Iterative Widening

Tristan Cazenave[1]

Abstract. We propose a method to gradually expand the moves to consider at the nodes of game search trees. The algorithm begins with an iterative deepening search using the minimal set of moves, and if the search does not succeed, iteratively widens the set of possible moves, performing a complete iterative deepening search after each widening. When designing carefully the different sets of possible moves, the algorithm can save some time in the game of Go tree search, as shown by experimental results.

Key words: Computer Go, Search.

1 Introduction

We propose a method to gradually expand the moves to consider at the nodes of game search trees. The algorithm begins with an iterative deepening search using the minimal set of moves, and if the search does not succeed, iteratively widens the set of possible moves, performing a complete iterative deepening search after each widening. When designing carefully the different sets of possible moves, the algorithm can save some time in the game of Go tree search, as shown by experimental results.

The second section describes the search algorithm and compares it with related existing algorithms. The third section gives hints on how to define and combine the gradually expanding sets of moves and defines some of these sets for the capture game in the game of Go. The fourth section details experimental results and underlines future work.

---

[1] Laboratoire d'Intelligence Artificielle, Département Informatique, Université Paris 8, 2 rue de la Liberté, 93526 Saint Denis, France. e-mail: cazenave@ai.univ-paris8.fr tel: 33 1 49 40 64 04 fax: 33 1 49 40 64 00

2 The search algorithm

We use Abstract Proof Search [Cazenave 2000] to develop AND/OR proof trees for the game of Go. This is an iterative deepening Null Window Search [Marsland & Björnsson 2000], that uses some game specific functions to efficiently prove theorems about goals in games.

We define sets of abstract possible moves, that can be tried at the node of the search tree at a given widening threshold. Sets are numbered, the following set always contains the previous set. Our algorithm uses the sets of moves in their number order.

For example, if the sets of possible moves to be tried at different widening threshold are the sets S1, S2,…,Sn. We have $S1 \subset S2 \subset \ldots \subset Sn$. The algorithm begins with an Abstract Proof Search, trying the moves in the set S1. If this search fails, it then makes another search with the S2 set. And so on until all the possible searches have failed, or the allotted time is elapsed.

For each goal to compute, two searches have to be performed. The first one with White playing first and the second one with Black playing first. However, when the goal can be reached by one player and prevented by the other, depending on who plays first, it is also very important to know all the moves that reach (i.e. prevent) the goal. It is more efficient to begin with a simple search that stops as soon as the goal is reached or prevented, and then to check if the goal can still be reached or prevented even if the opponent plays first. If so, it is useless to find all the working moves, as there is no need to play them because the goal is reached/prevented even if they are not played. On the contrary, if reaching the goal depends on who plays first, it is necessary to know all the working moves. In this case only, the same search is performed again, except that it is not stopped after the first working move, it continues until a predefined threshold .

First call the search algorithm with the first move function that sends back the moves of the first set. If the search does not succeed, continue with the following sets until the search succeeds or the time threshold is finished, or the search fails with the ultimate set.

In our experiments, the search are performed looking for all the working moves. So at the root, all the possibly interesting moves are tried, whatever the widening threshold is.

When a search fails at a given widening threshold, the transposition table is reinitialized and a new search is performed with the next set if possible. This can be improved by reusing the same transposition table for all the searches.

After having designed the method, we found that it has links with Iterative Broadening [Ginsberg & Harvey 1992]. This method is successful in constraint satisfaction search [Meseguer & Walsh 1998]. However, Iterative broadening is not the same algorithm as ours because it sets an artificial breadth cutoff c, and backtracks at most c times at any node of the tree. It iteratively increases c, and information can be memorized for the next iteration. Experiments by Ginsberg and Harvey in applying Iterative Broadening to Chess gave disappointing results because the move ordering of current Chess program is already near the optimum.

3 Designing the gradual sets of moves

It is quite important to carefully choose the sets of moves. The first set is better if it contains the moves that have high chances to reach the goal. Typically, the last set contains all the moves worth trying. We have separated the sets for the OR nodes and the AND nodes of the tree, as they have completely different properties.

We have defined two sets of moves at OR nodes: OR1 is constituted by the liberties of the string to capture only. OR2 is constituted by all the moves worth trying, including the liberties of the string to capture, the liberties of the liberties of the string to capture and, the liberties of the strings adjacent to the string to capture that have less liberties than it.

Similarly, we have defined two sets of moves at AND nodes : AND1 is constituted by the ip1 and ip2 moves, AND2 is constituted by the ip1, ip2 and ip3 moves. the ipn moves are the moves that prevent a string to be captured in n moves by the opponent. For example, the ip1 moves are the moves that may prevent a string in atari to be captured in one move (i. e. playing the liberty, or capturing an adjacent string).

