Generalized Widening

Tristan Cazenave[1]

Abstract. We present a new threat based search algorithm that outperforms other threat based search algorithms and selective knowledge-based $\alpha\beta$ for open life and death problem solving in the game of Go. It generalizes the Iterative Widening algorithm which consists in iteratively increasing the threat searched at the root. The main idea of Generalized Widening is to perform Iterative Widening at all max nodes of the search tree instead of performing it only at the root. Experimental results show it can be three times faster than selective knowledge-based $\alpha\beta$ using the same knowledge, and eight times faster than simple Iterative Widening. The performance against $\alpha\beta$ can possibly be greatly enhanced by adding more knowledge in the selection of moves during the verification of the threats.

1 INTRODUCTION

Generalized Widening (gw) is an improvement on current search algorithms to solve games. It improves on threat based search algorithms such as Generalized Threats Search (gts) [6], and it is a generalization of the Iterative Widening (iw) optimization [4]. The basic idea is to apply Iterative Widening at all max nodes of the search tree instead of applying it only at the root of the search tree.

Another goal of the paper is to show the applicability of threat based search algorithms to open life and death problems in the game of Go. The game of Go is a challenging game for AI, it is the only classical board game where humans are still clearly superior to machines despite lot of efforts [2, 10]. Open life and death problem solving is an important part of a Go program.

Section 2 presents the problem of life and death in the game of Go. Section 3 presents algorithms that are related to Generalized Widening. Section 4 explains the Generalized Widening algorithm. Section 5 details its application to open life and death problems in the game of Go. Section 6 details experimental results. Section 7 gives hints for future work.

2 LIFE AND DEATH IN THE GAME OF GO

Assessing the life of groups is an important subgame of the game of Go. For strong human players, improving their level in life and death problem solving is considered as a good way to improve their overall reading abilities and their level at Go.

For completely enclosed life and death problems, GoTools [14] is the best program. It can compete with high dan level players. Unfortunately, it is not able to solve open problems as those commonly arising in real games. It is restricted to completely enclosed problems with thirteen or fewer intersections. In open problems from real games, the boundaries are not well defined and the number of possible moves can easily be twenty. Some of the best Go programs use static heuristics to evaluate the life and death property in real games [7, 8] while others rely on selective pattern based search [3].

3 RELATED ALGORITHMS

In this section we start with presenting Generalized Threats Search which is an algorithm that selectively chooses the nodes to expand based on the verification of threats, then we present the Iterative Widening optimization for selective search algorithms, and the last subsection deals with Df-pn which relates to Generalized Widening in the sense that it also performs multiple iterative deepening and selects nodes to expand.

3.1 Generalized Threats Search

Generalized Threats Search tries to verify a threat at min nodes. It stops search when the threat is not verified. A threat at a min node always starts with a max move. Moves in a generalized threats are associated to an order. The order of a position is the number of moves in a row by the same player that are required to win the game. Moves of order n are moves associated to positions of order less or equal to n. An example of a threat is the (6, 3, 2, 0) threat. Six is the number of order one moves allowed in the verification of the threat, three the number of order two moves and two the number of order three moves. Only threats that have less moves than specified in the vector for each order are verified at min nodes. If no threat is verified at a min node, the node is cut and returns the $NoThreat$ value which is the lowest non terminal value. Generalized Threats Search is able to solve some games much faster than $\alpha\beta$. Examples of games it can solve faster are 11x11 Philosopher's Football, 6x6 Ponnuki Go, and the capture game of Go.

3.2 Iterative Widening

Iterative Widening is an optimization of selective search algorithms which has been used with success in the capture game of Go [4]. It has also recently worked for endgame play in Scrabble [13]. The idea is to iteratively increase the sets of moves that will be considered by the search. For threat based algorithms, it can consist in increasing the threats. The algorithm starts with trying a full iterative deepening search with the smallest threat, and if it does not work, it continues to perform full iterative deepening searches with the following threats until the last available threat.

