Cages in Retrograde Analysis

Theodore Hwa

December 31, 2022

1 What is a cage?

Consider the "Last Move" problem below.

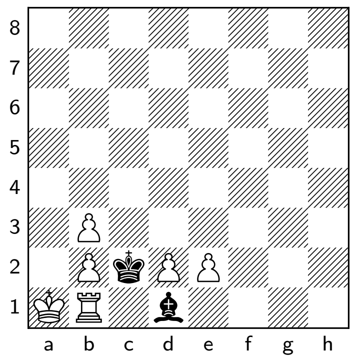

Figure 1

Brandis, Albercht
Last move?[1]

The only possibilities that are not quickly ruled out are Ka2-a1 (the correct answer) and a2xb3. Indeed, these are the possibilities reported by the software Retractor 2.[2] Why is a2xb3 more difficult to rule out? Because after retracting, say, a2xRb3 Rc3xRb3, both sides have a free piece so there is no immediate danger of retrostalemate. However, the configuration in the lower left corner (see Figure 2) will never be resolved: No matter what new units are uncaptured or how they are moved, the units in the lower left corner are stuck there. We say that Figure 2 is a **cage**.

[1]https://www.janko.at/Retros/Records/LastMove/index.htm
[2]https://github.com/hwatheod/retractor

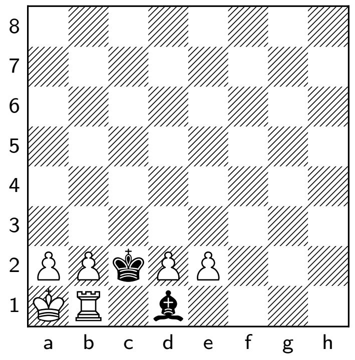
Figure 2

Position after retracting a2xb3 from Figure 1
(uncaptured unit not shown)

Definition 1 *A position is a* **cage** *if the position remains illegal in any* **extension** *of the position, i.e., no matter what other units are added to the empty squares in the position.*

Cages are an important part of automated solving of retro problems. If a position contains a known cage, then it can be immediately rejected as illegal regardless of the rest of the position. But currently, Retractor 2 can only detect a limited set of cages, and Figure 2 is not one of them, so it cannot solve Figure 1. Automated detection of cages is difficult. In this paper, we propose an alternative solution, which is an algorithm that can **verify** many (though not all) cages. A human supplies the claimed cage, and the computer verifies it. This allows a form of human+computer verification that should allay concerns expressed by Elkies that "human reasoning sometimes contain[s] holes big enough for a cook to slip through."[3]

A prototype implementation of this algorithm in Python can be found at (insert github link here).

Figure 3 shows another example of a cage.

[3]https://pairlist1.pair.net/pipermail/retros/2021-April/004918.html

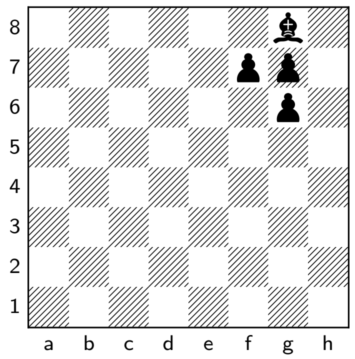

Figure 3

A cage with a perpetually mobile piece

Here, retracting h7xg6 would leave an illegally trapped bishop (outside of the Black bishop home squares c8 and f8). Otherwise, we may shuffle the bishop between g8 and h7, but this does not help. The position is still a cage because we can never return the bishop to a home square. The units on the board cannot leave the *zone of squares* g8, f7, g7, h7, g6, and they cannot reach a position within the zone where every unit is on a home square.

The general idea of the algorithm is as follows: a human supplies a position (believed to be a cage), and a zone of squares (which must contain at least the occupied squares).[4] The algorithm certifies the position as a cage if it can verify that no matter what retractions are made, there will always be at least one unit in the zone that is not on a home square.

Figure 4 shows a position which is **not** a cage, but may appear to be at first glance.

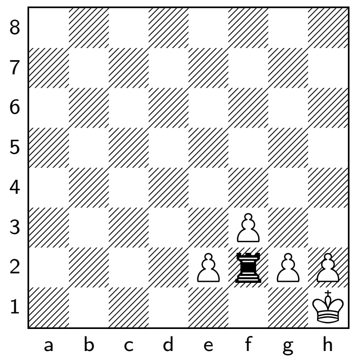

Figure 4

A position which may appear to be a cage, but is not.

[4]In all cases I have encountered in practice, it is sufficient to take the zone to be all squares which are occupied or a king's move away from an occupied square.

By adding a suitable piece at g1 (and a Black king), we obtain a legal position where the Black rook at f2 can escape from the lower right corner, thus allowing the White king to escape too.

Figure 4 illustrates an important principle. For cage analysis, we must consider all retractions that are legal in **some extension** of the position. We call this the **extension principle**. So in Figure 4, we must consider Rf2-f1 since it would become a legal retraction if a piece were added to g1. So line piece checks (Q, R, B) have to be treated a little differently than in typical retro analysis. The simple approach used below is to consider only unblockable checks from line pieces. This can cause us to search some illegal retractions but it can never miss any legal retractions. Since we only want our algorithm to guarantee illegality (not legality), this is okay.

2 Basic principles behind the algorithm

Before describing the algorithm, let us enumerate the general principles that the algorithm will follow to verify cages (with a given zone of squares). We emphasize the points which differ from usual retro-analysis of illegal positions.

