An Artificial Intelligence Approach to Das Bohnenspiel

Saleh, Zeyad
zeyad.saleh@mail.mcgill.ca
260556530 - ECSE 456

Submitted to:

McGill Comp 424 McGill

Abstract

This project is focused on the development of an AI agent that would play a variation of the game Das Bohnenspiel "the bean game" which is part of the Mancala family. This paper discusses the implementation of such an agent using Min-Max search algorithm and alpha-beta pruning. More importantly it focuses on the choice of heuristics and techniques to dominate an average agent.

Motivation

Searching and game playing have long been the main topics in AI. The main idea is to examine the problems that arise when trying to plan ahead in an environment where other agents are planning against us and to explore the different strategies and techniques that can overcome these problems. The two most common techniques are searching and learning. Search algorithms look ahead in the future by expanding on every legal move in the game, resulting in a tree that can be searched for the best score depending on a certain evaluation method and/or heuristic function. As for Learning techniques, they can be thought of as the process of finding weighted functions that reason from externally supplied inputs to produce general hypotheses which then can make predictions about future events. There are 2 major learning techniques: Supervised where the inputs are labelled, and unsupervised where the inputs aren't.

Since my first encounter with Das Bohnenspiel was at the start of this project, using learning techniques was hard since I did not have the expertise to provide the algorithm with expected outputs which would be the basis for its supervised training. This is why I decided to use Min-Max searching algorithm that uses state searching to find the best move using a linear evaluation function.

Technical Approach

As specified earlier, the goal of the technique discussed in this paper is to beat the average agent. The reason for this is that this competition is between students who are usually busy with other classes and have little time to optimize their AI agents. In order to understand the next sections we have to define the average agent. The assumption here is that the average agent uses Min-Max algorithm. Since an average number of turns in a game is 50 (by speculations) and given the time constraints in each turn, the average agent cannot go to the maximum depth of the tree in its first turns, and hence needs to use a heuristic function to estimate how close it is to its goal. It will be assumed that it uses a heuristic function that maximizes its score and minimizes its opponent's score.

Heuristic Function

Due to the time and processing power limitations, my agent is not able to speculate the whole search tree and thus needs to use a heuristic function to be able to evaluate if it is going in the right way to win the game. After reviewing existing literature on similar game playing techniques[1], 5 heuristics were deemed the most promising.

- H1: Total score difference (my score - opponent's score)
- H2: Seeds difference (my seeds - opponent's seeds)
- H3: Overflow of opponent's seeds to my pits
- H4: How far I am to winning ((max score/2) - my Score)
- H5: How close is the opponent to winning (opponent's score - (max score/2))

The use of H1, H4 and H5 is obvious. However H2 and H3 are less obvious. The idea behind H2 is to prevent losing due to the absence of legal moves, causing the game to end and the opponent to collect all seeds in its pits. H3 is related to H2 and tends to minimize the effect on the opponent changing the current plan of action of our agent on purpose by sowing into our pits. H4 and H5 however are directly related to H1. In fact H1 = -(H4 + H5). This is why it was a valid reason to disregard them. A utility function was used to combine the effect of all heuristics:

$$Utility(S) = \sum_{i=1}^{3} W_i * H_i$$

Where S is the state and W_i is a weight associated with the corresponding heuristic function.

For an average agent the weights are $W_1 = 1$ and all the other weights are 0. A tournament was made to choose the best heuristic or combination of heuristics for evaluation. We start by playing an average agent against itself.

MyDepth	OppDepth	First Move	Second Move
9	9	Win	Lose
10	10	Lose	Win
11	11	Lose	Win
12	12	Lose	Win
10	11	Lose	Lose
11	12	Lose	Lose
12	11	Win	Win
11	10	Win	Win

Table 1: Results of the Tournament between H1 against H1

It is clear that playing an average agent against itself is not a good strategy since winning only depends on the depth of the Min-Max tree and there is no general trend for winning. The next strategy was to incorporate H2 into our utility function. The results of playing H1, H2 against H1 is the following:

MyDepth	OppDepth	First Move	Second Move
9	9	Win	Win
10	10	Lose	Tie
11	11	Tie	Win
12	12	Win	Lose
10	11	Win	Lose
11	12	Win	Win
12	11	Win	Win
11	10	Win	Win

Table 2: Results of the Tournament between H1,H2 against H1

These results are also non satisfactory since the there is no criteria for winning except for the depth of my agent to be greater than the average agent's depth. The next tournament will be a weighted combination of H1, H2 and H3 versus the average agent. By reasoning and speculation the weights were initialized to 10 for H1, 1 for H2 and -1 for H3. It would have been more adequate to use learning in this scenario to figure out the most accurate weights, but for the average agent, as we will see, these weights seem to be good.

