12-18-2015

Optimizing a Game of Chinese Checkers

Nicholas Fonseca

Optimizing a Game of Chinese Checkers

Nicholas Fonseca

Submitted in Partial Completion of the
Requirements for Departmental Honors in Mathematics

Bridgewater State University

December 18, 2015

Dr. Jacqueline Anderson, Thesis Director
Dr. Shannon Lockard, Committee Member
Dr. Ward Heilman, Committee Member

Optimizing a Game of Chinese Checkers

Nicholas Fonseca

December 18, 2015

Abstract

Chinese Checkers is a multi-player strategy game in which game play can become surprising complex as the game progresses. In spite of this game's complexity, questions involving games with multiple players have received little research attention. This paper considers the three player case and discusses how to describe short games. By utilizing these tendencies for short games, a heuristic function can be defined which associates a player's possible move with a heuristic value. These heuristic values guide a search algorithm which searches through all the possible moves made in a game. To guide this discussion for three player games, the results regarding single player and two player cases will be mentioned.

1 Problem Background

Chinese Checkers is a strategy based game created in the 1880's. Developed in Germany by an American mathematician, George Monks, Halma (as it was called) did not become popular in America until the early 1920's. Because the colorful board and star shaped pattern, Americans', fascinated with the Far East during the time, changed Halma's name to Chinese Checkers. Currently little research has been conducted on this game; the research that exists seeks to optimize the shortest path across the board and the number of turns needed to win a game between two players [2].

Chinese Checkers is classified as a pebble game featuring fixed slots and movable pieces; the purpose of such games being to move pebbles to a new location following specific criteria. The fixed slots are arranged into a six pointed star on the game board and which form a hexagonal grid inside the star. Marbles represent the pebbles in Chinese Checkers and each player begins with ten marbles. The goal of a game is to transfer a player's "army" from its initial position directly across the board into an opposing player's initial position. This is called the goal space. A man, or marble, can move into an unoccupied, adjacent slot in one turn (called a step) or can move over another adjacent man into an unoccupied position which lies in the direction of the motion (called a jump). Although only one step is permitted per turn, any number of jumps can be made in a single turn as there are available for one single piece. Any piece can move in any one of six directions which can be seen in Figure 1.

Chinese Checkers first received attention in 1993 by a group of student researchers:Joel Auslander, Arthur Benjamin, and Daniel Wilkerson [1]. They proved that there were three optimal arrangements of n pieces which yielded the fastest transfer of men across the board. Later, in his 2009 paper *The Shortest Game of Chinese Checkers and Related Problems*, George Bell expands upon these ideas in his section on army transfer problem. In the first section of his paper, Bell analyzes two player games and provides a list of criterion which describe short games. Among these criterion are three particularly ideal conditions which could be adapted to the three player case: army displacement, centroid displacement, and game length bounds. By considering the distance between armies, Bell concludes that a lower bound exists on short two player games such that no possible game could be shorter than 27 moves. Bell uses an existing short game to serve as an upper bound to shortest possible game length and this game was provided to Bell by David Fabian in 1979. With an upper and lower bound established, Bell performed a depth first iterative A* search of games with lengths between the established bounds. This search yielded no such games and Bell concluded that the shortest possible two player games were

30 turns long [2].

To alleviate the burden of the search, Bell uses his criterion for short games to guide his search algorithm into exploring turns meeting ideal conditions. More specifically, he guides his search through possible games with a heuristic function which assigns certain turns greater priority according to the changing lower bound and critical move. For each possible move a player can make, this heuristic function calculates the lower bound for the game's length using the pieces in this arrangement. The search algorithm first explores turns with the greatest decrease in the lower bound's value. The search algorithm also keeps track of a critical move in which a winning player must have a piece in the goal space. If a possible game doesn't meet this criterion, the search backs up a turn and considers a different move. Bell's heuristic is an example of a uniform-cost function because it chooses nodes based upon a cost to reach a node and a cost to reach his goal. Although Bell describes other criterion for short games, such as man type, ladder formation, and army displacement Bell emphasizes that these criterion are not strict which is why he excludes them from the search. Short games, however, will have a tendency to exhibit these features, but are not required in order for games to be short.

2 Introduction

To find the shortest possible game for three player games, I modify many ideas that Bell and the student researchers discussed in order to provide criteria which describe optimal moves and short games involving more players. I will provide two methods for describing an army's progression and movement using the centroid and displacement. I also discuss bounds and critical moves on both game play and playable space as a way to simplify the possible outcomes for a game and quicken my search. I finish my discussion on defining optimal strategies and situations by describing ideal configurations and the formation of ladders which aid in speedy progression through the playable space.

Using these criterion, I define a heuristic function which is responsible for deciding which possible move is chosen during a turn. Working under the assumption that ideal moves meet a maximal number of criteria, I designed a heuristic function that has a tendency to choose moves which meet several criteria while making progress toward the goal. More specifically, my heuristic function assigns a turn a value based on the resulting displacement of the army, and the change in its centroid. This heuristic function then outputs a list containing coordinates describing the move and an associated integer value describing the move's optimality. This heuristic function recursively defines the optimality of moves at every turn during game play.

