How to Play Computer Games

Ryan Bunney
email: 21298244@student.uwa.edu.au

March 2017

Abstract

Developing algorithms and methods to play games has been a corner-stone of artificial intelligence research for more than half a century. The journey to create algorithms that can defeat the best of human intelligence has proven a perpetual challenge for AI researchers. The last of these challenges has been to establish an AI that can play the game Go. This paper discusses how AI researchers can represent games like Go for computers, and provides the popular Minimax algorithm as an example solution. It then addresses the recent uses of Monte Carlo Tree Search in game-playing research area; why it is better than previous methods; and comments on its successes and potential applications in other disciplines.

Keywords: Artificial Intelligence, AlphaGo, Monte-Carlo Tree Search, Go, Adversarial search.

1 Introduction

Since the term was first coined in the Dartmouth Conference of 1956, Artificial Intelligence (AI) has captured the imagination of the public and the scientific community [3]. The continual endeavour to create computer programs that are able to outperform humans in their favourite intellectual past-times - checkers, backgammon, chess - has seen numerous developments in the realms of game-playing algorithms, and the ability for computers to master tasks that it was thought only people could perform [10]. Since the defeat of Gary Kasparov in 1997 by IBM's Deep Blue, the 'last bastion of human superiority' has been the ancient Chinese game of Go [2]. Finding efficient solutions to playing complex games such as Go has led to exciting developments in how computers play games. One of these exciting developments is the Monte-Carlo Tree Search algorithm, which was used in 2016 - in collaboration with deep learning techniques - to defeat the European Champion of Go. [13]

In this paper, I introduce how games are represented to computers, and describe the Minimax algorithm with respect to tic-tac-toe. Certain disadvantages of this approach are discussed, after which I introduce the concept of the Monte-Carlo Tree Search. I make the claim that Monte-Carlo Search Trees are a pivotal moment in game-playing algorithms, both in their ability to deal with highly complex games as in the case of Go, and also in the application to wider areas of computing.

2 Background

2.1 Representing Games for Computers

Most common games, like that of chess or checkers, are deterministic, turn-taking and 'zero-sum games of perfect information [12]. That is, each player has the same information about the current state of the game, and there is no chance associated with the next player's move. This means the scoreboard at the end of the game will always be equal and opposite; if one person wins in chess, the other must have lost.

Games are depicted in the form of a decision tree and a game of tic-tac-toe provides a simple example.

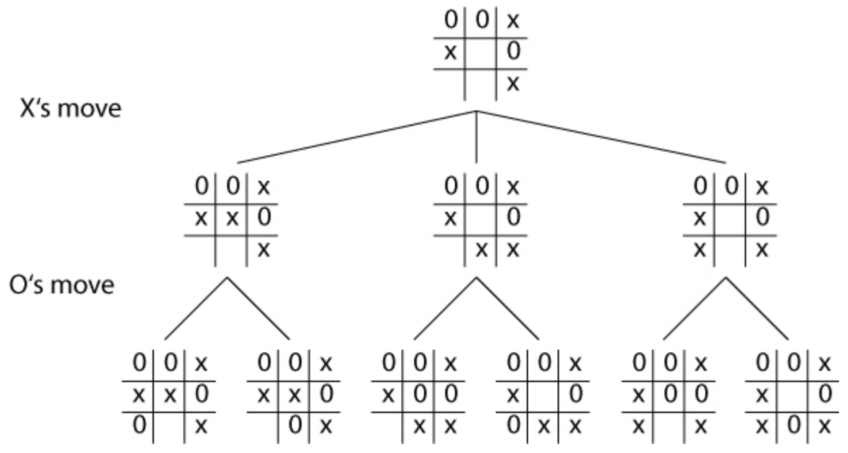

Figure 1: End-game moves of tic-tac-toe game [14]

Figure 1 shows the final moves of a tic-tac-toe game with each layer representing the set of decisions that could be made. Figure 1 demonstrates that even though there are only 3 moves left in the game, we have 9 different possible game states after 2 turns. The computer must search through this game tree to determine which moves it must make to win. One of the pioneering algorithms used to navigate search trees like the example above is the Minimax algorithm [12].

2.2 Minimax

In the Minimax algorithm, we depict the first player ('X') as MAX, and their opponent ('O') as MIN. This is simply because MAX wants to maximise their own final score, and MIN wants to minimise MAX's final score. The score is an arbitrary way of representing the outcome of the game to a computer. In the example of a tic-tac-toe game, the score can be measured by the final outcome of the game; winning is 1, losing is -1, and a draw is 0. The algorithm then chooses moves that will lead to states that prove the best for MAX, whilst taking into accounts moves that MIN make that are supposed to minimise MAX's score.

