Complexity of Retrograde and Helpmate Chess Problems: Even Cooperative Chess is Hard

Josh Brunner
Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, MA, USA

Erik D. Demaine
Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, MA, USA

Dylan Hendrickson
Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, MA, USA

Julian Wellman
Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, MA, USA

Abstract

We prove PSPACE-completeness of two classic types of Chess problems when generalized to $n \times n$ boards. A "retrograde" problem asks whether it is possible for a position to be reached from a natural starting position, i.e., whether the position is "valid" or "legal" or "reachable". Most real-world retrograde Chess problems ask for the last few moves of such a sequence; we analyze the decision question which gets at the existence of an exponentially long move sequence. A "helpmate" problem asks whether it is possible for a player to become checkmated by any sequence of moves from a given position. A helpmate problem is essentially a cooperative form of Chess, where both players work together to cause a particular player to win; it also arises in regular Chess games, where a player who runs out of time (flags) loses only if they could ever possibly be checkmated from the current position (i.e., the helpmate problem has a solution). Our PSPACE-hardness reductions are from a variant of a puzzle game called Subway Shuffle.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases hardness, board games, PSPACE

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2020.33

1 Introduction

Chess problems [9, 11, 12, 14] are puzzles involving Chess boards/pieces/positions, often used as exercises to learn how to play Chess better. Perhaps the most common family of Chess problems are of the form ***mate-in-k***: is it possible to force a win within k moves from the given game position (board state and who moves next)? While this problem can be solved in polynomial time for $k = O(1)$, it is PSPACE-complete if k is polynomial in the board size n [13] and EXPTIME-complete if k is exponential in the board size (or infinite) [5].

In this paper, we analyze the complexity of two popular families of Chess problems that are fundamentally ***cooperative***: they ask whether gameplay could possibly produce a given result, which is equivalent to the two players (Black and White) cooperating to achieve the goal. This cooperative means the two players effectively act as a single player (in the sense that quantifiers no longer alternate), placing the problem in PSPACE (see Lemma 1.1). We prove that the following two problems are in fact PSPACE-complete.

First, ***retrograde Chess problems*** ask about the moves leading up to a given position. For example, Figure 1 gives the puzzle on the cover of Raymond Smullyan's classic book *The Chess Mysteries of Sherlock Holmes* [11]. Other classic books with Chess problems (and descriptions of how to do retrograde analysis) are by Nunn [9] and Smullyan [12]. Many retrograde Chess problems (including Figure 1) ask what the final few moves of the two players must have been to reach this position. Fundamentally, these problems are about

31st International Symposium on Algorithms and Computation (ISAAC 2020).
Editors: Yixin Cao, Siu-Wing Cheng, and Minming Li; Article No. 33; pp. 33:1–33:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

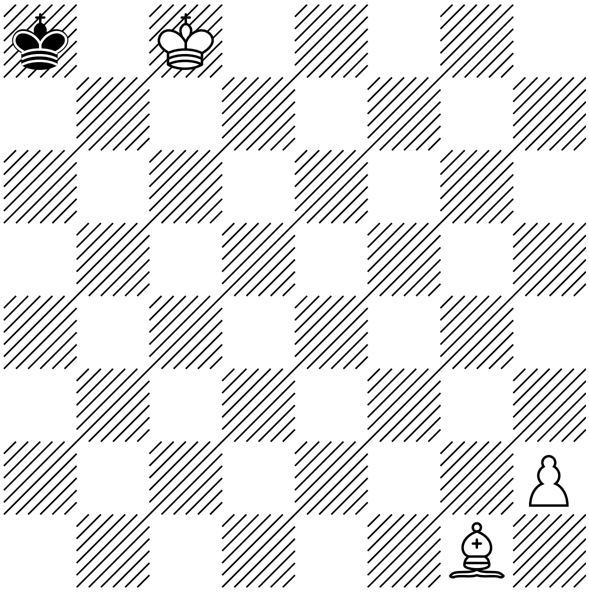


Figure 1 The retrograde Chess problem on the cover of [11]. What move did Black just make? What move did White make before that?

the ***reachability*** of the given position from the starting position, and focus on the final k moves where the puzzle is most interesting (as the moves are most forced). For exponentially large k, we find a core underlying decision problem: can the given position be reached at all from the starting position? Such positions are often called "valid" or "legal" because they are possible results of valid/legal gameplay; in Section 3, we prove that characterizing such positions is PSPACE-complete.

Second, ***helpmate Chess problems*** [8, 15] ask whether it is possible for a player to win via checkmate by any sequence of moves, i.e., when the players cooperate (help each other, hence "helpmate"). This problem could also naturally be called ***Cooperative Chess***, by analogy to Cooperative Checkers, which is NP-complete [1]. In addition to being a popular form of Chess problem, helpmate problems naturally arise in regular games of Chess, as FIDE's[1] "dead-reckoning" rule says that any position without a helpmate is automatically a draw:

> "The game is drawn when a position has arisen in which neither player can checkmate the opponent's king with any series of legal moves. The game is said to end in a 'dead position'." [4, Article 5.2.2]

In practice, this condition is often checked when a player runs out of time, in which case that player loses if and only if they could ever possibly be checkmated from the current position (i.e., the helpmate problem has a solution) [4, Article 6.9]. In Section 2, we prove that characterizing such non-dead positions is PSPACE-complete. Amusingly, this result implies that it is PSPACE-complete to decide whether a given game position is already a draw (***draw-in-0***) for Chess.

[1] The International Chess Federation (FIDE) is the governing body of international Chess competition. In particular, they organize the World Chess Championship which defines the world's best Chess player. All top-level Chess competitions (not just FIDE's) follow FIDE's rules of Chess [4].

