Challenges for Implementing Monte Carlo Tree Search into Commercial Games

Matthew Bedder[*]
Department of Computer Science
University of York
Heslington, UK
mb708@york.ac.uk

Daniel Kudenko
Department of Computer Science
University of York
Heslington, UK
daniel.kudenko@york.ac.uk

Simon M. Lucas
School of CSEE
University of Essex
Colchester, UK
sml@essex.ac.uk

Abstract

Monte Carlo Tree Search is a stochastic tree search technique that has seen much interest within academia due to its strength of play in card and board games. Despite this interest, it has rarely been used in commercial games. In this extended abstract I describe how MCTS works, suggest some of the issues facing its application to commercial games, and suggest some new areas of research that could address these challenges.

Introduction

For a long time the video game industry has been at the forefront of technology, with many innovations in computer hardware, rendering techniques, and artificial intelligence either being invented for or being first used in video games. Video games can provide a useful test-bed for state-of-the-art techniques by providing a wide range of challenges, and a clear path to commercialisation.

Within the academic community there has recently been considerable interest in an Artificial Intelligence technique known as Monte Carlo Tree Search (MCTS). This technique differs from many expert- and heuristic-based systems by using the results of simulations to guide the decision making process. This technique has seen many successes when applied to complex board and card games, yet has seen only a handful of implementations into commercial video games.

In this extended abstract I introduce the basics of Monte Carlo Tree Search and its strengths, and suggest some reasons why it may currently be unsuitable for the games industry.

Monte Carlo Tree Search

Monte Carlo Tree Search (Coulom 2006) is a stochastic tree search algorithm often used in game decision making. The approach iteratively builds a partial decision tree for a given domain, repeatedly performing the actions of *selecting* a part of the tree to expand, *expanding* the tree at this location, *simulating* how a game played from the selected state may turn out, and *backpropagating* the results of the simulation back up the tree. After a partial tree of sufficient size is built, a decision of which action to select from the root node is made.

Further explanations of these steps are as follows:

Selection First we *select* the part of the tree we want to look at next, navigating down the tree from the root until the first non-expanded node is found, taking into account the needs of *exploration* and *exploitation* (Kocsis and Szepesvári 2006).

Expansion Next we add a node, relating to an unexplored state, onto the partial decision tree.

Simulation The simulation step of MCTS involvves performing a rough assessment of the value of the new tree node by simulating the game using some *rollout policy* from the associated game state until a terminal game state is reached.

Random simulations, where the actions to perform are selected uniformly, are often used, although domain-specific simulations (*heavy rollout policies*) can sometimes improve performance by providing more accurate state-value estimations.

Backpropagation In the final stage of an MCTS iteration, the resulting reward from the simulation is used to update the knowledge of the value of the newly added node and all its ancestors.

Action Selection The final step of Monte Carlo Tree Search is to select an action to perform from the generated partial decision tree. Statistics over the nodes at the first level of the tree are used to make a decision, with a common approach being to select the action leading to the most visited node (Chaslot et al. 2008).

Successes of MCTS

Monte Carlo Tree Search has been successful over a variety of different board and card games, with the greatest successes arguably being seen in applying the technique to the game Go.

Go is a turn-based, two player game of perfect information, usually played on a 19×19 board, with players placing stones on the board to try to capture territory. Using a complex set of rules, Go is a very difficult game for humans to play, and was long considered too difficult for AI players to ever play well.

[*]This work is supported by the EPSRC Centre for Doctoral Training in Intelligent Games & Game Intelligence (IGGI) [EP/L015846/1]

