Using Partial Tablebases in Breakthrough

Andrew Isaac and Richard Lorentz(✉)

Department of Computer Science, California State University,
Northridge, CA 91330-8281, USA
andrew.isaac.37@my.csun.edu, lorentz@csun.edu

Abstract. In the game of Breakthrough the endgame is reached when there are still many pieces on the board. This means there are too many possible positions to be able to construct a reasonable endgame tablebase on the standard 8×8 board, or even on a 6×6 board. The fact that Breakthrough pieces only move forward allows us to create partial tablebases on the last n rows of each side of the board. We show how doing this enables us to create a much stronger MCTS based 6×6 player and allows us to solve positions that would otherwise be out of reach.

1 Introduction

Recently the game of Breakthrough has been attracting attention among researchers [1,4,6,7]. It has a simple set of rules, an interesting mix of strategy and tactics, and scales easily to different sizes. In this work we describe the construction of endgame tablebases for Breakthrough, focusing on boards of size 6×6.

Endgame tablebases are usually full board databases of a game at a stage when very few pieces remain and are typically constructed using retrograde analysis [8]. For example, in chess Nalimov tablebases have been used and expanded for many years now [5]. In Breakthrough, however, the endgame is reached while there are still many pieces on the board. Breakthrough is normally played on an 8×8 board and typically the endgame is reached while more than half the pieces are still on the board. With 8 or fewer pieces per player on a 64 square board there are simply too many legal positions to consider constructing a normal tablebase. In fact, even on a 6×6 board it is already impractical.

The rules of Breakthrough require that pieces always move towards the goal. Because of the forward running of pieces it is possible to recognize forced wins in positions by just looking at rows near the goal, that is, rows near the end of the board. The forward running also allows for a kind of iterative retrograde approach to constructing the tablebase. Given a tablebase containing all wins for White in the last $n-1$ rows we can use it to construct wins for White from the last n rows. Once constructed we can use the tablebases to improve the play of a Breakthrough playing program and use them to solve positions on smaller boards by simply applying the game playing program enhanced with the tablebase.

In Sect. 2 we briefly describe Breakthrough and give some examples of the endgame. Section 3 is where we describe our notion of tablebases, how we build them, and how we use them in an MCTS based Breakthrough player. In Sect. 4

A. Plaat et al. (Eds.): CG 2016, LNCS 10068, pp. 1–10, 2016.
DOI: 10.1007/978-3-319-50935-8_1

we show what we were able to accomplish using tablebases in 6×6 Breakthrough and Sect. 5 summarizes our results and points towards future directions of research.

2 The Game of Breakthrough and Its Endgame

Breakthrough is usually played on an 8×8 board but our studies will focus on the smaller 6×6 board. Each player begins with 12 pieces as shown on the left of Fig. 1. White pieces move one square at a time either diagonally or vertically to unoccupied squares and towards row 6, the goal row. White may capture a black piece if the piece is located where a diagonal legal move could be made – as a chess pawn might capture. A white piece cannot move straight forward if that square is occupied by either a white or a black piece. Black moves similarly in the other direction. The first player to have a piece reach the goal row or to capture all of the enemy pieces is the winner. White plays first.

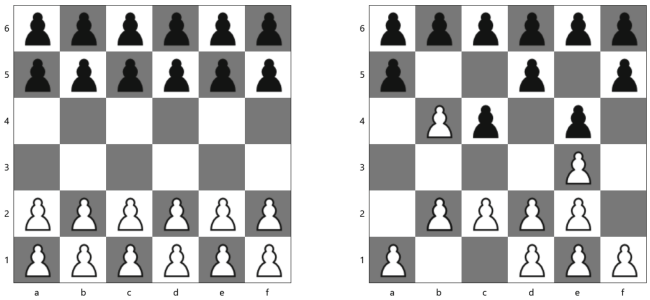

Fig. 1. Sample 6×6 breakthrough positions.

The right of Fig. 1 shows a position in the middle of a game. If it is White's turn to play, the piece on square b4 can capture the black piece on a5 or move to b5 or c5. If Black is to move, the black piece on square e4 can move to either d3 or f3 but not e3.

