorithms until recently [9].

There are also some characteristics about game AI that introduce further complexity into tuning problems. Scores or wins are usually awarded for achieving specific discrete goals, thus resulting in steps in the fitness landscapes instead of continuous changes. These steps in fitness can be comparably large because even small changes in the parameters can potentially result in considerable changes in game scores achieved by the corresponding AI. Even minor changes in player behaviour (e.g. going left instead of right in a maze) can lead to very different game experiences and resulting scores (e.g. finding the exit instead of falling into a trap). This factor is usually exacerbated in multi-player games, where a small change in player behaviour can cause a chain of different reactions.

Additionally, most modern game AI and game engines are stochastic. Stochasticity can also lead to small changes in AI behaviour, thus resulting in different game outcomes as described above. The fitness associated with a hyperparameter configuration is thus also non-deterministic, making the fitness landscape noisy. Due to the achievement-based manner in which scores are usually awarded in games, they would follow a multinomial distribution. Common assumptions in noise handling for evolutionary algorithms, such as normalcy or symmetry, would therefore not hold [10].

As argued above, hyperparameter tuning of algorithms is an important as well as complex problem in the context of game AI and beyond. We thus suggest to add a function suite representing hyperparameter tuning of game AI to the GBEA. In the remainder of this paper, we describe such a suite based around the game Splendor. To this end, we first describe the game in section 2.1 and motivate its suitability for this purpose. We also give a brief description of the implementation of the game using the Rinascimento research framework and specify the various game AI algorithms to be tuned. Following this, we propose several diverse functions for the suite and provide some preliminary characterisations of their fitness landscapes. We conclude the paper with a discussion of the identified characteristics and the associated difficulties for optimisation.

2 BACKGROUND

2.1 Splendor

Splendor is a 2-4 player card-based board game, first published in 2014 by Space Cowboys. It has received several awards and was a *Spiel des Jahres* nominee in 2014. The goal of the game is to collect prestige points by buying bonus cards using tokens that can be taken from the table. Cards have different values (three decks with three different prestige points distributions), prices and suits (gem colour: red, green, black, blue and white). Cards, in turn, can be used to buy other cards reducing the number of tokens needed. On the side there is a number of noble tiles (each worth 3 points) that

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
GECCO '19 Companion, July 13–17, 2019, Prague, Czech Republic
© 2019 Copyright held by the owner/author(s). Publication rights licensed to the Association for Computing Machinery.
ACM ISBN 978-1-4503-6748-6/19/07…$15.00
https://doi.org/10.1145/3319619.3326842

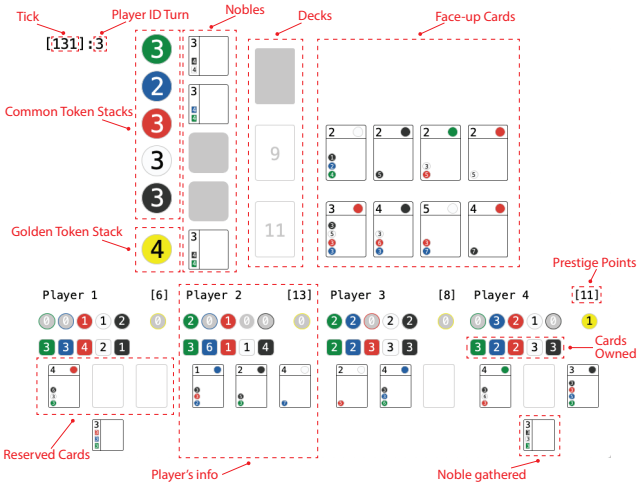

Figure 1: Rinascimento's UI.

independently move to players' hand when they own a specific combination of cards (e.g. 4 blue and 4 green). For a more detailed description of the game, please see [2].

Splendor is a popular game (ranked 16 among family games on boardgamegeek.com[1]). The complexity of the game involves dealing with limited resources (tokens and cards) available to the players during each turn. Moreover each player's action can significantly change the game state and consequently the available actions for the other players. This means that it's particularly hard to plan for long strategies. However it is possible to play very different kinds of strategies: (1) gather many cheap cards quickly and harvest many points from nobles, (2) focus more on expensive cards worth more points, (3) step gradually from cheap to medium and then expensive cards, or a mixture of them. Splendor is a stochastic game due to card shuffling but it is also partially observable as only few cards are accessible and face up. Nonetheless, the game state is very simple to read with very few game elements (cards, tokens and nobles), plus all the information is discrete. The actions have very clear instant outcomes without any timed or durable effects. This makes the game fairly easy to analyse with obvious differences between game states. Finally, since it is a board game without any heavy computation needs (e.g. physics modelling) we were able to write a fast implementation that can simulate more than 1.7 million actions per second[2].