After defining the heuristic function, a program was designed to search through all possible moves for games that were between 31 and 58 turns long. Because of the size of the search space between these two bounds, not all criteria discussed will be incorporated into the initial program. In the future, additional criterion and algorithm design can be added to make the program more efficient. The design of this program was broken into three stages. The first stage involved writing a code to play a game. This program runs a recursive procedure to create a list of all possible moves available during a turn, to execute any move at random, and repeat for each player until a desired number of turns passes. In the second stage, I coded my heuristic function so that it provided the previous program with the ideal move to take during each iteration. The final stage of this algorithm searches through every possible move that can be taken by working backwards through the game tree. At this particular stage, protocols are created to prune the search space. This search is designed to look through fewer turns if it finds a game shorter than 57 turns.

2.1 Definitions

Here we give the reader a list of definitions and terminology used frequently throughout the paper. Figure 1 describes what the Chinese Checkers Board looks like with the directions the men can move.

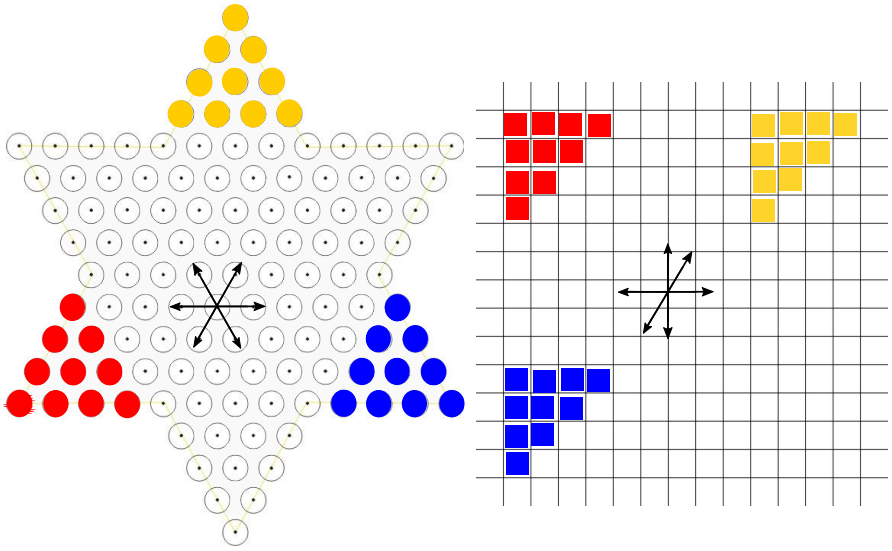

Figure 1: The figure on the left shows what the Chinese Checkers Board looks like during game play, while the right hand side shows a representation of the board on a two dimensional grid.

Def: A *cell* refers to the x and y coordinates for a slot on the board.

Def: A *man* refers to an individual piece or marble. Each player has ten pieces or marbles.

Def: An *army* refers to a player's collection of ten pieces. In this paper, I refer to armies by their color: Red, Blue, and Yellow.

Def: A *base* refers to the ten cells a players men occupy before the first turn.

Def: A *goal state* refers to the arrangement of men along the board in which all one player's men occupy an opposing players base. Since I assume that the red player is the winning player, the goal state implies that the red men will occupy an enemy's base.

Def: A *move* is a permissible action which displaces a single man from one point on the board to the another. A move is a list containing two ordered pairs. The first ordered pair describes the displaced man's initial board position and the second describes the man's final board position. Symbolically, M =

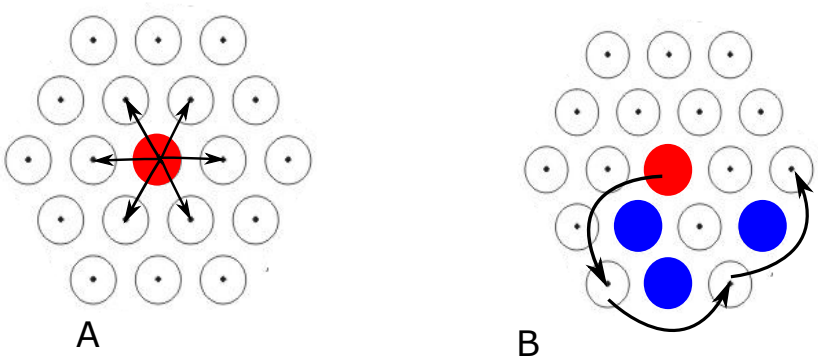

Figure 2: Figure 2A demonstrates how the red man can step. Figure 2B demonstrates the possible jumps the red man can make for the given arrangement. The arrows point from where the man can begin to the empty cell the man can enter. The man in Figure 2B can make any of the indicated moves in one turn.

$[(x_i, y_i) : (x_f, y_f)]$.

Def: A *turn* refers to the process of moving from one arrangement of men on the board to the another. A turn ends when a single player has moved a single man.

