CS 4701: Final Project Report
Optimal Heuristic For Checkers

Kevin Gregor (krg43) Marcus Boeck-Chenevier (mb2533)

I. Introduction

For our project, we decided to implement a game-playing Artificial Intelligence agent. We chose this because it was a very concrete topic covered in CS 4700 and because it would be an interesting project that we would be able to play in the future and share with others. After looking at a number of board, card, and dice games, we narrowed our choice down to the game of checkers (also known as English draughts). We thought that checkers would be interesting to implement since it is a simple, yet highly strategic game and also pieces can evolve as the game is played (normal pieces become kings), which could lead to an interesting heuristic evaluation function.

Furthermore we chose to implement the entire game and AI from scratch using Java. For this reason we decided to forgo making the game look aesthetically pleasing in favor of focusing on the functionality of the game, and more importantly the functionality of the AI. Visually keeping track of the board was not much of a challenge since we just needed to write a method that prints out the data stored in our grid in a manner that makes sense. Additionally, figuring out how to move pieces was done by scanning console inputs. The first input would be the coordinates of the checker that you'd want to move, then the console would print out all possible moves for that given piece, then the player would write down which move they'd want to make. The piece would then be moved and the board would be updated accordingly. Once we had the AI figured out, a lot of the printing would no longer be necessary since the AI determines what

moves to execute internally and not by looking at a visual board. This greatly sped up our testing, as a whole game could be resolved in a matter of milliseconds instead of possibly taking up to a whole minutes, depending on the length of the game.

II. Objective

In order to implement this game-playing AI, it was evident that we would need to use an adversarial game-tree search. We implemented the minimax search, which seeks to maximize your own benefit and minimize your opponent's by looking multiple moves ahead and assigning a mathematical value to each board configuration (based on the heuristic evaluation function).

Given these decisions on how to implement the checkers AI, our objective was to find the best heuristic function. We aimed to find out not only what elements to take into consideration when defining the evaluation function, but also what weight to give to each of these elements in order to create a better game playing agent. Thus, we asked the questions of:

1. What makes up an extensive heuristic function
2. What weights make that heuristic function optimal

III. Heuristic Evaluation Function

Initially, our evaluation function was simply the difference in number of pieces between the two players (so the AI would want to move towards board configurations where it has more pieces than its opponent), but we quickly realized that we would have to modify our function to add a higher weight for king pieces, since having a king is more advantageous for the player. Even once we created an agent playing with a heuristic function based on number of pieces and number of kings, it was evident that more adjustments needed to be made. Human players were able to handily beat the computer player, as many of the possible moves led to the same value from the evaluation function because a game can go many turns without a piece being taken.

After consulting sources on the internet (http://www.wikihow.com/Win-at-Checkers) as well as checkers strategy books and checkers players we know, we came up with a set of new features for the heuristic function. Our function now includes the following weights in this order:

- Number of regular "pawn" pieces
- Number of king pieces
- Number of pieces in the back row (a good strategy is to keep your pieces in the back row as long as possible to prevent your opponent from getting a king and to prevent your pieces from being taken)
- Number of pieces in middle 4 columns of middle 2 rows (controlling the middle of the board is advantageous)
- Number of pieces in middle 2 rows but not the middle 4 columns
- Number of pieces that can be taken by opponent on the next turn
- Number of pieces that can not be taken until pieces behind it (or itself) are moved

To evaluate our new heuristic function, we simply played against it with both ourselves (humans) and other AI agents with different heuristic functions. Since our objective was to find the weights that would lead to the optimal heuristic, a quantifiable way to measure this was to record the number of wins (and thus win percentage) of each heuristic when pitted against others. Clearly, the heuristic with the highest win percentage would be the "best" heuristic in our environment.

IV. Testing and Results

At the start of our testing implementation we manually assigned values/weights to the different heuristic attributes of each of the players. After multiple times of manually trying different combinations we came up with a basic set of weights to use for more robust testing. To further test our heuristic weights we decided to have player 1 loop through different values (0, 3, 6, and 9) for each of the different weights of the attributes, and testing player 2 with the manual weights we determined. The weights we decided to start with were: (4, 8, 1, 2, 1.5, -3, 3) for the weight of each normal piece, each king piece, keeping a piece in the back row, having a piece in the middle box, having a piece in the middle rows, having vulnerable pieces, having protected pieces, respectively. To determine if the weights would be best we would change them individually/one at a time by increasing and decreasing the weight, and compare the win percentages of each of them. If either of the new values lead to a win percentage better than the original we would further explore those values by applying the same strategy as we did originally (increasing and decreasing values). We would keep iterating through and modifying the weight values until we reached values, that if changed would only decrease the win percentage of player 2. The data we found led to the following results:

