Project

Nikoli Hovorka - hovor009

December 2020

1 Abstract

This paper presents the comparison of algorithmic expansions within an intelligent agent for playing the pen-and-paper game 'Dots and Boxes', a small 2-player game where opponents take turns drawing a line connecting two dots in a grid of dots. Since there has been some research into the topic, there will be some use of prior knowledge to aid in the choice of approach, so as to not simply repeat documented results. Firstly, there is some universally known strategical information for playing the game that could be incorporated in a rule-based agent: the primary goal of each player should be to control a certain number of 'long chains', which are "sequences of capturable boxes" with a magnitude of 3 or more [1] in which a player forces their opponent to queue up the player to capture every box. For mathematical reasons too detailed to present in a proposal, the first player should create a start comprised entirely of an even amount of long chains while the second should aim for there to be an odd number of long chains - both win if their goal is realized. The experimentation would likely involve agents incorporating rules pertaining to this strategy to a varying degree.

The two main algorithms that are most applicable to this problem is Alpha-Beta MiniMax and MCTS, as they both are well-suited for applying to two-player games (MiniMax especially so). From previous work in *Intelligent Agents For Solving The Dots And Boxes Game*, it was found that the use Monte-Carlo Tree Search performed worse against other strategies (Alpha-Beta pruning, mainly) [4], likely because Monte-Carlo is computationally more extensive and suffers from the 'best-first' nature in comparison to MiniMax. However, an extension point was suggested in a research paper wherein the agent terminates the play-out early . The use of MCTS with this strategy will be considered as an implementation of the agent, in addition to an agent implementing only MCTS, which will serve as a control in essence to determine if the adaptation improves upon it's core to any degree. There are also more extension points for an MCTS agent defined in other research which applies similar heuristic knowledge of the domain to prune nodes in each branch that are unpruned dynamically as time is made available [4].

Due to the existence of extensive previous research into the problem of constructing an agent, experimentation will center around comparing several strate-

gies for MCTS. This includes the improvement of several potential extensions as well as their performance against each other.

The research of artificial intelligence in games is almost universally accepted as being of great interest to the field in general. Pen-and-paper games such as Tic-Tac-Toe and Dots-and-Boxes offer an environment complex enough to require academic research, yet simple enough to achieve attainable results; one cannot feasibly apply the algorithms discussed to the problem of building an A.I. opponent in three-dimensional first person shooter video games, or an agent for driving a car through traffic. A Dots-and-Boxes agent also presents a problem that has many different solutions, which makes the implementation of algorithm modifications a valid avenue for such a project.

2 Literature Review

2.1 Problem-related Research

This literary analysis synthesizes a collection of works pertaining to the problem of creating an agent for playing the scratch-and-paper game 'Dots and Boxes'. These primarily involve the research of the extension of related algorithms, the related game theory, or implementation of intelligent agents outright. The research specifically involving agents capable of playing 'Dots-and-Boxes' should be analyzed first, as it is from their results that experimentation can be extended into previously undiscovered territory.

The first article scrutinized with reference to the problem directly, written by E. Simon et al., discussed the application of Monte Carlo Tree Search (MCTS), Minimax with $\alpha\beta$-pruning, and the Q-learning algorithm to the game [8].The article also highlights an interesting general strategy for gaining an advantage over the other player: 'Double-dealing', where the agent sets up a game state where the opponent's best option is to complete a square and then create a line that enables the agent to respond with a 'chain', or a sequence of finishing a box and using their required additional line to complete another box, and so on [8]. Each algorithm was then applied to an agent and tested against an opponent using it's own algorithm as well as the remaining algorithms; the results were summarized as evidencing that $\alpha\beta$-pruning was the dominant strategy, with the 'Double-dealing' agent failing to perform against a simple rule-based AI that did not employ this strategy.

With MCTS under-performing in the previous articles results, an article providing an extension to MCTS in a Dots-and-Boxes agent, as well as different implementations of a board simulation for computational efficiency, is considered. This research provides an artificial neural network (ANN) to replace the random decision-making employed in traditional MCTS, to supposedly make more informed decisions [10]. An ANN is a system of algorithms simulating the nodes in a human brain, which sequentially learns from previous data and operates with the learned information. Several other optimizations are introduced, briefly summarized: a 'greedy' auxiliary function that ensures the agent always

takes a box of valence 0 (no lines drawn) when it exists, an adjusted selection algorithm within MCTS, and an auxiliary Minimax algorithm that is employed during the late stage of the game to reduce the degree of randomness in the selection algorithm, and several more to be elaborated upon in the project. Incorporating all these optimizations, the ANN-enhanced MCTS algorithm won almost exclusively against the unaltered MCTS. It may be of use to compare a similarly optimized MCTS against A-B pruning, as it may outperform A-B pruning where MCTS itself could not.