2.2 Rinascimento

Rinascimento is a research framework that implements the game Splendor in an extensible and modular fashion intended to investigate various questions in games research. At the time of writing, it contains 3 parameterised AI agents, which are described in section 2.3 in more detail. It also includes a user interface depicted in figure 1 so that human players may join the game using the console to input actions. In addition, this feature allows observing the AI agents' behaviours within the context of a visualised game state, which makes it easier for a human to identify and understand patterns, as well as to find potential problems. One peculiarity of Splendor is the fact that the number of available actions changes with the game state. To account for this, Rinascimento provides a seeded random action generator (RAG) that samples valid actions.

2.3 Agents

In this section, we describe the three agents to be optimised, their algorithmic nature and their hyperparameters. More thorough details can be found in [2]. All agents implement Statistical Forward Planning (SFP), i.e. they search the game states using a forward model to simulate actions and future states. Such states are evaluated using a given heuristic that shapes the objective function of the search. The agents can also model opponent behaviour during their simulations. Therefore, all three agents have two parameters to define the opponent model and the simulation budget allotted to run them, respectively $om \in \{0, 1, 2\}$ and $ombs \in [0, 1]$. All the agent-specific hyperparameters are briefly described in Table 1.

Two of the agents (BMRH and SRH) are based on the Rolling Horizon Evolutionary Algorithm (RHEA) introduced in [8]. RHEA is an evolutionary algorithm that evolves and evaluates action sequences of predefined length. Once the budget is exhausted, it plays the first action of the best evolved sequence. Their key difference is that while BMRH evolves explicit action sequences, SRH finds seeds for the RAG. This allows the usage of standard variation operators instead of game-specific ones.

The third agent is Monte Carlo Tree Search (MCTS) [3]. It gradually builds a tree where each node represents an action and estimates the quality of it through Monte Carlo sampling of the action space. The main feature of the MCTS agent is that during the Tree Policy step with probability ep it will further expand the current node sampling ps-times the actions space through the RAG.

3 FUNCTION SUITE

3.1 Single-Objective Functions

Each function we propose in the following describes the success of hyperparameter tuning an AI agent with Rinascimento framework measured as the achieved winrate in 1 000 games against one or multiple opponents. However, hyperparameter tuning can be intended for several usecases, such as finding parameters that are suitable for beginner games or for robust performance. We have modelled 4 different usecases as function types, which are characterised by the opponents the tuned AI is tested against:

- Beginner: against simple agents (1-step look-ahead [2])
- Advanced: against agents that perform well in beginner usecase (identified via grid-search)
- Robust: against agents randomly picked from agent pool
- Exploits: against best agents found of the same type

To fully define the experiment, further specifications have to be made regarding the choice of AI algorithm to be tuned, as well as the number of opponents it is tested against and what budget is allowed for the agents during runtime. We thus implement several variants listed below that can be combined in order to create experiments:

- AI algorithm: BMRH, MCTS, SRH (see Section 2.3)

[1] https://boardgamegeek.com/familygames/browse/boardgame?sort=rank&rankobjecttype=family&rankobjectid=5499&rank=16#16
[2] Run on Intel(R) Core(TM) i7-3615QM (2012) CPU @ 2.30GHz, 8GB 1600 MHz DDR3 RAM.

Table 1: Agents' hyperparameters: the parameter type can be either boolean (B), integer (I) or double (D).