To get a feel for the number of endgame positions possible in Breakthrough we consider situations with six or fewer pieces for each player, with White to move, and note that we are only interested in positions where there is no white piece on either of the last two rows (else the game is over or White has a win-in-1) and there is no black piece on the first row (else Black has already won). In the case of a 6×6 board, for each placement of 1 through 6 white pieces in the first 4 rows, we place 6 black pieces on the last 5 rows on the unoccupied squares. We underestimate the true total by assuming that all white pieces occupy squares that a black piece might also occupy, even though this is not true if a black piece is on one of its first two rows. It leads us to Eq. (1) as an estimate of the number of positions possible with six or fewer pieces of each color.

$$\sum_{i=1}^{6}\left(\binom{24}{i}\times\sum_{k=1}^{6}\binom{30-i}{k}\right)\approx 4.03\times 10^{10} \tag{1}$$

Similarly, Eq. (2) estimates the number of endame positions on an 8×8 board with no more than 8 pieces of any color.

$$\sum_{i=1}^{8}\left(\binom{48}{i}\times\sum_{k=1}^{8}\binom{56-i}{k}\right)\approx 2.25\times 10^{17} \tag{2}$$

Both of the above counts include illegal positions, e.g., all of the white pieces are on the first 3 rows of column a and b. However, the number of illegal positions is quite small given how few pieces are on the board and so the estimates given above are reasonable.

3 Endgame Tablebases

Any board configuration with White to move that has a white piece on row 5 is a win-in-1 because White can simply move to the goal row. An effective win-in-3 is a board configuration where White can make a move that preserves a win-in-1 against any black defense. This either means that Black has no defense and White will win on the next move or Black can prevent the immediate win but White can follow with a move that restores it.

For example, in the position on the left of Fig. 2 White has a win-in-3 by moving 1. d4–d5. Black must continue to defend the win-in-1 by capturing the advanced piece, which White in turn recaptures until White finally lands on row 5 and is invulnerable to capture. Thus the win-in-3 actually takes a total of 7 moves.

The position in the middle of Fig. 2 shows an effective win-in-5 that actually requires 9 moves to complete. The idea is that White must clear the Black d6 piece so that White will be able to play f3–f4–e5–f6. After 1. c3 × b4 Black is forced to play 2. c5 × b4 else White wins by moving to or capturing a5. White then plays 3. c4–c5, Black is forced to capture and White wins with the f4 piece. Since Black must reply to White's first two moves, it is an effective win-in-5.

The rightmost position of Fig. 2 shows an effective win-in-9. It is a win-in-9 because two preparatory moves need to be made in the process of advancing to the goal. The main line is: 1. b3 × c4, 2. e5 × d4, 3. c3 × d4, 4. a5–a4, 5. f3–e4, 6. f5 × e4, 7. e3–f4, 8. a4–a3, 9. f4–f5, 10. e6 × f5, 11. d4–d5, 12. c6 × d5, 13. c4 × d5, 14. a3–a2, 15. d5–e6. We can see that most of the moves are either forcing moves or moves advancing towards the goal, but moves 1 and 7 are extra moves needed to prepare for the breakthrough at d5.

Using this concept of a forced win, where defensive moves by the opponent are not counted towards the move count, board configurations can be stored in a tablebase along with their effective number of moves required to win. During game play a player can consult the tablebase to determine if a forced win exists

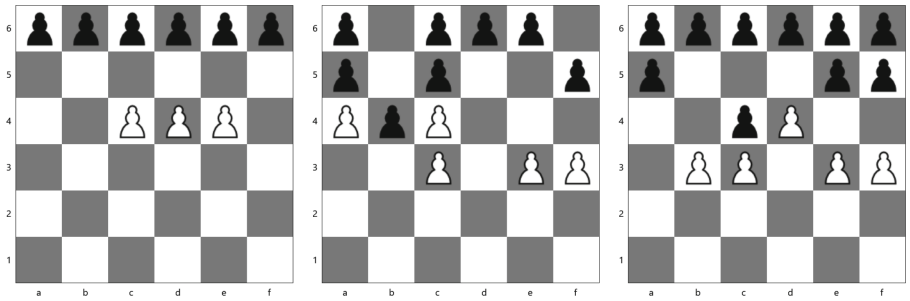

Fig. 2. Forced wins.

and in how many effective moves. By comparing any forced wins White or Black may have it is possible to determine who will win the game even with a significant portion of the game left to play.

The rules of Breakthrough require pieces to move in one direction up or down the board, so once an opponent's piece has progressed beyond a certain row, it is no longer defensively relevant to a win involving pieces in rows that it has already passed. For example, a forced win that only requires White to use pieces in the top 3 rows is certain to be an effective forced win-in-3 because every move threatens a win-in-1 and so must be responded to. A forced win that requires White to use pieces in the top 4 rows must be at least a forced win-in-5, since there will be at least one time when white moves from row 3 to row 4 and after Black answers at best a win-in-3 will have been created. In this case if Black has an effective win-in-3 moves (or less) after White makes its move, Black will have the winning position.