1.1 Chess Problem Definitions

To formalize the results summarized above, we more carefully define the objects problems discussed in this paper.

A ***Chess position*** is a description of an $n \times n$ square grid, where some squares have a Chess piece (a pawn, rook, knight, bishop, queen, or king designated either black or white) and a designation of which player (black or white) plays next.

We follow the standard FIDE rules of Chess [4], naturally generalized to larger boards. In particular, there must be exactly one king of each color; colors alternate turns; a king cannot be in check after its color's turn; and rooks, bishops, and queens can move any distance (as also generalized in [13, 5]).

To define reachability for $n \times n$ boards, we define a natural ***starting position*** to be a Chess position in which all of the following conditions hold:

1. The first two ranks (rows) are filled with white pieces; the last two ranks are filled with black pieces; and the rest of the board is empty.
2. The second and second-to-last ranks contain only pawns.
3. The first and last ranks contain no pawns and exactly one king each, and sufficiently many of each of the non-pawn piece types. (The exact composition and ordering of these ranks will not affect our reduction.)

Now we can define the two decision problems studied in this paper:

▶ **Problem 1** (***Reachability***)**.** Given an $n \times n$ Chess position, is it possible to reach that position from a starting position?

▶ **Problem 2** (***Helpmate***)**.** Given an $n \times n$ Chess position, is it possible to reach a position in which the black king is checkmated?

▶ **Lemma 1.1.** *Both helpmate and reachability are in PSPACE.*

Proof. An $n \times n$ Chess position takes only polynomial (in n) space to record. A nondeterministic polynomial-space machine can guess a sequence of moves, accepting when it achieves checkmate (for helpmate) or reaches the target position (for reachability); thus both problems are in NPSPACE = PSPACE. ◀

We prove that both problems are in fact PSPACE-complete. Evidence for these problems not being in NP were first given by Shitov's examples of two legal positions that require exponentially many moves to go between [10], using long chains of bishops locked by pawns. Our constructions to show PSPACE-hardness take on a similar flavor.

1.2 Subway Shuffle

Our reductions are from a one-player puzzle game called ***Subway Shuffle***, introduced by Hearn [7, 6] in his 2006 thesis, and shown PSPACE-complete in 2015 [3]. Recently, Brunner et al. [2] introduced a variation called ***oriented Subway Shuffle*** and proved it PSPACE-complete, even with two colors, a limited vertex set, and a single unoccupied vertex.

Our reductions to show PSPACE-hardness of Chess-related problems are from a slightly modified version of this restricted form of oriented Subway Shuffle, which we will call "Subway Shuffle" for simplicity, defined as follows:

▶ **Problem 3** (***Subway Shuffle***)**.** We are given a planar directed graph with edges colored ***orange*** and ***purple***, where each vertex has degree at most three and is incident to at most two edges of each color. Each vertex except one has a ***token***, which is also colored orange or purple. One edge is marked as the ***target edge***.

A legal move is to move a token across an edge of the same color, in the direction of the edge, to an empty vertex, and then reverse the direction of the edge.

The Subway Shuffle decision problem asks whether there is any sequence of legal moves which moves a token across the target edge.

This definition differs from that in [2] only in the goal condition; in [2], the goal is to move a specified token to a specified vertex. However, their proof of PSPACE-hardness also works for our definition, where the goal is to move a token across a specified edge; by examining the win gadget in [2], it is clear that the target token can reach the target vertex exactly when a specific edge is used, so we can set that edge as the target edge.[2] Thus we have the following result:

▶ **Theorem 1.2** ([2]). *Subway Shuffle is PSPACE-complete.*

2 Helpmate Chess Problems are PSPACE-Complete

In this section, we prove that Helpmate is PSPACE-complete by reducing from Subway Shuffle.

▶ **Theorem 2.1.** *Helpmate is PSPACE-complete.*

The structure of the reduction is to use a line of pieces of one type to represent and edge in the Subway Shuffle graph, where using the edge involves moving every piece in the line one space. A vertex is represented by a square where pieces from three different edges can move to. The two colors of Subway Shuffle are represented by which piece type is present in the vertex. All of the moving pieces involved in the reduction are white; black will be given a gadget to pass their turn with. In many of the figures, we label the relevant pieces that can move in red; all of the red pieces are white in the actual Chess position.

In order to make sure that players cannot make moves outside of the reduction, all of the edge and vertex gadgets are walled in with walls of bishops and pawns that are completely stuck.

2.1 Two-Orange One-Purple Subway Shuffle

First, we show how to modify Subway Shuffle slightly to make our reduction simpler. In Subway Shuffle, some vertices have two orange edges incident while others have two purple edges incident. Rather than trying to build separate gadgets for each of this cases, we use Lemma 2.2 to have every vertex have two orange edges and one purple edge. This way we only need to build gadgets for one type of vertex.

