Games of No Chance 4
MSRI Publications
Volume **63**, 2015

Retrograde analysis of Woodpush

TRISTAN CAZENAVE AND RICHARD NOWAKOWSKI

Retrograde analysis enables to solve games, starting from the end and moving toward the beginning. For games with repetition of positions and the ko rule, the standard retrograde analysis algorithm has to be adapted. We present an algorithm to solve games with the ko rule by retrograde analysis. The game of Woodpush has the ko rule and its limited state space enables its complete analysis with a modified retrograde analysis algorithm. The resulting Woodpush program plays perfect moves instantly.

1. Introduction

The game of Woodpush is a recent game, developed by the second author, that involves ko. (See [Nowakowski 2015, B2] this volume.) In this game, moves that repeat the previous position are forbidden. It is a game more simple than Go but it retains the complexity of managing ko situations. As yet there is no accepted starting position but

```
|L| |L| ... |L| | | |R| |R| ... |R| |R|
```

is a popular starting board where Left is represented with L, and Right with R. An example of a starting position for a game of width 9 is

```
|L| |L| | | |R| |R|
```

A Left move consists in sliding one of his pieces to the right as in the following example:

```
|L| |L| | | |R| |R|  =>  |L| | |L| | |R| |R|
```

If some pieces are on the way of the sliding piece, they are jumped over. For example, the Right move in the following position jumps over the Left piece contiguous to it:

```
|L| | | |L|R| | |R|  =>  |L| | |R|L| | | |R|
```

When a piece has an opponent piece behind it, it can move backward and push all the pieces behind, provided it does not repeat the previous position (i.e., creates a ko):

Keywords: none.

```
|L| | |R|L| | | |R|  =>  |L| |R|L| | | | |R|
```

For example, after this last Left move that moves backward, Right cannot move backward because it is a direct ko that repeats the previous position.

The game is won when the opponent has no more pieces on the board. Here is an example of a winning move by Right:

```
| | | | | | | |R|L|  =>  | | | | | | | | | |R|
```

The Left piece falls of the board and Right has won.

Retrograde analysis has been applied to many problems. It enables to generate databases of positions or databases of patterns. For each possible position or pattern it enables to find the status of the position and other information such as the minimal number of moves required to win in the position. Once generated, databases enable to control, reduce or even replace search.

Retrograde analysis was first used to solve chess endgames [van den Herik and Herschberg 1985; Thompson 1986; Stiller 1996; Thompson 1996] containing up to six pieces. Chess endgame databases enable to play endgames perfectly and even discovered new chess knowledge about endgames [Nunn 1993].

Another successful application of retrograde analysis is the computation of Checkers endgames by Chinook [Lake et al. 1994; Schaeffer 1997] which is an important part of the program that solved Checkers [Schaeffer 2007]. Retrograde analysis has also been used in single player games such as the 16 puzzle. It consisted in computing an admissible heuristics involving only some of the pieces [Culberson and Schaeffer 1998]. Pattern database can also be combined and improve on single pattern databases [Korf and Felner 2002]. Another application of pattern databases is Rubik's cube [Korf 1997] where separate databases for corner and side cubes can be computed and improve much the admissible heuristic. Pattern databases can also be used for the game of Go, computing for example databases on eyes or on life [Cazenave 1993; Cazenave 1996b; Cazenave 1996a]. Improvements include associating patterns to abstract conditions such as external liberties [Cazenave 2001] and reducing memory requirements using metarules [Cazenave 2003].

Some complex games such as Awari have been completely solved with retrograde analysis [Romein and Bal 2003].

In his thesis [Fraser 2002], Bill Fraser describes the BruteForce program that searches an endgame region in Go to calculate thermographs for every position. It enables his program to find means, temperatures, and orthodox lines of play. Our work is related since we use a brute force approach that takes ko into account, however we simply compute the values of positions and not the associated thermograph. Moreover we deal with long loops in the game graph, long loops only very rarely occur in Go positions.

Section 2 presents the retrograde analysis algorithms we have tested. Section 3 details experimental results. Section 4 concludes.

2. Retrograde analysis algorithms

In this section, the different algorithms we have tested are presented. The standard Retrograde analysis is first detailed. Then the adaptation of the algorithm to manage direct ko is explained. Retrograde analysis that scores positions with loops is then presented in order to evaluate positions that involve loops in the state space. Eventually, the combination of retrograde analysis with direct ko and of retrograde analysis with avoidance of loop enables to solve the game. In order to produce correct scores, retrograde analysis is used again at the end of this process.