BMRH			
Symbol	**Type**	**Range**	**Description**
l	I	[1,∞)	sequence length
n	I	[0,∞)	sequences evaluated
usb	B	$\{\bot, \top\}$	if it uses shift buffer
mo	B	$\{\bot, \top\}$	if it has to mutate once
mr	D	$[0, 1]$	mutation probability
SRH			
Symbol	**Type**	**Range**	**Description**
l	I	[1,∞)	sequence length
n	I	[0,∞)	sequences evaluated
usb	B	$\{\bot, \top\}$	if it uses shift buffer
mo	B	$\{\bot, \top\}$	if it has to mutate once
mr	D	$[0, 1]$	mutation probability
MCTS			
Symbol	**Type**	**Range**	**Description**
d	I	[1,∞)	max rollout depth
c	D	(-∞,∞)	exploration constant of UCB
e	D	(-∞,∞)	ϵ of UCB
ep	D	$[0, 1]$	expansion probability
ps	I	[1,∞)	actions sampled during expansion
rt	I	$\{0, 1, 2\}$	recommendation type

- Number of opponents[3]: 1, 3
- Budget [# of simulated actions][4]: 1 000, 5 000 and 10 000 (these budgets were chosen to keep a reasonable execution time, i.e. ∼ 13 minutes per evaluation when budget is 10 000).

The combinations of the proposed function types with the implemented variants results in 72 total functions. The number of functions can probably be reduced based on first results, e.g. if the problems are too easy or too similar to others.

3.2 Multi-Objective Functions

There are two main ways in this context to pose hyperparameter tuning as a multi-objective problem. The first is an extension of the robust and advanced usecases as described above. Winrates against different agents that ideally represent different types of gameplay can be interpreted as different objectives. Agents around the centre of the Pareto front of this problem would thus exhibit robust performance against different strategies. In contrast, configurations towards the extreme edges would result in very strong performances against specific strategies. Analysing the solutions found on the Pareto front could create better insight into the effects of different hyperparameters and allow for quick adaptation.

Alternatively, additional objectives could be introduced. Several usecases in AI-assisted game design require AI agents that exhibit specific types of behaviour, e.g. aggressive or defensive playing styles [6]. If the similarity to intended playing styles can be expressed numerically (e.g. by the percentage of aggressive moves), it is a suitable second objective. Further possible objectives are the similarity to human behaviour as well as successful collaboration with human players[5].

Furthermore, if the tuned AI algorithms are intended as opponents for human players in a single-player version of the game, creating strong opponents is probably not the sole objective. Instead, the game designers usually want to ensure a specific experience. Introducing an additional objective function that describes e.g. the tension[6] during the game could help finding configurations that result in interesting and skillful games against AI players.

3.3 Specifications for COCO

Objectives. As described above, functions could have 1 or more objectives. In case the different objectives are winrates against various AI agents, the resulting functions might even be considered many-objective problems. Because all proposed functions can be expressed as rates or percentages, the theoretical optimum is at function value 1, while the minimum is at 0.

Search Space Dimension, Scalability and Constraints. The search space dimensions and constraints are defined by the hyperparameters available to the different agents as defined in Table 1. For some parameters, it would be possible to specify a smaller area of interest based on further experiments and practical restrictions. For example, the sequence length and number of sequences evaluated is automatically restricted by the budget available to the agent.

Hyperparameter tuning problems are generally not easily scalable in search space. One option is to keep specific parameters constant, but this reduces the complexity of the problem drastically. Another option is to use an indirect representation as suggested in [4]. Alternatively, the search space of each function could also be kept constant, as scalability is not of particular interest in hyperparameter tuning problems. Tuning different agents already results in search spaces of different dimensions.

Instances. There are several options to introduce instances in this function suite. The most promising one is to vary the opponent behaviour that the tuned AI is tested against slightly, e.g. by small modifications of their hyperparameters. It would need to be ensured that the resulting fitness landscapes are similar but not identical in this case. The simplest option would be transformations such as rotations and shifts akin to instances in BBOB [5].

Runtime. Simulating 1 000 games with 4 players, i.e. evaluating a single configuration, usually takes around 60 seconds on the setup described in section 2.1. An estimate for the upper bound of an evaluation based on the average simulation speed of a game and on the maximum duration of games observed in our experiments for an algorithm using the maximally available budget is 80 seconds.

3.4 Characterisation of Fitness Landscapes

In order to be able to provide a first characterisation of the problems in the proposed suite, we discuss some preliminary experiments in the following. The first is a grid-search for the best configuration for all AI algorithms with a budget of 1 000 simulated actions,

[3]For advanced usecase: In case of 3 opponents, these are the best agent configurations found by grid-search on beginner function for BMRH, MCTS and SRH. In case of 1 opponent, it is randomly picked from this pool

[4]Budget of 1 000 used in previous work [2], results in ∼ 7 − 28 full games

[5]http://hanabi.fosslab.uk/

[6]Tension has been described as the number of times the lead changed during the game as well as closeness in score [12].

