Dealing with Uncertainty: a PiecewiseGrid Agent for Reconnaissance Blind Chess*

Timothy Highley, Brendan Funk, Laureen Okin
Department of Mathematics and Computer Science
La Salle University
Philadelphia, PA 19141
highley@lasalle.edu

Abstract

Reconnaissance Blind Chess (RBC) is a two-player chess variant where players do not have complete knowledge of the game state. Each turn, a player has a sense action and a move action. For the sense action, the player chooses a square on the board and then is informed of the identity of all pieces in that square and the eight surrounding squares. The only information about the locations of opposing pieces comes from the results of a sense action or when an opponent captures one of your pieces. The move action is a standard chess move with slight rule changes to adjust for the incomplete knowledge of the game state. Whenever a player captures an enemy piece, the player learns that a capture took place but not which enemy piece was captured. This paper presents a rules-based agent for playing RBC that took second place in the first worldwide RBC competition: the Fall 2019 NeurIPS RBC tournament sponsored by Johns Hopkins University Applied Physics Laboratory. Following a more detailed explanation of RBC, the fundamental underlying data structure that the agent uses to track game state is presented: the PiecewiseGrid. The PiecewiseGrid maintains a separate probability distribution for each piece, reflecting the agent's belief about where that piece might be located on the board. This allows the agent to track possible game states and their relative likelihood in a space-efficient manner. Strategies for choosing where to sense, making a move, and updating the PiecewiseGrid are also presented.

1 Introduction

The game of chess has long been a testbed for advances in artificial intelligence. Reconnaissance Blind Chess (RBC) is a recently introduced variation of regular chess [2] in which the players do not know exactly where the opponent's pieces are. That is the "blind" part of the game. The reconnaissance part of the game comes from the sensing actions that the players take. On a player's turn, they take a sense action and a move action. In a sense action a square is chosen, and then that square and the eight immediately adjacent squares – a three by three block – are revealed. The contents of those squares (if anything) are revealed but only to the player that took the sense action. Only what is in the sensed squares is certain; everything else is unknown. Aside from that, the rules of the game are generally the same as regular chess. The king is the target. However, rules for check and checkmate are ignored since players may not be aware that they are even in check. Instead, the game ends when a king is actually captured. If a player attempts an illegal move, such as a rook moving through an enemy piece, the piece will move as far as it can. This may result in an unexpected capture.

Markowitz, et al analyzed RBC and determined that when a player is facing an uncertain situation in RBC, on average the number of possible game states that are consistent with prior observations exceeds that of a player in Texas Hold 'Em poker [1]. They looked at several different approaches to RBC and how they impact the number of game states that a player is facing.

An RBC player, whether human or an AI agent, must deal with uncertainty in order to succeed. In the agent that is described in this paper, piecewise probability distributions are used to generate sample boards that are fed to a leading open-source chess engine: Stockfish [3]. The engine is used to both decide the agent's own moves and to guess the opponent's moves. The predictions of opponent moves are then used to update the piecewise probability distributions. For sense actions, the agent essentially senses the individual square that has the greatest uncertainty (i.e. a piece whose probability of being in a specific square is closest to 50%).

Some of the strategies employed here are refinements of other RBC strategies published by Markowitz, et al [1] or that have been discussed elsewhere. However, the piecewise probability distributions for representing the game state that are presented here is a novel approach in the context of RBC.

2 RBC Agent

Any RBC agent must tackle four problems. In the face of uncertainty, how will the game state be represented? As moves are taken, how will the game state be

updated? Based on the game state, what move should the agent take?Based on the game state, what sense action should the agent take? This section answers each of these questions in turn.

2.1 PiecewiseGrid: Representing the Game State

The board state is represented as a collection of 32 PiecewiseGrid objects: one for each piece in the game. Each PiecewiseGrid consists of a probability distribution that is represented as a two-dimensional array: a probability for each square on the board that indicates the agent's belief that the piece is in that particular square. The actual probability grid is the principal element in a PiecewiseGrid object, but there are also a few other pieces of information to help keep track of a piece's state. There are Boolean variables to indicate if a piece has been captured or is a promoted pawn. There is a list of other pieces (the capturedList) that is populated when an enemy piece is marked as captured. The list can be used to correct the game state if it is later determined that marking the piece as captured was an incorrect conclusion. For each pawn, there is a list of columns that the pawn might possibly be in.