3.3 Df-pn

The PN* algorithm [12] is a depth first version of the Proof-Number search algorithm [1]. Proof number search uses proof numbers and disproof numbers at each node. A proof number is the minimal number of leaves under a node that need to be expanded in order to prove

[1] Laboratoire d'Intelligence Artificielle, Université Paris 8, Saint Denis, France email : cazenave@ai.univ-paris8.fr

a win for the node. A disproof number is the minimum number of leaves that need to be expanded to prove a loss. PN* uses iterative deepening search at all max nodes, it stops when the threshold for the proof number is reached or when the node is proved or disproved.

Df-pn [11, 9] goes further than PN* by performing iterative deepening at all nodes, and having thresholds both for proof numbers and for disproof numbers.

4 GENERALIZED WIDENING

The idea of Generalized Widening is to perform Iterative Widening at all max nodes of the search tree. In this section we start by stating what is multiple iterative widening, then we state what is multiple iterative deepening, the third subsection details the use of transposition tables in Generalized Widening, and the fourth subsection gives and explains some pseudo-code for the max nodes search.

4.1 Multiple Iterative Widening

At max nodes, the algorithm starts with the smallest threat and increases the threat until the threat associated to the node is searched or the time is elapsed or the node is won or the node is lost. Threats are ordered and represented by an indice. For example the threat number 0 is usually (1,0), meaning that only min nodes where a single max move threatens a direct win will be developed. The threat number 1 is usually (2,1,0) which is the simplest threat of order 2.

4.2 Multiple Iterative Deepening

Multiple iterative deepening consists in performing an iterative deepening search at all max nodes of the search tree.

For threats that are less than the threat associated to the node, Generalized Widening performs a complete iterative deepening search until the result of the search is terminal or the time is elapsed. For the threat associated to the node, it performs an iterative deepening search that stops at the depth associated to the node.

4.3 Use of transposition tables

Beside the usual information, a transposition table entry for Generalized Widening contains the number of the threat used to search the position. Each entry in the transposition table contains information on the score of the position, the depth it was searched, a flag for determining if the score is exact, is a lower bound or is a higher bound, the threat used, the best move and the forced moves.

We give below the code used to detect transpositions. The function returns 1 when a transposition is successful, and 0 when the position has to be searched again. The alpha, beta and res variables are passed by reference.

```
transpo (d, alpha, beta, res, t) {
    if (score == Won) {
      res = Won
      return 1
    }
    else if (score == Lost) {
      res = Lost
      return 1
    }
    if (depthTranspo >= d) {
      if (flag == ExactScore) {
        res = score
        if (t <= threatTranspo)
          return 1
      }
      else  {
        if ((flag == MinScore) &&
            (t <= threatTranspo))
          alpha = max (alpha, score)
        else if ((flag == MaxScore) &&
                 (t <= threatTranspo))
          beta = min (beta, score)
        if (alpha >= beta) {
          res = score
          return 1
        }
      }
    }
    return 0
  }
```

4.4 Algorithm for max nodes

The Generalized Widening algorithm performs at each max node complete iterative deepening searches for all threats lower than the threat associated to the node. The multiple iterative deepening for these threats continues until the overall maximum possible depth is reached or the position has the maximum possible evaluation (Won or $WonByKo$ if a ko has been taken back before) or the search returns $NoThreat$ (meaning that the threat has to be increased to possibly find a win) or $LostByKo$ or $Lost$.

Concerning the threat associated to the node, the multiple iterative deepening stops at the depth of the node. The threats less than the maximum threat for the node are searched possibly deeper than the maximum threat for the node. This means that the lower threats work as quiescence searches of the main threat search.

The pseudo-code for the algorithm is:

```
MaxNode (alpha, beta, threat, depth) {
  eval = evaluation ();
  if (isTerminal (eval) ||
      (depth == 0) ||
      !moreTime ())
    return eval

  if (transpo (depth, alpha, beta,
               res, threat))
    return res

  Generate max moves

  // Multiple Iterative Widening
  for (t = 0;
       ((t <= threat) &&
        (res < MaxEval ()) &&
        (res > MinEval ()) &&
        moreTime ());
       t++) {
    res = eval

    // Multiple Iterative Deepening
    currentDepth = MaxDepth
```

```
    if (t == threat)
      currentDepth = depth
    for (d = 1;
         ((d <= currentDepth) &&
          (res > NoThreat) &&
          (res < MaxEval ()) &&
          moreTime ());
         d++) {
      res = MinEval ()
      alphaTemp = alpha
      betaTemp = beta
      for all max moves
        if (alphaTemp < betaTemp) {
          play move
          r = MinNode (alphaTemp,
                       betaTemp, t, d-1)
          if (r > res) {
            res = r
            if (res > alphaTemp)
              alphaTemp = res
          }
          undo move
        }
    }
  }
  return res
}
```