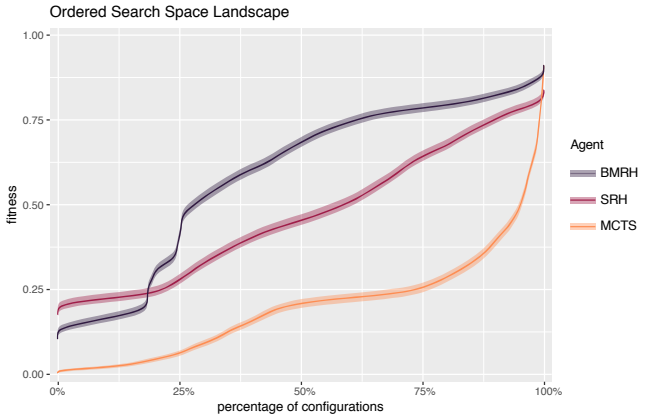

Figure 2: Ordered fitness landscape.

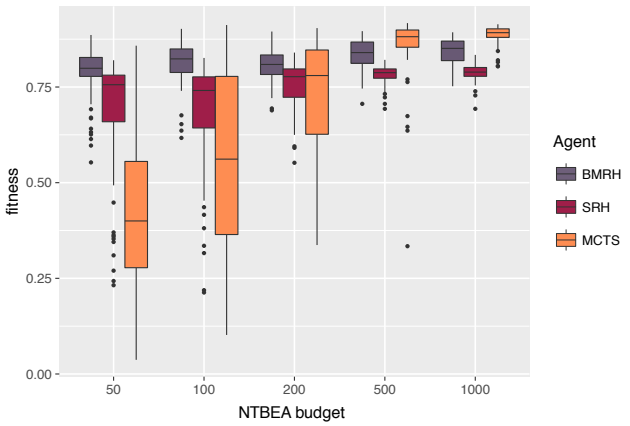

Figure 3: NTBEA results for the three agents with different budgets.

tested against 3 1-step look-ahead opponents (*beginner* usecase). The winrates that were obtained are plotted in figure 2, where the obtained values are sorted for easier visualisation. The widths of the coloured lines represent the observed standard deviation.

From the plot, we can see that the performance of all agents depends strongly on their hyperparameter configuration. This demonstrates that finding suitable hyperparameters is indeed crucial in game AI. The grid-search is able to reach winrates of above 90% for MCTS and BMRH, and above 80% for SRH. It is further promising to see the range of achieved winrates and the relatively low corresponding noise. This should create sufficient feedback for an optimisation algorithm to operate successfully. The plot additionally suggests that tuning MCTS with simple approaches such as random search is more difficult than optimising the other algorithms, because relatively fewer configurations obtain good winrates. This observation was confirmed by experiments with random search, which resulted in mean winrate of around 20%, whereas for BMRH, almost 60% winrates are reached on average.

We ran a preliminary test to support the above conjectures on the difficulty of the functions by running the N-Tuple Bandit Evolutionary Algorithm (NTBEA), a hyperparameter tuning algorithm specialised in game AI [7], on the *beginner* problems as specified above. The best winrates obtained by NTBEA for each algorithm and for different evaluation budgets are depicted in figure 3.

We find that even for very low budgets, NTBEA already finds solutions with high winrates for BMRH and SRH. Despite the seemingly easy problem, the results are still significantly better than those achieved by Random Search. It is however very striking how NTBEA is able to discover far better configurations of MCTS in a consistent manner (low variance) if given a higher budget. This also suggests that the fitness landscapes for the three AI algorithms are very different.

Instead of landscape walks, which are difficult in a mixed-integer space, we created boxplots of the distribution of fitness values for the various categorical and boolean parameters. These distributions mostly look similar, suggesting that the problems are not separable by any of these categories. The functions in the *beginner* usecase are thus of suitable complexity for testing evolutionary algorithms, especially for low budgets. The remaining function types will likely pose even more difficult problems, as the configurations will probably be less distinguishable by the achieved winrates. This is because a larger percentage of configurations will result in losses against the best agents found previously, thus resulting in larger plateaus in the corresponding fitness landscapes.

