Multi-Agent Retrograde Analysis

Tristan Cazenave[⋆]

LAMSADE
Université Paris-Dauphine

Abstract. We are interested in the optimal solutions to multi-agent planning problems. We use as an example the predator-prey domain which is a classical multi-agent problem. We propose to solve it on small boards using retrograde analysis.

1 Introduction

The predator-prey problem is a classical multi-agent problem. It was introduced in [3]. There are four predators and one prey and the goal of the predators is to capture the prey. In this seminal work the predators can occupy the same location and the prey moves randomly. In a posterior work the agents could not occupy the same location [17]. Richard Korf proposed a simple pursuit strategy using attraction between the predators and the prey and repulsion between predators [10].

The predator-prey problem has been used to test multiple agent based algorithms. For example it has been use to analyze a general model of multi-agent communication with a message board, using a genetic algorithm to evolve multi-agent languages [9]. It has also been used to test genetic algorithms with lamarckian learning operators in multi-agent environments [7].

Genetic programming has also been used to co-evolve predators and preys populations [8]. In this work the authors acknowledge that the approach fails and claim that a simple prey algorithm is able to evade capture from the predators algorithms. Another work evolving multi-agent teams for the predator-prey game is presented in [11].

In a system evolving neural networks in separate subpopulations for different agents, it was advocated that the learning is easier than with a single controller and that communication is unnecessary and even detrimental in the predator-prey problem [21].

The other topic we address in this paper is retrograde analysis. Retrograde analysis computes the optimal solution to a large number of game states starting from terminal positions and going up towards deeper positions. It was first used in Chess and Checkers and related to dynamic programming [1]. Chess endgames have been completely solved up to 6 pieces using retrograde analysis [18, 19]. The 6 piece endgame tables requires 1.2 TB. Endgame tables have also been instrumental in solving Checkers [16]. Awari was completely solved thanks

⋆ cazenave@lamsade.dauphine.fr

to retrograde analysis [12] leading to an optimal and instantaneous player. Retrograde analysis has also been applied to many other games such as Nine Men's Morris [6], Go [4], Fanorona [14], Chinese Chess [5] or Chinese Dark Chess [13] among others.

Games can also be solved by search. A standard algorithm for solving games is iterative deepening $\alpha\beta$ with a transposition table [15]. Search was used to solve Checkers [16], small board Go [20] and small board Atarigo [2].

The outline of the paper is to present the predator-prey game in the next section, then to present retrograde analysis and search, followed by experimental results.

2 The Predator-Prey Game

In the predator-prey game we have designed, three predators are trying to capture a prey. In our implementation there are five possible moves for each agent: going up, down, left, right or staying on the same location. Predators cannot occupy the same location and when a prey moves to a predator location it is captured.

A state is terminal either if the prey is on the same location as a predator or if the prey is blocked by the predators and cannot move to an empty location.

A state is legal if no two predators are on the same location.

In previous work, moves by the predators and the prey can be either simultaneous or sequential. We have chosen sequential moves with the prey moving after the predators. When the prey is the second player he can choose the move the most beneficial to him knowing the future locations of the predators. It should be better for the prey but the evaluation must be made only after the prey move so as to simulate simultaneous moves. If the predator moves to the location of the prey, the prey can still escape since it can still move and that the evaluation of a state is only made after the move of the prey.

In our implementation it is possible for the prey to swap locations with a neighbor predator. It could also be possible to forbid such swaps. Enabling swaps as we do should be beneficial to the prey.

Overall when we had to make choices for the design of the game we chose the design the most beneficial to the prey.

3 Retrograde Analysis

In order to store the results of retrograde analysis in a table we have to design a bijection between the states of the problem and the indices in the table. We call the index associated to a state its code. A simple way to compute a code is to number each agent and each cell on the board and to compute the code of a board as:

$code = \sum_{agent} cell(agent) \times MaxCell^{agent}$

In this formula the agent variable is an integer between 0 and 3 that represents an agent. Agent 0 is the prey and agents 1, 2 and 3 are the predators. The function *cell*(*agent*) returns an integer that represents the location of the agent on the board, each cell is associated to an integer between 0 and $MaxCell - 1$.