Any board configuration can be represented as an integer using Gödel numbering, so if a certain configuration can be identified as a forced win, the integer value of that configuration can be added to a database of forced wins, along with any other information related to the configuration, such as the effective number of moves required or the next move the player needs to make to achieve the forced win. If every forced win can be identified, the game itself would be solved. To limit the size of the tablebase of forced wins to be solely resident in RAM and to be able to construct the tablebase in a reasonable amount of time, only forced wins that involve pieces located on the last n rows, for some suitable n, are included. During game play, the current board state in those n rows can be converted to its integer value and the endgame tablebase can be consulted to determine if a forced win has been achieved.

3.1 Building an Endgame Tablebase for Breakthrough

Below we discuss the following three topics. Building an endgame tablebase for Breakthrough in Sect. 3.1; n x 6 tablebases in Sect. 3.2; using endgame tablebases in Sect. 3.3.

To construct a tablebase for 6×6 Breakthrough that is limited in size and can be quickly queried, the top 3 rows are first examined to identify any effective forced wins-in-3. This is done by iterating through all valid configurations of pieces that may exist in the top 3 rows. Assuming White to move, any white pieces in row 6 can be skipped, since that is already a win, as well as any white pieces in row 5 as that is a trivial win-in-1. Only configurations where there are some white pieces on row 4 and Black pieces on any of the top 2 rows need to be examined as possible forced wins. We start with 1 white piece and build up to more pieces. Configurations that either have White to move to row 5 and where it is invulnerable to capture, or that will eventually lead to White being able to move to row 5 and be invulnerable to capture as the result of responses by Black get added to the tablebase until all possible forced wins in 3 moves are found.

Once all forced wins only requiring pieces that belong to the top three rows are found, this process is repeated for the all possible combinations within the top 4 rows, skipping any forced wins already found in the top 3 rows from the previous step. We repeat the process similarly for the top five rows.

The order the pieces are placed goes from fewer to more white pieces and within a fixed number of white pieces from higher numbered to lower numbered rows. This allows us to fill the tablebase in a single pass.

In order for a configuration to be added to the tablebase it has to be determined if White has a forced win. For every possible white move, all possible black responses are tested. For every black response, if White has a forced win that is already in the tablebase the configuration is added to the tablebase.

To determine the total number of moves required for the forced win once it is identified it is assumed that White will choose the move that leads to the shortest previously found forced win, while Black will choose the move that leads to longest inevitable loss. In order to determine the effective number of moves of a forced win, an imaginary offensive move by Black is applied. If making the offensive move by Black results in a better forced win for White, it is assumed that Black instead will choose to make a defensive move. If the effective number of moves for a forced win is the same whether Black makes a defensive or offensive move, it is assumed that Black will use the opportunity to make an offensive move over a defensive move.

For every board configuration entered into the tablebase, the symmetric configuration is added as well. An important feature of this optimization when combined with the order that white pieces are added to the board as described above is that all forced wins get added to the tablebase in a single run. Without the addition of symmetries or by using a poor ordering of positions, forced wins may go undetected when the forced win testing program is run a single time, requiring a second run (or more) to find the remaining wins.

3.2 $n \times 6$ Tablebases

A summary of the complete tablebase that we created for 6×6 Breakthrough can be seen in Table 1. Notice that there are no wins-in-3 for the 4-row tablebase.

Table 1. Tablebase contents

Rows	3	4
Wins-in-3	227,547	0
Wins-in-5	0	19,888,955
Wins-in-7	0	642, 417
Wins-in-9	0	6,170
Wins-in-11	0	190
Wins-in-13	0	5
Wins-in-15	0	0
Total forced wins	227, 547	20,537,737
Time to generate	15 s	2.5 h

This is because it is impossible for a White piece on row 3 or lower to affect such a fast win.

In contrast, it is possible to have a win-in-5 that involves row 2 or 1. Consider the first diagram in Fig. 3. After White has moved d2–d3 Black is in zugwang and once Black moves White has a straightforward short win. It is because of the zugzwang that White is suddenly presented with a shorter win than was available before Black moved.

The right side of Fig. 3 shows one of the longest wins in the tablebase. Being a win-in-13 means that White will make a total of 7 moves. Three of the moves advance the piece from row 3 to row 6 and the other 4 moves aid in this advancement.

