On the computational cost of GVG-AI games and tree search algorithms

Peter Mlakar, Tom Vodopivec

Faculty of Computer and Information Science, University of Ljubljana, Slovenia
E-mail: tom.vodopivec@fri.uni-lj.si

Abstract

The general video game AI (GVG-AI) competition is becoming an established venue for benchmarking video-game-playing algorithms. There, most high-ranking algorithms use Monte Carlo tree search (MCTS) upon simulated future moves. The computation time of such simulations strongly impacts the algorithms' performance and the values of their learning parameters. In our study we overview the computational cost of the GVG-AI games and explore which features contribute most to it – the presented results can help MCTS researchers with offline parameter tuning and with implementing approaches for automatic (online) tuning based on the game-features we identified as impactful.

1 Introduction

Most of the highest-ranking algorithms that compete in the general video game AI (GVG-AI) competition [1] use Monte Carlo tree search (MCTS) methods [2]. The canonical MCTS algorithm, UCT [3], balances the exploitation and exploration dilemma [4, 5] with the help of the UCB equation [6] and an exploratory constant C_p. The optimal value of the constant – the optimal rate at which the algorithm prefers exploration over exploitation – is strongly dependant on the number of simulated moves the algorithm can produce in a single search [7]. Consequently, the optimal value of the exploratory constant is dependant on the computational cost of the underlying problem – and it can vary wildly across different problems. Given that the GVG-AI games significantly differ both in terms of their dynamics, as well as in terms of their computational cost, setting an overall efficient value of C_p is non-trivial. To ease the process, in this study we overview the computational cost of the GVG-AI games and identify game features that are highly correlated with it, and which could therefore be used for automated parameter-tuning. This can help improve the base performance of several general game-playing algorithms that use MCTS (or similar) algorithms as their backbone.

2 Background

We explain the MCTS framework in the context of the GVG-AI competition, the difference between closed-loop and open-loop implementations, and the dynamics of the UCB equation used by MCTS algorithms.

2.1 The General Video Game AI Competition

The GVG-AI competition [8, 1] is a worldwide platform on which researchers tackle the problem of general intelligence in video games. Participants face the challenge of creating a game agent, that has to play many different game types. One game might require collecting items, while another pushing boxes into right places. The games run in real-time, allowing 40 ms of computational time per each move. The games feed back to the controller the current game state: a win/lose flag, a score, and a list of observations – entities in the game. The constructed controllers are tested in these games and ranked according to the score they accumulate. The problem of general intelligence is far from trivial since one has to construct an agent that has to play a variety of games without knowing the type of game it is playing. The results from all GVG-AI competitions and the source code of the competing controllers are available on the official GVG-AI web page [9].

2.2 Monte Carlo Tree Search

Monte Carlo tree search [2, 7] represents a class of algorithms that try to find the best set of actions that yield the optimal result over a period of time. It works by incrementally building a tree structure, which it uses for predicting the best possible actions for maximising the reward. The nodes in the tree are game states derived from taking actions from the initial state to the following (simulated) states. The edges in the tree represent possible actions. The incremental building of the tree is broken in iterations, where each iteration is made of four phases: selection of actions based from the tree root till tree leaf, expansion of the tree with new nodes, simulation of the game till a terminal state is reached, and backpropagation of the received feedback up the tree to the root state.

2.3 Closed-loop and open-loop implementations

The GVG-AI framework provides two implementations of the MCTS algorithm described earlier: the closed-loop and open-loop implementations [10] – the *sampleMCTS* and *sampleOLMCTS* controllers, respectively. The two differ in how they handle the simulation of game states and expansion of the tree with new nodes.

The sampleMCTS controller, when expanding the tree, stores the complete current game state into the newly-added node. In this way, when it revisits the same state, it does not need to simulate the move to get to this state again, but rather only observes the copy of the state already stored in the node. This approach might require less computational time when states get revisited often and when copying them is less costly than re-simulating moves, but it requires more memory and does not account for stochasticity in the simulations.

The sampleOLMCTS controller does not save states into nodes – it always moves to the next state by simulating moves, regardless if the algorithm already visited them before. This implementation is more flexible in cases of stochastic games where the game state depends on a random variable that governs some of its changes.

