Gradual Abstract Proof Search

Tristan Cazenave[1]

Labo IA, Université Paris 8, 2 rue de la Liberté, 93526, St-Denis, France

ABSTRACT

Gradual Abstract Proof Search (GAPS) is a new 2-player search technique. It has been used to prove that 11x11 Phutball is a win for the first player in 25 plies or less, that 6x6 Atari-Go with a crosscut in the center is a win for the first player in 15 plies or less. I give domain dependent optimizations for the two games. I also describe theoretical and experimental comparisons with other related algorithms.

1. INTRODUCTION

Recently, selective proof search algorithms that preserve the correctness of the solution have been applied quite successfully to the game of Go (Cazenave, 2001a; Cazenave, 2001b). The usefulness of this family of search algorithms in other games has yet to be explored. Our goal is to understand the kind of games and problems that can benefit from such selective proof search algorithms. This goal can be achieved applying the algorithm to different games and problems. We have tested such a search algorithm for the game of Phutball and the game of Atari-Go. In Winning Ways (Conway, Berlekamp, and Guy, 1982), the final conclusion of the section on Phutball is 'Like Chess and Go, and unlike most of the games in this book, Phutball is not the kind of game for which one can expect a complete analysis'. This assertion was an incentive to try to solve it with a recent selective game proof search algorithm.

In the second section, we explain the game of Phutball. In the third section, we explain the game of Atari-Go. In the fourth section, we compare GAPS with related selective proof search algorithms. In the fifth section, we give some optimizations of the search related to some properties of Phutball. In the sixth section, we give the optimizations used to solve 6x6 Atari-Go. The seventh section details experimental results. The eighth section outlines future work and the last section concludes.

2. PHUTBALL

The game of Phutball was invented by J. H. Conway. It is described in Winning Ways (Conway *et al.*, 1982). The game is usually played on a 19x15 grid (or a 19x19 Go board). The ball is represented by one black stone which is at the center of the board at the beginning of the game. White stones represent men. Initially the board is empty except for the ball. The two players are named Left and Right. There are two types of legal moves: (i) placing a new man at an empty intersection or (ii) moving the ball, jumping over the neighboring men, and removing the men jumped over. If many men are adjacent on the same line, they can be jumped over, and the whole line is removed. The ball is allowed to make many jumps in the same move. The Left player's goal is to put the ball on the rightmost line of the board, or to drop it after this line. The Right player tries to put the ball on or after the leftmost line of the board. Each player moves in turn with Left beginning. The two players have the same material and the same legal moves. A player either places a man or moves the ball, he cannot do both.

On an 11x11 board, the initial position consists of an empty board with the ball (a black Go stone) in the middle of the board (the 6,6 or F6 intersection).