1. Only unblockable checks matter, and double unblockable checks are impossible.
2. Retractions need not alternate between the two sides, except when one side retracts its king into an unblockable check, then this must immediately be followed by a retraction by the opposing checking unit.
3. When uncapturing a unit, we do not need to explicitly add the unit to the board, nor enumerate all the possibilities for the uncaptured unit.
4. If a unit leaves the zone, it is immediately removed from the board.
5. The verification fails if all units are removed from the board or if all remaining units are on home squares.

3 An algorithm for verifying a position is a cage

Now we describe the algorithm. Given a position and a zone of squares containing at least the occupied squares of the position, we perform a tree search as follows. If the algorithm returns True, then the position is verified to be a cage.

1. If the current position repeats a position on the current search branch, prune this branch by returning True and dropping back to the previous level.

2. If the current position *can be proved illegal* (see below), prune this branch by returning True and dropping back to the previous level.

3. If the maximum search depth is reached, verification has failed: terminate the search and return False.

4. If all remaining units are on home squares, verification has failed: terminate the search and return False.

5. At this point, none of the previous situations apply. Enumerate the possible retractions from the position. If the previous retraction left the retractor's king in unblockable check, then only enumerate retractions by the checking unit. Otherwise, enumerate retractions by all units of both sides.

6. If any retraction moves a unit outside of the zone, remove these units from the board. Recursively call the algorithm on the new position. If the recursive call returns False, then terminate the search and return False. If the recursive call returns True, then restore the removed units and return True.

7. Otherwise, for each retraction, perform the retraction and recursively call the algorithm on the new position. If the recursive call returns False, then terminate the search and return False. If the recursive call returns True, undo this retraction and continue with the next retraction in the list.

8. If all retractions have recursively returned True, then return True.

What positions *can be proved illegal* in step 2? The simplest cases are (a) a retraction which leaves the opposing king in unblockable check; (b) a retraction which leaves the retractor's own king in double (or higher) unblockable check. But we can increase the known illegalities by a bootstrapping process. Once the algorithm has verified a position to be a cage, then that position can be added to a set of known cages to be used for future runs of the algorithm. We could also add other logic for detecting illegalities as long as we obey the extension principle. See section 4 for more on this.

A significant optimization can be made by adding a cache. This has been done in the prototype implementation.

4 Limitations and Extensions

Once this algorithm is implemented in Retractor 2, many more "Type A" Last Move problems (at the link in the footnote for Figure 1) should be solvable by Retractor 2 compared to what is currently solves. This is based on a manual review of the currently unsolved problems. Of course, a human will have to supply the necessary cages for each problem for verification.

Some of the remaining unsolved problems are due to reasons unrelated to cages. However, there is one problem, by Darvall[5], which contains a cage that is not verified by the above algorithm. To verify that this problem has a unique solution, we must rule out the last move Ba7xNb8. In other words, we must show that the position in Figure 5 below is illegal. It is an example of a cage that cannot be verified by the algorithm as it stands.

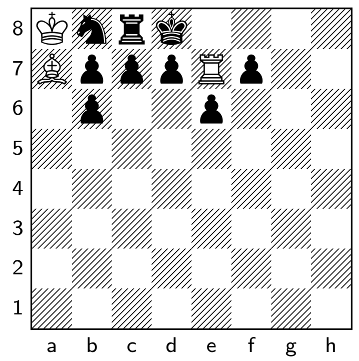

Figure 5

A cage that is not verified by the algorithm

This position is a cage because the only possible retraction is with the knight on b8, but it can't retract without uncapturing a piece. This can only be another knight, but then the new knight also can't retract without uncapturing another knight, ad infinitum.

The algorithm, as it stands, can't detect this because its treatment of checks is too simplistic. The check from c8 to a8 is blockable, but the need to block it means that the White bishop can never retract to b8, as the algorithm wants to do. Perhaps it is possible to modify this algorithm to detect this type of cage, but we have to be careful to obey the extension principle.

Another possible modification is to take into account global features of a position such as pawn captures, in order to limit the possible retractions

[5]KBrk4/1pppRp2/1p2p3/8/8/8/8/8

in a local cage. This sort of analysis is needed in a problem by Ceriani which is cited in an article by Kornilov[6] on knight evasion routes. To solve this problem, we need to establish that the following position in Figure 6 is illegal:

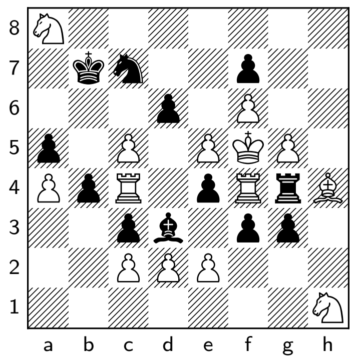

Figure 6

(13+12) Why is this position illegal?

White is missing 3 units, and Black is missing 4, from which it is easy to see that a pawn capture of the form dxe by either side is impossible. Taking this fact into account, we see that no retractions are possible in Figure 7, which is a subposition on the right side of Figure 6. In other words, Figure 7 would be a cage if a dxe pawn retraction is forbidden for both sides. The existing algorithm can easily verify it given this additional restriction.

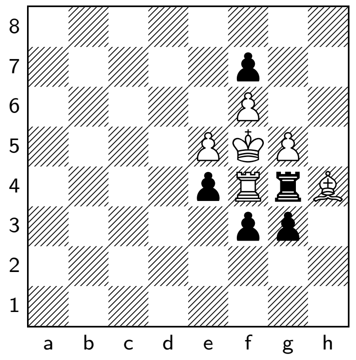

Figure 7

[6]See Position A at https://www.janko.at/Retros/KnightEvasionRoutes/index.htm or *Die Schwalbe* Heft 181, Feb. 2000.