Alex Zwart
Evan Fannin
Jace Locke
CSC 3400

An Assessment of Search Tree Heuristics for Optimizing Performance on the Board Game Fox and Hounds

Abstract:

The problem is one of the general family of full information abstract strategy games and is specifically of a class of games known as “hunting games,” where the two sides are asymmetrical in size and have different winning goals, one usually to trap the other in some way. The solution to this problem will use domain specific adversarial search methods for general full information games. As such, the optimal algorithms and heuristics will be related to many real world game theoretic situations where there are two sides that have full information about each other. The contributions in this paper will be novel in that they focus on a situation where the adversaries are pursuing different goals and have different valid moves. In order to develop search methods, we intend to use the Minimax and Alpha-Beta searches. To optimize these, we will reduce the size of the search tree by pruning bad moves. In doing so we hope to come up with an optimal heuristic for selecting moves. Given that each player pursues a different goal, each will need a separate heuristic.

Background:

Fox and Hounds is a two-player board game that takes place on an 8x8 checkered board. One player controls a single fox while the other controls 4 hounds. The hounds begin the game lining the top edge of the board on the black spaces only. The fox begins the game placed in the bottom left corner of the board on the black space. In order to win the game, the fox must make its way to one of the initial starting places of a hound. The hounds win by “trapping” the fox before it makes it there.

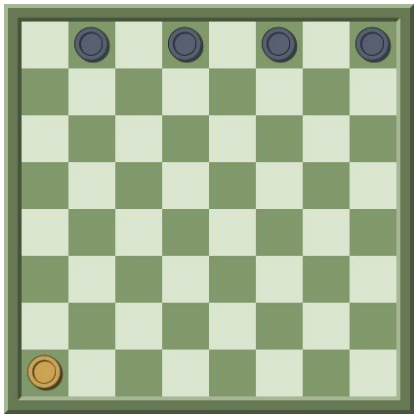
Figure 1: The initial board state

On each turn, the fox has up to 4 possible moves; diagonal in any direction. The hounds only have up to 2 moves per turn; diagonally down right or left. There are no capture, elimination, or hopping mechanics. Pieces are not allowed to move to a space where there is already a piece. The game is over once the fox has a turn with no move options, or once the fox is in one of the initial hound positions.

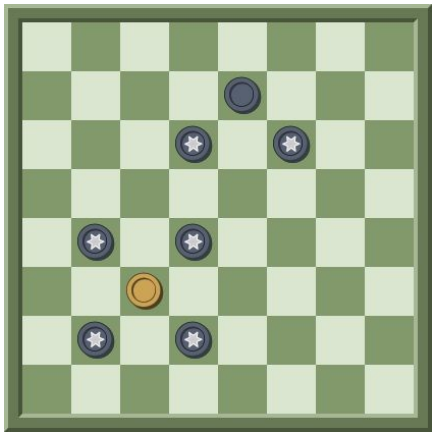
Figure 2: Possible moves for the fox and hounds represented by stars. Pieces cannot move to occupied positions.

What we did:

At the beginning of a game, an empty board needs to be initialized. The type of the state is a tuple of strings. Each string represents a row while each string index represents a column. Hounds are then represented by ‘h’ and the fox is represented by an ‘f’. We chose to use periods as open spaces. The first element of the first string is either a 1 or a 0 depending on whose turn it is (fox is 1, hounds are 0). It is important that the 4 hounds and the fox are placed on the same horizontal planes (either all on the spaces that would be red, or all on the spaces that would be black).

In order to implement this game, there were a few things that needed to be taken into account. First of all, how do the players win? In this particular game, each player has a different objective. This means defining a different winning state for each of the players. For the player controlling the fox, it's simple. If the fox is in one of the initial spaces of a hound, it wins. All we need to do is check the location of the fox and we would know if it wins. For the hounds, it is a bit different. This check requires determining if the fox has any available moves. If the fox can't move and it is not in one of the hound's initial positions, the hounds win.

Next to consider are the possible moves of each player at any given time. Although moves are similar between the fox and the hounds, there are slight differences. To know a piece's possible moves, it is important to first know the position of the pieces. For the fox, that means iterating through all of the points on the board until one of them is an 'f'. This is done by a call to a get_fox function. It will return a set of coordinates where the row is defined by the index of the string in state, and the column is defined by the index of the character within that string. From there the get_fox_moves function checks the diagonal spaces on the board to see if they are open. It is important to check to make sure that each of the checks is within the bounds of the board. For each one that is open, an action is returned. The format of the action is a list of tuples each containing the coordinates of the open position.

For the hounds it gets a little bit more complicated. First, a get_hounds function is run in order to gather where each of the hounds is located. It does so by iterating through all spaces on the board and checking to see if it contains an 'h'. Once it has found 4 'h's, it knows where they all are. A list of tuples containing the appropriate coordinates is returned to the get_hounds_moves function. It will then iterate through each of the hound locations and find open downward diagonal spaces. We then append a tuple containing the coordinates of the hound followed by all possible spaces to move to into the moves list.

Once the moves have been collected, a result function is called. Based on a selected search algorithm, a move will be chosen. From that move, the location of the player's piece will be replaced with a '.' while the 'move-to' space will be replaced with the character of the player's piece. The search algorithms will be responsible for deciding the best moves to get from the current state to the goal state.

In order to select the most optimum move and in an effort to prune the alpha-beta search tree, we experimented with implementing several heuristics for the fox and the hounds. These were focused on the fox's distance from its goal locations and on the

fox's degrees of freedom: how many possible moves it has, remembering that zero moves means that the hounds win.