One of the most common operations involving the PiecewiseGrids is to generate a possible board state. Most of the time, the exact board state is unknown, but the PiecewiseGrids represent the agent's beliefs about what the board might look like. To generate a possible board, each piece is placed onto the board, one after the other, each into a square based on its own probability distribution. If two pieces happen to be randomly placed into the same square, the second piece that would be placed there is simply placed in a different square instead. This skews the probabilities for any piece that gets bumped from its initially chosen square, but it is a quick way to generate a potential board without maintaining a massive collection of possible boards. When generating a board, the pieces are placed onto the board in order of importance, beginning with the kings and queens and ending with the pawns. In that way, if any piece suffers from skewed probabilities, it is less likely to be one of the more important pieces.

2.2 Updating the PiecewiseGrid

At the start of the game, the location of every piece is known with 100% certainty. Throughout the game, a player always knows the locations of their own pieces. As a consequence, the 16 probability grids for a player's own pieces always have the same format: 1.0 for the square where the piece is located and 0.0 for the other 63 squares. (If the piece has been captured, all 64 squares are 0.0.) The probability grids for the opponent's pieces are not so simple, and they must be updated regularly.

There are three points at which the game state is updated: after the opponent's move action, after a player's own sense action, and after a player's own move action.

After the opponent's move action, there was either a capture or not. If there was a capture, the agent will make a guess about which enemy piece performed the capture. If there was no capture, the agent makes a guess about what move the enemy made.

To guess which enemy piece made a capture, a board is first generated that contains only the pieces where the agent is fairly certain of the location (i.e. probability > 0.99). This includes all of the player's own pieces and some of the enemy pieces. After these pieces are "locked" in, it is then determined, based on each piece's probability grid, how likely it is that each enemy piece (including those not locked in) would be able to attack the square where the capture occurred. The pieces that are locked in may block certain pieces from attacking the square in question, eliminating those pieces from consideration and allowing the agent to better zero in on the actual attacking piece. Ultimately, the exact piece might not be determined. In either case, the probability grids are updated under the assumption that the probability a particular piece was the attacker is proportional to the probability that the piece was in a position to attack. Naturally, if only one piece had a non-zero probability of being in a position to attack, then the agent is sure about which piece performed the capture.

For each enemy pawn a list of possible columns is maintained. A capture is the only way that a pawn can change columns, so whenever the enemy captures a piece, the pawns possible-columns lists are updated. For example, a pawn that begins in column A is known to still be in column A as long as no capture has occurred in column B.

If no capture took place on the opponent's turn, then updating the game state after an opponent's move takes a very different approach. Using the probability grids, a number of possible boards are generated. Each board is analyzed using Stockfish. The best move on each board according to Stockfish is noted. Other good moves on each board according to Stockfish are also noted as good moves, and any moves that might put the king in check or pin a piece are also noted as good moves. (Those moves get special attention because in RBC, putting the king in check and pinning a piece are stronger than they are in regular chess. Those moves can lead to an immediate win if the opponent does not sense them.) Moves that are not noted as the "best" move or a "good" move are still noted as "other" possible moves. The agent places all of the possible moves into one of those three categories, and the probability grids are updated with the assumption that the opponent was most likely to have taken one of the "best" moves, less likely to have taken one of the "good" moves, and very unlikely to have taken one of the "other" moves. As each move is processed,

the probability that a piece is in the move's starting square is decreased and the probability that the piece is in the move's ending square is increased by an equal amount. Figure 1 gives a simple example, where the two possible enemy knight moves are processed, and one of the moves is deemed much more likely than the other.

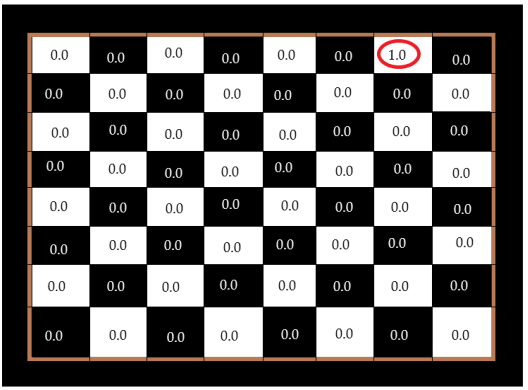

Figure 1: An enemy knight's probability grid before an opponent's move

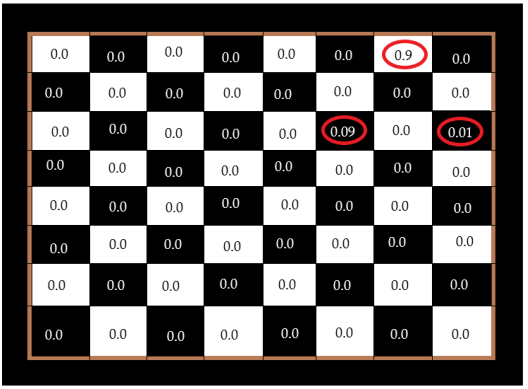

Figure 2: An enemy knight's probability grid after an opponent's move. The agent thinks the knight probably didn't move, but if the knight did move, it probably moved toward the center of the board.