2.4 Upper Confidence Bound applied to trees

The canonical MCTS algorithm – UCT [3] – uses the UCB1 formula (1) [6] at each tree node to balance exploration and exploitation of the search space:

$$UCB_i = Q_i + C_p\sqrt{\frac{\ln(n+1)}{n_i}} \quad (1)$$

where UCB_i is the estimated UCB-value of the i-th child in the current node, Q_i is the $[0,1]$-normalized value of the node based on previous experience, n is the total number of visits of the parent node, and n_i is the number of visits of the evaluated child node. The parameter C_p governs the exploratory tendency of the algorithm – its value can highly impact the performance, it is often set to a default constant value of $\sqrt{2}$. The equation consists of two terms: the left term promotes the exploitation of the child nodes that offer high average reward values, whereas the right term promotes exploration.

3 Experiments

We describe our experimentation process, the selected control variables, and the observed features.

We used the official GVG-AI framework from October 2016. The control variables for our experiments were the choice of the GVG-AI game, the game level (5 per game), the framework-given game-playing controller (a random-selection algorithm, *sampleMCTS*, or *sampleOLMCTS*), and the value of the exploratory parameter C_p ($\frac{1}{2}\sqrt{2}$,$\sqrt{2}$, or $2\sqrt{2}$). For each combination of control variables in each 40-ms interval given per move we observed the following: the number of simulated MCTS iterations, the number of simulated moves – *advances*, number of simulated episodes (of each game), the average duration of the simulations (in moves), game board size, the average score the agent achieved in its simulations, the number of simulated wins and the number of observations in the current (real) game state. The counts of simulated advances and iterations allows us to compare the computational cost of games and controllers. The number of episodes differs from the number of MCTS iterations, as iterations get truncated after 10 moves from the root node (according to the original GVG-AI implementation of MCTS), whereas episodes do not.

The measurements were conducted on all 72 games of the GVG-AI framework. Each game contains 5 levels for the agent to play through. Considering that games have a duration of up to 2000 moves (of 40-ms each) in which we measured the features described above, and that we performed 20 repeats of each game, we collected up to 200000 samples of data per single combination of control variables.

The measurements were conducted on a computer with Intel(R) Core(TM) i7-5960X 3.00GHz CPU, 32GB DDR4 2133MHz memory, and Windows 7 operating system.

4 Results

Here we first provide a comparative overview of the computational cost of the GVG-AI games, and then analyse the correlation between the observed features and the computational cost.

4.1 Comparison of computational costs

In the majority of GVG-AI games MCTS algorithms manage to simulate several hundreds of advances per 40 ms (Figure 1). However, there is a substantial difference between the two MCTS implementations: closed-loop MCTS in average invokes 892.0 advances, whereas open-loop MCTS invokes 1352.0 advances. This is expected, due to the nature of closed-loop MCTS not invoking an advance when visiting a state already stored in the tree (as explained earlier). Nevertheless, the closed-loop implementation in average computes 119.8 MCTS iterations, whereas open-loop 141.3 iterations (Figure 2). This shows that an open-loop implementation is faster for GVG-AI games – it seems more efficient to re-simulate previously visited states rather than losing time and memory in copying them in the tree nodes.

Table 1 points out the three most demanding and the three least demanding games[1]. When developing GVG-AI algorithms it is meaningful to benchmark them on these games, since they represent the two extremes – the performance and behaviour of the algorithms might differ wildly. In computationally heavier games the algorithms perform less simulated moves per 40 ms and therefore less MCTS iterations – the algorithms explore a smaller portion of the search space and build a smaller tree. In such case it is often better to set a smaller exploratory tendency – to set C_p closer to 0, and vice versa.

The upward spikes of the number of iterations happen near ending positions in games – the number of advances per iteration gets low, because the game is terminating, and thus the number of iterations (and episodes) per 40 ms gets higher.

4.2 Feature correlations

Whereas the results above help GVG-AI practitioners in offline tuning of their game controllers, here we analyse

[1] Contact the authors for full results.

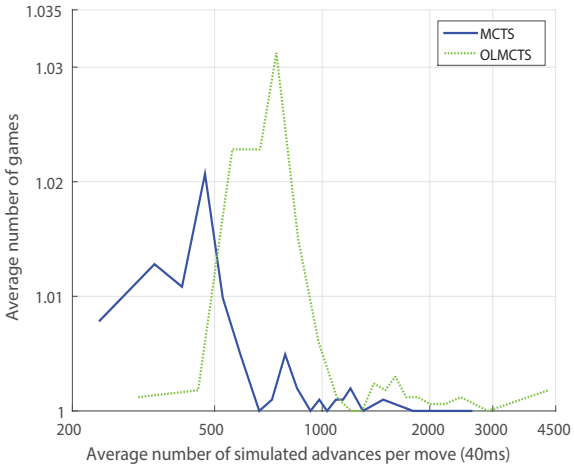

Figure 1: The distribution of GVG-AI games according to the computational cost. The latter is expressed as the number of simulated moves (advances) that an algorithm produces in 40 milliseconds (this is the time allowed by the GVG-AI framework to output a move), whilst the average number of games expresses the average amount of games in which the agent managed to simulate a certain (average) number of moves (advances) in 40 milliseconds.