Def: A *step* moves a man into an adjacent empty space in one of the six directions shown in Figure 1. A player can only make one step.

Def: A *jump* refers to a move in which a man, M_1, moves around a directly adjacent man, M_2, into a an empty spot directly adjacent to M_2. Note that this empty spot must lie in the same line as M_1 and M_2.

Figure 2 demonstrates some possible step and jump moves.

Def: A *game space* or *game tree refers to the collection of all possible games or all possible moves that can be created during any one game.*

Def: A solution *refers to the sequence of moves taken by the three players to complete a game.*

3 Conditions for Optimally Short Games

In order for a program to search fro the shortest game, it must create a list of all possible games. Using a combinatoric argument it can be shown that the number of possible board arrangements is on the order of 10^{101}*. This is a huge amount of options that a search program will need to look through and therefore the program will suffer from space and time limitations. To efficiently search for a short game solution, moves need to be ranked based on how well they advance a player's progress to the goal. Because the players are assumed to be cooperating in an ideally short game, different criteria are used by each player when making decisions about which move to make. More specifically the winning player's decisions are guided by concepts such as distance and centroid while the losing players are concerned with traffic in some playable area and the creation of ladders.*

The goal for the Red player is to transfer his army across the board into the Blue player's base. The best way to describe the red players progress toward this goal would be to consider his distance away from the goal state. Therefore the underlying assumption for optimal criteria is that they aid in minimizing this distance as game play progresses. Within this assumption, we can also conclude that optimal moves avoid increasing this distance. Therefore each criteria is developed assuming that men never backtrack. Def: The distance between two armies will be defined as the fewest number of step moves it takes to be adjacent to a single man from an opposing army. This will be symbolically written as $d(\mathbf{B}, \mathbf{R})$*, where* **B, R are the heads of two distinct armies. The distance between these two cells can be calculated using equation 1.**

$$\mathbf{d}((x_i.yi) : (x_f, y_f)) = max\{|x_f - x_i|, |y_f - y_i|\} \tag{1}$$

Definition: $\mathbf{R_H}$ **is the Head of R, or Head({R}), is the man with cell coordinates,** (x, y) **where** $x = max\{x_i\}$ **and** $y = min\{y_j\}$ **where j is a man with an x-coordinate** $max\{x_i\}$ **and i a man in R.**

Using the definition of distance, bounds on the length of the game can be defined to guide a search. A lower bound on game length is defined to be the shortest possible game played under ideal conditions.

Proposition: The lower bound for the remaining length of the game can be calculated at any point during game play and is denoted L_B.

$$L_B = min\{d(\mathbf{R_h}, \mathbf{G_i}) + \mathbf{3(10 - n)})\} \tag{2}$$

Where $\mathbf{R_h}$ is the head of the red man's configuration and $\mathbf{G_i}$ corresponds to any cell in the goal, and n represents the number of red men already in the goal state.

Proof for the Lower Bound:

Consider any configuration of the red men on the board. The minimal distance required to get any one man into the goal equals the distance between the man closest to the goal, by definition this is the head of a configuration, and the closest empty cell in the goal. This quantity is defined as $min\{d(R_h, G_i)\}$. Once this man is placed in the goal the best case scenario would have every remaining man placed into the goal space in each subsequent turn. Because each other player must make a turn before the red player can move another man, three turns pass for every one man displaced. It will take an additional $3 * (10 - n)$ turns to get the remaining men into the goal. The lower bound on game length is the sum $min\{d(R_h, G_i) + 3(10 - n)\}$.

An upper bound, U_B, on a solution can be defined as the number of turns required to guarantee that a solution exists. An upper bound can be found by playing multiple games of Chinese Checkers, the upper bound is the shortest of these games. Doing this myself gives an initial upper bound as 58 turns. By knowing an upper bound, the critical move can be defined where at least one man must be in the

goal in order for a game with the Upper Bound's length to exist. The critical move, α, occurs at $\alpha = U_B - 3 * (10 - 1)$. Therefore the critical move occurs at $\alpha = 58 - 27 = 31$ turns.

Proof for the Critical Move:

During the critical turn, the red player must have at least one man in the goal. This implies that there are still nine men that need to be displaced. The best case scenario has one man entering the goal every turn after α. Therefore $\alpha + (9 * 3) = U_B$. The upper bound is defined before the game begins, thus solving for α gives $\alpha = U_B - 3 * (10 - 1)$

These bounds help define the parameters for a search but do not guide a player's decision during the turns leading up to those bounds. To guide the search between these bounds, other criteria needs to be considered. To create these criteria I consider three assumptions that tend to lead to short games. First, an optimal move yields the smallest lower bound possible during that turn. The lower bound discussed above depends upon $\min\{d(R_h, G_i)\}$, where G_i is an empty cell in the goal space, and gives a measure of turns remaining in a game. This can be achieved by moving men into cells which minimize this distance. Therefore we assume optimal moves decrease the distance between the head and the goal. Secondly, the red army seeks to cross the playable game space as quickly as possible in order to reach the goal. From this point of view we assume that optimal moves maximize this increase in the distance any man travels. Also notice that to move quickly across the board, the distance an army needs to travel needs to decrease. Thus the final assumption states that optimal moves decrease the maximum distance the army needs to travel to reach the goal. Using these assumption a list of optimal criteria can be given depending on distance.