[1]email:cazenave@ai.univ-paris8.fr

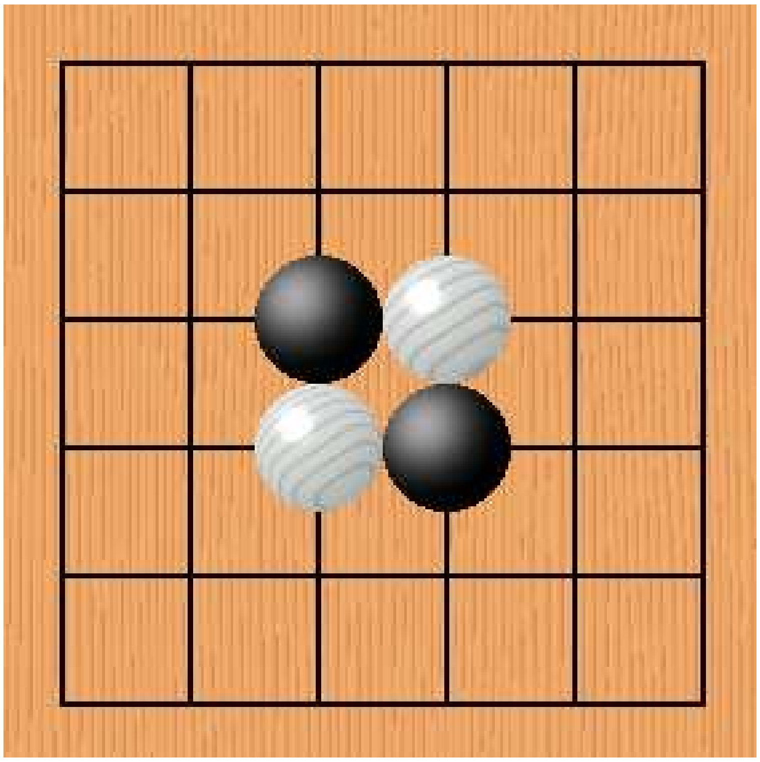

Figure 1: The starting position for 6x6 Atari-Go with a crosscut.

The complexity of Phutball has not yet been proved. However, even determining whether the current player has a move that immediatly wins the game has been proved to be NP-complete (Demaine, Demaine, and Eppstein, 2002). The number of possible moves on an 11x11 board is 120 on average. And the depth of the solution we have found is 25. Phutball is typically the kind of game that is hard to solve using traditional search algorithms, such as brute force Alpha-Beta, that consider all possible moves at each node.

3. ATARI-GO

Atari-Go is used to teach beginners to play the game of Go. The goal is to be the first to capture a string. It can be played on any board size. It is usually played on a small board so that games are not too long. Teacher also often choose to start with a crosscut in the centre of the board in order to have an unstable position. GAPS solves the version with a crosscut in the centre on a 6x6 board. The starting position is shown in the Figure 1.

The rules are similar to Go, Black begins, Black and White alternate playing stones on the intersections of the board, strings of stones are formed by stones of the same color that are linked by the lines on the board. The number of empty intersections neighbor to a string is the number of liberties of the string. A string is captured if it has no liberty. For example in the Figure 1, the four strings all have two liberties. A string that has only one liberty left can be captured by the other color in one move, it is in Atari, this is where the name of the game comes from.

4. SELECTIVE PROOF SEARCH ALGORITHMS

Usually, selective search makes search results less reliable. The family of algorithms we are interested in are very selective but they ensure the correctness of the solution, when they find a solution. These safe and selective search algorithms can greatly outperform Alpha-Beta in games with a high branching factor, and with easy to define abstract properties on the minimum number of moves in a row needed to acomplish a goal (Cazenave, 2001b). For example in the game of Go, such a property is the number of liberties: at least n moves by the opponent are needed to capture a string that has n liberties. In this section we present the selective proof search algorithm we have used. We start with Threat Space Search, then we redefine Abstract Proof Search. We continue with Lambda Search, a strongly related algorithm that behaves differently. Then we describe the iterative widening optimization. The next subsection deals with a modification of the previous definitions of the game functions used to select forced moves at the Min nodes of the search tree. The games are defined more gradually than in previous algorithms. It enables to treat efficiently cases that are not handled easily with the games as they are defined in Abstact Proof Search and Iterative Widening or as with lambda trees in Lambda Search. The final subsection gives the search algorithm based on the gradual games that solved 6x6 Atari-Go and that was decisive in solving 11x11 Phutball: the Gradual Abstract Proof Search algorithm.

4.1 Threat Space Search

Go-Moku has been solved by V. Allis et al. using a selective proof search algorithm based on threats and proof number search for the main search when no threats are available (Allis, van den Herik, and Huntjens, 1996). The threats are given names that correspond to patterns: Four, Straight Four, Three, Five. APS, GAPS and lambda search are a generalization of Threat Space Search. They are based on tree search to find threats of increasing orders instead of fixed patterns.