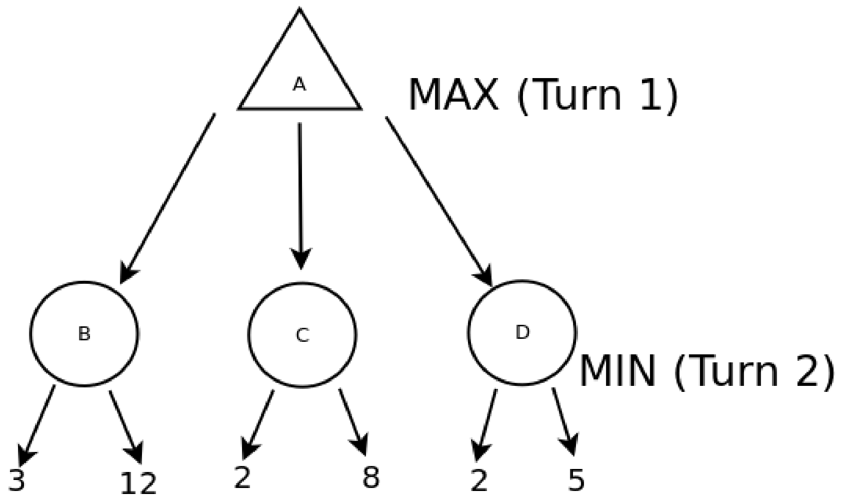

Figure 2: 2-turn Minimax game

Figure 2 shows a simple 2-turn Minimax game, in which the aim is to get the highest score. In Turn 1, MAX is in state A and has 3 options; B, C, and D. The way Minimax works is by picking searching through each potential tree and working out which end-result MAX will end up with. In this example, if MAX chooses B, it knows that in Turn 2 MIN will choose 3, because that is the minimum possible score for MAX. If MAX were to then consider state C, it would see that MIN is going to pick 2; this is not a good state for MAX to pick, because it knows from its search that the previously known high-score is greater. This is true also for state D. Hence, the Minimax algorithm will suggest that MAX should choose to move to state B.

In a tic-tac-toe game, Minimax works the same way; by searching through the tree and identifying which states will win, the computer can ensure MAX makes the best intermediate steps to win the game.

3 Monte-Carlo Tree Search

3.1 Drawbacks of Previous Work

If we look back at Figure 1, we can see how that in one state with three moves left, there are 9 potential sub-states we have to search through. If it were the beginning of a tic-tac-toe game, we can see how the combinations start to pile up quickly. If we apply the same algorithm to a game like Chess, the number of different tree branches is prohibitively high. One can use algorithmic heuristics that limit the depth of the search - that is, only search a pre-defined number of moves into the future - or use a ‘pruning’ algorithm that identifies branches that are unlikely to be useful in the future. These have had success in defeating grandmasters in Chess [1, 8], but are unable to play in the amateur leagues of Go. This is because thenumber of possible positions in Go is more than 10^{100} times that of Chess, which is more than the number of atoms in the universe.

3.2 Monte-Carlo Tree Search

The Monte-Carlo Tree Search (MCTS) provides a general approach to address the problem of very large search trees. Monte-Carlo methods have been used in games that are incomplete in knowledge and contain random element, such as scrabble or dice-games [4]. The goal of MCTS is to balance exploration of the game tree, and exploitation of the game tree. Exploration is the same as searching, as we have done in algorithms Minimax, whereas exploitation is when the algorithm chooses a node that seems beneficial at that point in time, and continues with that node. Exploration and exploitation are thus a balancing act of being ‘greedy’ by choosing options that may be a good idea immediately, and searching through the game states to see if there are better options in the future.

The algorithm has 4 main stages:

- Selection
- Expansion
- Simulation; and
- Back-propagation

When the algorithm goes through Selection, a leaf at the bottom of the tree that has been reached previously is selected (Refer to Figure 3). If this is not an end-game state, the algorithm Expands the leaf to explore child nodes. Similar to Minimax, a value is placed on a node to indicate its potential as a 'winning' path; we also keep track of how often we reach this node throughout the Simulation phases. If we keep coming across the same node, and reaching positive end-game states, this is a good sign it is a worthwhile node to exploit. Once the Simulation phase is complete, we have a revised value for the child node, and can choose to exploit that node in the Selection process again, or select another node in the tree to Expand and inform the algorithm [7].

The algorithm selects a node through the use of a 'policy'. This policy is a form of statistical weighting that can be obtained through a number of methods. [5]. The advantage of a policy like this is that as the search progresses, the policy weights different nodes more accurately based on the explored success; this happens without the need for programming any game-specific knowledge into the algorithm. This is how, in combination with artificial neural networks, the MCTS algorithm was implemented in the AlphaGo computer in 2016. By analysing its own game tree and learning what paths were better as it played the game, it defeated the European Champion 5 games to 0, in a stunning and pivotal victory for in AI [13] .

3.3 Other implications

One of the exciting aspects of the MCTS algorithm is the generalist nature of its policy. This means that it is applicable to a wide variety of different areas. A classic programming problem is scheduling tasks into a particular order of execution. [15], and Monte Carlo search has been demonstrated as effective in its application to task scheduling [11]. An example implementation of significance is the DALiuGE framework, which is designed to process the science data that will arise from the Square Kilometre Array [9] which has requirements to process huge amounts of data [16].