Criterion: The red player only jumps.

A jump increases the displacement of a man by at least two steps

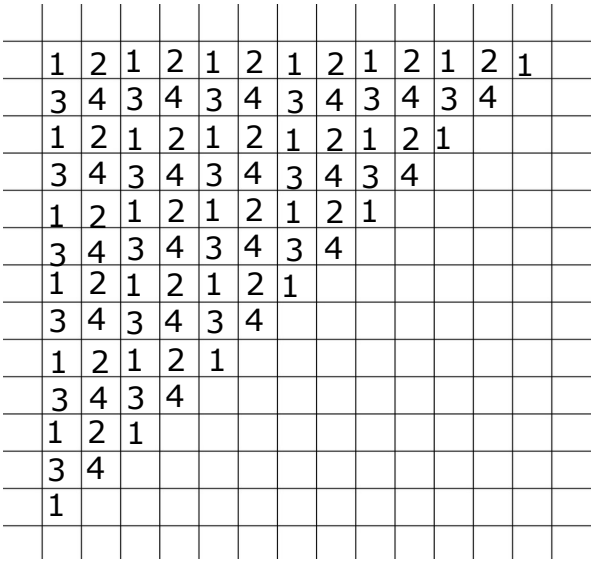

Figure 3: The two dimensional grid representing the Board has been labeled according to how men can jump. 1/4 men can only jump over 2/3 men and visa verse. Two types of ladders exist; 2-3 ladders and 1-4 ladders.

while a step only increases the displacement by one. Therefore ideal moves for the red army involve jumping. To ensure that a game can be played where the red player only jumps consider Figure 3. The cells are labeled from one to four in manner that groups those cells which can be reached only by jumping and therefore only step change a man's type. The frequency of men types in the red army's base matches the frequency of men types in the blue army's base. This implies that the red player can transfer his army into the blue player's base solely by jumping. Therefore R's goal space is the base occupied by the blue army. Also, notice that the number of type 4 men in the red player's base does not match the number of type 4 men in the yellow players base. This implies that at least one man needs to change its type in order for the red player to transfer his men into yellow player's base. This implies that the red player's optimal strategy is to jump into the blue player's base.

Criterion: Blue and yellow players move into cells directly adjacent

to a red man.

Definition: $\mathbf{R_T}$ is the tail of R, or Tail({R}), is the man with cell coordinates, (x, y) where $x = min\{x_i\}$ and $y = max\{y_j\}$ where j is a man with an x-coordinate $max\{x_i\}$ and i a man in R.

In order for the red player to jump, the space directly adjacent must be occupied. If a red man only jumps over a red man, then the ideal outcome would decrease $d(R_t, G_i)$ by at most one every turn. The ideal situation involving the red man jumping over another player man would have the R_h jumping over another player to the goal. Here $d(R_T, G_i)$ can decrease by at least one making jumping over other player's men optimal. In order for the red player to jump over a different colored man however, the Blue or Yellow players must move men directly adjacent to a red man.

Criterion: The red player displaces the tail of R.

The tail is the man in R with the furthest distance from the goal state. The maximum displacement for this man would be a jump move over $\mathbf{R}_H$, because the head is, by definition, the man closest to the goal. Here $\mathbf{R}_T$ becomes $\mathbf{R}_H$ and $d(R_h, G_i)$ decreases and the value of L_B decrease. Also notice that a new man, $\mathbf{R}_j$, now becomes the tail of the arrangement. In order for $\mathbf{R}_i$ to be the original tail, $minx_i \leq minx_j$.Therefore, the maximum distance the army needs to travel decreases.

Criterion: The red player moves a man into a cell such that $d((x_i, y_i) : (x_f, y_f))$ is maximal and $d((x_i, y_i) : (x_f, y_f)) - d((x_f, y_f) : G_i)$is minimal.

Now consider the case where the tail cannot jump over the head. According to the first assumption, we seek to decrease the value of the bound. This suggests that we move the head of the arrangement. This type of move will only displace a man a distance one and therefore is not ideal. Therefore the player should move the man which undergoes

the greatest possible displacement. However a move which displaces a man the furthest may not be ideal. Rather, the red player chooses moves for which the displacement is the greatest and distance to the goal is smallest.

Distance arguments provide the strongest sense of how well the red player approaches his goal, but are not the only way to describe optimal moves. The red player's progression toward the goal can also be described using the spread of the army across the board. To minimize the spread of an army, bounds can be used to restrict which moves a player may make, or the player can track the centroid of their army as it progresses across the board. In their paper on *Optimal Leapfrogging*, Auslander, Benjamin, Wilkerson describe the centroid as a vector quantity such that:

$$C(\mathbf{R}) = \frac{1}{10}\sum_{i=1}^{10} \mathbf{x_i} \tag{3}$$

. C(R) is a vector pointing from the cell (1,1) to the point on the board where the summed distance of d($\mathbf{R_i}$, C(R)) for all men in R is zero. The length of this vector represents the mean distance traveled by the army. The angle between this vector and the ideal centroid measures how far the army deviated from an ideal path across the board. Using the concept of a centroid, a new list of criteria can be used to describe the red army's optimal moves.