4.2 Abstract Proof Search

Abstract Proof Search (Cazenave, 2001b) is a very selective search algorithm that ensures that winning moves are correct. It is much faster than brute force Alpha-Beta. It consists in developing small search trees at the Min nodes of the main search in order to select the interesting moves or to decide to stop search. Given that Left plays at Max nodes, and Right at Min nodes, an Abstract Proof Search of order one consists in verifying at each Min node if the Left player can win in one move. If it is not the case, the search is stopped and the Min node is labeled as lost for Left. Otherwise, if Left can win in one move, only the Right moves that can prevent from winning in one move are considered and tried at this node. A position that is won in one move for Left is labeled as Win-1 for Left, and Forced-1 for Right. The search of order one consists in developing small 'trees' with one Left move at each Min node. A search of order two consists in developing trees with two Left moves (depth 3 plies trees) at each Min node. A search of order three consists in developing small trees at each Min node with at most three Left moves from the root in the small tree (depth 5 plies search trees at each Min node).

4.3 Lambda Search

Lambda Search (Thomsen, 2000) is a search algorithm that has strong links with Abstract Proof Search. It can be defined using lambda trees and lambda moves. A lambda tree of order n is a search tree that contains lambda moves of order n. A lambda move of order n for the attacker is a move that implies that there exist at least one subsequent winning lambda tree of order strictly inferior to n. A lambda move of order n for the defender is a move that implies that there is no winning tree of order strictly inferior to n.

Abstract Proof Search imposes limits on the depth and the order of the trees developed at each node, whereas Lambda search imposes limits on the global order and on the global depth of these trees. Abstract Proof Search relies more on abstract properties of the game to select a few interesting moves and reduce the number of moves to look at for each order. Apart from these distinctions, they are based on similar principles.

4.4 Iterative Widening

The Iterative Widening algorithm (Cazenave, 2001a) consists in performing a full Abstract Proof Search at a given order, before increasing the order of the search. It has been successfully tested on the capture game in the game of Go. Practically, it consists in trying an order one Abstract Proof Search, and if it fails in trying an order two search, and if it fails in trying an order three search. And so on until the time alloted for the search is elapsed. It gives a speed-up of two for the capture game of the game of Go.

4.5 Gradual Games

Gradual games are designed to implement a simplest-first proof strategy. They enable to bound a search tree to a pre-defined tree which verifies that winning the game can be proved with the pre-defined search tree. If the attacker cannot win a position by playing a move followed by a bounded pre-defined search tree starting with another move of the attacker (a gradual game), then the position is considered too complex and not searched anymore. Gradual games have different sizes and complexities.

The Figure 2 gives some examples of trees representing different gradual games. The two players are named Left and Right. Left tries to win the game and Right tries to prevent Left from winning. A game is a Win game when it is associated to winning Left moves. Each Win game is also a Forced game, except that Forced games are associated to Right moves to prevent Left from winning. In these trees, a branch that goes on the left represents a Left move, and a branch that goes on the right represents some Right moves. Left branches are associated with winning moves for Left, and right branches are associated to the complete set of Right moves that can possibly prevent the win of the corresponding left branch (the left branch directly at the left of the right one with the same parent). All the leaves of the trees are positions won for Left. In order for the game to be verified, all Left moves have to be winning moves, and all Right moves have to be refuted by Left.

The tree labeled Win-1 in the Figure 2 is the most simple one. The Win-1 game is verified when Left can win in one move. The Won-1 tree represents a position where Left has won in one move even if Right plays first. The left branch below the root represents the winning move for Left if he plays first, and the right branch below the root is followed by a left branch. It means that all the Right moves that prevent the first winning Left move are followed by another winning Left move, and are therefore failing to prevent Left from winning. The next tree is labeled Win-21, it is a position where Left can move to a Won-1 type position. A Win-21 game means that Left can win in at most two moves.

