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. To develop search methods, we have used the Minimax and Alpha-Beta searches. To optimize these, we have constructed heuristic algorithms to reduce the size of the search tree by pruning bad moves. We have done this in two iterations. The first set of algorithms dealt with the fox and hounds’ positions relative to their goals. The performance of this first generation was far exceeded by our second iteration of heuristics, which used graph theoretical properties of the board and game state. This AI then combines adversarial search with path finding algorithms for optimal performance.

Background:

Fox and Hounds is a two-player board game that takes place on the black spaces of an eight by eight checkered board. One player controls a single fox while the other controls four hounds. The hounds begin the game lining the top row of the board, while the fox begins the game placed in the bottom left corner of the board. 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 so that it has no admissible moves.

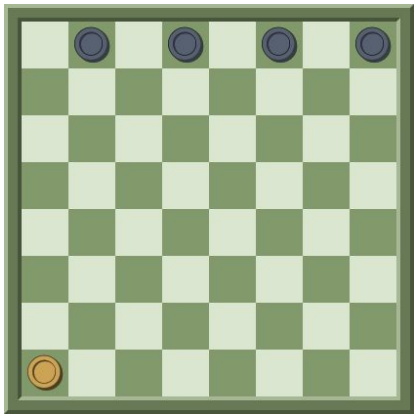
Figure 1: The initial board state

On each turn, the fox has up to four possible moves, diagonal in any direction. The hounds only have up to two 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.

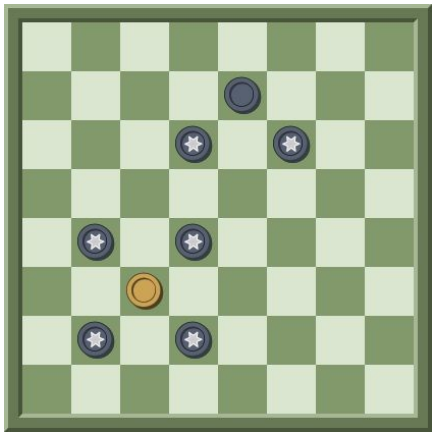
Figure 2: Possible moves for the fox (yellow) and hounds(black) marked by stars

What we did:

We set out to make an AI that can play the game against an average player. Our work fell into three parts: we had to build a representation of the game, wrap the basic methods required by an Alpha Beta search around that representation, and then develop heuristics that optimally pruned the search tree.

Part 1: Building a Representation of the Game

First, we discuss the board representation. This is done by entering into the program a tuple of strings, with each string representing a row of the board. Hounds are demarcated by ‘h’s, the fox by an ‘f’, and blank spaces by ‘.’.

When gameplay begins, the program must know whose turn it is. Because each adversary is looking through successive plies of a search tree, this turn state changes quite often, so it needs to be hard coded into the board state. This was done by keeping either a “1” or a “0” as the first element of the first string in the game state. This was achievable because the players never move onto that space; keep in mind that they can only move on black spaces.

In order to be manipulated by each player, the game state is then converted into a two-dimensional array containing the same entries in the same locations. Each player can then move according to rules coded into the program, the same rules described above.

Finally, a terminal test was created to determine when the game is over. This again follows the rules described above; it checks whether the fox is in one of the hounds’ original positions or if it has any free moves.

Part 2: Setting up the Alpha Beta Search

The functions loosely described above already form the basis for implementing a game-specific Alpha Beta search program using a general one. We have a function that determines the possible actions available to each player based on the positions of the pieces on the board. We have a function that generates the result of an action by converting the board from a string to an array and then back, moving the selected piece in between, and knowing which player is moving by reading the aforementioned ‘1’ or ‘0’. We also have a terminal test that checks for the conclusion of the game.

All that’s remaining is to implement a heuristic function that the fox or hounds, depending on whose turn it is, can use to judge possible moves in the search tree. This judging is done by assigning each move a certain amount of points based on a heuristic and then pruning the search tree based on maximizing those points for one’s self and minimizing them for one’s opponent.

