Cmput 670 Project Report - A Dots and Boxes Player

Michele Albach

Apr 9th 2019

Introduction

Dots and boxes is a 2-player game in which the goal is to collect the most boxes. The game starts with grid of unconnected dots (Figure 1), and players take turns by connecting two dots with a line. Whenever a player draws the 4th line surrounding a specific box, they capture that box, and then must take another turn (Figure 2). The game continues until there are no more available moves, and then the player with the most boxes wins. Dots and boxes can usually be divided into two parts: while there are still remaining safe moves (that do not add a third line around a box and open it up for capture), and after there are no more safe moves. In the second part, the game can be described as multiple chains and loops, which each have the property that they can be entirely captured in one series of moves by a player.

I decided to create a player for the game dots and boxes because the game holds a special place in my heart. When I was a child, my family would often go out for dinner at restaurants and, while waiting for the food to arrive, one of our go-to activities was this game. Once we had decided to play, I would grab a pen and a napkin and draw the largest grid of dots that I could possibly fit, and all 4 of us would each take turns picking lines to draw. To me, the activity was more of a look-and-find game than a strategy game, as I would scour the page for any remaining safe moves or small chains. However, upon researching the game for this project, I discovered new strategies that neither I or my family had ever been aware of while playing at restaurant tables.

For this project, I aimed to create a working player for a 5x5 dots and boxes board that was good enough to beat myself and my family. I wanted to create a player using only strategy, as well as a searching player (for smaller board sizes). I was able to complete a working strategy player that consistently beats my friends and family, and beats me about half of the time. I created a minimax searching player with alpha-beta pruning, but it is too slow for any board size larger than 2x2. In this report, I will describe the steps that I took to create my players, followed by the results that I obtained in testing them, and lastly, I will outline the improvements that could be made to my players with more time.

Making the Players

To start, I needed to create a system for keeping track of the board state. I initially decided to label each box as a digit from left to right and top to bottom. Some may find it more intuitive to label the dots than the boxes, but to me the boxes are the most important information in the game and I wanted them to have specific labels. I later switched from digits to lower-case letters as I realized that I could not make a board larger than 3x3 (9 boxes) without running out of digits (I realize that using letters does not completely solve this problem since now I cannot make a board larger than 5x5, this is a flaw that I will address in the improvements section). To label the lines, I used the two boxes that a line is separating, and cardinal directions (upper-case) for the edges (Figure 3). This system allowed me to easily access a box's neighbours by finding the lines surrounding it. To keep track of the board in python, I used dictionaries for both the boxes and lines with their labels as the keys and the owner of the box (or ‘empty’) and the state of the line (0 or 1) as the items respectively. Once I had the board state saved, I created a basic, non-strategic player, at a similar level to myself as a child. This player will avoid unsafe moves, capture boxes when available, and otherwise play randomly.

Next, I started to work on my strategic player by incorporating chain and loop analyses. To do this, I created a function that returns a list of the chains and loops in a given board state. This function, called *returnChains*, iterates through the boxes in the board to find what chain/loop they are a part of. If the box is part of a chain/loop already found, it is skipped, else the function finds all of the box’s neighbours that are not disconnected by a line. If the box has exactly two neighbours, then a chain is created and the function iteratively finds the string of neighbours on each side until a box with more or less than two neighbours is found (or the edge is reached) to end the chain. Or, if a box already in the chain is seen again, the chain is declared a loop and ended. *returnChains* does this for every box in the board and returns a list of the chains and loops. Using this tool, my player could now look for the shortest chain possible to open up once no safe moves are left. Additionally, I could now incorporate double-crossing.

Double-crossing is a strategy in which a player sacrifices two or four boxes in order to not open up a new chain to their opponent. Double-crossing is always an option when capturing a chain or loop of length three or more, and is sometimes an option in a chain of length two (if your opponent makes a mistake). When capturing the boxes in a chain, the idea is to capture up until two boxes remain, and then leave those boxes and instead close the other end of the chain. Since no new box was captured, this ends a player’s turn and does not force them to play somewhere else potentially opening another long chain (whereas if they took the boxes, they would need to play again). In a loop, this can be done by capturing boxes until four remain, and then splitting the 4 boxes into two closed chains of two. As long as there are only loops and chains of length at least 3 remaining on the board, this strategy can be used repeatedly on every opened chain until the last (and likely the longest) chain is opened, which can all be captured. An example of double-crossing is given in Figure 4.

To incorporate double-crossing, my player makes a decision whether or not to use the strategy every time that it is faced with a chain with two boxes that is closed on one side and open on the other. First, the player checks if there are other boxes available to be captured, if so those boxes are taken first. Next, the player checks if these two boxes are the only remaining unclaimed boxes in the game, if so there is no reason to double-cross. Lastly, the player checks if there are other remaining unopened chains that are possibly too short to be double-crossed, if so, it only double-crosses if there is an even number of short chains. This step is because the player doesn’t want to be given the last too short chain, since then their opponent could adopt a double-crossing strategy. When given a loop, the player will only double-cross if it can gain more than four boxes by doing so, since a greater sacrifice is needed.