5 APPLICATION TO LIFE AND DEATH

In this section we present the features of the program specific to Life and Death. We start with explaining how groups are built. The second subsection describes how some properties of the group are evaluated. The third subsection deals with the selection of moves. The fourth subsection details the evaluation function.

5.1 Construction of the Group

A fundamental task of an open life and death solver is to list all the strings that will be considered as related concerning their life and death. The set of related strings is called a group, this terminology comes from Go. In order to build the group, the program start from a string and iteratively adds all the strings that share a liberty with a string already in the group, it also adds strings that are adjacent to opponent strings that can be captured in a ladder, if the opponent string is itself adjacent to a string of the group.

The set of strings of the group is updated incrementally during the search.

5.2 Properties of the Group

Once a group is built, we can compute properties of the group such as its skin and the list of possible eye points. These properties are useful to evaluate the life of the group.

The skin of a group is a set of strings and empty points that surround the group. The skin includes the opponent strings that have one or two liberties and that are adjacent to the group, the strings adjacent to these adjacent strings that have less liberties than the adjacent strings. It also includes the liberties of all the strings of the groups, as well as the empty intersections adjacent to these liberties. Moreover, it includes the adjacent opponent strings that have four liberties or less and that are completely enclosed by the group.

False eye points are intersections that cannot possibly be transformed in eyes. An intersection of the skin is a false eye if it has a neighbor of the color of the opponent which is not part of the skin, or if it has two empty neighbors that are neighbors of opponent strings that are not part of the skin, or if it is empty and has at least one enemy neighbor, or if it is on the first line and it has at least one diagonal owned by the opponent, or if it is on the first line and it has two empty diagonals that the opponent can own in one move, or if the number of empty diagonals that the opponent can own in one move divided by two plus the number of diagonals owned by the opponent is strictly greater than one.

The set of possible eye points of a group is its skin minus the false eye points.

5.3 Selection of moves

The program uses two specialized functions to select moves at max and min nodes. The max nodes moves are the moves that try to make the group alive. The possible moves for the max player are the liberties of opponent strings adjacent to the group that can be captured in a ladder, and the liberties of the friend strings adjacent to the opponent string and which have less liberties than the opponent string, moves that threaten to capture an adjacent opponent string in a ladder, moves on the liberties of the strings of the group and on the empty intersections neighboring these liberties.

The possible moves for the min player are roughly the same as the move for the max player, except for moves that escape a ladder of an opponent string instead of moves that capture the ladder.

Move of order one are only moves that make an eye, a special function is called to select this kind of moves. Moves of order one are used in the evaluation function to detect if the group can live in one move. In our implementation, the move selection functions are the same for the $\alpha\beta$ and for the threat based algorithms. More selective move selection functions of order two and three can be written for threat based move selection and can improve by much the behavior of threat based search algorithms.

5.4 Evaluation function

The group is considered alive if two different eyes are recognized on two different liberties of the same string and if the two eyes do not share a non protected intersection (this last condition tests the non-dependence of the two eyes). The group is not dead if there are at least two possible eye points that are not neighbors, otherwise it is considered dead and the evaluation function returns $Lost$ or $LostByKo$ if a ko has been taken back earlier.

The evaluation function returns values that are ordered as follow: $Lost < LostByKo < NoThreat < Unknown < WonByKo < Won$.

The $NoThreat$ value means that the current threat has not been verified at a min node, it is different from the $Lost$ value which means that the threat to live has been verified, and that one of the min moves has lead to a dead group.

At max nodes, the evaluation function tests if there is an order one living move, if so it returns Won or $WonByKo$ depending on the ko status of the node.