Part 3: Developing optimal heuristics

First Iteration of Heuristics:

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 on top of a basic winning heuristic, which just uses the terminal test to check whether a player can achieve a win within the horizon of the search tree.

The first time through we created heuristics that we thought a human player might implicitly use. There were two: a distance-from-goal heuristic and a degrees-of-freedom heuristic. Both were used by both the fox and hounds, but were positive for one and negative for the other.

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.

Second Iteration of Heuristics:

The behavior of our first iteration was suboptimal. This will be discussed in the results section. We observed that the AIs were missing an important strategy that a human would use when looking at the board: looking out for paths to the goal and their lengths. So we decided to explore the implementation of a heuristic exploiting the graph-like nature of the board to search for paths between nodes and their respective lengths. A function with this knowledge could count paths to the goals for the fox and take the minimum length one. This could be used as a positive factor in the fox's search and a negative one in the hounds'. When a path to the goals isn't present because the hounds are temporarily blocking the way, the algorithm could then default back to our first generation of heuristics.

To begin, we had to add yet another function to transform the representation of our game in order to implement Dijkstra's Algorithm, which we chose to find shortest paths. This is because any graph algorithm needs a graph, so we had to transform each board cell into a node with data pointing to its neighbors: the moves reachable from that position.

We were able to find a library called Networkx, that handled most of the heavy graph-manipulation and path-finding functions. We then only had to focus on creating a graph that kept the rules of the game intact.

To do so, we first created a graph by explicitly linking each black tile of the 8x8 board to the appropriate diagonal tiles. Thus the graph had 32 vertices and each vertex had between 1 and 4 connections. This allowed us to only consider the black tiles as legal moves for both players.

To persist the rule that the fox can't move to tiles occupied by the hounds, we found the location of each hound and removed that corresponding vertex from the graph along with its edges, thus leaving a graph representing the current game state of cells open to the fox.

Our heuristic then finds the fox's location and searches for paths from there to each of the goals. If there aren't paths, it defaults to the previous heuristics. If there are paths, it chooses the minimum distance among them. At this point, a weight can be assigned to a possible move in the search tree by subtracting the path length from a high valued integer, thus giving more points to a shorter path versus a longer one examined in another possible state.

Results:

The first iteration of heuristics:

The fox was simply too clever. The fox tended to win against the hounds unless placed in constrained positions where the hounds are certain to win in a few ply by viewing a terminal state within their search tree's horizon. This was because the heuristics applied were sufficient for the fox to naively reach its goal, but not enough for the hounds to perform the more complicated process of coordinating to trap the fox. We knew that this result was suboptimal because, in perfect play, the hounds should win this game every time. That is, this is a solved game according to game theory.

We attempted to adjust the weights of the various heuristics. We found that the best combination of heuristics for the hounds (using the first generation) 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. However, we had hopes that we could improve performance further with a new heuristic.

The second iteration of heuristics:

This time around results were much more in line with what would be expected, that is, in perfect play on both sides, the hounds win every time. This heuristic indeed seemed to be optimal under all conditions that we tested it in, which can be found in our accompanying repository's log file. The only thing that it lacked was optimal play under the rare condition that the path-finding heuristic did not apply because there was no path to the goal for the fox due to the hounds being in the way. In this case both sides had to default back to the original sub-par heuristics to weight potential moves.

Aside from continued suboptimality in a handful of possible states, the other issue with that we found with this path heuristic is that it takes a long time to run. It selects good moves, but is not time efficient. In every newly examined move in every ply of each search tree, the game board must be converted to an appropriate graph and Dijkstra's algorithm must run. This adds up to a significant amount of computation that causes a delay of up to five seconds, depending on the complexity of the current positioning on the board. That is, if the fox is in the middle of the board with plenty of path possibilities, more must be examined then if it has few options available.

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, as well as optimizations inherent to the game to improve computation time and make the game playable on larger boards with more hounds.

Graph theoretic algorithms continue to hold promise for possible improvements to AI gameplay. The problem of finding optimal moves for the fox and hounds while no paths from the fox's position to the goal exist in the gamestate is an outstanding one.