After incorporating double-crossing, the only other strategies that I encoded were opening chains of two and loop-avoidance. When required to open up a chain of length two, my player will always draw the line in between the two boxes (Figure 5). This is because if the chain is opened on an end then the opponent could double-cross. When choosing safe moves in the first part of the game, my player will choose lines inside of created corners (Figure 6) in an attempt to avoid small loops. I chose to implement this because loops require a greater sacrifice to double-cross and so can be annoying.

Lastly, I attempted to create a searching player using minimax with alpha-beta pruning. This part of the project was the most difficult for me as I am not very experienced with optimizing searchers. I created a recursive function *search* which calls itself for every possible move by either player until an end-of-game state is reached. At each point, it saves the move by the player that results in the highest final score. I added alpha-beta pruning by stopping any search into the opponent's possible moves if a score lower than the current highest was found. Additionally, I pruned out isomorphic game states and unsafe moves (while safe moves exist). It could be that the best possible move is not a safe move, but in an attempt to speed up the searcher I left them out. However, after pruning my searcher is still too slow for a 3x3 game board. The longest that I let it run was a couple hours, but it did not terminate. On a 2x2 board, my searcher is fast enough to compete.

Results of Players

Throughout the process of creating my players I frequently tested them against myself and friends and family. Many of the strategies that I encorporated were inspired by those tests. But once I had determined that I was satisfied with my players (for the scope of this project) I performed some additional final tests. Since my goal was to make a player on a 5x5 board, most of the testing I did for my strategy was on this size. In all the tests that I was able to complete against friends and family, my strategy player only lost once. Not all of my friends and family are familiar with double-crossing, but I tried to give them a chance to learn the strategy before the final tests. Against myself, the player wins about half of the time. However, I don't think that I am the best opponent since most of the time I know exactly what my player will do. In all tests I switched between having my player go first and second. Since none of my friends and families are super familiar with dots and boxes, I think that it would be interesting to test my player against someone who is (but also isn't me). As for my searching player, I was only able to test it on a 2x2 board. Against me, it wins most of the time. Against the strategy player, I found that it either wins or ties about half-and-half, but there was one game in which the strategy player won 3-1.

Improvements and Conclusion

There are many possible improvements that could be made to my players. First of all, my system for the board state does not allow a board size larger than 5x5. It would be a lot of work to change this system as I rely on it a lot to determine chains and loops. Also, the current set-up only allows square boards. To allow rectangular boards, the setup of the dictionaries would need to be altered. This could likely be done somewhat easily, I just did not think that it was a priority. Another thing that I didn't prioritize is user-interface. At the moment, all my code runs in the terminal and someone must become familiar with my system to use it. Someone with experience in graphics could easily adapt my program to allow for more traditional simple play. In terms of strategy, I did not have time to better use the total number of chains in a board. Albert, Nowakowski, and Wolfe describe an algorithm that determines the prefered number of long chains to a player, but I struggled to think about how my player could try to affect the final number of long chains. Lastly, my program treats all safe moves the same and chooses randomly over them, but choosing a better safe move could affect the number of chains or who will be offered the first long chain. My strategy could be greatly increased with an anlyses of the best safe moves. For my searching player, many obvious improvements could be made. I am sure that an expert on optimization would have many suggestions to allow my player to terminate on larger board sizes.

In conclusion, I created a decent strategy based dots and boxes player for up to a 5x5 square board. I am satisfied with my results, and all in all I think that I provided a fun competitor for both myself and my friends and family. I created a searcher for very small boards, but failed to optimize it for larger boards. I thoroughly enjoyed this project and learned a lot while also reliving my childhood. This report, the list of figures, and all of my code is available on GitHub at https://github.com/michelealbach/Dots-and-Boxes.

Bibliography

Albert, M., Nowakowski, R. and Wolfe, D. (2007). *Lessons in Play*. AK Peters.
Allcock, D. (2018). *Best play in Dots and Boxes endgames*. [online] arXiv.org. Available at: https://arxiv.org/abs/1811.10747 [Accessed 13 Feb. 2019].
Barker, J. and Korf, R. (2012). Solving Dots-And-Boxes. *Association for the Advancement of Artificial Intelligence*, pp.414-419.
Berlekamp, E. (2001). The dots-and-boxes game: sophisticated child's play. *Choice Reviews Online*, 38(08), pp.38-4503-38-4503.
Cao, Y., Lu, J. and Yin, H. (2018). Endgame database for Dots-and-Boxes game. *2018 Chinese Control And Decision Conference (CCDC)*.
Li, S., Li, D. and Yuan, X. (2012). Research and Implementation of Dots-and-Boxes Game System. *Journal of Software*, 7(2), pp.256-262.
Manners, O. (n.d.). *Dots And Boxes*. [online] Dotsandboxes.org. Available at: http://dotsandboxes.org/ [Accessed 14 Feb. 2019].
Prince, Jared, "Game Specific Approaches to Monte Carlo Tree Search for Dots and Boxes" (2017). Honors College Capstone Experience/Thesis Projects. Paper 701.
En.wikipedia.org. (2019). *Dots and Boxes*. [online] Available at: https://en.wikipedia.org/wiki/Dots_and_Boxes [Accessed 14 Feb. 2019].
Wilson, D. (2019). *Dots-and-Boxes Analysis Index*. [online] Wilson.engr.wisc.edu. Available at: http://wilson.engr.wisc.edu/boxes/ [Accessed 13 Feb. 2019].