Expanding past MCTS and $\alpha\beta$-pruning, which had been the only algorithms expanded in the previously presented research pertaining to Dots-and-Boxes, the consideration of Lex Weaver and Terry Bossomaier's implementation of a neural network with a back-propagated genetic algorithm is imperative. A genetic algorithm mirrors the biological inspiration of ANN by simulating natural selection: states, like genes in a gene pool, are mutated and reproduced along a selective lineage that in theory produces the 'fittest' state. The actual selection process is actually the variable, with two different algorithms: one that pairs each member of the population against each other 'round-robin'-style, and an 'Antibody/Antigen' evolution that randomly determines one pairing for members [9]. When paired against human players, both algorithms succeed, which serves only to make evident the validity of both solutions.

A final Dots-and-Boxes-playing agent that introduced many core concepts for the underlying game theory was analyzed. This mathematical decomposition of the game was used to improve the efficiency of really any search applied to it; a game state that mirrors another across horizontal or vertical symmetry also mirrors it's optimal choice, and thus does not have to be calculated [1]. The well-known concept of 'chains', or a game state that allows the agent to capture 3 or more boxes in one turn, is also incorporated into the agent. Finally, transposition tables are used with some modification to further improve computation time, at the expense of space [1]. Transposition tables are in essence a data store of previous states and evaluations to be used in game-playing algorithms, which in theory improve computation time at the expense of space efficiency. The agent that uses all of the above attributes outperforms all those to which it was compared, including all copies of the agent that each left out one different modification [1].

2.2 Algorithm-related Research

Another article related to the improvement of MCTS was considered, applied to an agent playing Go. Aptly entitled "Progressive Strategies for Monte Carlo Tree Search", the article highlights two extensions to the traditional MCTS algorithm: *progressive bias*, which applies heuristic knowledge to the weight of nodes to choose, and *progressive unpruning*, which applies similar heuristic knowledge of the domain to prune nodes in each branch that are unpruned "progressively as more time becomes available" [4]. The heuristics are explained in the article with reference to an implementation of these concepts in an agent to play Go, however the general principle is applicable to the Dots and Boxes game: the

heuristic H_i is only computed once per node, after a certain threshold T of games have been played through this node [4]. This property ensures that the computation of heuristics is not a significant degree of time complexity to be inapplicable in consideration of more efficient algorithms. [4] runs an experiment comparing the performance of MCTS and an agent that uses MCTS with these two progressive strategies against GNU Go, another intelligent agent, with the results concluding that the progressive strategies greatly enhance MCTS performance in the game of Go [4].

A much more general article was selected that proposed the use of modular problem solving to decompose a complex game into an amalgam of smaller, trivial games. Written by J. K. Barker and R. E. Korf, it defines a w-function that assigned values to the positions of sub-games within composite games, i.e. games whose states can be broken up into various sub-games [6] that are solved to provide a general solution to the super-game. It also considered 'degeneration', or the simplification of a win-lose game by solving an optimal initial strategy; the Dots and Boxes game was referenced, with the initial strategy of being the first playing to finish a box [6]. The hypothesis of the degeneration defined in was that it would in general provide a 'good' strategy for complex 2-player games in linear time [6]. While fabricating a goal of creating the first box is revealed to only generally guide one towards a more optimal solution, it is worth considering in the implementation of rule-based algorithm, which in previous articles had failed to perform against the search algorithms [1].

As with MCTS, a paper exploring the modification of Minimax with $\alpha\beta$-pruning would be an appropriate addition to the reviewed literature. Yngvi Bjorsson and Tony Marsland, the article's authors, propose an heuristic that determines further which subtrees to prune beyond traditional Minimax: the number of 'good moves' available for the player upon node expansion [2]. Other heuristics are considered, including a 'history heuristic' and several others, all showing substantial improvement over traditional Minimax with $\alpha\beta$-pruning [2].

Another $\alpha\beta$-pruning article entitled "Best-First Minimax Search: First Results" highlights more extensions of Minimax specifically, with it's core focus being a 'best-first' Minimax algorithm that, instead of evaluating each branch, selectively decides the best node to expand; in essence, a best-first search of a minimax game tree. This is quite similar to the previous article in terms of a 'greedy' choice of ndoe expansion, however the heuristics are entirely different. In playing the game Othello with random game trees, the experiment's results conclude that best-first minimax outperforms $\alpha\beta$-pruning; however, the random nature of the game tree makes this finding unhelpful for comparison to previous algorithms. However, a hybrid best-first approach is then constructed that incorporates both $\alpha\beta$-pruning and best-first in the Minimax search, which outperforms simple best-first Minimax [5], and may be an interesting modification to a Dots-and-Boxes $\alpha\beta$-pruning agent.