▶ **Lemma 2.2.** *Subway Shuffle is PSPACE-complete even when every degree three vertex has exactly two orange and one purple edge incident.*

Proof. Given an instance of Subway Shuffle, every vertex with two purple incident edges can be replaced with one with two orange incident edges as shown in Figure 2. Note that while we need to transform vertices with one or two purple outgoing edges, we don't need

[2] This is the middle purple edge in the bottom row in Figure 6(a) in [2]. We also remove the target vertex (and the edge incident to it) so there is only one unoccupied vertex.

to worry about vertices with zero purple outgoing edges. This is because a purple vertex with zero purple outgoing edges can never move, so the entire vertex is stuck and can safely be ignored. It is easy to check that the set of legal moves is the almost the same in every configuration. The only difference is that in Figure 2(d), the purple token can leave the vertex twice through each of the two outgoing edges; however the second purple token that leaves doesn't allow any further moves except moving the purple token back into place. With the assumption that only one vertex is ever empty, this situation is never useful, so this replacement perfectly simulates the original vertex. ◀

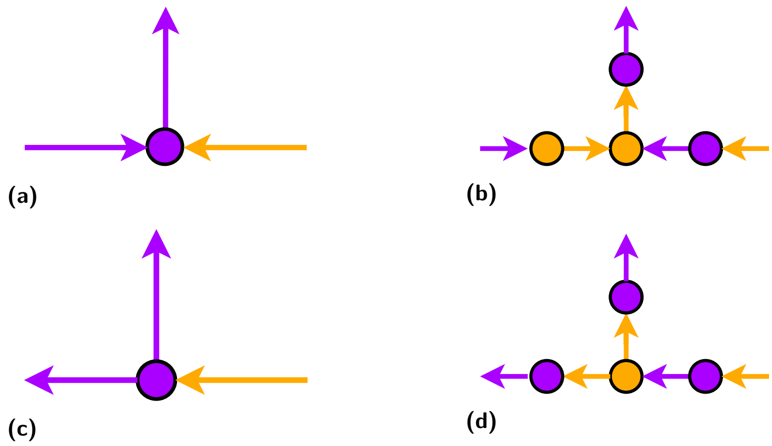


Figure 2 The color changing gadget. (a) and (b) show the transformation for vertices with one coming purple edge, and (c) and (d) show the transformation for vertices with no incoming purple edges. In both cases, the vertex behaves identically after the change, and using this technique we can give every degree-3 vertex two orange edges.

2.2 Gadgets

We start with the ***edge gadget***. To represent an edge, we simply use a line of bishops, shown in Figure 3. To move a token along the edge, move all of the bishops one space in that direction. The net effect will be a bishop entering one end and another bishop leaving the other end, representing a token moving.

Now we move on to the ***vertex gadget***. There are two cases for a Subway Shuffle vertex: a vertex which can have two edges of one color both pointing into the vertex when it is empty, and a vertex which has one edge pointing in and one pointing out of the same color when it is empty. Note that a vertex which has all of the edges of a color pointing out does not make sense because a token of that color could never reach the vertex, so those edges are provably unusable. We implement both of these with the same vertex gadget, shown in Figure 4. This gadget has three edges coming out of it from the left, right, and bottom. The left and right edges are orange, and the bottom edge is purple. Which type of vertex the gadget represents depends on which red knights are present in the middle. The empty square in the middle is the vertex square. When it contains a knight, it represents a vertex occupied with an orange token, and when it contains a rook, it represents a vertex occupied with a purple token. To use the gadget, white moves all of the red pieces one step away from the vertex along one of the edges until the vertex square becomes empty. This represents a token leaving the vertex along that edge. Then white moves one of the red pieces that can