After a player's own sense action, more information is known about what the opponent did than was known immediately after the opponent's turn. Many potential moves that were processed after the opponent's turn may be known to be impossible after the sense action. The agent rolls back all impossible moves.

If a square that was known to be empty now contains an enemy piece, any move not involving that piece is impossible. If the agent was sure of an enemy piece's location and that square is now empty, then any move not involving that piece is impossible. Similarly, if the piece is still in the same location, then any move involving the piece is impossible. This allows the agent to narrow down the opponent's possible move to just a small set of viable candidates, or possibly even to identify the move exactly. After rolling back any moves that are deemed impossible, the probabilities of the remaining viable candidates are proportionally increased and updated.

The reader may wonder why there is a two-step process to determining the opponent's move. After an opponent's move, why not wait until after the agent's own sense action to guess what the move was? By first processing a larger number of potential moves immediately after the opponent's action, more information is available for the agent to decide where to sense. Then, after the sense action, the information from the sense action can be used to better determine which move the opponent actually took.

After a player's own sense action, the probability grids are also updated for the 9 squares that are sensed. (This is in addition to refining the guess about the opponent's move, as described above.) For any square that contains a piece, the corresponding piece's grid is updated to 1.0 for that square and 0.0 for every other square, since the piece's location is known with 100% certainty. If there are multiples of that type of piece still on the board (e.g. if a rook is seen but it is not known which rook it is), the best guess is assumed to be the correct piece. Every piece's probability grid is zeroed out for any squares where it is known that they are not located. Whenever a probability is zeroed out, the probabilities for that piece being in other squares are proportionally adjusted upward to so that each piece's grid remains a valid probability distribution. If the agent sees an enemy piece in the sense result but thought that all enemy pieces of that type were already captured, then it considers two possibilities. First, the agent considers that perhaps it was wrong in an earlier guess about which enemy piece was captured. If that possibility is detected, a correction referred to as a "resurrection" is issued. Second, if no resurrection is possible then the agent considers that perhaps an enemy pawn was promoted.

The third time that the agent updates the game state is after the agent makes its own move. If no capture occurred, the update is simple: change the moved piece's probability grid and zero out the destination square in all other pieces' probability grids. If a capture did occur, the agent knows where the capture occurred but is not told which enemy piece was captured. In most cases, the capture happens because the agent knew where the enemy piece was and intentionally captured it. However, there is almost always at least a small chance of being wrong, and sometimes the capture is unexpected. Whenever

an enemy piece is captured, the agent makes its best guess about which piece was captured, and updates that piece's probability grid by zeroing out every square and marking it as captured. The agent also creates a list, known as the capturedList, of all other enemy pieces that had a non-zero probability of being in the square where the capture occurred.

The probability grids are merely the agent's best guesses about the game state. They are almost never completely accurate. Notably, it is possible for a grid to indicate a 1.0 probability for a particular square when the enemy piece is not actually in the square. Similarly, it is possible for an enemy piece to show up in a square even though the probability grid indicates a 0.0 probability. This can happen because not every possible board is investigated and not every possible move is accounted for. When the agent generates possible boards to consider, the actual, ground truth board might not be (and in fact probably isn't) actually generated. As a result, some moves that are possible on the actual, ground truth board might never be seen, considered, or taken into account in the probability grids. This can lead to the agent truly being caught off guard.

For example, the agent might end up capturing a piece in a square where the probability grid for every single enemy piece indicates a 0.0 probability of being in the square. In this case, the agent has no idea what piece was just captured, but the agent will try to guess anyway. If there is a pawn that is not captured and could possibly be in the given column, then the agent marks such a pawn captured. That keeps the count of defeated enemy pieces accurate. The agent also adds almost every enemy piece to the capturedList for that pawn, so that if the guess proves to be incorrect the agent can roll back and try to correct the error.

2.3 Choosing the Move

When deciding where to move, the agent generates a number of possible boards based on the probability grids. The number of boards that are generated depends on the amount of uncertainty in the probability grids and the amount of time remaining. (In a standard game, each player has a total of 15 minutes for the whole game.) For example, if the agent knows where every piece is located then only one board is considered. If the agent knows where almost every piece is located, then only a few boards are considered. If there is a greater degree of uncertainty, more boards are considered.

The agent generates a number of boards and for each board, it uses Stockfish to determine the best move on that board. After considering those boards, the result is a set of candidate moves. All of the candidate moves are then evaluated on a new set of generated boards. Stockfish gives every candidate move a numerical score on each of the new boards, and the scores are tallied.

The move that has the highest total score is chosen as the move to take.