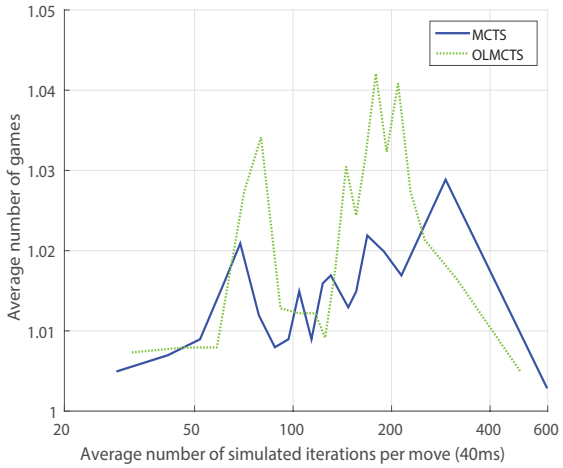

Figure 2: The distribution of GVG-AI games according to the computational cost, when the latter is expressed as the number of simulated MCTS iterations in 40 milliseconds.

which game features are correlated with the computational cost and could be as such used for automated or online parameter tuning.

The assessed features are listed in Section 3. Some exhibit significant correlation with the computational cost (Table 2), but the *average number of observations in game states* is most strongly correlated both with the number of advances per move (Figure 3) and with the number of iterations per move (Figure 4). The computed Pearson linear correlation coefficients are in the range $[-0.35, -0.58]$ (for the four data series presented in the two figures). These results confirm that with increasing the number of observation objects in the current game state, the computational cost of simulating the game increases as well.

Table 1: The most and least computationally-demanding GVG-AI games according to the number of simulated moves of an open-loop MCTS algorithm.

Game name	Average number of simulated advances in 40 ms
blacksmoke	291.0
boulderdash	377.7
lasers	438.0
jaws	3916.1
missilecommand	4381.5
whackamole	4487.3

Table 2: The correlation (Pearson coefficient) of GVG-AI game features with the computational cost of a GVG-AI game (expressed as the number of simulated advances per 40 ms).

Game feature	Correlation with number of simulated advances
Num. simulated episodes	0.24
Average win rate	0.22
Score in current state	0.03
Avg. episode duration	-0.21
Num. observations	-0.56

The correlation seems to be non-linear (note the logarithmic scale on the figures).

Considering the results above, we suggest that MCTS practitioners measure the average number of observations their agents encounter in the real game states and use this as an input vector for online tuning of the exploratory parameter C_p (for example, with linear regression or with more complex methods). Based on rough local experiments with our controller *ToVo1* we confirm such an approach improves the performance of the algorithm.

5 Related work

To our knowledge there are no studies of the impact of GVG-AI game features on the computational cost of games and MCTS-based agents, nor studies in using such features for online parameter tuning. However, GVG-AI game features were thoroughly studied for the purpose of finding the most suitable algorithm for each game [12] or for estimating game difficulty, which helped the researchers identify well-performing combinations of MCTS and evolutionary algorithms[11].

In the broader scope of MCTS, the learning parameters are usually set and adjusted manually [7]. Automated tuning of C_p was studied by Kozelek [13], but he based the process on the variance of the feedback and not on game features. Chaslot et al. [14] used a feature-based cross-entropy method to tune parameters for playing the game of Go, but the process was run offline – prior the playing the game. There are some other methods for tuning parameters [15, 16], but these are both specific to Go, which is unlike ours, which covers a wide range of (GVG-AI) games.

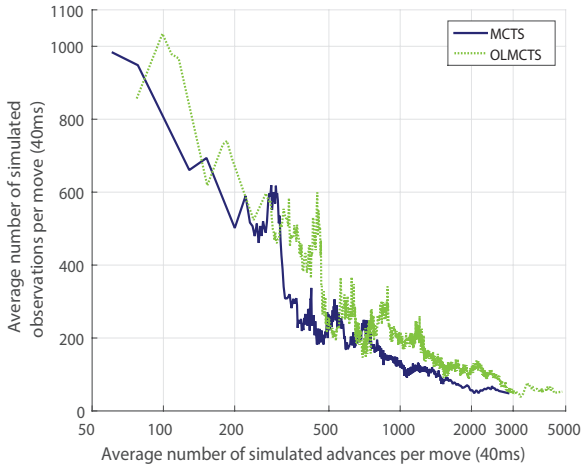

Figure 3: The correlation between the number of simulated advances per move and the number of game-state observations.