2.1. ***Standard retrograde analysis.*** At the start of the algorithm the database is initialized with the scores of terminal positions (i.e., positions where only one player has pieces). Other positions are marked as unknown. The score of a terminal position is positive if Left has won and equals the number of moves Left can play in a row, it is the opposite of the number of moves in a row Right can play if Right has won the position.

The retrograde analysis algorithm uses two tables, one table for the positions where Left is to move and one table for the positions where Right is to move. Each entry of the table is an integer that contains the score of the position.

The algorithm iteratively calls two functions one after another. The first function finds new values for Left given the values for Right. The second function finds new values for Right given the values for Left. When no new values are found, the algorithm terminates. The pseudocode for the function that finds new values for Left is given in Algorithm 1.

Algorithm 1. Step for Left of the retrograde analysis algorithm.

```
stepForLeft ()
for position in all possible positions do
  if value for Left to play of position is unknown then
    if no children of position is unknown for Right then
      value for Left to play of position ← value for Right to play of the best
      child of position
    end if
  end if
end for
```

A problem with this standard algorithm is that it does not take the possible ko status of a position into account. A position can either have a move that is forbidden by ko or the same move allowed because there is no ko. Therefore the same position can have different values according to its associated ko moves. We show in the next subsection how we solve this problem.

2.2. ***Retrograde analysis with direct ko.*** To take ko into account, every possible position is associated to a set of possibly forbidden moves. The evaluation of a position now takes its ko status into account in algorithm 2. This algorithm searches through all possible positions and all possible ko moves in these positions. It does not consider the child with the selected ko move to compute the value of a position associated to a ko move. Note that the children of *position* for Left have an associated ko move for Right and that this ko move is used to find the relevant value for Right of each child.

Algorithm 2. Step for Left of the retrograde analysis algorithm with ko moves.

```
stepKoForLeft ()
for position in all possible positions do
    for ko in all possible ko moves of position do
        if value for Left to play of position with the ko move forbidden is unknown
        then
            let children be the children of position except the child for ko
            if no position in children is unknown for Right to play then
                value for Left to play of position with ko ← value for Right to play
                of the best child in children
            end if
        end if
    end for
end for
```

This algorithm correctly makes the difference between the different ko moves of a position. However it does only score a small subset of all positions. This is due to loops in the state space. If a position is inside a loop, it cannot be scored since a position is scored only if all its descendants are scored. Moreover the child in the loop cannot be scored since the position to score is also one of its descendant and is not scored. We give in the next subsection an algorithm that enables these positions to be scored.

2.3. ***Retrograde analysis with avoidance of loops.*** In order to evaluate positions that are in a loop, we adopt the following policy: if there is a winning move for the color to play, and if another move leads to a position in a loop, the policy decides to ignore the loopy move and to evaluate the position with the value of the best winning move. This is described in Algorithm 3.

Algorithm 3. Step for Left of the retrograde analysis algorithm with ko moves and avoidance of loops.

```
stepKoForLeftAvoidLoop ()
for position in all possible positions do
  for ko in all possible ko moves of position do
    if value for Left to play of position with the ko move forbidden is unknown
    then
      let children be the children of position except the child for ko
      if at least one child in children is won for Left then
        value for Left to play of position with ko ← value for Right to play
        of the best child in children
      end if
    end if
  end for
end for
```

This algorithm is only to be applied once the retrograde analysis with direct ko is finished and cannot find new positions. When it is the case, all unknown positions are either in a loop or have a descendant that is in a loop. The policy that avoids loops is necessary to find new values.

However applying the function that avoids loops until no new values are found is not enough. Because these new values enable new losing positions to be evaluated for the player to play. Therefore new passes of the direct ko algorithms are needed to evaluate these positions.

The function that avoids loops is not guaranteed to find the exact values for the positions, but as it only scores positions that are won for the color to play, it cannot evaluate a position as lost if it is won. The score may be underestimated but the sign of the score is correct.

2.4. ***Interleaved retrograde analysis.*** In order to evaluate the maximum number of positions, retrograde analysis with direct ko and retrograde analysis with avoidance of loops are interleaved and consecutively called until they are both unable to find new values. When it is the case, the remaining unknown positions are considered to lead either to a loop or to a loss. Instead of losing both colors will choose to loop, making these positions a draw. The algorithm that interleaves retrograde analysis is given in Algorithm 4.