MyDepth	OppDepth	First Move	Second Move
9	9	Win	Win
10	10	Win	Win
11	11	Win	Win
12	12	Win	Tie
13	13	Win	Win
10	11	Win	Win
11	12	Lose	Win
12	11	Win	Win
11	10	Win	Win

Table 3: Results of the Tournament between H1,H2,H3 against H1

There is a visible performance improvement with this heuristic function, but how will it compare to the combination H1,H2? The following are the results of such a tournament:
As we can see, although this method works well against the average agent, it does not generalize well against other agents. A possible solution to this problem might be in adjusting the weights to the heuristic function through learning.

MyDepth	OppDepth	First Move	Second Move
9	9	Win	Lose
10	10	Win	Lose
11	11	Win	Lose
12	12	Tie	Lose
13	13	Tie	Lose
10	11	Win	Lose
11	12	Lose	Tie
12	11	Lose	Tie
11	10	Win	Tie

Table 4: Results of the Tournament between H1,H2,H3 against H1,H2

Min-Max Vs Alpha-Beta Pruning

With a branching factor of 6, Min-Max can speculate the tree until depth 9. With Alpha Beta Pruning the agent was able to go till depth 14 before any timeout. On average, it is better for the agent to search deeper and return its best current move. The following diagram shows the average time performance (in ms) of a Min-Max agent (in orange) and an Alpha Beta pruning agent (in blue) as a function of the depth of the tree. Although Alpha Beta Pruning clearly provides a better time performance than Min-Max, its effectiveness is highly dependent on the order in which the states are examined[2]. sorting the moves at each node was disregarded since it was found to be expensive and results in a slower exploration rate.

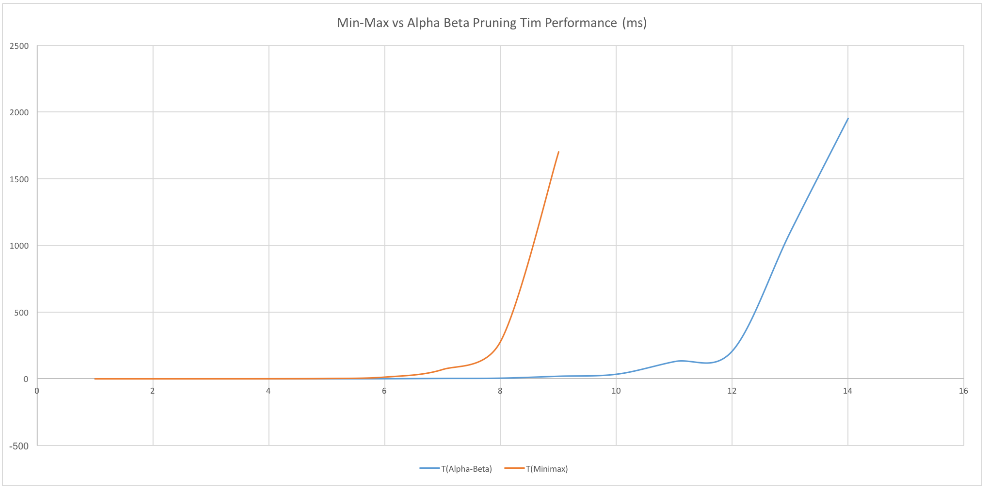

Figure 1: Simple Binary SVM maximizing margin

Conclusion

In conclusion, this paper discusses the feasibility of developing an agent that would dominate the average case agent in Das Bohnenspiel. Such an agent exists by combining 3 heuristics in one evaluation function: the score difference, seeds difference, and opponents' seeds overflow and assigning them different weights.

Advantages

As we can see from Table3, it seems that the agent developed for this project succeeded in dominating the average agent in most of the cases. That is also putting into account that the average agent will be exploring the tree at a depth that is equal or less than the depth of our agent.

Disadvantages

It can also be seen from Table4 that this method does not generalize quite well to the more sophisticated agents. This might be because of the weights that were assigned to each one of the heuristic function. Another disadvantages comes from the fact that we are using Min-Max search algorithm which implies that the agent does not account for any surprising moves made by the opponent, it always assumes that the opponent is smart and chooses the move that will directly get it closer to winning, which might enable the opponent to trick the agent into losing.

Further Improvements

One possible improvement is to use learning techniques to calculate the best weights for the heuristic functions. Another improvement arises from the fact that the agent does not account for previously encountered scenarios. By caching any moves that proved to be killer moves or winning moves based on the past games played, the tree will not need to be constructed from scratch every turn and the optimal move for these states would be chosen if encountered before.

References

[1] Chris Gifford, James Bley, Dayo Ajayi, and Zach Thompson "Searching and Game Playing: An Artificial Intelligence Approach to Mancala" `https://fiasco.ittc.ku.edu/publications/documents/Gifford_ITTC-FY2009-TR-03050-03.pdf`

[2] Imran Ghory: Reinforcement learning in board games, `https://www.cs.bris.ac.uk/Publications/Papers/2000100.pdf`