The last four gradual games in the Figure 2 are more interesting. They show the distinction between gradual games and the classification of games based on the same order and depth as in Abstract Proof Search, or on the global order and depth as in Lambda Search. The Won-211 game is verified in a position where Left can win in one move, and where all the Right moves that can possibly prevent this winning move lead to a Win-21 position. The Figure 4 gives such a position. Left can win if he plays first by jumping the ball over the three white stones up to the H6 intersection, then jumping again in the same move to the K8 intersection, and eventually scoring a goal by jumping over the white stone at L9. In this position, the only move that can prevent this one ply win for Left is a ball move. The right branch below the root of the Won-211 tree represents the Right moves that may prevent the Left win in one ply: all the ball moves and the stone placements on the path of the ball to the goal. None of the stone placements prevents the win in this position (in some other positions, they can be useful). The only ball move that is not followed by a winning Left move is jumping the ball to the F8 intersection (marked with a 1 in the Figure 4). This is the best move for Right as it is the one that maximizes the depth of the Left win. Following this Right ball move comes the Left placement at G7 marked with a 2 in the Figure 4 (the white stone has been removed by the precedent ball move, therefore Left can put a stone again at G7). It corresponds to the second left branch in the Won-211 tree. The position is Won-1 for Left after this branch (the subtree below the branch is a Won-1 tree).

A Win-311 game is verified when a Left move leads to a Won-211 game (as can be seen on the Figure 2 where the subtree following the rooted left branch is a Won-211 tree).

The name of the gradual games are given as follow: the first number (2 in case of a Won-211 game) gives the maximum number of Left moves before the win, the subsequent numbers give the maximum order of the game associated to the Right moves when Right plays optimally (forcing Left to play the longest sequence).

The Win-311 and the Win-321 games have the same depth, but are different gradual games. In the Abstract Proof Search and in the Iterative Widening algorithms there was non distinction between the two: they both belong to depth five plies trees of order 3. However they do not have the same complexity. The Win-311 tree is faster to check than the Win-321 one as the the left game following the rooted left branch in the Win-321 game is Win-21 and not Win-1 as in the Win-311 game. In practice there are more moves preventing a Win-21 game than a Win-1 one, and the Win-21 games take more time to check.

This observation is amplified in greater order of games. For example the Win-4111 game is checked very rapidly, whereas a full depth seven tree search corresponding to a Win-4321 game can take a relatively very long time to be checked.

The most complex gradual game used to solve 6x6 Atari-Go is the Win-4221 game. It is drawn in the Figure 2. The dashed arrows indicates the nodes of the tree where the 221 in the name of the gradual game comes from. It means that after the forced move there is a Forced-21 game verified for Right, then after all Right moves, there is always a Left move that lead to another Forced-21 game, and after all the Right Forced-21 moves, there is still always a Left move that leads to a Forced-1 game.

The Figure 3 gives examples where some of the gradual games are verified in Atari-Go.

A Win game always starts with a rooted left branch and no rooted right branch. All the Right moves that can prevent a Win game are associated to the corresponding Forced game. For example, the Forced-311 moves are the Right moves that prevent a Win-311 game for Left. The possibly preventing Right moves can be found by memorizing the intersections involved in the search of the corresponding Win game. It is described in the section on optimizations related to Phutball, in the subsection on finding forced moves.

4.6 Gradual Abstract Proof Search

Gradual Abstract Proof Search consists in iteratively widening the scope of the gradual games, instead of widening the games based on the depth of the games as in (Cazenave, 2001a). The order we use for widening is Forced-1, Forced-21, Forced-311, Forced-4111, Forced-51111, Forced-321, Forced-4221. It is a generalization of Iterative Widening. The Iterative Widening algorithm can be stated as a Gradual Abstract Proof Search using the order Forced-1, Forced-21, Forced-321, Forced-4321 etc... for the gradual games. The Forced-311, Forced-4111 and Forced-51111 games have respectively depth 5, 7 and 9, but are rapidly verified as they represents very narrow trees. On the contrary, Forced-321 is slower to verify mainly because there are much more Right moves to refute after the Left move at the root: the order of the second moves of the Forced-321 tree is two whereas the order of the second moves of the Forced-21, Forced-311, Forced-4111 and Forced-51111 trees is only one.

Gradual Abstract Proof Search mixes the good properties of Abstract Proof Search and Lambda Search. Instead of only selecting moves mainly based on the depth as in Abstract Proof Search, or based on the order as in Lambda Search, it selects the moves using these two criteria. It enables more control on the search behavior than the two other algorithms. It can also easily use abstract properties of a game in order to be very selective on the moves to try as in Abstract Proof Search.