In summation, the preliminary research of Artificial Intelligence and Dots-and-Boxes has been considered greatly, in several different avenues of approaches. However, the articles outside of the game-specific problem discussed show substantial opportunities for exploring new algorithm expansions.

3 Experiment

3.1 Methodology Description

Since the literature around constructing an agent to play Dots and Boxes is well established, and with the knowledge that $\alpha\beta$-pruning outperformed MCTS, the extension of both algorithms was the primary focus of the experiment. Wannes Meert of the KU Leuvin Department of Science has constructed a dots-and-boxes application that interfaces with two websocket servers as players or agents, published as a previous course assignment [3]. One can construct an agent in Java or Python3 that creates a server and acts as a player in the simulation. However, since the intricacies of runtime and space consumption are not able to be computed without additional logging modification, and the problem is primarily concerned with the algorithm that 'wins' the most games, this is not included in the experiment results. This decision was also made due to the already extensive research into runtime and storage comparisons from works cited, so as not to be redundant this is left out of consideration.

A solution to Meert's application is available publically, published by KU Leuvin graduate students Stijn Caerts and Wouter Baert, which implements 6 variations of MCTS: with their permission, their basic agent for MCTS was used, for both the control MCTS agent as well as the basis of agents employing MCTS extensions [**?**]. It should be noted that the construction of the $\alpha\beta$-pruning agent did not employ any algorithmic code, however the intricacies of implementing a connection to the KU 'dotsandboxes' application as well as a Node class was used [3]. Their code was vetted for error with no flaw found in their programming.

The first agents selected are simply Monte Carlo Tree Search and $\alpha\beta$-pruning, as they were in essence the control of previous literature related to the problem, in accordance with the result that $\alpha\beta$-pruning almost always wins against Monte Carlo Tree Search. Their basic functionality is provided in many of the related articles, so only a concise summary shall be given in the next section. As testing became rather time-intensive, only one grid dimension of 6x6 'boxes' was selected - this was to further differentiate from the realm of previous experimentation, as a larger environment can better showcase algorithmic behavior and trivially offers a more complex game tree. Each agent was tested against each other (but not themselves) 50 times for each player position; two agents are put in competition for 100 games, both going first an even number of times as the first player has a distinct advantage, showcased in [8].

3.2 Core Algorithm Description

The Monte Carlo Tree algorithm progressively simulates game playing throughout it's search process, with node exploration randomized somewhat according to the following selection formula:

$$max\ n{\in}children\ (\frac{win(n)}{times(n)} + \sqrt{\frac{2ln*times(parent(n))}{times(n)}}) \quad (3.2.1)$$

With n representing the current node selected for consideration, *win()* denoting the number of times the simulation at the node resulted in a win, *play* denoting the number of times the node has been played in total, and *parent()* denoting the nodes' parent node.

Minimax with $\alpha\beta$-pruning was to be a second core algorithm upon which extensions were made. However, implementation for an agent compatible with the board definition in Wannes' application [7] proved inapplicable.

3.3 Algorithm Extension - Progressive Bias

The first algorithm considered adopts the concepts of progressive unpruning and progressive bias from [4]. The latter works to redirect the search according to some heuristic knowledge, which will be defined in this problem shortly. It is factored in MCTS by altering the node selection algorithm:

$$max\ n{\in}children\ (\frac{win(n)}{times(n)} + \sqrt{\frac{2ln{*}times(parent(n))}{times(n)}} + f(n))\ [4]\ (3.3.1)$$

f(n) is defined as the heuristic function equating to:

$$\frac{H_n}{plays(n)+1}\ [4]\ (3.3.2),$$

with H_n denoting the heuristic value of node n, calculated by:

$$H_n = C_n + \sum_k \frac{1}{(2_{d_{k,n}})^{a_k}}\ (3.3.3),$$

Where C_n denotes the *capture value* calculated at node n - ostensibly, how many boxes would be captured by the move. $d_{k,n}$ is the euclidean distance from the kth last move, and $a_k = 1.25 + \frac{k}{2}$. This is modified somewhat from [4]; there was originally a value associated with a pattern-matching evaluation, however it is not considered as it it computationally quite expensive and somewhat outside of the scope of this course. a_k is a tuned factor that was found to improve performance in the implementation of the Go agent, weighing but was modified later on in the experiment to better perform in the Dots-and-Boxes environment.