Consider an ideal path from R's base to B's base as a vector quantity connecting R's centroid to B's centroid. This vector describes the displacement R needs to make in order to complete the game. This will be refered to as the ideal centroid, $\mathbf{C_I}$, see Figure 4.The unit centroid is defined as $\mathbf{U_I} = \frac{\mathbf{C_I}}{||\mathbf{C_I}||}$ and gives a sense of the direction the army should travel. The deviation an army makes from this centroid can be described using the angle between $\mathbf{C_R}$ and $\mathbf{C_I}$. The projection

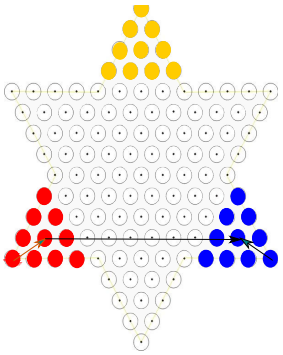

Figure 4: This figure demonstrates how the ideal centroid can be found. The pink arrow represents the red player's centroid and the purple arrow represents the blue player's centroid. The black arrow connects these two centroids and represents the ideal centroid.

coefficient, P, between vectors relates the angle between two vectors to the their magnitude. Therefore, the deviation from an ideal path can be calculated using the value P.

Criterion: Ideal moves maximize the value of P, where P is the projection coefficient between the ideal and red army's centroid .

By assumption, the red army's progress takes priority over the blue and yellow armies. The red army's ideal trajectory lies along the direction of the ideal centroid, $\mathbf{U_I}$. This only happens when the angle between $\mathbf{C_R}$ and $\mathbf{U_I}$ is zero. By definition, the projection of $\mathbf{C_R}$ onto this vector equals $||\mathbf{C_R}|| \cos\theta$.This implies:

$$\frac{P}{||\mathbf{C_R}||} = \cos\theta.$$

Because the cosine function decreases when $0 < \theta < 90$, the maximum value for $\frac{P}{||\mathbf{C_R}||}$ occurs when $\theta = 0$. Therefore $\frac{P}{||\mathbf{C_R}||}$ is a maximum when $\mathbf{C_R}$ lies on top of the ideal centroid, which is ideal. Figure 5 provides an example of two different arrangements and their centroid vectors.

In addition to bounds on length, the playable game space can also be restricted. In ideal games, a majority of game play is spent

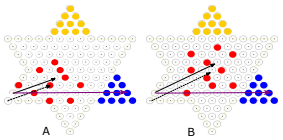

Figure 5: Figure 4 depicts two different arrangements for the red army. The purple arrows represent the ideal centroid while the dotted arrows represent centroids for the red army. To better compare the two centroids, I recenter the origin to the tail of the ideal centroid and draw the red army's centroid using a solid black arrow. Arrangement A has a smaller angle between the ideal and army's centroids, therefore it is optimal compared to Arrangement B.

maneuvering men within the cells where the i coordinate is less than 10 and whose j coordinate is less than 6. By restricting movement to this region, the distance for any red man to the goal either decreases or remains the same. Because the red men are restricted to this region of the board, the other players can only aid the red player's progress to the goal by moving into this region.

4 Searching for the Shortest Game

Game play in Chinese Checkers can be modeled by using a tree diagram making a tree search the ideal means to find the shortest game. Trees are a special class of graphs. A graph is a mathematical structure used to describe how physical states of a system are influenced by actions taken which alter these states. Graphs can be described using two sets of objects interacting with one another: vertices (or nodes) and edges. Vertices are the objects which represent the physical states of the system, or the possible arrangements of men on the board. Edges represent the set of possible actions that can be made to alter the state of the system. The set of edges represent moves in this analysis. Two vertices, or two distinct arrangements, are connected by an edge if there exists one single move which transforms the first arrangement into the other. Trees have an additional property that only one path exists between any two vertices. This means that

there exists a unique sequence of moves which takes one arrangement of men into another after a set number of turns.

An algorithm begins searching a tree by looking into the nodes that are connected to the root node, or initial state, by an edge: the root node here is the initial board arrangement. There are two basic types of brute force tree searches and they are categorized by the order in which nodes are expanded; these searches are called breadth-first search and depth-first search. A node is "expanded" when the search algorithm performs the action required to make the system look like this node. In other words, a node is expanded when the algorithm moves a player's man. Beginning with the root node, a Breadth-First search creates a list, S, of vertices which are connected to the root node in the order the search program sees them. The algorithm then expands the first node in S, removing it from the list, and adds every adjacent node to the end of S in the order the program sees them. The program then expands the next node in S and repeats this process. The search is completed when every vertex is expanded, or S is empty. A depth-first search differs only in the order by which the nodes are appended to S. Instead of appending the new vertices to the end of S, Depth-First Searches appended these vertices to the beginning of S [3].