It is possible to put a global depth limit to Lambda Search and to use iterative deepening on the depth. This is quite different of what we propose. We rather put limits on the shapes of the trees that can be searched at each Min node of the overall search. When choosing the gradual games appropriately, GAPS can model either Iterative Widening or Lambda Search. It is a generalization of both. The nice property of GAPS is that it enables more control on the search than previous algorithms.

The order of the gradual games was chosen according to their estimated average verification time.

5. OPTIMIZATIONS RELATED TO PHUTBALL

In this section, we give some Phutball specific optimizations. However the ideas behind many of the optimizations can prove useful in other problems. We start by analyzing the properties of the winning moves in Phutball related to the depth of the win. Then we explain how forced moves (a restricted number of Min nodes moves that prevent the opponent from winning) can be selected. Other optimizations are used at Max nodes, this is what the third subsection is about. Eventually, the last section describes the overall search algorithm used to solve 11x11 Phutball.

5.1 Properties of the game

A winning move in Phutball can only be a ball move. Therefore, when trying to prove that a position can be won in one move, the only moves to look at are the ball moves. A consequence of this is that a winning move

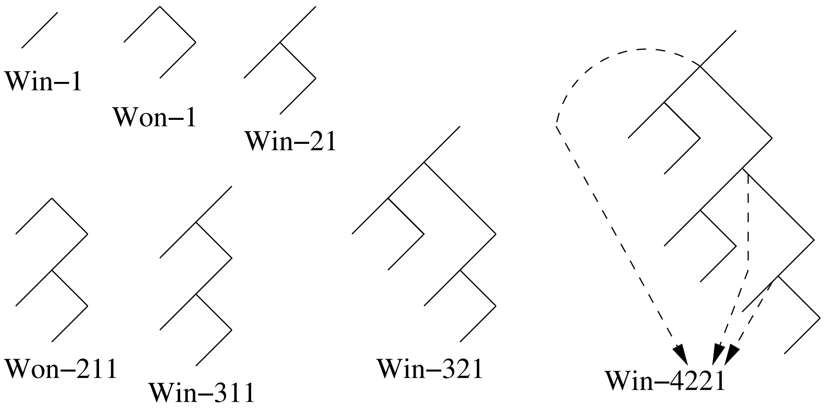

Figure 2: Some trees representing gradual games.

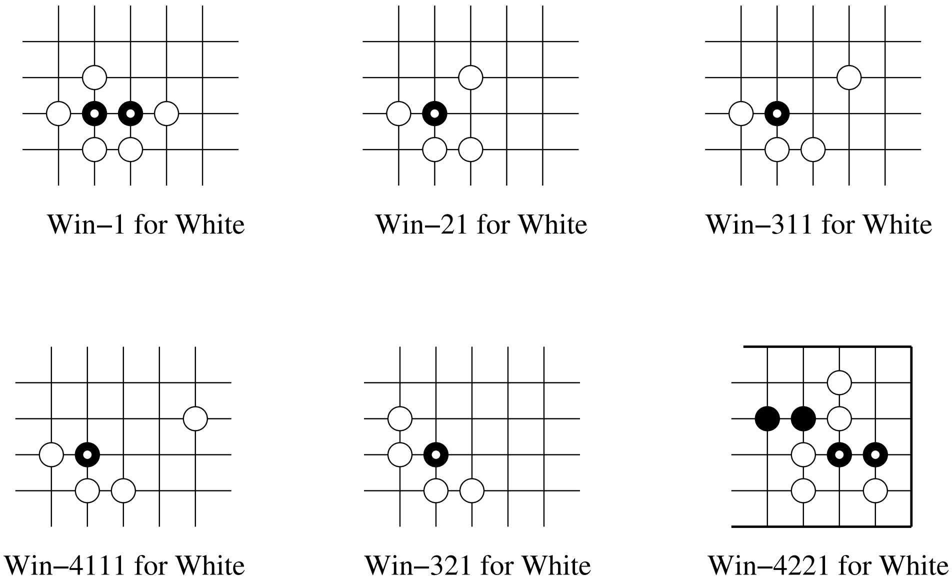

Figure 3: Example of gradual games for Atari-Go.