6 EXPERIMENTAL RESULTS

The test suite consists of fifty life problems. In some problems the goal is to find a move that makes the group live, while in the others the goal is to prove the group is alive whatever the opponent plays. The problems are taken from games between computers, or from games between human and computers. They are representative of the kind of open problems that a life and death solver has to deal with in real games.

The threat number 0 is the (1,0) threat, number 1 is (2,1,0), number 2 is (6,5,0) and number 3 is (6,3,2,0).

The $\alpha\beta gw(t)$ algorithm is a generalized widening algorithm that reverts to usual $\alpha\beta$ when all the threats in [0,t-1] have failed. Instead of stopping the increase of threats when the higher threats returns NoThreat as in the gw algorithm, the $\alpha\beta gw(t)$ considers all the min moves for threat number t without verifying any threats (as in the usual $\alpha\beta$). This can also be understood as an $\alpha\beta$ with a Generalized Widening algorithm for quiescence search. The $\alpha\beta gw(0)$ algorithm is an $\alpha\beta$ with multiple iterative deepening at each max node.

The $\alpha\beta$ uses the same knowledge as the other algorithms for move generation and evaluation.

The window search for all the algorithms has been set to $[WonByKo$ - 1, $Won]$ as they all try to prove the life of the groups.

The machine used is a 1.7 GHz Pentium with 100 Mb of memory. The transposition table contains 16384 entries. All the algorithms use the transposition move, two killer moves and the history heuristic, in this order, at all nodes, so as to have a good move ordering. $\alpha\beta$ uses iterative deepening.

Table 1 gives the number of problems solved and the total time used for the fifty problems under different time constraints. For each algorithms, three different time constraints have been tested. The thresholds used are 0.1 second, 1 second and 10 seconds. All algorithm stop search as soon as the time threshold is reached, or before the threshold when the result is not modifiable.

The 0.1 second threshold per problem fits well for Go programs based on a global selective search where many positions including some life and death problems have to be evaluated for deciding a move[7]. The 1 second threshold is better associated to program based on the evaluate and play architecture, which allocates more time for the accurate evaluation of the position [3]. The 10 seconds threshold is given to show the behaviour of the algorithms with more time.

gts(0) does not figure in the table 1 because it is equivalent to iw(0).

The result of 250 seconds for $\alpha\beta$ with 25 problems solved, may seem strange, but it is due to one problem quickly solved by the $\alpha\beta$ but incorrectly labeled as won, whereas it is won by ko. The 24 other unsolved problems account for 240 seconds, the other 10 seconds is the time used for the 25 solved problems and the incorrectly assessed one.

For all algorithms, only one problem is incorrectly labeled by all the algorithms, the other unsolved problems are all due to a lack of problem solving time and not to an incorrect answer. The incorrectly labeled problem is due to the evaluation function that does not recognize ko, not to the search algorithm which can handle kos.

The results of the table 1 show that for all threats, the Iterative Widening algorithm is better than the Generalized Threats Search (it solves more problems in less time). The Generalized Widening algorithm is clearly better than the Iterative Widening algorithm (except for small occasional overheads, gw usually solves more problems in less time than iw for all threats). Especially, $gw(3)$ solves more problems in less time than $\alpha\beta$ and $iw(3)$. The $\alpha\beta gw(3)$ generalized widening algorithm appears to be the algorithm of choice, it solves more problems than $\alpha\beta$ in less time for all time constraints.

Table 1. Different algorithms with different time constraints.

	$< 0.1s$		$< 1s$		$< 10s$	
$Algorithm$	$solved$	$time$	$solved$	$time$	$solved$	$time$
$\alpha\beta$	13	3.69	21	30.71	25	250.50
$\alpha\beta gw(4)$	19	3.46	24	26.89	26	235.91
$\alpha\beta gw(3)$	19	3.43	24	26.89	27	235.50
$\alpha\beta gw(2)$	19	3.46	23	27.26	26	238.46
$\alpha\beta gw(1)$	20	3.38	21	28.75	24	262.02
$\alpha\beta gw(0)$	14	3.72	19	32.57	25	252.92
$gw(3)$	18	3.51	24	26.30	26	198.46
$gw(2)$	17	3.35	22	20.08	24	131.61
$gw(1)$	18	2.78	21	10.94	21	40.79
$gw(0)$	16	1.31	18	3.66	18	14.39
$iw(3)$	16	3.56	21	29.04	23	228.10
$iw(2)$	16	3.30	22	20.26	22	145.93
$iw(1)$	16	2.66	21	11.25	21	36.93
$iw(0)$	16	1.04	18	3.32	18	13.17
$gts(3)$	10	4.06	12	36.37	19	288.66
$gts(2)$	11	3.64	18	25.31	21	176.57
$gts(1)$	11	2.96	20	13.40	21	64.26