Using the previous code we consider that each agent is different from the other ones. However we could consider that two predators can exchange their locations and that it is still the same state. In this case the total number of states and the greatest possible code are quite reduced [14], thus reducing the size of the retrograde analysis table.

For this paper, we kept things simple and in our experiments we used the simple code considering each agent different from the other ones.

The overall algorithm that performs retrograde analysis is given in algorithm 4. It calls two subsequent algorithms. The initialisation algorithm that is given in algorithm 1 and which initializes the table with terminal states, and the step algorithm that is given in algorithm 2 and which computes the states won by the predators in currentDepth moves by the prey. The step algorithm performs a one ply search in order to discover the states won at currentDepth given the states won in less than currentDepth. This one ply search algorithm is given in algorithm 3.

In the algorithms the constant $MaxAgent$ is set to 4 and represents the number of agents including the prey. When the *agent* variables reaches $MaxAgent$ it means that either all the agents have been placed in the initialisation algorithm or that all the predators have moved in the step and the one step lookahead algorithms.

In the algorithm 3 the predators try to minimize the depth to the capture and the prey tries to maximize it. The unknown states are initialized to ∞ in the init algorithm. If the prey can escape to an unknown state then the algorithm returns ∞ and the predators have to keep trying other moves.

The table can be used to decide the predators moves that wins in the smallest number of steps. It can also be used to decide the prey move that will take the most number of steps before capture. In some Chess endgames, even in lost states, computers using an endgame table can lure grandmasters and keep them away from victory as the human players do not always play optimal moves.

4 Search

The worst branching factor for 4 agents and vertical and horizontal moves is $5^4 = 625$. A simple depth 8 problem for size 5×5 can already visit at most $625^8 = 2.33 \times 10^{22}$ leaves. In practice the exact number of leaves should be less but still quite a large number. It would be clearly more than the state space size of a 5×5 problem which is 345,000. The state space complexity of the problem is far lower than its game tree complexity.

A possible solution to avoid searching again the already visited states is to use iterative deepening search with a transposition table as a search algorithm. It avoids searching again the same state multiple times and it could significantly

Algorithm 1 The initialisation algorithm

```
init ($agent$)
if $agent$ = MaxAgents then
  if board is legal then
    nbStates ← nbStates + 1
    depth [board.code ()] ← $\infty$
    if board is terminal then
      depth [board.code ()] ← 0
      nbStatesDepth [0] ← nbStatesDepth [0] + 1
    end if
  end if
else
  for $cell$ in possible locations on board do
    board.cell [$agent$] ← $cell$
    init ($agent + 1$)
  end for
end if
```

Algorithm 2 The step algorithm computing the next depth of retrograde analysis

```
step ($agent$)
if $agent$ = MaxAgents then
  if depth [board.code ()] = $\infty$ then
    if board is legal then
      if min (1) = currentDepth - 1 then
        depth [board.code ()] ← currentDepth
        nbStatesDepth [currentDepth] ← nbStatesDepth [currentDepth] + 1
      end if
    end if
  end if
else
  for $cell$ in possible locations on board do
    board.cell [$agent$] ← $cell$
    step ($agent + 1$)
  end for
end if
```

Algorithm 3 The one step lookahead algorithm

```
min ($agent$)
if $agent$ = MaxAgents then
  return max ()
end if
mini ← min ($agent + 1$)
for $move$ in possible moves for $agent$ do
  make $move$ for $agent$
  $eval$ ← min ($agent + 1$)
  undo $move$ for $agent$
  if $eval < mini$ then
    $mini \leftarrow eval$
  end if
end for
return $mini$

max ()
if board is illegal then
  return $\infty$
end if
maxi ← depth [board.code ()]
for $move$ in possible moves for the prey do
  make $move$ for the prey
  $eval$ ← depth [board.code ()]
  undo $move$ for the prey
  if $eval > maxi$ then
    $maxi \leftarrow eval$
  end if
end for
return $maxi$
```