It is worth pointing out that Black actually has a shorter win in this position but the tablebase only reports wins for White. When the tablebase is actually used, after finding the win-in-13 for White it will make White's first move and query the database to see if Black indeed has a faster win. Here is a summary of White's win-in-13 with a shorter win for Black pointed out. 1. e3–e4, (the key to the winning attack is to breakthrough using the c4 square) 2. f5 × e4, 3. f3 × e4,

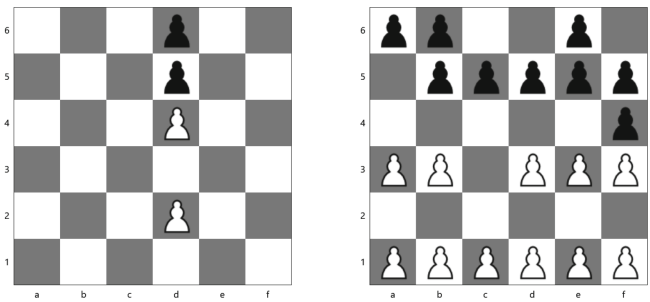

Fig. 3. 5 row win-in-5 (left) and 5 row win-in-13 (right).

4. f4–f3, 5. a3–a4, 6. c5–d4, 7. a4 × b5, 8. a6 × b5, (a forced move by Black and does not count towards the mate count) 9. b3–b4, (as far as the tablebase is concerned, Black is now in zugzwang, but in this particular board position Black can make a move that threatens a win that is shorter than White's) 10. d4–e3, 11. b4–a5, 12. b6 × a5, (forced) 13. d3–d4, and now White has an obvious win-in-3 that completes the original win-in-13, though Black has an equally obvious win-in-3 and so would actually win this game.

3.3 Using Endgame Tablebases

There are a number of natural places in an MCTS based program one might use tablebases. We have an EPT (early playout termination [3]) program named Wanderer [4] that plays the game of Breakthrough and we use it to study the various options.

In the spirit of EPT it makes sense to access the tablebases during the random playouts. The simplest approach would be to augment the evaluation function to test if either player has a faster forced win than the other and if so, return accordingly. In contrast, if at any time during the playout, not just at the end when the evaluation is invoked, it can be determined that a player has a forced win it would be a shame to ignore this important information. But it is not cheap to query the tablebase so the multiple queries could prove to be too expensive. As it turns out the advantage of immediately recognizing the forced win overshadows the cost of the queries and so checking the tablebase after every move in the playout is to be preferred. This is not entirely surprising because in a similar way with both Havannah and Breakthrough we have shown it is preferable to detect a win-in-1 and make that move during a playout rather than just make a random move [2,4].

The other place to consider querying the tablebase is during tree expansion. When expanding a node in the MCTS tree, if the tablebase shows that a child node allows a forced win by the opponent there is no need to add the node to the tree as a player will never select a move that allows the opponent to win. A simplified version of this is already done in Wanderer in order to aid the MCTS solver: if a position is found that allows a win-in-1 by the opponent (or an easy to see win-in-3) then that child is not added to the tree.

But querying the tablebase at every node creation can be very slow. Our tests show that given a fixed amount of time the size of the tree we are able to create when doing these queries is less than half the size than when we do not make the queries. Further, the tests show that this is too high a price to pay and the version of Wanderer without the queries easily outperforms the one with them under normal time constraints (see Table 2 in Sect. 4).

It is possible that a forced win for White requiring pieces in the top 5 rows can actually be a shorter effective win than a forced win for White only requiring pieces in the top 4 rows, and likewise an effective forced win requiring all 6 rows can be a shorter forced win than a forced win requiring fewer rows. Since the tablebase only holds entries for all forced wins located 5 rows or less from the winning row, it is possible that a forced win requiring all 6 rows, which is not

contained within the tablebase due to size and time limitations, could be a better effective forced win. The best forced win requiring pieces located 6 rows from winning would be at least an effective win in nine moves, so only effective forced wins in the tablebase that are nine moves or less are considered to be reliable forced wins when the tablebase is queried.

We are also interested in solving Breakthrough positions with an ultimate goal of solving 6×6 Breakthrough. So far the best results along these lines are by [6] where they solve 5×6 Breakthrough. When trying to solve positions the faster we can eliminate nodes, the better. In this case it is more important to prove and remove nodes quickly than to find promising moves. Our tests show that by querying the tablebase at the time of node creation and not adding losing nodes we can solve positions eight or more times faster and use comparably less RAM.

4 Results

In order to gauge any change in performance we ran a number of tests that are summarized in Table 2. TB$n$rows denotes a version of Breakthrough using a tablebase that checks n or fewer rows and noTB represents a version of Breakthrough using no tablebases at all. TBtree denotes a version of Wanderer that queries the 4-row tablebase both in the random playouts and when adding nodes to the tree. None of the other versions check during node expansion.