My tree search program can be broken up into two major phases: a game playing phase and a searching phase. The program plays a game by calling a function which creates a list of possible moves for a particular player at a particular time. A heuristic function then assigns each of these moves a value depending on how well that move brings us to our goal and outputs a list of possible moves sorted by their heuristic function values. The program then chooses the first element in this list and performs the move. The program repeats this process using the same procedure above for the different players until

a certain number of iterations has been performed or until a game reaches a certain length. After the program plays a game it outputs a list of moves made and searches through every game with this length employing the depth-first search method.

In order for my program to play a game of Chinese Checkers, I needed to create a means to track the arrangement of men on the board. I did this by simulating the board with an array containing 15 arrays where each array contains fifteen elements. This structure describes the location of any man using an ordered pair (i, j) where i tell us the particular array the man is located in and j indicates the position of the man in this array. Each element within the individual arrays can be labeled either R,B,Y,E,W. The labels R, B, Y, indicate that there is a colored piece occupying the corresponding cell with coordinates (i,j). An E corresponds to an empty cell on the playable board space while a W indicates that the corresponding (i, j) ordered pair does not exist on the board. Figure 6 shows what the beginning arrangement of the men looks like in this model.

To find the possible moves for some player, a possible moves function was created. This function uses three different call sequences: 'steps", 'jumps", and 'adjacent jumps". Each of these functions takes three arguments: an x coordinate, a y coordinate, and the Board described above. 'Steps" creates a list, S, of all the steps that the man with coordinates (x,y) can make by looking to see if any adjacent cells in the array Board are empty (or equal to E). 'Adjacent jumps" creates a list of single jumps by looking to see which adjacent cells are empty. If a cell is non-empty, then the program looks another step in that particular direction to see if that cell is empty. When both of these conditions are met the function adds the move to a list called J. Jumps modifies this list by adding all multiple jumps to the list. To do this, a 'while loop" unions a set Jnew, containing the jumps adja-

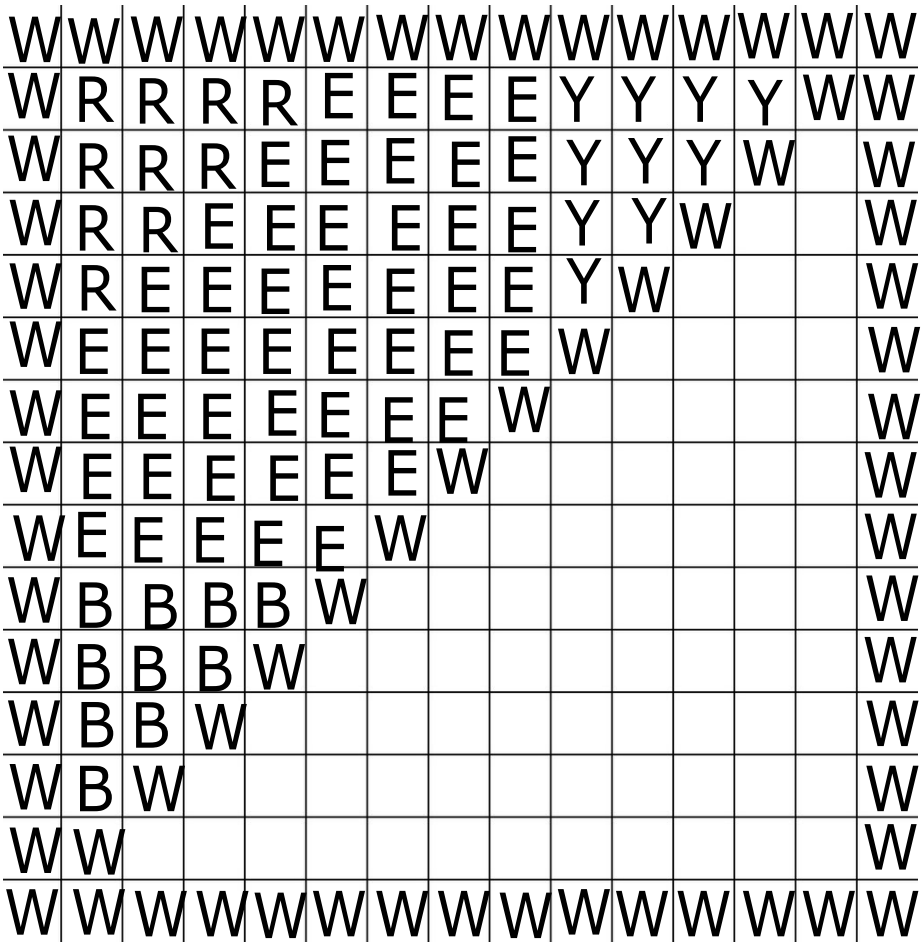

Figure 6: The two dimensional grid above represents the starting state of the Board in the computer. The rows of the grid correspond to the i component of the ordered pair and are lists in the array. The columns represent the individual elements in the list described above. Any unmarked square represents a 'W".

cent to the moves defined in set J, with the set J. The 'while loop" will continue to union the two sets until the sets Jnew equals J and outputs the set as a list J. The possible moves procedure then concatenates the lists S and J. A procedure, named 'transitions", then forms a list representing the initial coordinates for a man with its final coordinates if the move were made. The program uses the list of order pairs to play a game.