The $\alpha\beta gw(4)$ algorithm has been tested because it is the extension of the $gw(3)$ algorithm. The same threats as in the $gw(3)$ algorithm are used in the same order, but when no threats work, it reverts to $\alpha\beta$ instead of stopping the search. It is interesting to note that $\alpha\beta gw(4)$ performs slightly worse than $\alpha\beta gw(3)$ for the ten seconds threshold. It might be the case that the overhead of verifying the threat number 3 (i.e. the (6,3,2,0) threat) is too large for the unsolved problem compared to simply searching all the moves selected by the $\alpha\beta$. The choice of using high order threats versus the choice of using all the moves selected by the $\alpha\beta$ is driven by two factors. The first factor is the selectivity obtained in the move generation for the $\alpha\beta$. In the case of open life and death problem, we have designed a quite selective move generator, therefore the difference between the set of moves generated by the high order threats and the one generated by the selective move generator is not so large. The second factor is the time needed to verify the high order threats, which is the overhead of using threat based move selection. We have not used abstract knowledge related to the threats for the move generation in the threat verification, therefore the threat based move selection is relatively slow. The combination of these two factors might explain why using higher order threats does not always improve the $\alpha\beta gw$ algorithm, and particularly in the case of the (6,3,2,0) threat.

In the experiments of table 1, the total time is dominated by the unsolved problems. This is realistic for approximating the behaviour of the algorithms in a Go program. It is less relevant to compare the merits of the different algorithms. So we did another experiment which consisted in restricting the problems to the 27 solvable problems, and then comparing the time and number of moves used by the different algorithms to solve these problems. For each algorithm, we ran it with a time limit of 100 seconds. The number of moves for threat based algorithms includes the moves played in the main search, the moves used to verify the threats, the moves played to generate possible moves, and the moves played in the evaluation function. Concerning $\alpha\beta$ the number of moves includes the same numbers as for threat based algorithms except that it does not verify threats.

Table 2 shows that Generalized Widening is superior to Iterative Widening, which is in turn superior to simple Generalized Threats

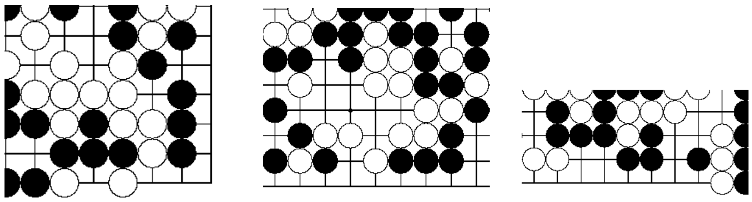

Figure 1. Some problems of the test suite

Table 2. Timing the different algorithms.

$Algorithm$	$solved$	$time$	$\#moves$
$\alpha\beta gw(3)$	27	15.40	916458
$gw(3)$	26	18.71	1108289
$\alpha\beta$	27	50.80	7266459
$iw(3)$	26	116.61	12040874
$gts(3)$	23	> 463.36	> 39859002

Search. It also shows the superiority of Generalized Widening on $\alpha\beta$ with the same knowledge: $\alpha\beta gw(3)$ solves the problems three times faster than $\alpha\beta$. This result is obtained without giving any special abstract knowledge to the threat based algorithm, which is remarkable as giving it such knowledge in other games gave speed-ups of an order of magnitude [5]: the potential of the threat-based algorithms is underestimated in these experiments.

The gw(3) algorithm is also much faster than $\alpha\beta$ with the same knowledge, but it fails to solve one problem because in this problem, the opponent can find moves that lead to min nodes where the (6,3,2,0) threat is not verified, and the result for this problem is therefore NoThreat.