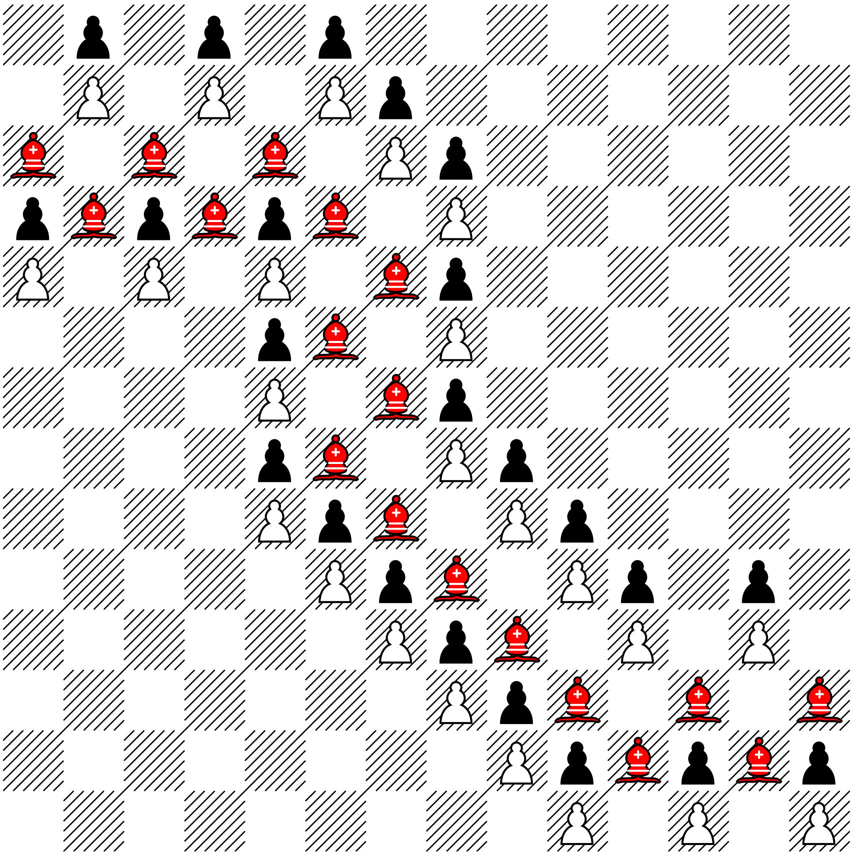


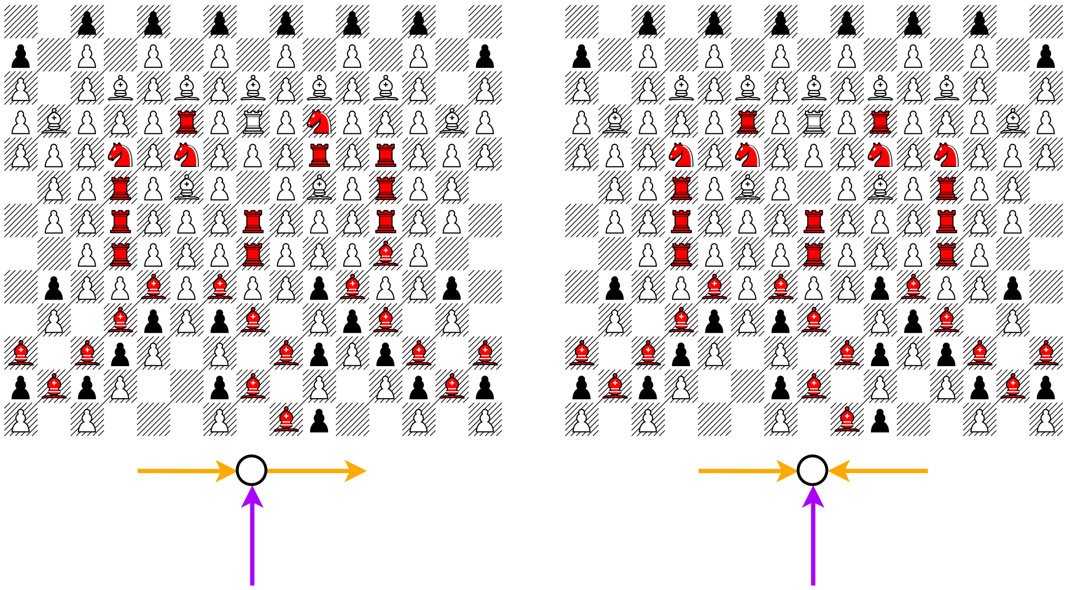
Figure 3 Our edge gadget. Since pawns care about their orientation, the gadget looks slightly different when the edge runs vertically compared to horizontally. This figure shows what both look like and how it can turn. Red represents movable white pieces.

move into the vertex square and continues moving all of the pieces along that path of red pieces; this represents moving a token into this vertex along that edge.

We use the argument about color changing from Lemma 2.2 to allow all three edges leaving the vertex to be bishop lines. The transition from the rooks in the gadget to the bishop edges is essentially this color-changing. All of the bishops are on dark squares, and our edges can be routed to connect arbitrary squares of same color, so we will not need to worry about parity issues with connecting different vertices.

Now we have to implement the Subway Shuffle target edge. This means making a ***win gadget*** which checks whether a particular edge is used. In order for an edge to be used, a piece must leave the vertex at the tail of the edge to move along that edge. Our win gadget is a modified version of the vertex gadget which allows white to checkmate if they can get a knight (representing an orange token) to leave the vertex along a specified edge. Our win gadget is depicted in Figure 5. Note that it is identical to the vertex gadget except for the replacement of one of white's pawns with a black king.

Lastly, we have a ***do-nothing gadget***. The entire puzzle is solved by white; all of the pieces that can move in the gadgets are white's and the helpmate in question is white trying to checkmate black. A legal Chess game, however, must have alternating moves by each side. Thus, we need to give black something to do. The gadget in Figure 6 accomplishes this, by giving black a trapped bishop that they can (and must) move back and forth in between



(a) An empty vertex with one orange edge pointing in and one pointing out.

(b) An empty vertex with both orange edges pointing in.

Figure 4 The vertex gadget. The two edges coming out of the sides are orange edges, and the middle edge coming out of the bottom is purple. Which of two red knights which threaten the center empty space are present determines whether the orange edges are pointing in or out.

white's moves.

It is worth noting that the positions that result from this reduction are reachable from a starting position, provided we make the board size a polynomial in the size of the Subway Shuffle instance large enough to have enough pieces and pawns to make the gadgets. Any extra pieces can capture each other prior to beginning to construct the position. We can also lock the white king away in a cage similar to the do-nothing gadget.

2.3 Correctness

Now we show that the only way white can ever checkmate black is by solving the Subway Shuffle problem and using the gadgets as they are intended to be used.

For the edge gadget, it is easy to check that no piece can move except the red bishops can move one space along the edge when it is in use.

For the vertex gadget, there is one other move white can try, but it does not do anything productive. White can try moving one of the pawns below the rook columns up one space when the rook above it moves up. This results in an immediately stuck position, so it is never useful for white to do. There are no other moves white can legally make outside of the reduction.

3 Reachability Retrograde Chess Problems are PSPACE-Complete

We reduce from the same problem, max-degree-3 two-color oriented Subway Shuffle, as in the previous section. As in the previous section, the basic structure will have white solving an instance of Subway Shuffle while black effectively passes their turn. But this time, rather than making moves, white will "undo" moves, which allows for pawns to move backwards or