Algorithm 4. Interleaving retrograde analysis with ko moves and with avoidance of loops.

```
interleavedAnalysis ()
score all terminal positions
while new positions with ko or with avoidance of loops are scored do
    while new positions with ko are scored do
        stepKoForLeft ()
        stepKoForRight ()
    end while
    while new positions with avoidance of loops are scored do
        stepKoForLeftAvoidLoop ()
        stepKoForRightAvoidLoop ()
    end while
end while
score unknown positions as draw
```

2.5. ***Retrograde analysis again.*** When interleaved retrograde analysis is finished, all positions have the correct sign but not always the correct score. In order to find the correct scores, another algorithm is required which reevaluates non terminal positions according to the scores of their children. The program alternates Left and Right passes until there is no more changes.

3. Experimental results

For size 7, retrograde analysis uses 128 passes. All positions are scored. Right, the second player, wins by four. Here is the winning line with perfect play from both players with succeeding positions written one after another:

```
|L| |L| |R| |R|
| |L|L| |R| |R|
| |L|L|R| | |R|
| | |L|R|L| |R|
| | |L|R|L|R| |
| |L|R|L| |R| |
| |L|R|L|R| | |
|L|R|L| |R| | |
|L|R|L|R| | | |
| |R|L|R|L| | |
| | |R|L|R|L| |
| |R|L| |R|L| |
| |R|L| | |R|L|
|R|L| | | |R|L|
|R|L| | | | |R|
|L| | | | | |R|
|L| | | | |R| |
| |L| | | |R| |
| |L| | |R| | |
| | |L| |R| | |
| | |L|R| | | |
| | | |R|L| | |
| | | | |R|L| |
| | | | |R| |L|
| | | |R| | |L|
| | | |R| | | |
```

For size 9, our algorithm uses 366 passes. 933 positions cannot be scored by the algorithm. They are given value 0 which means a draw. The initial position, scored with the rule of direct ko, is a draw due to a superko (repetition of a position after more than two moves):

```
|L| |L| | | |R| |R|
|L| | |L| | |R| |R|
|L| | |L| |R| | |R|
|L| | | |L|R| | |R|
|L| | | |L|R| |R| |
| |L| | |L|R| |R| |
| |L| | |L|R|R| | |
| | |L| |L|R|R| | |
| | |L|R|L|R| | | |
| |L|R|L| |R| | | |
| |L| |R|L|R| | | |
| | |L|R|L|R| | | |
| | |L| |R|L|R| | |
| | | |L|R|L|R| | |
| | | |L| |R|L|R| |
| | | |L|R|L| |R| |
| | | |L|R|L|R| | |
| | |L|R|L| |R| | |
| | |L| |R|L|R| | |
| | | |L|R|L|R| | |
```

For size 11, our algorithm uses 860 passes. 8,563 positions are scored 0 by the algorithm. The initial position is also a draw:

```
|L| |L| | | | | |R| |R|
|L| | |L| | | | |R| |R|
|L| | |L| | | |R| | |R|
|L| | | |L| | |R| | |R|
|L| | | |L| |R| | | |R|
|L| | | | |L|R| | | |R|
|L| | | | |L|R| | |R| |
| |L| | | |L|R| | |R| |
| |L| | | |L|R| |R| | |
| | |L| | |L|R| |R| | |
| | |L| | |L|R|R| | | |
| | | |L| |L|R|R| | | |
| | | |L|R|L|R| | | | |
| | |L|R|L| |R| | | | |
| | |L| |R|L|R| | | | |
| | | |L|R|L|R| | | | |
| | | |L| |R|L|R| | | |
| | | | |L|R|L|R| | | |
| | | | |L| |R|L|R| | |
| | | | |L|R|L| |R| | |
| | | | |L|R|L|R| | | |
| | | |L|R|L| |R| | | |
| | | |L| |R|L|R| | | |
| | | | |L|R|L|R| | | |
```

4. Conclusion

An algorithm that evaluates Woodpush positions using retrograde analysis has been presented. It correctly evaluates the score of positions and scores positions that lead to a superko as a draw. The resulting program plays perfect moves instantaneously.

For size 7 Woodpush, the game is a loss by 4 for the first player. For size 9 and 11 the game is a draw with the direct ko rule due to a repetition of positions.