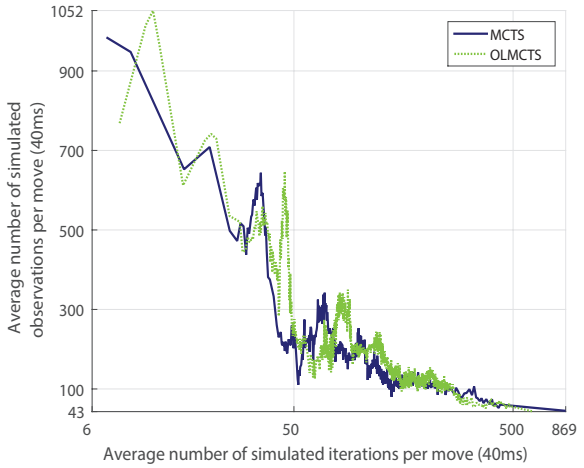

Figure 4: The correlation between the number of compute MCTS iterations per move and the number of game-state observations.

6 Conclusion

In this paper we briefly introduced the general video game AI (GVG-AI) competition and two basic Monte Carlo tree search (MCTS) implementations. We experimentally assessed the computational cost of the first 72 GVG-AI games – we expressed it with the number of simulated moves (game-state advances) and simulated MCTS iterations in a single 40-ms time span. We evaluated the correlation of the computational cost with several game features and identified that the number of observations (game entities) in the real game states are highly suitable for predicting the computational cost of the game, and as such are also suitable for automated online tuning of MCTS parameters, especially for tuning the exploratory tendency of the algorithm.

We aim at providing GVG-AI practitioners with guidelines about configuring their algorithms best according to the computational cost of the games. We understand that some parameters are dependant upon the number of iterations or advances an agent is capable of executing. An agent could collect the amount of observations in games it is simulating, and from that data estimate the computational complexity of the game. We are positive that this study can help GVG-AI practitioners with optimizing their parameters given some additional knowledge about the game features, which can be gathered during real-time with little overhead.

References

[1] D. Perez-Liebana et al.: General Video Game AI: Competition, Challenges and Opportunities, Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, 2016

[2] R. Coulom: Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search, Computers and Games, 2006

[3] L. Kocsis, C. Szepesvari: Bandit Based Monte-Carlo Planning, Proceedings of the Seventeenth European Conference on Machine Learning, 2006

[4] R. Sutton, A. G. Barto: Reinforcement Learning: An Introduction, 1998

[5] R. Sutton, A. G. Barto: Reinforcement Learning: An Introduction, Second Edition, in progress, 2017

[6] P. Auer, N. Cesa-Bianchi, P. Fischer: Finite-time Analysis of the Multiarmed Bandit Problem, Machine Learning, 2002

[7] C. B. Browne et al.: A Survey of Monte Carlo Tree Search Methods, IEEE Transactions on Computational Intelligence and AI in Games, 2012

[8] D. Perez et al.: The 2014 General Video Game Playing Competition, IEEE Transactions on Computational Intelligence and AI in Games, 2015

[9] The General Video Game AI Competition, http://www.gvgai.net/

[10] D. Perez et al.: Open Loop Search for General Video Game Playing, Proceedings of the 2015 on Genetic and Evolutionary Computation Conference, 2015

[11] H. Horn et al.: MCTS/EA Hybrid GVGAI Players and Game Difficulty Estimation, Proceedings of the IEEE Conference on Computational intelligence and Games, 2016

[12] P. Bontrager et al.: Matching Games and Algorithms for General Video Game Playing, Artificial Intelligence and Interactive Digital Entertainment, 2016

[13] T. Kozelek: Methods of MCTS and the game Arimaa, Master's thesis, 2009

[14] G. Chaslot et al.: Cross-entropy for Monte-Carlo tree search, European Workshop on Reinforcement Learning, 2008

[15] G. Chaslot et al.: On the huge benefit of quasi-random mutations for multimodal optimization with application to grid-based tuning of neurocontrollers, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, 2009

[16] A. Bourki et al.: Parameter tuning by simple regret algorithms and multiple simultaneous hypothesis testing, International Conference on Informatics in Control, Automation and Robotics, 2010