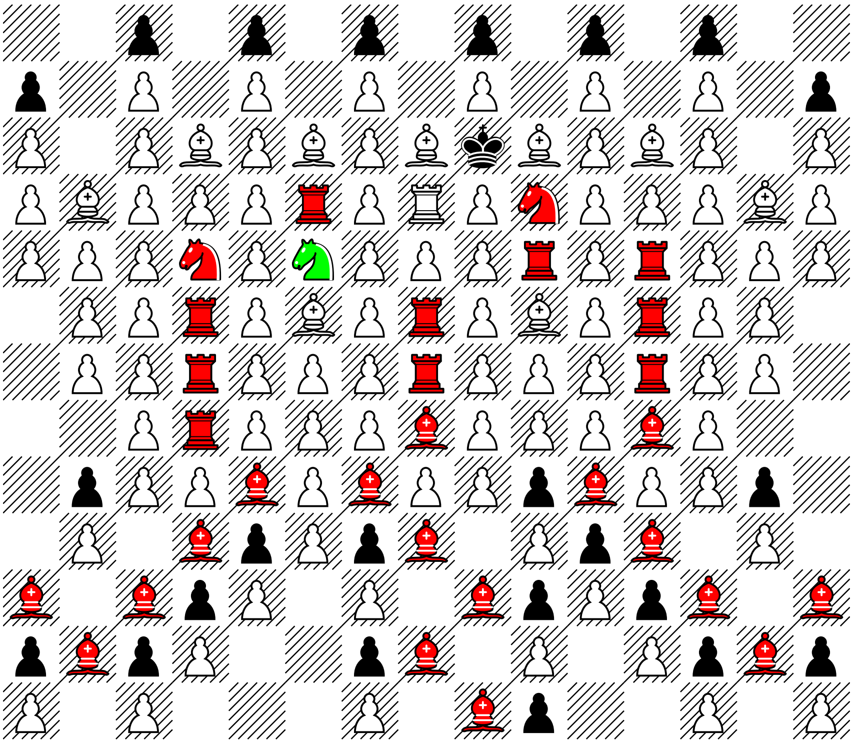


Figure 5 The win gadget. The black king is completely stuck; if it gets checked it will be immediately checkmated. The white knight highlighted in green is the only piece ever capable of accomplishing this. In order to do so, it must first move to the vertex, and then from there move to the right edge.

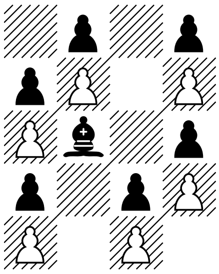


Figure 6 Do-nothing gadget, which allows black to pass forever.

pieces to be uncaptured, among other things. If white succeeds, the win gadget will allow a piece to escape from the walls of the reduction. Once this hole appears, it will let more pieces forming the walls of the gadgets to start leaving, eventually unravelling all of the gadgets. At this point once the pieces are spread out, it is easy for the players to find a sequence of moves that could get there from the starting position.

Before we describe the gadgets, we will first make some observations about how moves work in retrograde puzzles. Instead of thinking about moves that can be made from a position, we will think about moves that could have just been done; we will call these ***unmoves***. All Chess pieces other than pawns unmove the same way that they move. Pawns are different, and all captures are different as well. A piece is never captured in an unmove; to undo a capture, a piece will unmove and the captured piece appears in its place. This means that, unlike in the checkmate reduction before, we will not have to worry about pieces being capturable, so the color of non-pawn pieces in the reduction is irrelevant.

Another important distinction is that pawns do not need a piece to be able to uncapture, so any pawn can always move diagonally backward to uncapture a piece unless the space it would unmove to is occupied. Due to Corollary 3.2, this will make walling our gadgets much harder than before since every Chess piece can unmove to some square to its left and some square to its right. This means that we cannot have any isolated gadgets in the middle of the board; in order for any block of pieces to be stuck, the block must extend to both the left and right edges of the board. This results in needing an additional gadget, a terminator that we attach to the ends of the construction which anchors everything to the edge of the board.

▶ **Lemma 3.1.** *If the five nearest spaces in either file (column) immediately adjacent to a piece are empty, and the piece is not in the first two or last two ranks, then that piece can unmove into that file, leaving an empty space where it came from.*

Proof. We simply look at each piece and note that every Chess piece can unmove into a square in the immediately adjacent file. For every non-pawn piece, it simply unmoves there and leaves an empty space immediately. For a pawn, it must uncapture to do this, which it can do because it's not in the first two or last two ranks. It can uncapture a non-pawn piece, and that piece can then unmove into the empty file immediately, leaving a hole. ◀

▶ **Corollary 3.2.** *If a region of the board which does not include the first two or last two ranks has at least one piece and has an empty file adjacent to it, then a piece can unmove (possibly with multiple unmoves) to escape the region.*

Proof. Without loss of generality, let the empty file be on the left. Consider the leftmost piece in the region. Then the conditions of Lemma 3.1 are satisfied, so the piece can unmove into that file. From here the piece can continue unmoving until it leaves the region. ◀

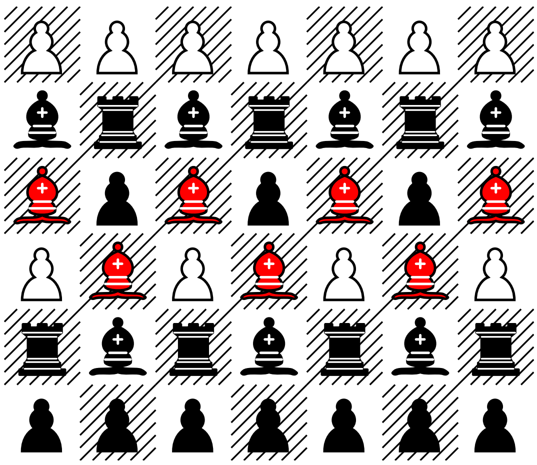
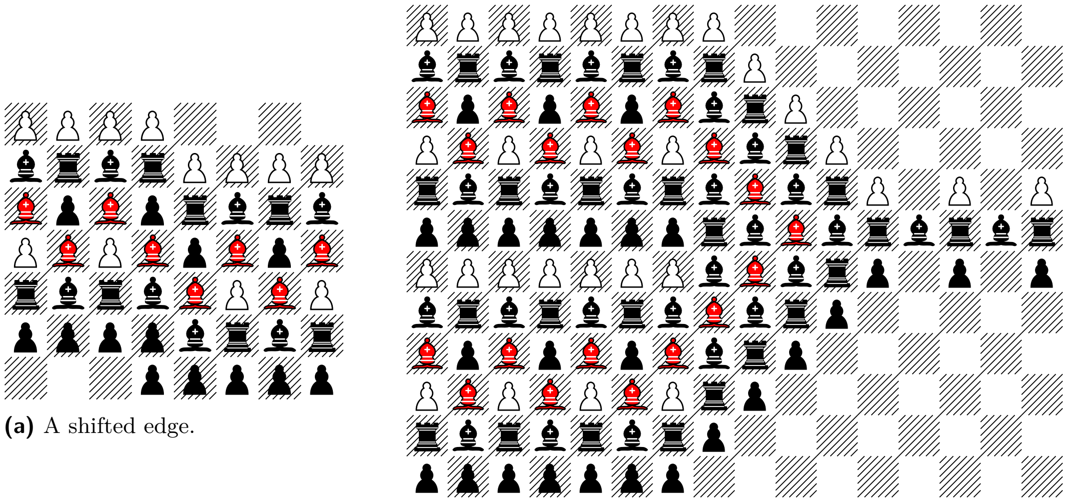