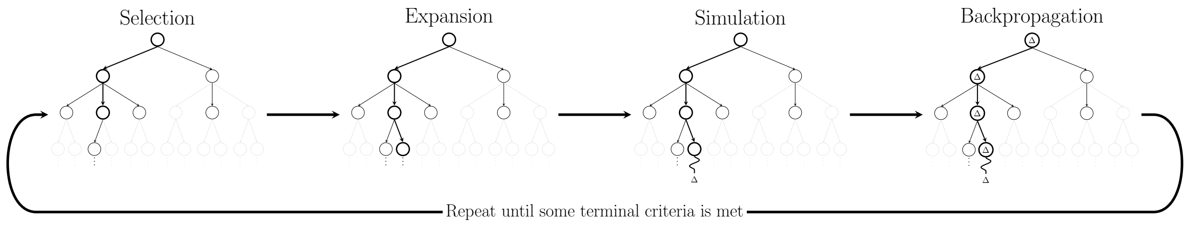

Figure 1: The four main stages of an iteration of Monte Carlo Tree Search

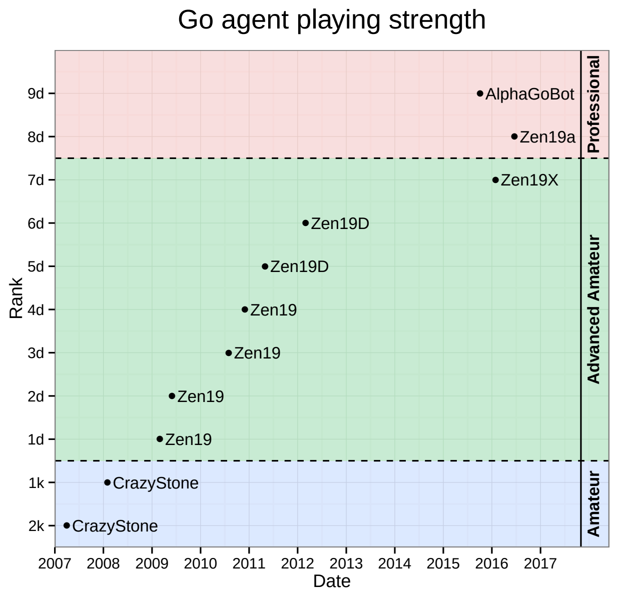

Figure 2: Go Rankings from (Sensei's Library 2016) for the best Go AI agents of each year. Each agent listed uses a variant of MCTS for decision-making.

Since the technique's proposal in 2006, MCTS-based Go agents quickly began to show their competence at playing the game. Over time the performance of these agents has gradually and steadily increased (Figure 2). Recently AlphaGo, a Go agent using a combination of MCTS and Deep Learning approaches, was able to beat the world champion of Go (DeepMind 2016), further demonstrating the strength of the technique.

A different area in which MCTS has seen great success is that of general videogame playing (Perez et al. 2015). In this domain, an agent must learn to how to play a game that it has not seen before, a task well suiting base MCTS implementations due to its aheuristic and domain-independent nature.

Challenges for MCTS in video games

Although complex board games can still be difficult for AI players, the challenges in commercial video games are often much larger. Go, for example, may have a large state-space and branching factor, but these absolutely dwarfed by strategic video games like Civilization, with the larger state-space having a dramatic impact on sampling-based techniques like MCTS (Branavan, Silver, and Barzilay 2011).

Another concern when implementing techniques like MCTS into commercial video games is the amount of computational resources required. Research-based MCTS implementations are often designed to run on very powerful, custom compute servers (DeepMind 2016), dramatically more than can be afforded to videogame AI implementations (Mountain 2015).

A final limiting factor to the uptake of the technique by the video game industry that we have identified is that many optimisations of MCTS have been designed for a single domain, with the reason for their effect on performance being ill-understood. This is clearly seen in the use of heavy rollout policies, where it is not unusual for increased rollout policy strength to result in reduced MCTS strength (Gelly and Silver 2007) This lack of understanding into the exact impact of MCTS optimisations means that potential implementers of the technique have to undergo trial-and-error to find a solution that works sufficiently well.

References

Branavan, S.; Silver, D.; and Barzilay, R. 2011. Non-Linear Monte-Carlo Search in Civilization II. In *Proc. International Joint Conf. on Artificial Intelligence*.

Chaslot, G.; Winands, M.; Uiterwijk, J.; van den Herik, H.; and Bouzy, B. 2008. Progressive Strategies for Monte-Carlo Tree Search. In *Proc. Joint Conf. on Information Sciences*.

Coulom, R. 2006. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In *Computers and Games*. Springer. 72–83.

DeepMind. 2016. AlphaGo. http://deepmind.com/alphago.html. Accessed March 2016.

Gelly, S., and Silver, D. 2007. Combining Online and Offline Knowledge in UCT. In *Proc. IMLS International Conf. on Machine Learning*.

Kocsis, L., and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In *Proc. European Conf. on Machine Learning*. 282–293.

Mountain, G. 2015. Tactics in Fable Legends. http://gwaredd.github.io/nuclai_mcts. Presented at nuclai.

Perez, D.; Samothrakis, S.; Togelius, J.; Schaul, T.; Lucas, S.; Couetoux, A.; Lee, J.; Lim, C.; and Thompson, T. 2015. The 2014 General Video Game Playing Competition. *Trans. Computational Intelligence and AI in Games*.

Sensei's Library. 2016. KGSBot Ratings. http://senseis.xmp.net/?KGSBotRatings.