Tests showed that versions of Wanderer that use the tablebase performed considerably better than the version that did not. Additionally we found that, despite the slowdown required to make additional queries to the tablebase, Wanderer performed best when making use of the 4 or 5-row tablebase, while the 5-row tablebase did not seem to show significant game playing improvement over the 4-row tablebase.[1] Test #7 shows that querying the tablebase when building the tree is too expensive for real-time play.

When trying to solve positions, however, querying the tablebase when building the tree is fruitful. In order to aid in the solving, we made some plain modifications to Wanderer (mainly in terms of its evaluation function) and found that adding tablebase queries as the tree is being built reduces the solving times by at least a factor of six. We have been able to solve every position we have tried 6 moves from the beginning of the game, most positions 5 moves from the start of the game, and many from 4 moves. We have not found any positions after just 3 moves that we can solve, indicating we are still some distance from actually solving 6×6 Breakthrough. Our hope is to be able to solve 6×6 Breakthrough by simply using existing tools of tablebases and Wanderer.

[1] Time and space requirements prevented us from creating a complete 5-row tablebase so we created an abbreviated version where we further restricted the number of pieces on the board.

Table 2. Test results with 6×6 breakthrough

Test #	Player A	Player B	Player A white	Player B white	Player A vs. player B
1	noTB	noTB	124–76	127–73	197–203
2	TB3rows	noTB	157–43	65–135	292–108
3	TB4rows	noTB	173–27	38–162	335–65
4	TB5rows	noTB	182–18	35–165	347–53
5	TB4rows	TB3rows	175–25	77–123	298–102
6	TB5rows	TB4rows	132–68	127–73	205–195
7	TB4rows	TBtree	159–41	96–104	263–137

5 Conclusions and Future Work

We have shown that incomplete tablebases can significantly improve the performance of a 6×6 Breakthrough playing program. Even a straightforward 3-row tablebase provides significant improvement and adding a 4th row improves quite a bit more. We were unable to complete a full 5-row tablebase, but we were still a bit surprised that adding the partial 5th row did not improve the program. This point requires further research.

Ultimately we want to extend these ideas to 8×8 Breakthrough and improve Wanderer, our competitive program. The increase in size from 6×6 to 8×8 provides some difficulties. We can produce a 3-row tablebase and expect some improvement from that, but a complete 4-row tablebase is just too big. As a result, we plan to create a 4×6 tablebase and query that on both sides of the 8×8 board.

Finally, though we are still some distance from solving 6×6 Breakthrough we conjecture that it is a win for White since all reasonable positions that we have solved after 4 or 5 opening moves have proven to be wins for White. This goes against the opinion of most Breakthrough players because (1) 5×5 Breakthrough is a win for Black and (2) there are a number of common positions reached in competitive 8×8 Breakthrough games where Black has the forced win.

References

1. Finnsson, H., Björnsson, Y.: Game-tree properties and MCTS performance. In: IJCAI Workshop on General Game Playing (GIGA11) (2011)
2. Lorentz, R.: Experiments with Monte-Carlo tree search in the game of Havannah. ICGA J. **34**(3), 12–21 (2011)
3. Lorentz, R.: Early playout termination in MCTS. In: Plaat, A., Herik, J., Kosters, W. (eds.) ACG 2015. LNCS, vol. 9525, pp. 12–19. Springer, Heidelberg (2015). doi:10.1007/978-3-319-27992-3_2
4. Lorentz, R., Horey, T.: Programming breakthrough. In: Herik, H.J., Iida, H., Plaat, A. (eds.) CG 2013. LNCS, vol. 8427, pp. 49–59. Springer, Heidelberg (2014). doi:10.1007/978-3-319-09165-5_5

5. Nalimov, E.V., Haworth, G.M., Heinz, E.A.: Experiments with Monte-Carlo tree search in the game of Havannah. ICGA J. **23**(3), 148–162 (2000)
6. Saffidine, A., Jouandeau, N., Cazenave, T.: Solving breakthrough with race patterns and job-level proof number search. In: Herik, H.J., Plaat, A. (eds.) ACG 2011. LNCS, vol. 7168, pp. 196–207. Springer, Heidelberg (2012). doi:10.1007/978-3-642-31866-5_17
7. Skowronski, P., Björnsson, Y., Winands, M.H.M.: Automated discovery of search-extension features. In: Herik, H.J., Spronck, P. (eds.) ACG 2009. LNCS, vol. 6048, pp. 182–194. Springer, Heidelberg (2010). doi:10.1007/978-3-642-12993-3_17
8. Thompson, K.: Retrograde analysis of certain endgames. ICGA J. **9**(3), 131–139 (1986)