Figure 7 The edge gadget.

3.1 Gadgets

Now we describe the gadgets in our reduction.

First is the ***edge gadget*** shown in Figure 7. Like in the previous section, this gadget uses a line of bishops each of which move one space to represent the movement of a token. To keep the bishops locked in, we use a repeating pattern of pawns, rooks, and bishops across the top and bottom of the edge. Because pawns care about the orientation of the board, we cannot actually make vertical edges. Instead, we use the ***turn*** and ***shift gadgets*** shown in Figure 8; if you wiggle an edge back and forth with turns and shifts you can make it travel vertically up the board.



(a) A shifted edge.

(b) A U-turn gadget for edges.

Figure 8 Edge routing gadgets: shift and U-turn.

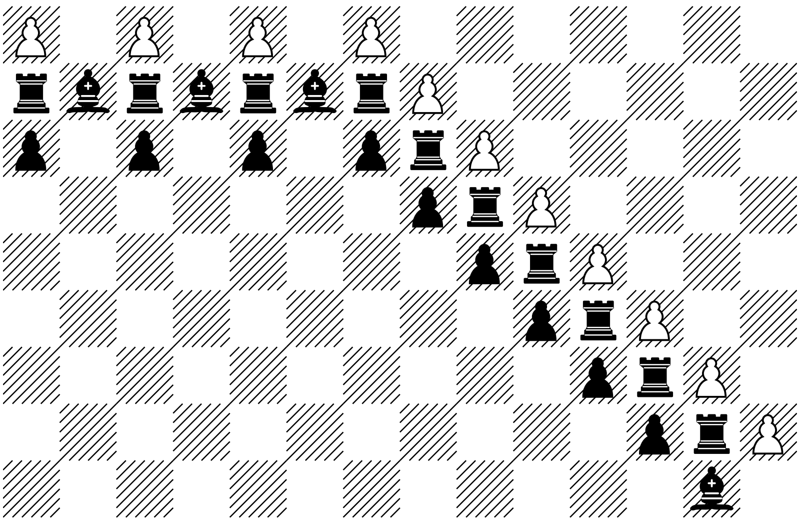


Figure 9 Terminator gadget that connects the loose ends of gadgets to the bottom rank. Unlike other figures, here we care that the first rank of this figure is the actual first rank of the Chess board. In particular, this means that the white pawn on the second rank cannot unmove, and that allows us to prove that everything is stuck.

At the edge of the turn gadget, we have the ***terminator gadget***, shown in Figure 9. As previously stated, because every piece is capable of unmoving to both adjacent files, we need a terminator gadget which connects these loose ends to the edge of the board. We have all of our terminator gadgets terminate either on the first rank of the board or on any other gadget.

The terminator gadget in Figure 9 terminates on the first rank. To have one terminate on another gadget, it simply runs (diagonally) into the wall of pawns on either side of any of our gadgets, including another terminator. We will have only a single terminator gadget on each side of the construction terminate on the first rank, and all others will terminate on another gadget.

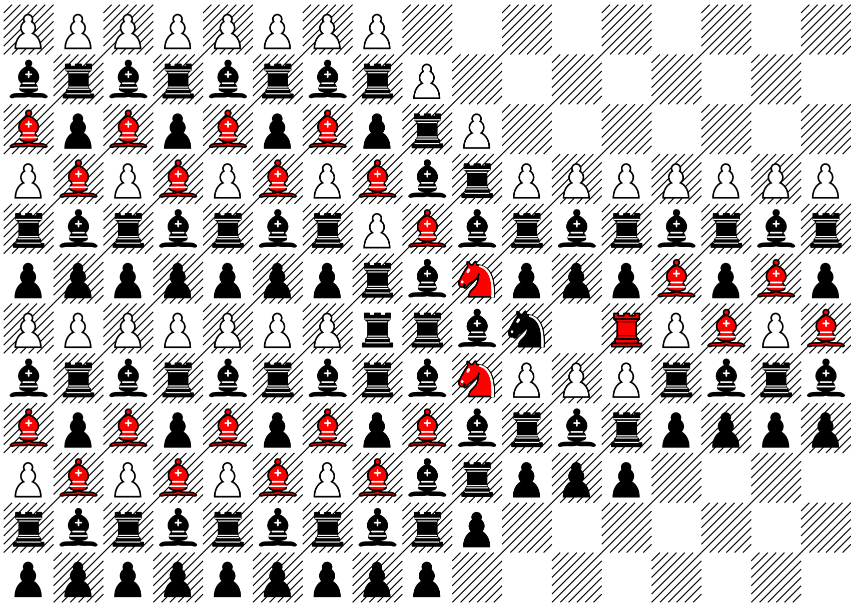


Figure 10 The vertex gadget. The empty square in the middle is the vertex; whether it is occupied by a knight or a rook determines the color of the token at this vertex.