2.4 Choosing Where to Sense

When choosing where to sense, the agent chooses the area of the board with the greatest uncertainty.

Each piece is given an uncertainty score for each square. As a first step, the uncertainty score is a number from 0 to 0.5 that indicates how certain the agent is about whether the piece is in the square. If the piece's probability grid contains a 0.0 or 1.0 for that square, then the agent is certain about whether or not the piece is in the square and the uncertainty score for that piece/square combination is 0.0. If the probability grid contains a 0.5 for the square in question, then the agent has no idea whether that piece is in the given square. That indicates a great deal of uncertainty about whether that piece is in the square, and the uncertainty score for that piece/square combination is 0.5. The uncertainty score for a piece/square combination is the absolute value of 0.5 minus the probability that the piece is in the square.

That describes the basic uncertainty score, but then several adjustments are made to the score. If the piece can attack the agent's king from the given square, then the uncertainty score is increased. If the piece pins one of the agent's pieces from the given square, then the uncertainty score is increased. If the piece cannot attack any of the agent's pieces from the given square, then the uncertainty score is greatly reduced. If the piece is a pawn that is near promotion, the uncertainty score is increased.

After calculating the uncertainty scores for every piece/square combination, an uncertainty score for each individual square is calculated. To calculate the uncertainty score for a square, the agent considers every piece in combination with that square, and the uncertainty score for the square is the maximum uncertainty score for any of those piece/square combinations.

The agent will choose to sense the square with the maximum uncertainty score, but it does not necessarily sense that square directly. Instead, it chooses the 3-by-3 block of squares that contains the chosen square while maximizing the sum of the uncertainty scores of all 9 squares that will be sensed.

3 Tournament Results

The agent described here took second place in the Fall 2019 NeurIPS RBC tournament sponsored by Johns Hopkins University Applied Physics Laboratory. There were 22 entries in the tournament, and each entrant played every other entrant 24 times (12 as black and 12 as white). The agent had a record of 11-13 against the eventual winner and 10-14 against the fifth place finisher,

but other than that it had a winning record against every other entrant, and at least 20 wins against every finisher outside the top 5.

4 Opportunities for Improvement

Some losses in the tournament came because the agent lost track of a piece. For example, in one game a knight moved forward from its starting position and aggressively went for (and reached) the agent's king. Opponent moves that were terrible chess moves but good RBC moves (like moving a knight aggressively toward the king) were less likely to be sensed by the agent. Stockfish is an engine that plays regular chess. The agent uses Stockfish to both decide its own moves and predict the opponent's moves. If the agent used an engine that was tuned specifically for RBC to decide moves and predict opponent moves, it could do better.

There are a large number of constants embedded within the agent. Tuning of those constants could improve the performance of the agent. Here are just a few of the constants that could be used for tuning purposes:

- When choosing a move, how many boards should be considered? What about when predicting an opponent's move?
- When choosing a move, how long should Stockfish think about each board? What about when predicting an opponent's move?
- When predicting an opponent's move, how many "good" moves should Stockfish return?
- When predicting an opponent's move, what percentage of the time should the agent assume the opponent is taking one of the "best" moves versus a "good" move versus all the other moves?
- When evaluating a move, bonus or penalty points are granted for capturing the opposing king, putting the opposing king in check, moving the agent's own king into check, and moving a piece away from allied pieces (so that it is harder for the opponent to sense the whole board). How many bonus points are each of these situations worth? Should the bonuses differ when moving versus predicting the opponent's moves?
- When evaluating the uncertainty of a square, a bonus is given if the king could be in check or if a piece is pinned. A penalty is given if the piece would not be able to make any attacks from the given square. How much should the bonuses and penalty be?

Reconnaissance Blind Chess is a new chess variant that has been proposed as a testbed for AI and machine learning research that requires agents to deal with uncertainty. As one of the top finishers in the first worldwide RBC tournament, the approaches taken by this agent offer a promising foundation for

future RBC agents as the research community explores approaches to dealing with uncertainty in this context.

References

[1] Jared Markowitz, Ryan W Gardner, and Ashley J Llorens. On the complexity of reconnaissance blind chess. *arXiv preprint arXiv:1811.03119*, 2018.

[2] Andrew J Newman, Casey L Richardson, Sean M Kain, Paul G Stankiewicz, Paul R Guseman, Blake A Schreurs, and Jeffrey A Dunne. Reconnaissance blind multi-chess: an experimentation platform for isr sensor fusion and resource management. In *Signal Processing, Sensor/Information Fusion, and Target Recognition XXV*, volume 9842, page 984209. International Society for Optics and Photonics, 2016.

[3] Tord Romstad, Marco Costalba, and Joona Kiiski. Stockfish chess. http://www.stockfishchess.org.