There are different orders in which the widening can be performed. Here are the one we have tested:

- OR2-2AND2-2: This is the original non-widening, iterative deepening Null Window Search algorithm. The OR2 set of moves is used at OR nodes, and the AND2 set of moves is used at AND nodes.
- OR1-2AND2-2: The algorithm begins with the OR1 and AND2 sets of moves, and if the search fails, it searches again with the OR2 and AND2 sets of moves.
- OR2-2AND1-2: The algorithm begins with the OR2 and AND1 sets of moves, and if the search fails, it searches again with the OR2 and AND2 sets of moves.
- AND1-2OR1-2: The algorithm begins with the OR1 and AND1 sets of moves, and if the search fails, it searches again with the OR1 and AND2 sets of moves. If the search fails again, it searches again with the OR2 and AND2 sets of moves.
- OR1-2AND1-2: The algorithm begins with the OR1 and AND1 sets of moves, and if the search fails, it searches again with the OR2 and AND1 sets of moves. If the search fails again, it searches again with the OR2 and AND2 sets of moves.
- ORAND1-2: The algorithm begins with the OR1 and AND1 sets of moves, and if the search fails, it searches again with the OR2 and AND2 sets of moves.
- OR1-2ANDOR1-2: The algorithm begins with the OR1 and AND1 sets of moves, and if the search fails, it searches again with the OR2 and AND1 sets of moves. If the search fails again, it searches again with the OR1 and AND2 sets of moves. If the search fails again, it eventually searches with the OR2 and AND2 sets of moves.

- ORAND1-2AND1-2: The algorithm begins with the OR1 and AND1 sets of moves, and if the search fails, it searches again with the OR1 and AND2 sets of moves. If the search fails again, it searches again with the OR2 and AND1 sets of moves. If the search fails again, it eventually searches with the OR2 and AND2 sets of moves.

4 Experimental results and future work

This section gives experimental results on a standard test set for capturing strings in Go: we call them ggv1 [Kano 1985a], ggv2 [Kano 1985b] and ggv3 [Kano 1987]. We have selected all the problems involving a capture of a string, including semeai and some connection problems. There are 114 capture problems in ggv1, 144 in ggv2 and 75 in ggv3. Experiments were performed on a Pentium 266 MHz micro-processor.

Algorithm	Total time	Number of nodes	% of problems
OR2-2AND2-2	18.15	4809	99.12%
OR1-2AND2-2	17.67	2667	99.12%
OR2-2AND1-2	12.81	4291	99.12%
AND1-2OR1-2	12.26	2576	99.12%
OR1-2AND1-2	12.38	3044	99.12%
ORAND1-2	12.11	2730	99.12%
OR1-2ANDOR1-2	12.31	2913	99.12%
ORAND1-2AND1-2	12.13	2587	99.12%

Table 1. Results for ggv1

Algorithm	Total time	Number of nodes	% of problems
OR2-2AND2-2	62.96	30182	86.81%
OR1-2AND2-2	47.99	19096	86.11%
OR2-2AND1-2	32.62	28008	86.81%
AND1-2OR1-2	39.74	19721	86.11%
OR1-2AND1-2	37.15	24244	87.50%
ORAND1-2	39.57	19566	87.50%
OR1-2ANDOR1-2	35.99	23450	87.50%
ORAND1-2AND1-2	45.85	19544	85.42%

Table 2. Results for ggv2

Algorithm	Total time	Number of nodes	% of problems
OR2-2AND2-2	41.43	21226	78.67%
OR1-2AND2-2	30.03	15526	77.33%
OR2-2AND1-2	23.70	22647	81.33%
AND1-2OR1-2	23.78	15073	77.33%
OR1-2AND1-2	20.68	16281	81.33%
ORAND1-2	25.11	13206	78.67%
OR1-2ANDOR1-2	21.85	18106	80.00%
ORAND1-2AND1-2	32.74	13844	74.67%

Table 3. Results for ggv3

Every combination of widening sets gives speed-ups compared to the original algorithm. However, some combinations also decrease the percentage of solved problems. Luckily, some combinations both decrease the time to solve problems and increase the percentage of solved problems. In particular, OR1-2AND1-2 seems to be the combination of choice. It does not only reduce significantly the computation time, it also solves more problem than the original non iterative widening algorithm (OR2-2AND2-2).

In the original non iterative widening algorithm, the liberties of the string are tried first, and the order of the moves at each node is the same as in the iterative widening algorithm, therefore the observed speed-ups are due to the iterative widening, not to another factor such as move ordering.

In these experiments, the transposition table is completely initialized before each widening. It would me more clever, to keep the same transposition table, and to put a flag on the transpositions, memorizing the widening step of the transposed board. Therefore reusing the information from the previous and less wide search in order to save computation time.

5 Conclusion

Gradually widening the sets of moves in the game of Go search trees enables to reduce the search time when performing an iterative deepening Null Window Search. It also appears that some more problems can be solved by using this technique. However, one has to be careful when choosing the widening sets, only some combinations of them give good results. Results could be even better by reusing transposition table information from the previous and less wide search.

6 References

Cazenave T.: *Abstract Proof Search*. Submitted. 2000.

Ginsberg M. L., Harvey W. D. : *Iterative Broadening.* Artificial Intelligence 55 (2-3), pp. 367-383. 1992.

Kano Y.: *Graded Go Problems For Beginners. Volume One.* The Nihon Ki-in. ISBN 4-8182-0228-2 C2376. 1985.

Kano Y.: *Graded Go Problems For Beginners. Volume Two.* The Nihon Ki-in. ISBN 4-906574-47-5. 1985.

Kano Y.: *Graded Go Problems For Beginners. Volume Three.* The Nihon Ki-in. ISBN 4-8182-0230-4. 1987.

Marsland T. A., Björnsson Y.: *From Minimax to Manhattan.* Games in AI Research, pp. 5-17. Edited by H.J. van den Herik and H. Iida, Universiteit Maastricht. ISBN 90-621-6416-1. 2000.

Meseguer P., Walsh T. : *Interleaved and Discrepancy Based Search.* Proceedings ECAI98 (ed. H. Prade). John Wiley & Sons Ltd., Chichester, England. ISBN 0-471-98431-0. 1998.