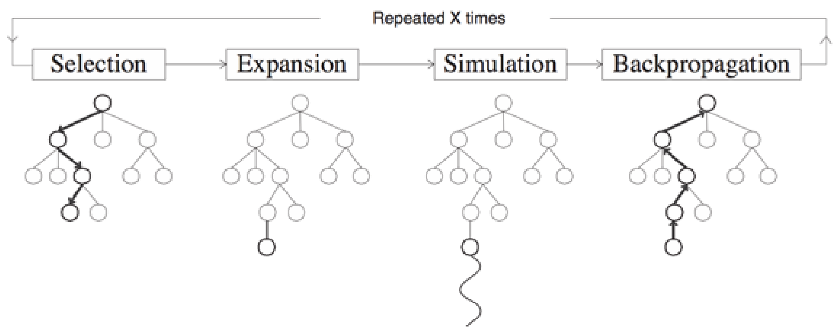

Figure 3: Four stages of Monte-Carlo Tree Search [6]

4 Conclusion

This paper has provided a comprehensive discussion on previous methods used to play 'classic' games such as Chess and Go. The Minimax algorithm was put forth as a seminal meSthod for approaching adversarial games with deterministic outcomes, and then set about demonstrated with an example. This approach has key deficiencies when it came to the number of moves (on average) made in a particular game; as a result, heuristics such as pruning were mentioned as additions to the search. These improvements, however, have been shown to be inferior to human skill when playing Go. This paper then demonstrated how the development the Monte-Carlo Tree Search algorithm has proved pivotal in how computers play games against humans. The use of the MCTS method, in conjunction with deep learning techniques, led to the computer AlphaGo beating a human champion in 2016. The success of the MCTS is such that computers have now beaten what was believed to be the final test in the battle between human and artificial intelligence. Other exciting applications, such as that of task-scheduling in big-data pipelines, also demonstrate the benefits the algorithm has in general solutions to difficult problems.

Acknowledgements

Acknowledgements must be made for Georgia Hay and Tomas Cokis, who helped with grammar, prose and gave suggestions for what help a non-technical audience may require in this paper.

References

[1] Deep Blue. http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/. Accesed: March 2017.

[2] Antonick, G. Google artificial intelligence beats expert at go game. https://wordplay.blogs.nytimes.com/2016/02/01/brilliant-go/?_r0. Accessed: March 2017.

[3] Bostrom, N. *Superintelligence: Paths, Dangers, Strategies*, 1st ed. Oxford University Press, Oxford, UK, 2014.

[4] Bradberry, J. Introduction to Monte Carlo Tree Search. https://jeffbradberry.com/posts/2015/09/intro-to-monte-carlo-tree-search/. Accessed: March 2017.

[5] Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games 4*, 1 (March 2012), 1–43.

[6] Chaslot, G. M. J.-B., Winands, M. H. M., van den Herik, H. J., Uiterwijk, J. W. H. M., and Bouzy, B. Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation 4* (2008), 343–357.

[7] Coulom, R. Efficient selectivity and backup operators in monte-carlo tree search. In *Proceedings of the 5th International Conference on Computers and Games* (Berlin, Heidelberg, 2007), CG'06, Springer-Verlag, pp. 72–83.

[8] Harding, L., B. L. From the archive, 12 may 1997: Deep blue win a giant step for computerkind. https://www.theguardian.com/theguardian/2011/may/12/deep-blue-beats-kasparov-1997. Accessed: March 2017.

[9] ICRAR. DALiuGE: Using the Logical Graph Editor, 2016.

[10] Kasparov, G. The Day I Sensed A New Intelligence. http://content.time.com/time/subscriber/article/0,33009,984305-1,00.html. Accessed: March 2017.

[11] Runarsson, T. P., Schoenauer, M., and Sebag, M. Pilot, rollout and monte carlo tree search methods for job shop scheduling. *CoRR abs/1210.0374* (2012).

[12] Russell, S. J., and Norvig, P. *Artificial Intelligence: A Modern Approach*, 2 ed. Pearson Education, 2003.

[13] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of Go with deep neural networks and tree search. *Nature 529*, 7587 (Jan. 2016), 484–489.

[14] Tarakajian, C. Solving Tic-Tac-Toe, Part II: A Better Way. `http://catarak.github.io/blog/2015/01/07/solving-tic-tac-toe/`. Accessed: March 2017.

[15] Ullman, J. D. Np-complete scheduling problems. *J. Comput. Syst. Sci. 10*, 3 (June 1975), 384–393.

[16] Wu, C., Tobar, R., Vinsen, K., Wicenec, A., Pallot, D., Lao, B., Wang, R., An, T., Boulton, M., Cooper, I., Dodson, R., Dolensky, M., Mei, Y., and Wang, F. DALiuGE: A Graph Execution Framework for Harnessing the Astronomical Data Deluge. *ArXiv e-prints* (Feb. 2017).