The second concept, *progressive unpruning*, was removed from consideration for the algorithm as it required the use of prior knowledge and thus would consume a great deal more space, and since that cannot be determined from the agent application of [7], it would be indeterminable whether it's time complexity greatly outweighed it's performance against other, less space-prohibitive agents. A new agent Biased MCTS (BMCTS) was created, only implementing the bias heuristic.

3.4 Re-purposed Extension - 'Greedy' Alpha Beta

This algorithmic expansion utilizes the central heuristic of [2], which enumerates the moves available for a given node. The theory behind this is rather trivial; the more opportunities for node expansion could generally navigate towards

optimality. However, the results of [2] showed considerable improvement over traditional $\alpha\beta$-pruning with Minimax, so it was deemed interesting as an extension, and was implemented within the minimizing and maximizing conditions of the *alphabeta* algorithm:

$$value = (min/max)(value + movesLeft, alphabeta(board, depth - 1, a, b, false)), \quad (3.4.1)$$

Where the value is essentially equivalent to the size of it's subsequent moves to be made, with much less consideration given to the win or loss state as in the core algorithmic agent. As the core algorithm could not be implemented, neither could this extension. However, this heuristic is also applicable within MCTS, specifically as a metric for *progressive bias*. This became another agent, Move-based Biased MCTS (MBMCTS), implemented identically to H_n in Figure 3.3.3, however C_n is the following:

$$C_n = totalBoardMoves(), (3.4.1)$$

denoting the amount of available moves within the state.

3.5 Experiment Overview

In summation, the agents used in the experiment are: MCTS, BMCTS, and MBMCTS, with their formal definitions provided previously.

4 Results and Analysis

4.1 MCTS vs. BMCTS

First Player	MCTS Win	BMCTS Win
MCTS	38%	62%
BMCTS	24%	76%

4.2 MCTS vs. MBMCTS

First Player	MCTS Win	MBMCTS Win
MCTS	42%	58%
MBMCTS	2%	98%

4.3 Analysis

The results garnered from pairing MCTS against BMCTS are not entirely absolute, yet still encouraging to the degree that it seems to show the slight improvement of utilizing progressive bias, generally winning in both player 1 and 2 positions. Other values for a_k were considered, with no tangible improvement, however further testing could result in a more optimally-tuned extension.

In regards to the experimental simulation of MCTS and MBMCTS in competition, the simulation results are more muddled in nature with a playout closer to even, however with a slught margin of victory for MBMCTS

5 Conclusion

In the realm of points of improvement or consideration, an obvious element is the inclusion of Minimax with $\alpha\beta$-pruning against the extension agents. Although it would require too great an effort to implement within the constraints of the project for which this report is written, a two-agent simulation constructed solely for the project would be an ideal environment in which to run comparison simulations.

The use of Chains, as defined within [8], would also provide a distinct agent for comparison against BMCTS and MBMCTS. They were not used as to differentiate the current experimentation from that of previous works, however their basis in the mathematical underlying of each state would provide another useful control, in the sense of previous research, as comparison to other avenues of heuristic modification to the problem space.

In summary and in summation, the product of several other works would be included within the results of this project; especially the computational interest of symmetrical game states, and those which were included in the literature synthesis but not implemented in the experimentation phase. Future work to expand upon these results would likely implement these aforementioned works.

References

[1] J. K. Barker and R. E. Korf. Solving dots-and-boxes. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, AAAI'12, page 414–419. AAAI Press, 2012.

[2] Y. Björnsson and T. A. Marsland. Multi-cut -pruning in game-tree search. *Theoretical Computer Science*, 252(1):177 – 196, 2001. CG'98.

[3] S. Caerts and W. Baert. Ku leuvin machine learning project: Dots and boxes a.i., Oct 2018.

[4] G. Chaslot, M. Winands, H. Herik, J. Uiterwijk, and B. Bouzy. Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation*, 04:343–357, 11 2008.

[5] R. E. Korf and D. M. Chickering. Best-first minimax search. *ICGA Journal*, 19(3):187–187, 1996.

[6] R. Lenhardt. Composite mathematical games. 2007.

[7] W. Meert. wannesm/dotsandboxes, Sep 2018.

[8] E. Simon. Intelligent agents for solving the game dots and boxes. 01 2020.

[9] L. Weaver and T. Bossomeier. Evolution of neural networks to play the game of dots-and-boxes. *Artificial Life V (1996)*, May 1996.

[10] Y. Zhuang, S. Li, T. V. Peters, and C. Zhang. Improving monte-carlo tree search for dots-and-boxes with a novel board representation and artificial neural networks. In *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 314–321, 2015.