To make a man in the 'Board" array move from one cell to another, a set of three functions are called upon. These are called turn functions and there is one for each of the three players. A turn function takes five arguments 'a",'c",'gs", 'iterator", and 'Board". The first four of the arguments will be discussed later, during the search phase of the program. At this stage, the function takes the ordered pairs outputted by the heuristics function and makes the cell corresponding to the first ordered pair in the list equal to 'E' while making the cell corresponding to the second ordered pair equal to the player's color, either 'R', or 'B', or 'Y'. The turn functions output the Board and move made. The program plays a game with length i using a while loop which cycles through the three turn functions and adds an element to a list called iterator. To track the moves made during a game, the program also adds the moves made in this while loop to a "Moves" list The while loop terminates when the iterator has length i.

Recall that this is a massive tree diagram and any unguided brute force search would suffer from space and time limitations. In order to efficiently search for a short game solution, I employ a heuristic function to aid the search process. A heuristic function assigns each node a value determined by a set of optimal criteria. The aim of my heuristic function is to assign optimal nodes with higher values which will guide a search into expanding short games first. The heuristic

function will also delete, or prune, nodes with a lower bound greater than 57 which will not produce short games.

Each turn function has a heuristic function which maps the possible moves available to a particular player to an integer indicating the optimality of that particular move. This integer value is calculated by considering the distance and centroid. The program then relates each move with its integer value in an array which is added to a list, 'HF" and sorts this list by increasing heuristic value. Each heuristic function outputs an array with two coordinates corresponding to the moves with the greatest value. Because the red player wins the game and the blue and yellow players are cooperative, the heuristic function uses different criteria for the red player than it does for the other players.

The heuristic function calculates this integer value by encoding the criteria described above. It operates under the assumption that optimal moves meet as many of these criteria as possible. Therefore the function increases the moves value for each criteria it meets and decreases its value for those moves that violate these criteria. Since the program is designed to search through every possible game that is thirty to fifty-seven turns long, the heuristic function assigns values such that the search algorithm has a tendency to find short solutions sooner.

The red player's heuristic function focuses on the displacement of his men. As described above, optimal moves for the red player maximize the displacement any man makes while minimizing the distance that man has left before reaching the goal. Thus the program initially relates each move with the difference between these two quantities. Therefore $c = d((x_i, y_i) : (x_f, y_f)) - min\{d((x_f, y_f) : G_i)\}$**. Th heuristic function calculates this using two procedures: the distance function defined above and a minimum distance function which uses a 'for**

loop" in combination with this distance procedure. After this calculation the heuristic function adds one to this value for each move that positions a man next to or a distance two away from another man. This forces the creation of ladders. Rather than compute the army's centroid and projection coefficient at each iteration of the red heuristic in the search process; instead we imposed a restriction on the game space for the game. Those moves that are ideal create as little deviation from the ideal centroid as possible therefore large deviations will occur less frequently if the men are confined to a section of the board. Because R's ideal centroid is $[0, 12]$ and the maximum y coordinate for any piece is 4, I kept gameplay restricted to the region between $y = 0$ to $y = 6$ to allow the the men with $y = 4$ the ability to jump in the y direction without any penalty. Therefore I subtracted one from any man whose initial y coordinate was less than 6 and whose final y coordinate is greater than or equal to 6. However one is added if a man's initial y coordinate was greater than 6 and whose final coordinate is less than 6.

The red heuristic function also calculates another integer value which serves as a cutoff value for the search. This cutoff value is equal to the length of the Upper Bound or the shortest game the program has currently found. The program determines whether a node exceeds this cutoff value by calculating the sum between the current lower bound on game length, using the lower bound theorem, and the number of turns passed. If the value of the node exceeds the cutoff, that node is deleted from the list "HF". If it does not, then the node is added to the list as indicated above.

The blue and yellow heuristic functions aim to aid the red army's progression across the board. This is done by forcing the formation of ladders. Each move starts with a value zero. Then the heuristic function adds one to a move's value if it places a man directly adjacent

to a red man. However if that move places the man directly adjacent to another man so that the red man cannot jump, then the function subtracts two so that the program searches through this turn last. If a move places a man into a cell that is a distance three away from a red man, the function adds one to its value.

The heuristic functions for the blue and yellow players differ only during the beginning of game play. Because the red player needs to transfer into the blue army's base, the blue men must exit their base as quickly as possible. To accomplish this, the function adds one to any move which begins inside the blue army's base and moves them outside while decreasing the value if a move places a blue man back into the base from outside of it. The yellow player begins the furthest away from the restricted game space. In order to force the formation of ladders, the yellow army needs to advance into this area quickly. Therefore the heuristic adds the difference between the distance displaced and $y_f - 5$.