Algorithm 4 The overall algorithm for retrograde analysis

```
nbStates ← 0
nbStatesDepth [0] ← 0
init (0)
currentDepth ← 1
while true do
  nbStatesDepth [currentDepth] ← 0
  step (0)
  if nbStatesDepth [currentDepth] = 0 then
    break
  end if
  currentDepth ← currentDepth + 1
end while
```

decrease the search time as there are many transpositions in the predator-prey problem.

We have implemented a perfect transposition table. A perfect transposition table is a table that has exactly one entry per possible state. When a state has been searched the result can be stored in the corresponding entry and it can be reused when reaching the state again. We use the code of the board as the index in the transposition table.

The search algorithm for the predator-prey game is given in algorithm 5. It uses a perfect transposition table and two functions. The minTT function tries all the possible combinations of the predators moves and selects the one leading to the capture of the prey if it exists. If no combination enables the capture in *depth* steps it returns false. The maxTT function tries all possible moves for the prey and selects the one that avoids capture. If all possible moves lead to capture it stores the result in the transposition table and returns true.

The TT table contains the depth of the search that solved the state. It contains ∞ if the state was not solved. If a state has already been solved with a smaller or equal depth, the algorithm returns true. The other table is the depthTT table, it contains the maximum search depth performed for the state. If a state has already been searched with a greater or equal depth, the search is cut as it is not necessary to search it again.

5 Experimental Results

The experiments were run on a 1.9 GHz computer running Linux and the algorithms were written in C++.

The retrograde analysis algorithm was used to compute the depth to mate of every state for various board sizes. The number of states for each depth to mate is given in the table 1 for board sizes ranging from 4×4 to 9×9.

The table 2 gives the total number of states, the maximum code used and the time to perform retrograde analysis for 4×4 to 9×9 boards.

We wrote an algorithm similar to the initialisation algorithm in order to verify that all states are won for the predators. It is run after the retrograde analysis is finished and verifies that the depth to mate is finite for every possible state. We have found that it is the case for all the board sizes we solved.

A 7×7 state with maximum depth 12 is the following state:

```
o....xx
......x
.......
.......
.......
.......
.......
```

Algorithm 5 The search algorithm

```
minTT ($depth$, $agent$)
if $agent$ = MaxAgents then
  return maxTT ($depth - 1$)
end if
if minTT ($depth$, $agent + 1$) then
  return true
end if
for $move$ in possible moves for $agent$ do
  make $move$ for $agent$
  $eval \leftarrow$ minTT ($depth$, $agent + 1$)
  undo $move$ for $agent$
  if $eval = true$ then
    return true
  end if
end for
return false

maxTT ($depth$)
if board is illegal then
  return false
end if
if prey is blocked then
  return true
end if
if $depth = 0$ then
  return false
end if
if TT [board.code ()] $\leq depth$ then
  return true
end if
if depthTT [board.code ()] $\geq depth$ then
  return false
end if
if depthTT [board.code ()] $< depth$ then
  depthTT [board.code ()] $= depth$
end if
if the prey is not on the same location as a predator then
  if not minTT ($depth$, 1) then
    return false
  end if
end if
for $move$ in possible moves for the prey do
  make $move$ for the prey
  if the prey is not on the same location as a predator then
    if not minTT ($depth$, 1) then
      undo $move$ for the prey
      return false
    end if
  end if
  undo $move$ for the prey
end for
TT [board.code ()] $\leftarrow depth$
return true
```

Table 1. Number of states for each depth to mate and for different board sizes.