Now we describe the ***vertex gadget***, shown in Figure 10. This vertex has a similar structure to the vertex gadget from the previous section, with potentially two knights and a rook representing the three tokens that can move in from connecting edges into the vertex. The two edges connected by a knight to the vertex are the orange edges; the rook is the purple edge. When both knights are present, we get a vertex which has both orange edges pointing into the vertex. If only one knight is present and the other is replaced by a bishop, only the edge with the knight points into the vertex and the other orange edge points out of the vertex.

3.2 Win Gadget and Self-Destruction

Finally we have the ***win gadget*** shown in Figure 11. Here we have modified the vertex gadget to add an extra hole a knight's move away from the top orange edge's connection to the vertex. When the top edge is used by having a knight enter it from the vertex, it can hop into this hole. This creates a second hole, which will be key to allowing the construction to unravel. Protruding from the top left and top right of the gadget are two terminator gadgets. The unravelling of these will be crucial to winning.

Normally, the green knight just to the right of the vertex is capable of unmoving to the vertex when it is empty. After this the second green knight can follow it unmoving into the square it just left. This then allows the purple (white) pawn to uncapture the square the knight was on. However, regardless of which piece the pawn uncaptures, the new piece is completely stuck and incapable of moving.

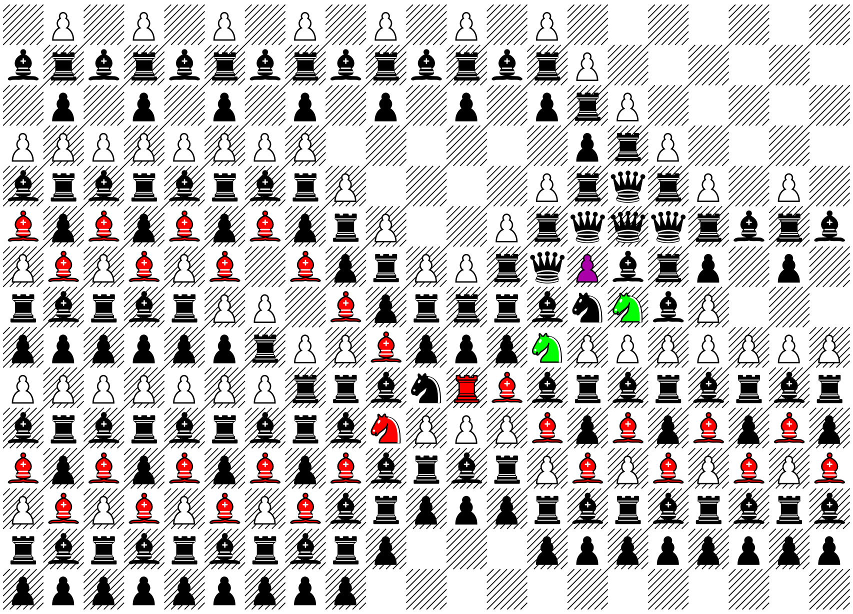


Figure 11 The win gadget. The purple pawn is a white pawn. If a knight leaves the vertex by the top edge, it allows the two white knights highlighted in green to follow it. Then the purple pawn can uncapture a knight where a knight was, which can move away, starting the unravelling process.

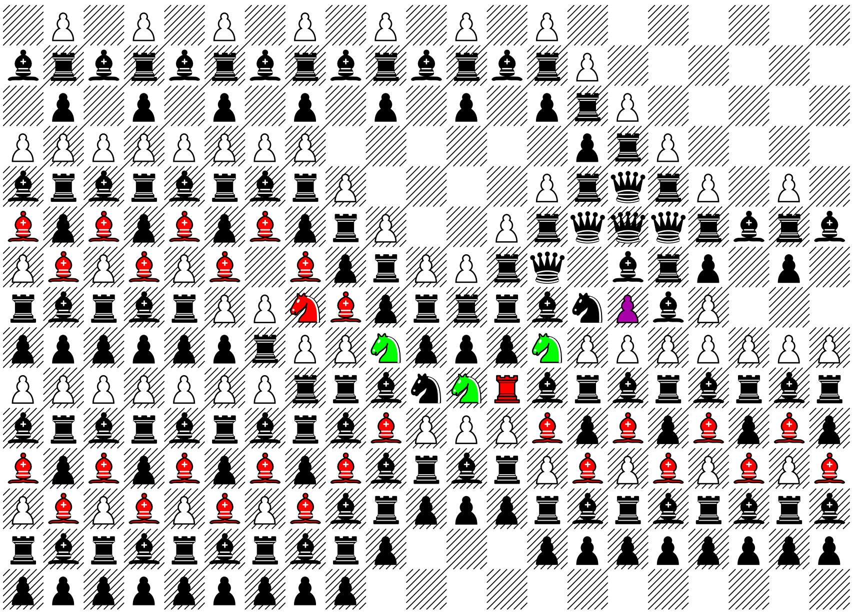


Figure 12 The win gadget after the first few unmoves to begin the unravelling are made. From here, the queens can leave allowing the terminator gadget in the top right to start unravelling.