After the program plays a game it outputs a list of moves made and searches through every game with this length. It does this by employing a depth first search. Working backwards, the program undoes the last move and recreates a list for all the possible moves. It then performs the second move in the list. If the board is in the goal state then the program prints a list of the moves for this game and decreases the game length it is searching for by one. If the board is not in the goal state then the program back tracks again and takes the third move on the list. This process continues until all possible moves after the second to last turn have been generated. When this happens, then the program moves back one more turn, undoing the last and second to last turn, and searches through the possible moves at this state. The program only backtracks and looks at possible moves in the prior turn, if there are no more unexplored

moves available during that particular turn.

The search algorithm works by using five different 'while loops" in the function called play. The 'while loops" change based upon the value for 'gs". The program starts by creating an empty list called iterator, makes the value for 'a" and 'gs" equal to zero, and makes the value of 'c" two. In this program, 'c" represents the turn the program is currently searching through (this turn number is the length of the game minus the value c) and 'a" represents the number of iterations made during that turn. While 'gs" equals zero, the play function cycles through the three turn functions, adds the element 1 to the iterator list for each turn function called, and adds the moves made to a list called 'Moves". The value of 'gs" becomes one when the play function has performed thirty one moves; this corresponds to the value α described above.

After making 'gs" equal to one, the procedure 'critical" checks the board arrangement to see if any red man occupies the blue base. If it does, the program makes 'gs" equal to two; there is no need to perform a search at this level. If there is no red man in the goal state, the program performs a search to find a turn in which there is a red man in the goal state. The search proceeds in this 'while loop" by initiating another 'while loop" that iterates as long as the value for 'c" is not zero. First it makes the value of 'a" equal to the value of c'th element in the iterator. Then a 'for loop" in conjunction with the 'undomoves" procedure, resets the Board to the arrangement before the last c turns were made. The program then makes the c'th element in the "Moves" list equal to the output of a turn function. The program decides which turn function to evaluate by indexing through the 'TChooser" list which contains the three possible turn functions depending upon the value, c mod 3. While evaluating the turn function, the value of gs triggers the heuristic function to per-

form its second set of procedures. Instead of outputting the largest valued element in the sorted HF list, called Q, the procedure first deletes the first "a" largest valued elements in HF. If the sorted list contains more than one element, the heuristic function outputs the highest valued element in the list to the turn function. At the same time it increases the value for the cth element in the iterator by one. If there exists only one element in Q, the heuristic outputs that move to the turn function while making the value of the c'th element in the iterator zero and increasing the value of c by one. In order to leave the while loop, the procedure runs the critical move procedure after every red player's turn. If the Board meets the critical move criteria, the extra elements in the iterator,representing moves that do not need to be made, are deleted; the value of c becomes zero, guaranteeing that the 'c" while loop terminates; and the value of 'gs" becomes 2.

The third and fourth 'while loops" operate in a similar fashion as the first two loops. The program plays a game until the length of the iterator becomes fifty eight, the length of the upper bound. Then the program searches through these moves until the Board array contains the goal state arrangement. Once it does, it makes 'gs" equal to four and prints the moves for this short game. To ensure that the program searches through all possible games, the function checks the value of the elements from the critical move found up until the last element in the iterator. If all these elements equal zero the program makes "gs" equal to two and searches for a new critical move to continue a search from. If there exists at least one non-zero, the program makes "gs" equal to three and searches for a shorter game with the critical move found. If every element in the iterator equals zero, the program has exhausted all possible options and the program terminates.

5 Conclusion

Some issues with my program developed while trying to perform the final search. During the initial stage of the search algorithm, the heuristic function chooses to move a man and then undo this move during the second turn. It continues to cycle for the 31 turns it is designated to play. Since the function never makes more than one different turn for the red army, the red army will never enter the goal space and the program will not terminate. I decided to revert back to an earlier form of my heuristic function which does not incorporate the lower bound heuristic or distance functions. However this program has yet to yield a solution shorter than length 58, which was expected, and has not terminated. In the future, I will patch the bug in the heuristic program and re-run the algorithm. By fixing this bug, the program should search through short games quicker as it prunes away undesirable options. I will also edit the program so that it runs more efficiently and can incorporate more criterion into the heuristic function.

References

[1] **Auslander, J., Benjamin, A. T., Wilkerson, D. S.. (1993).** ***Optimal Leapfrogging.*** **Mathematics Magazine, 66(1), 14–19. http://doi.org/10.2307/2690465**

[2] **Bell, George I. (2009).** ***The Shortest Game of Chinese Checkers and Related Problems.*** **Integer, Volume 9 Issue 1, 17-39**

[3] **Korf, R. E. (2005).** ***Search.*** **In L. Nadel, Encyclopedia of Cognitive Science. Hoboken, NJ: Wiley. Retrieved from http://maxwell.bridgew.edu/login?url=http://search.credoreference.com/content/entry/w**

1