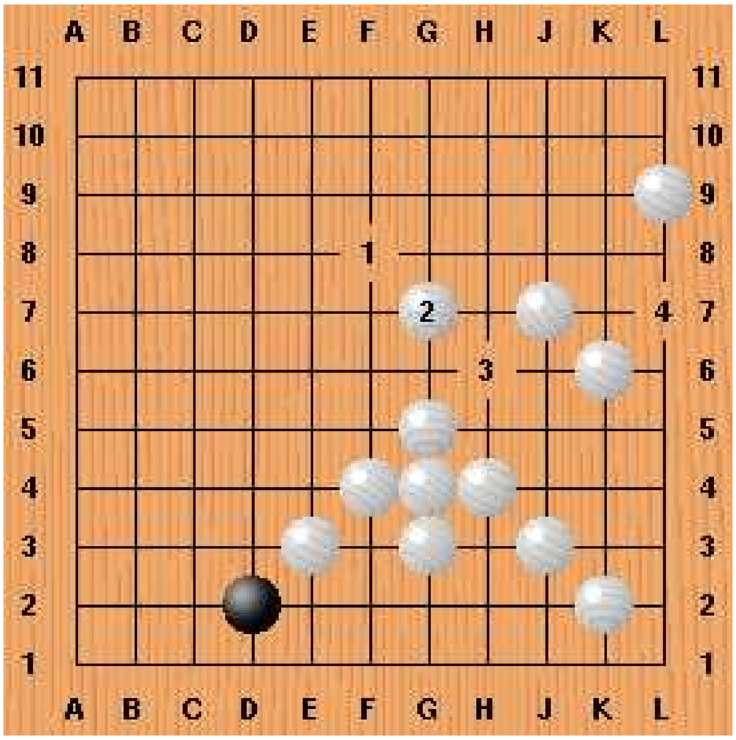

Figure 4: The Won-211 game for Left is verified in Phutball.

in three plies is always a stone placement. It cannot be a ball move, as the depth three move would also be a ball move, and in this case the position could be won directly in only one ply. An optimization taking into account this property of the game is to try only stone placements that are on the intersections reachable in one move by the ball and their neighbors, when searching a three plies deep tree.

5.2 Finding forced moves

When a search is performed, all the intersections involved in the search are memorized in a bitmap. An intersection is involved in a search if a stone has been put on it during the search, or if the ball has moved on the intersection. Once this bitmap is available for a given search, it is dilated, adding to the bitmap all the intersections that are neighbors of the bitmap. The complete set of moves that can possibly invalidate the result of the search are all the possible ball moves, plus all the stone placements on the empty intersections of the dilated bitmap.

A special optimization is used for forced moves when the opponent can win in one move (Forced-1 games). In this case, the only stone placements that need to be considered are the stone placements on the empty intersections of the ball path to the goal. All other stone placements are useless, and cannot prevent the opponent from winning. Of course, all the ball moves have to be taken into account to try to prevent a one move win.

5.3 Max nodes moves

Only a restricted number of moves are tried at the Max nodes of the search tree. An example is given in the Figure 5 of the Max nodes moves for Left considered in this position (the empty intersections involved in the stone placements are marked with a triangle). The only moves tried at a Max node are (i) the stone placements at an intersection neighboring the ball, which is closer to the goal line than the ball (stone placements at C8, C6, D8 and D7 in the Figure 5), (ii) the stone placements on the empty intersections involved in all ball paths (stone placements at F4, D2 and H6 in the Figure 5), (iii) the stone placements on the empty neighbors of the empty intersections involved in all ball paths that are the closest to the goal line (stone placements at J5, J6