Weight being changed (in order)	Weights Format: (pieces, kings, back row, mid box, mid rows, vulnerable, protected)	Number of matches won (out of 4374)	Win percentage
Baseline heuristic function (only evaluates the number of checkers at each turn)	**(1, 0, 0, 0, 0, 0, 0)**	538	12.3%

First iteration			
Normal pieces	**(4, 8, 1, 2, 1.5, -3, 3)**	1999	45.7
King pieces	**(4, 8, 1, 2, 1.5, -3, 3)**	1999	45.7
Pieces in back row	**(4, 8, 4, 2, 1.5,-3, 3)**	3678	84.1%
Pieces in middle box	**(4, 8, 4, 2.5, 1.5, -3, 3)**	3700	84.6
Pieces in middle rows	(**4, 8, 4, 2.5, 0.5, -3, 3**)	3823	87.4
Pieces vulnerable	**(4, 8, 4, 2.5, 0.5, -3, 3)**	3823	87.4
Pieces protected	**(4, 8, 4, 2.5, 0.5, -3, 3)**	3823	87.4
Second iteration			
Normal pieces	**(5, 8, 4, 2.5, 0.5-3, 3)**	3862	88.3
King pieces	**(5, 7.75, 4, 2.5, 0.5, -3, 3)**	3875	88.6
Pieces in back row	**(5, 7.75, 4, 2.5, 0.5, -3, 3)**	3875	88.6
Pieces in middle box	**(5, 7.75, 4, 2.5, 0.5, -3, 3)**	3875	88.6
Pieces in middle rows	**(5, 7.75, 4, 2.5, 0.5, -3, 3)**	3875	88.6
Pieces vulnerable	**(5, 7.75, 4, 2.5, 0.5, -3, 3)**	3875	88.6
Third iteration			
Normal pieces	**(5, 7.75, 4, 2.5, 0.5, -3, 3)**	3875	88.6

After running the initial test with the baseline function, we set a goal for ourselves to find a set of weights for a heuristic function that leads to a win percentage higher than the losing percentage of the baseline. As it can be seen the best weights that we found for each attribute is: Weight of each regular piece = 5, weight of each king piece = 7.75, weight of each piece in the back row = 4, weight of having a piece in the middle box positions = 2.5, weight of having a

piece in the middle two rows = 0.5, weight of each vulnerable piece = -3, weight of each protected piece = 3. These values don't deviate much from our initial estimate, yet they do yield a much higher win percentage (almost twice as high), as well as leaving the "basic heuristic function" in the dust in terms of performance (over 7 times higher win percentage). In addition, the baseline's losing percentage was 87.7%, as it only won 538 games, while our optimal heuristic's winning percentage was 88.6%, with only 499 games lost. Clearly, this is a very large improvement over our initial intuition of simply comparing the number of each player's pieces.

V. Conclusion and Further Improvements

While this does give an answer to our initial question of "what would be the best heuristic function to use for a checkers?", this answer may possibly be further improved. A major improvement that can be made on our strategy of determining the best heuristic would be to increase our resources. Our biggest limitation would be the resource restriction we faced when testing our various heuristic. This is primarily due in part to the long processing time required to evaluate a certain set of values as well as the immense variety of values that the weights could take on. We primarily tested our optimal heuristic with a multitude of loop-generated heuristic weights for the opponent. The heuristic values for the opponent could be potentially be improved by looping through more specific numbers rather than the larger integers we used (example iterating over 0.1 rather than 3). Also, as it has been stated, we primarily tested our heuristic on other AI. An improvement could also be made if a sort of "beta-testing" were to be used, in which we would test our different heuristics to play against a large numbers of Human players to see if humans behave similar to the AI or if there are different factors that we would need to take into account. Finally another uncertainty can be pointed out by saying that this is only a "locally optimal" heuristic value. The argument could be made that our approach matches a hillclimbing approach to finding a best heuristic for our checkers limits our result to only being a "local best" since we would refuse to explore any options that gave a worse result than what we would previously have. So while we are very confident that our heuristic function for our checkers-play artificial intelligence can more than hold its own we do believe that further improvements could be made if provided with the right resources.