An extension of our work is to extend our results to games with superko. Either using superko together with retrograde analysis, or using the direct ko database to solve superko with search.

Retrograde analysis algorithms that undo moves instead of browsing all possible positions are usually more rapid. It is certainly possible to adapt our algorithm in this way even though speed of the retrograde analysis algorithm is not a matter for the sizes that were solved.

References

[Cazenave 1993] T. Cazenave, "Apprentissage de le résolution de problèmes de vie et de mort au jeu de Go", Rapport de DEA, Université Paris 6, 1993.

[Cazenave 1996a] T. Cazenave, "Automatic acquisition of tactical Go rules", pp. 10–19 in *Game Programming Workshop in Japan '96* (Hakone, Kanagawa, Japan), edited by H. Matsubara, Computer Shogi Association, Kanagawa, Japan, 1996.

[Cazenave 1996b] T. Cazenave, *Système d'apprentissage par auto-observation: Application au jeu de Go*, Ph.D. thesis, Université Paris 6, 1996, http://www.lamsade.dauphine.fr/~cazenave/papers/these.pdf.

[Cazenave 2001] T. Cazenave, "Generation of patterns with external conditions for the game of Go", pp. 275–293 in *Advance in Computer Games 9*, edited by H. van den Herik and B. Monien, Universiteit Maastricht, Maastricht, 2001.

[Cazenave 2003] T. Cazenave, "Metarules to improve tactical Go knowledge", *Inform. Sci.* **154**:3-4 (2003), 173–188.

[Culberson and Schaeffer 1998] J. C. Culberson and J. Schaeffer, "Pattern databases", *Comput. Intelligence* **14**:3 (1998), 318–334.

[Fraser 2002] W. E. Fraser, *Computer-assisted thermographic analysis of go endgames*, Ph.D. thesis, University of California, Berkeley, 2002, http://search.proquest.com/docview/304802211.

[van den Herik and Herschberg 1985] H. van den Herik and I. S. Herschberg, "The construction of an omniscient endgame database", *ICCA Journal* **8**:2 (1985), 6–87.

[Korf 1997] R. E. Korf, "Finding optimal solutions to Rubik's Cube using pattern databases", pp. 700–705 in *AAAI-97* (Providence, RI), AAAI Press, Menlo Park, CA, 1997.

[Korf and Felner 2002] R. E. Korf and A. Felner, "Disjoint pattern database heuristics", *Artificial Intelligence* **134**:1-2 (2002), 9–22.

[Lake et al. 1994] R. Lake, J. Schaeffer, and P. Lu, "Solving large retrograde-analysis problems using a network of workstations", pp. 135–162 in *Advances in computer chess*, vol. 7, edited by H. J. van den Herik et al., University of Limburg, Maastricht, 1994.

[Nowakowski 2015] R. Nowakowski, "Unsolved problems in combinatorial game theory", in *Games of no chance 4*, edited by R. Nowakowski, Mathematical Sciences Research Institute Publications **63**, Cambridge University Press, New York, 2015.

[Nunn 1993] J. Nunn, "Extracting information from endgame databases", *ICCA Journal* **16**:4 (1993), 191–200.

[Romein and Bal 2003] J. W. Romein and H. E. Bal, "Solving awari with parallel retrograde analysis", *IEEE Computer* **36**:10 (2003), 26–33.

[Schaeffer 1997] J. Schaeffer, *One jump ahead: Challenging human supremacy at checkers*, Springer, New York, 1997.

[Schaeffer 2007] J. Schaeffer, "Game over: Black to play and draw in checkers", *ICGA Journal* **30**:4 (2007), 187–197.

[Stiller 1996] L. Stiller, "Multilinear algebra and chess endgames", pp. 151–192 in *Games of no chance* (Berkeley, CA, 1994), edited by R. J. Nowakowski, Math. Sci. Res. Inst. Publ. **29**, Cambridge Univ. Press, 1996.

[Thompson 1986] K. Thompson, “Retrograde analysis of certain endgames”, *ICCA Journal* **9**:3 (1986), 131–139.

[Thompson 1996] K. Thompson, “6-piece endgames”, *ICCA Journal* **19**:4 (1996), 215–226.

cazenave@lamsade.dauphine.fr *LAMSADE, Université Paris-Dauphine, 75775 Paris, France*

rjn@mathstat.dal.ca *Mathematics and Statistics, Dalhousie University, Halifax B3H 3J5, Canada*