Depth	4×4	5×5	6×6	7×7	8×8	9×9
0	10,440	42,000	129,408	332,856	751,560	1,537,800
1	2,712	4,920	7,464	10,344	13,560	17,112
2	3,960	5,976	7,560	8,616	9,672	10,728
3	10,200	16,056	20,808	24,792	26,760	28,728
4	19,584	42,840	48,960	57,888	64,752	68,352
5	6,864	79,752	119,040	128,616	138,912	150,912
6	0	83,928	240,864	273,600	283,416	298,080
7	0	68,760	367,896	531,000	584,280	580,128
8	0	768	387,816	888,696	1,122,336	1,131,336
9	0	0	211,848	1,218,600	1,927,536	2,067,480
10	0	0	576	1,170,576	3,021,264	3,533,112
11	0	0	0	755,424	3,478,080	5,575,200
12	0	0	0	15,648	3,005,280	7,666,608
13	0	0	0	0	1,551,816	8,301,696
14	0	0	0	0	19,752	6,625,848
15	0	0	0	0	0	3,816,432
16	0	0	0	0	0	55,968
17	0	0	0	0	0	0

Table 2. Number of states, maximum code and time to solve with retrograde analysis in seconds for different sizes

Size	Number of states	Maximum code	Time to solve
4×4	53,760	65,536	3.82
5×5	345,000	390,625	56.16
6×6	1,542,240	1,679,616	386.75
7×7	5,416,656	5,764,801	1,900.91
8×8	15,998,976	16,777,216	8,618.76
9×9	41,465,520	43,046,721	27,002.54

The iterative deepening search for this state evolves as indicated in the table 3. The times indicated are the cumulative times, the times for all inferior depth. The time to solve the corresponding 9×9 problem with search at depth 16 is 1714.53 seconds. This is only to solve one problem when retrograde analysis can be computed offline in 1,900 seconds for size 7×7 and for all problems and results in instantaneous and optimal moves.

Table 3. Times for searching a 7×7 depth 12 state with iterative deepening and a perfect transposition table.

Depth	Time
1	0.006094
2	0.006181
3	0.007001
4	0.013914
5	0.046882
6	0.162278
7	0.492648
8	1.453308
9	4.410059
10	12.724911
11	91.924769
12	164.675883

In order to illustrate the predators strategies, we give an example of the solution to the 7×7 problem above of maximum depth 12, with the prey randomly choosing among the moves leading to a maximum depth state:

```
o....xx   o....x.   o....x.   ....x..   ...x...   ..x....
......x   .....xx   ....xx.   o..xx..   ..xx...   .xx....
.......   .......   .......   .......   o......   .......
.......   .......   .......   .......   .......   o......
.......   .......   .......   .......   .......   .......
.......   .......   .......   .......   .......   .......
.......   .......   .......   .......   .......   .......
  12        11        10         9         8         7

.x.....   .......   .......   .......   .......   .......
.x.....   .x.....   .......   .......   .......   .......
..x....   .x.....   .x.....   .......   .......   .......
.o.....   .ox....   .xx....   .x.....   x......   x......
.......   .......   .o.....   oxx....   xx.....   .......
.......   .......   .......   .......   o......   xx.....
.......   .......   .......   .......   .......   o......
   6         5         4         3         2         1
```

6 Conclusion

Retrograde analysis of the predator-prey problem is tractable in time and memory until 9×9 boards. It results in instantaneous decisions and optimal multi-agent strategies.

A result from our research is that the predator-prey game is always lost for the prey even when there are only 3 predators, when the prey knows the moves of the predators before moving and when the prey is allowed to swap locations with a neighbor predator. The maximum number of moves by the prey before capture is 14 for size 8×8 and 16 for size 9×9.

Another result is that iterative deepening search with a perfect transposition table is slow even for small board sizes. It cannot compete with retrograde analysis with respect to solving time.

In future work we will explore the use of abstraction so as to solve boards of large sizes, learning of agents strategies, and compression of tables. Another line of research is to solve a continuous version of the game.

There are multiple possibilities for learning using endgame tables. For example, learning an evaluation function for a depth one search, learning the move to make for an agent or learning a move ordering heuristic with an evaluation of states or with an evaluation of moves.