4 CONCLUSIONS AND FUTURE WORK

Hyperparameter tuning is an important optimisation problem, especially in context of real-world applications such as games. In this paper, we suggest a function suite consisting of several diverse hyperparameter tuning problems for game AI implemented using the Rinascimento framework. An implementation of a working interface to GBEA can be found in our public github repository[7]. A preliminary analysis of the fitness landscapes demonstrates reasonable optimisation complexity as well as diversity.

There are several ways to further extend the proposed function suites. For example, the number of tuneable agents implemented in the Rinascimento framework could be increased. Another option is to introduce an additional hyperparameter for all agents by providing different heuristics to evaluate the game states. This would be especially interesting as finding appropriate heuristics is an open question in current research [1]. Furthermore, the Rinascimento framework also supports modifications of the deck available in the game. A function suite around deck generation could thus be implemented using the same framework and agents.

Besides the possible extensions, a more detailed investigation of the fitness landscapes in the proposed suite should be conducted. It would for example be interesting to investigate the correlation between search and objective space for the continuous and integer-valued parameters. In addition, noise handling could be made an essential part of problems in GBEA. For example, the reported fitness values in our implementation of the proposed function suite are currently averaged over 1 000 simulated games in order to allow for relatively low noise (see figure 2). However, it would be interesting to allow the optimiser to pick which configurations should be evaluated repeatedly (as done by NTBEA, see figure 3).

[7] https://github.com/TheHedgeify/coco/tree/rw-new-suite

ACKNOWLEDGMENTS

This work was funded by the EPSRC CDT in Intelligent Games and Game Intelligence (IGGI) EP/L015846/1.

REFERENCES

[1] Ivan Bravi, Ahmed Khalifa, Christoffer Holmgård, and Julian Togelius. 2017. Evolving game-specific UCB alternatives for general video game playing. In *European Conference on the Applications of Evolutionary Computation*, Giovanni Squillero and Kevin Sim (Eds.). Springer, Cham, Switzerland, 393–406.

[2] Ivan Bravi, Simon Lucas, Diego Perez-Liebana, and Jialin Liu. 2019. Rinascimento: Optimising Statistical Forward Planning Agents for Playing Splendor. In *Conference on Games (CoG)*. IEEE, Piscataway, US, accepted.

[3] Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in games* 4 (2012), 1–43.

[4] 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)*, Anne Auger et al. (Eds.). Springer, Cham, Switzerland, 296–307.

[5] Nikolaus Hansen, Anne Auger, Olaf Mersmann, Tea Tusar, and Dimo Brockhoff. 2016. COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting. *CoRR* abs/1603.08785 (2016). arXiv:1603.08785 http://arxiv.org/abs/1603.08785

[6] Christoffer Holmgård, Michael Cerny Green, Antonios Liapis, and Julian Togelius. 2018. Automated Playtesting with Procedural Personas through MCTS with Evolved Heuristics. *IEEE Transactions on Games* (2018), accepted. http://arxiv.org/abs/1802.06881

[7] Simon M Lucas, Jialin Liu, and Diego Perez-Liebana. 2018. The N-Tuple Bandit Evolutionary Algorithm for Game Agent Optimisation. In *Congress on Evolutionary Computation (CEC)*. IEEE, Piscataway, US, 221–229.

[8] Diego Perez, Spyridon Samothrakis, Simon Lucas, and Philipp Rohlfshagen. 2013. Rolling Horizon Evolution Versus Tree Search for Navigation in Single-Player Real-Time Games. In *Genetic and Evolutionary Computation Conference (GECCO)*. ACM, New York, NY, 351–358.

[9] Tea Tušar, Dimo Brockhoff, and Nikolaus Hansen. 2019. Mixed-Integer Benchmark Problems for Single- and Bi-Objective Optimization. In *Genetic and Evolutionary Computation Conference (GECCO)*. ACM Press, New York, accepted.

[10] Vanessa Volz and Boris Naujoks. 2019. Investigating Uncertainties with a Game-Benchmark. In *Uncertainty Quantification and OPtimization (UQOP)*. https://uqop.sciencesconf.org/246597

[11] 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)*. ACM Press, New York, accepted.

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