The gts(3) algorithm only solves 23 problems because 3 problems reach the 100 seconds time limit without finding the solution. Given more time it would solve 26 problems as the iw(3) and the gw(3) algorithm.

7 FUTURE WORK

The current approach tests all possible max moves at max nodes, and cuts min nodes if the threat is not verified. It is interesting to test for threats to kill at max nodes, and to test Generalized Widening also at min nodes on the threats to kill.

The program currently heuristically selects min moves in the $\alpha\beta$. This heuristic function for selecting min moves is also used to select min moves in the threats. The selectivity of min moves can be safely improved by only selecting the min moves associated to the trace of the verified threat.

Another possible improvement concerns the evaluation function. Life can be detected earlier by recognizing larger eyes and using patterns.

The selectivity of max nodes moves can also be improved. For example, in the threat based search, except for order one moves, the max nodes moves are the same as the max nodes moves of the $\alpha\beta$. It is possible to be more selective for order two moves, only considering eye making moves and moves threatening to make an eye near a string that already has an eye.

More generally, it is possible to be more selective in the threat based move generation by increasing the knowledge about the abstract properties of the game. Abstract game knowledge about moves generated in the threats and in the main search has been shown to be an important factor for the speed of threat based search algorithms [5]. It can give a large speed-up for $\alpha\beta gw$, gw, iw and gts to give this knowledge to the program.

The combination of Generalized Widening with the idea of Df-pn search could also lead to substantial improvements.

Testing Generalized Widening in other games also looks promising.

8 CONCLUSION

We have shown that Generalized Widening can be associated to the $\alpha\beta$ algorithm to improve problem solving for open life and death problems in the game of Go. The resulting algorithm is faster and solves more problems in less time than $\alpha\beta$ alone and that other related threat-based algorithms.

REFERENCES

[1] L.V. Allis, M. van der Meulen, and H. J. Herik, 'Proof-number search', *Artificial Intelligence*, **66**(1), 91–124, (1994).

[2] B. Bouzy and T. Cazenave, 'Computer Go: An AI-Oriented Survey', *Artificial Intelligence*, **132**(1), 39–103, (October 2001).

[3] D. Bump, G. Farneback, et al., 'gnugo', web page, http://www.gnu.org/software/gnugo/, Free Software Fondation, (1999-2004).

[4] T. Cazenave, 'Iterative Widening', in *Proceedings of IJCAI-01, Vol. 1*, pp. 523–528, Seattle, (2001).

[5] T. Cazenave, 'Admissible moves in two-player games', in *SARA 2002*, volume 2371 of *Lecture Notes in Computer Science*, pp. 52–63, Kananaskis, Alberta, Canada, (2002). Springer.

[6] T. Cazenave, 'A Generalized Threats Search Algorithm', in *Computers and Games 2002*, volume 2883 of *Lecture Notes in Computer Science*, pp. 75–87, Edmonton, Canada, (2002). Springer.

[7] K. Chen and Z. Chen, 'Static analysis of life and death in the game of Go', *Information Sciences*, **121**, 113–134, (1999).

[8] D. Fotland, 'Static eye analysis in "The many faces of Go"', *ICGA Journal*, **25**(4), 203–210, (2002).

[9] A. Kishimoto and M. Müller, 'Df-pn in Go: an application to the one-eye problem', in *Advances in computer games 10*, pp. 125–141, (2003).

[10] M. Müller, 'Computer go', *Artificial Intelligence*, **134**(1-2), 145–179, (2002).

[11] A. Nagai, 'Df-pn algorithm for searching AND/OR trees and its applications', Phd thesis, Department of Information Science, University of Tokyo, (2002).

[12] M. Seo, 'The C* algorithm for AND/OR tree search and its application to a tsume-shogi program', Master's thesis, Departement of Information Science, University of Tokyo, (1995).

[13] B. Sheppard, 'Endgame play in scrabble', *ICGA Journal*, **26**, 147–165, (September 2003).

[14] T. Wolf, 'Forward pruning and other heuristic search techniques in tsume go', *Information Sciences*, **122**, 59–76, (2000).