With an eye to computation, we would like to look into ways to reduce the amount of time taken to perform a sweep through the adversarial search tree. A possible algorithmic improvement lies in the use of A* search over Dijkstra's Algorithm based on Breadth First Search when searching for paths and their respective lengths between the fox and its goals. Through further research, we found that A* could be a superior algorithm to implement instead of Dijkstra's algorithm because it finds the shortest path more efficiently. While A* requires some overheard with space complexity, our board and any likely expanded board is small. Furthermore, the information of the board is fully available, allowing A* to develop a guess of the fox's distance from the goal.

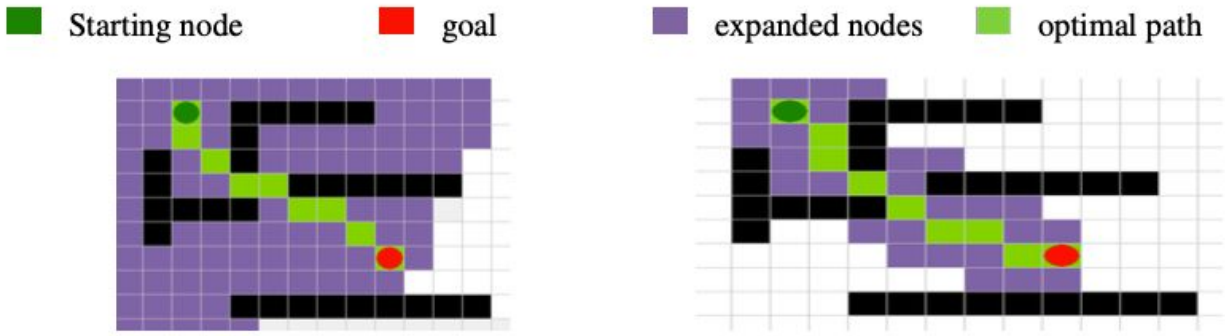

Figure 3: A diagram showing the nodes expanded by breadth first search (left) and A* (right) - Source [3]

Yet another avenue of future work would be the optimizable of inherent game representation mechanics, both to improve computation time and to allow the board to be scaled up for a larger game with more hounds. It would be ideal if a way could be found to keep a constructed graph representation of the board instead of creating one with every new state expansion in the search tree. This permanent representation could then be updated with the exploration of each move possibility.

The game also is currently hard coded to create a graph based on an 8x8 game board. To play on boards of larger size with more hounds, an algorithmic way of creating the board will need to be thought of. Among other things, one would have to calculate the number of hounds to be placed in the top row and incorporate that into the game state. Functions will also be needed to create a graph of only black spaces on a board of any size, connecting those black spaces to each other appropriately.

Works Cited

[1] 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.**

[2] Cui, Xiao, et al. "Direction Oriented Pathfinding in Video Games." *SemanticScholar* Oct. 2011,pdfs.semanticscholar.org/1e92/858a275c37c0e2b94c3ad280014bf810f12e.pdf**. This paper addresses the significant performance issue that comes with utilizing search algorithms such as Dijkstra's algorithm and A* algorithm. Also, when a visible graph or map representation is included. This is relevant to our project since we are currently using both and we are beginning to see the issues occur.**

[3] Cui, Xiao, et al. "(PDF) A*-Based Pathfinding in Modern Computer Games." *ResearchGate*, Nov. 2010, www.researchgate.net/publication/267809499_A-based_Pathfinding_in_Modern_Computer_Games.
This document verges deeper into using A* algorithm for pathfinding and provide various techniques and algorithms based off of the popular search algorithm. Also real game examples that explain how to use it in what ways. A very useful addition to further our understanding for the future!

[4] Hart, Peter E. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths." *AI.Stanford*, 1967, ai.stanford.edu/~nilsson/OnlinePubs-Nils/PublishedPapers/astar.pdf. **This paper acknowledges the issue of minimum cost paths occurring and provides heuristic information incorporated into formal mathematical theory. It really helps the overall project by providing useful examples and proofs on search strategies.**

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.