Another line of research is to analyze endgames of multi-agent games with a much larger state space.

References

1. Richard Bellman. On the application of dynamic programing to the determination of optimal play in chess and checkers. *Proceedings of the National Academy of Sciences of the United States of America*, 53(2):244, 1965.
2. Frédéric Boissac and Tristan Cazenave. De nouvelles heuristiques de recherche appliquées à la résolution d'Atarigo. In *Intelligence artificielle et jeux*, pages 127–141. Hermes Science, 2006.
3. M. Brenda, V. Jagannathan, and R. Dodhiawala. On optimal cooperation of knowledge sources-an empirical investigation. *Boeing Adv. Technol. Center, Boeing Comput. Services, Seattle, WA, Tech. Rep. BCSG2010-28*, 1986.
4. Tristan Cazenave. Generation of patterns with external conditions for the game of go. *Advances in Computer Games*, 9:275–293, 2001.
5. Haw-ren Fang, Tsan-sheng Hsu, and Shun-chin Hsu. Construction of chinese chess endgame databases by retrograde analysis. In *Computers and Games*, pages 96–114. Springer, 2001.
6. Ralph Gasser. Solving nine men's morris. *Computational Intelligence*, 12(1):24–41, 1996.
7. John J. Grefenstette. Lamarckian learning in multi-agent environments. Technical report, DTIC Document, 1995.
8. Thomas Haynes and Sandip Sen. Evolving behavioral strategies in predators and prey. In *Adaption and learning in multi-agent systems*, pages 113–126. Springer, 1996.

9. Kam-Chuen Jim and C. Lee Giles. Talking helps: Evolving communicating agents for the predator-prey pursuit problem. *artificial life*, 6(3):237–254, 2000.
10. Richard E. Korf. A simple solution to pursuit games. Working papers of the 11th international workshop on distributed artificial intelligence, 1992.
11. Sean Luke and Lee Spector. Evolving teamwork and coordination with genetic programming. In *Proceedings of the 1st annual conference on genetic programming*, pages 150–156. MIT Press, 1996.
12. John W. Romein and Henri E. Bal. Solving awari with parallel retrograde analysis. *IEEE Computer*, 36(10):26–33, 2003.
13. Abdallah Saffidine, Nicolas Jouandeau, Cédric Buron, and Tristan Cazenave. Material symmetry to partition endgame tables. In *Computers and Games*, pages 187–198. Springer, 2014.
14. Maarten P. D. Schadd, Mark H. M. Winands, Jos W. H. M. Uiterwijk, H. Jaap van den Herik, and Maurice H. J. Bergsma. Best play in fanorona leads to draw. *New Mathematics and Natural Computation*, 4(3):369–387, 2008.
15. Jonathan Schaeffer. The history heuristic and alpha-beta search enhancements in practice. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 11(11):1203–1212, 1989.
16. Jonathan Schaeffer, Neil Burch, Yngvi Björnsson, Akihiro Kishimoto, Martin Müller, Robert Lake, Paul Lu, and Steve Sutphen. Checkers is solved. *science*, 317(5844):1518–1522, 2007.
17. Larry M. Stephens and Matthias B. Merx. Agent organization as an effector of dai system performance. In *Ninth Workshop on Distributed Artificial Intelligence, Rosario Resort, Eastsound, Washington*, pages 263–292, 1989.
18. Ken Thompson. Retrograde analysis of certain endgames. *ICCA Journal*, 9(3):131–139, 1986.
19. Ken Thompson. 6-piece endgames. *ICCA Journal*, 19(4):215–226, 1996.
20. Erik C.D. van der Werf and Mark H.M. Winands. Solving go for rectangular boards. *ICGA Journal*, 32(2):77–88, 2009.
21. Chern Han Yong and Risto Miikkulainen. Cooperative coevolution of multi-agent systems. *University of Texas at Austin, Austin, TX*, 2001.