However, if there were a second hole for the green knights to jump into, then once the green knights unmove again, the purple pawn uncaptures a knight, which can then unmove to where one of the knights was. This is shown in Figure 12. At this point, the queens and rooks can shift around to let the queens start escaping. From here, everything begins to unravel. This is where the two terminator lines come in. Once a few queens leave, both of these lines can start to unravel.

We choose a layout of the Subway Shuffle instance such that the win vertex is the furthest north vertex, and these two terminator lines are on the outside face of the graph. We extend these lines very far away from the rest of the construction, which is possible because the choice of layout implies no other part of the construction is in as high a rank as win vertex. We have any other terminator lines from U-turn gadgets which have not already terminated on another gadget terminate on these two terminator lines. Only these two terminators will eventually reach the first rank of the board, as shown in Figure 9.

Once these two lines have unravelled, it is now the case that our construction is in a region in the middle of the board with no pieces on either side of it. This means we can repeatedly apply Corollary 3.2, until every piece has left the construction. It is fairly easy once all of the pieces are free in the middle of the board to find a sequence of unmoves to send them home.

3.3 Counting Pieces

Now we need to do a piece counting argument, to show that the position is even plausible. One property of a starting Chess position is that it has only one pawn of each color in each file. Not only this, but pawns also cannot stray too far from their starting file. In particular, a pawn on rank n must come from a file at most n files away from its current position. Unfortunately, our construction can have many pawns in each file, and furthermore is constrained in how far it can be away from the bottom edge of the board due to the terminator gadgets.

However, the terminator gadget has the property that along the horizontal part, it has a white pawn (and similarly black pawn) density of only one pawn every two files. If we stretch the horizontal part far enough, and put the construction on a sufficiently far forward rank, we can use this to get the pawn density below one pawn per file. As long as this area with low pawn density is at a higher rank on the board than the number of files it is wide, with enough uncaptures the pawns can sort themselves into one pawn per file. The number of uncaptures required is at most quadratic in the number of pawns.

We also need to check the number of non-pawn pieces. To make sure that the board has the right amount of pieces, we simply have the board be much larger than our construction. Our pawns will need to make a large number of uncaptures during the unravelling, and to handle this we will have the board be much larger than the number of pieces in our construction. It is always possible to keep uncapturing additional pieces and pawns so we do not need to worry about having too large of a board.

Finally, every legal Chess position needs to have one king of each color. Since we don't use kings anywhere in the construction, and their ability to roam free doesn't allow the players to unravel the position without solving the Subway Shuffle instance, we simply put the two kings in their home positions. This also ensures that both players always have legal unmoves allowing white and black to alternate making unmoves as is required in Chess.

Acknowledgments

This work was initiated during open problem solving in the MIT class on Algorithmic Lower Bounds: Fun with Hardness Proofs (6.892) in Spring 2019. We thank the other participants of that class — in particular, John Urschel — for related discussions and providing an inspiring atmosphere.

References

1. Jeffrey Bosboom, Spencer Congero, Erik D. Demaine, Martin L. Demaine, and Jayson Lynch. Losing at Checkers is hard. In *The Mathematics of Various Entertaining Subjects (MOVES 2017)*, volume 3, pages 103–118. Princeton University Press, 2019.
2. Josh Brunner, Lily Chung, Erik D. Demaine, Dylan Hendrickson, Adam Hesterberg, Adam Suhl, and Avi Zeff. 1×1 Rush Hour with fixed blocks is PSPACE-complete. In *Proceedings of the 10th International Conference on Fun with Algorithms*, pages 7:1–7:14, La Maddalena, Italy, 2020.
3. Marzio De Biasi and Tim Ophelders. Subway Shuffle is PSPACE-complete. Manuscript, February 2015. http://www.nearly42.org/cstheory/subway-shuffle-is-pspace-complete/.
4. FIDE. FIDE Laws of Chess. https://handbook.fide.com/chapter/E012018, 2018.
5. Aviezri S. Fraenkel and David Lichtenstein. Computing a perfect strategy for $n \times n$ Chess requires time exponential in n. *Journal of Combinatorial Theory*, Series A, 31:199–214, 1981.
6. Robert A. Hearn. *Games, Puzzles, and Computation*. PhD thesis, Massachusetts Institute of Technology, 2006. URL: http://erikdemaine.org/theses/bhearn.pdf.
7. Robert A. Hearn and Erik D. Demaine. *Games, Puzzles, and Computation*. A K Peters/CRC Press, 2009.
8. David Hooper and Kenneth Whyld. *The Oxford Companion to Chess*. Oxford University Press, 2nd edition, 1992.
9. John Nunn. *Solving in Style*. Gambit Publications, 2nd edition, 2002.
10. Yaroslav Shitov. Chess God's number grows exponentially. arXiv:1409.1530, 2014. https://arxiv.org/abs/1409.1530.
11. Raymond M. Smullyan. *The Chess Mysteries of Sherlock Holmes: 50 Tantalizing Problems of Chess Detection*. Alfred A. Knopf, 1979. Reprinted by Dover, 2012.
12. Raymond M. Smullyan. *The Chess Mysteries of the Arabian Knights: 50 New Problems of Chess Detection*. Alfred A. Knopf, 1981.
13. James A. Storer. On the complexity of Chess. *Journal of Computer and System Sciences*, 27(1):77–100, 1983. URL: http://www.sciencedirect.com/science/article/pii/0022000083900302, doi:https://doi.org/10.1016/0022-0000(83)90030-2.
14. Wikipedia. Chess problems. https://en.wikipedia.org/wiki/Chess_problem, 2020.
15. Wikipedia. Helpmate. https://en.wikipedia.org/wiki/Helpmate, 2020.