For the distance heuristic we tried two variations. These were the distance that the fox is from its goal as a minimum of the distances to the four different goals. The other variation was the sum of the distances from each hound to the fox. In measuring the fox's distance from the goal, we hoped to incentivize the fox to move closer to the goal, while making the hounds keep the fox down-board. In keeping track of the sum of the hounds' distance from the fox, we wanted to get the hounds to all close in on the fox.

In creating the degrees of freedom heuristic, our intention was to have the hounds be able to move towards states where the search algorithm would be able to detect winning moves within its horizon. This also keeps the fox from being able to freely move around the board. Conversely, the fox using this as a weight in its search tree keeps it from moving into dangerous situations that it can't get out of.

Results:

The fox is really clever! Even for human players, maneuvering the hounds into a trap is a much more difficult process than simply evading the hounds. The fox tends to win against the hounds unless placed in constrained positions where the hounds are certain to win in a few ply.

Knowing this, we focused on improving the hounds' performance as a metric of overall success. We found that the best combination of heuristics for the hounds was one measuring the fox's distance from the goal combined with one counting the fox's degrees of freedom. The performance was best when the degrees of freedom were weighted heavier than the distance metric, which makes sense considering the priority is to trap the fox, not just keep it away.

Future Work:

While there may be mechanical methods of improving AI game performance such as representations that decrease computation time and machine setups that increase processing power, in our future work we would primarily like to look at algorithmic improvements to optimize the alpha-beta search.

Because this game is on a grid, algorithms based in graph theory hold the most promise. One worth looking at involves finding every path that the fox has to the goal,

similar to the process in Dijkstra's algorithm. This would particularly benefit the hounds in their enclosure of the fox, as the biggest challenge that they face is making a hole in their formation and letting the fox through as a result. By weighting the fox's number of paths negatively in the hounds' alpha-beta search, they would be incentivized to make moves that tend to block the fox's progression up the board.

Works Cited

Lim, Darren. *Taking Students Out for a Ride: Using a Board Game to Teach Graph Theory*. Mar. 2007, http://www.cs.xu.edu/csci390/13s/p367-lim.pdf
This paper serves as a way to teach graph theory with the use of a board game. Providing a useful example and scenario how to learn the basic principles and design better algorithms/programs. This relates to our project since the board game state is graphical and the movement is as well. Overall giving us more information regarding graph theory to apply.

August, Stephanie, and Matthew Shields. *Adversarial Search: Nine Men's Morris, Minimax, and Alpha Beta Pruning Search Algorithms*. Oct. 2015, seaugust.lmu.build/TAILSweb/html/Nine-Mens-Morris-master/program/files/module.pdf.
The authors utilizes an example strategy board game called Nine Men's Morris to model the adversarial search: minimax. This lab experiment is useful by giving us an example to compare our game too in multiple ways. One being Alpha-Beta pruning.

Herik, H.Jaap van den, et al. "Games Solved: Now and in the Future." *Artificial Intelligence*, Elsevier, 12 Sept. 2001, www.sciencedirect.com/science/article/pii/S0004370201001527.
This paper focuses on how two players' games are solved, with what characteristics should be kept in mind when the solution is the main target. As Fox and Hound is a zero-sum game, the game ending and results are important.

Kang, Xiyu, et al. "Research on Different Heuristics for Minimax Algorithm Insight from Connect-4 Game." *Journal of Intelligent Learning Systems and Applications*, Scientific Research Publishing, 7 Mar. 2019, https://www.scirp.org/journal/paperinformation.aspx?paperid=90972
This journal shares the insight on different heuristics gained from another example game similar to ours. Doing so by conducting three experiments on important factors that affected the heuristic. Helps our project in regards to game optimization and possible inefficiency.

Khalil, Elias B., et al. “Learning to Run Heuristics in Tree Search.” *Learning to Run Heuristics in Tree Search | Proceedings of the 26th International Joint Conference on Artificial Intelligence*, 1 Aug. 2017, https://dl.acm.org/doi/10.5555/3171642.3171737
The authors focus on improving and overall optimizing the performance by running the heuristic dynamically. Specifically picking what node in the search tree to run it.

Mathew, Geethu. “Direction Based Heuristic For Pathfinding In Video Games.” *ScienceDirect*, www.sciencedirect.com/science/article/pii/S1877050915004743/pdf?md5=77fe483c4ef5a28aff69d24ca658042e&pid=1-s2.0-S1877050915004743-main.pdf.
The authors of this document mainly discussed the concept of pathfinding in graph theory. Since our project is based on a zero sum, two player game(A.I can be used), pathfinding is less significant compared to achieving the goal. But being able to find an optimal path on the graph to avoid obstacles(the Hounds), could be quite useful.

Schiffel, Stephan. “Automatic Construction of a Heuristic Search Function for General Game Playing.” *ResearchGate*, Jan. 2006, www.researchgate.net/publication/228697273_Automatic_construction_of_a_heuristic_search_function_for_general_game_playing.
This article gives a general overview of what our game functionalities and techniques for constructing a heuristic search. Key information to keep in mind while designing the game.

Wittpohl, Milan. *Adversarial Search Methods In Two-Player Games*. Dec. 2019, https://milanwittpohl.com/projects/adversarial/Adversarial-Search-Methods-Report.pdf
The author covers the adversarial search methods that are used in most two player games. But reminds us that the performance of said search methods can suffer depending on the depth. An example and reminder to help keep our game in adequate shape.