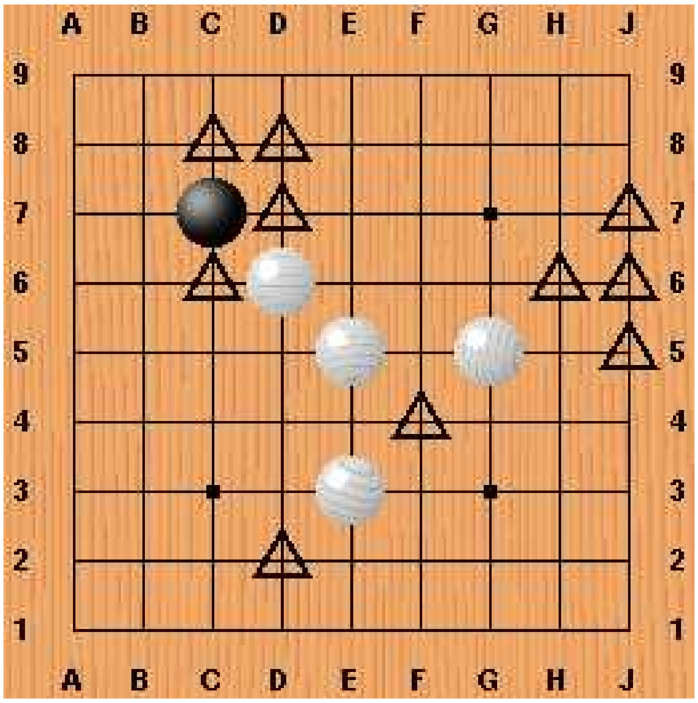

Figure 5: Possible Max node moves in Phutball.

and J7 in the Figure 5), and (iv) all the ball moves (at F4, D2 and H6 in the Figure 5).

This heuristic for Max node moves is also used when verifying the gradual subgames defined in the previous section.

5.4 Main Search for Phutball

The main search consists in an iterative deepening search with all AND moves, as long as no pre-defined gradual game is verified, and only the Max node moves advised by our heuristic. Once a game has been verified at a given depth, the algorithm switches to a gradual proof search for this depth. Gradual proof search can be considered as a kind of generalized quiescence search for the main search.

Transposition tables are used in the main search, but only to find the move to try first if the position has been already searched before.

6. OPTIMIZATIONS USED FOR ATARI-GO

6.1 Optimizations of Alpha-Beta

I have previously used a simple Alpha-Beta with iterative deepening and transposition tables for Atari-Go (Cazenave, 2002). It does not solve 6x6 Atari-Go in a reasonable time. Erik Van der Werf has used other usual Alpha-Beta optimizations that improve a lot the solving of Atari-Go (van der Werf, 2002). I improved the Alpha-Beta algorithm with similar optimizations, and it also improved the behavior of the GAPS algorithm which is based on Alpha-Beta.

The optimizations used are the use of transposition tables, containing the score and the best move. The memorization and the use of two killer moves after the transposition move. The history heuristic with a weight of 2^{Depth}. An incremental evaluation function which computes the difference between the number of liberties of the black string that has the less liberties and the number of liberties of the white string that has the less liberties. The number of liberties of strings are updated incrementally too.

More optimizations can lead to even better results for Alpha-Beta according to (van der Werf, 2002).

6.2 Optimizations of GAPS for Atari-Go

Verifying complex gradual games such as Forced-4221 can take a relatively long time. Abstract knowledge can be used to detect early that a gradual game cannot be verified, for example when the number of admissible moves to win is greater than the order of the game. For example in AtariGo, it means that no order 2 game can be verified when the minimum number of liberties of all the defender strings is 3.

Abstract game knowledge is also used to select a few candidate moves, that are the only relevant moves to capture a string given some order.

The abstract knowledge of order one is the knowledge to optimize the generation of possible moves when looking for a winning move. It consists in testing if there is an opponent string with only one liberty left. The attacker move generator returns the liberty if it is the case, and returns an empty set of moves when there is no such string. The defender move generator of order one first tries to capture an opponent string with only one liberty, and if there is none, it looks for friend strings with only one liberty. If there is one such string, it plays its liberty. If there is none, it returns an empty set of moves.

The order two abstract knowledge for the attacker move generator consists in sending back the liberty of a defender string if it has only one liberty, otherwise in sending back an empty set if the minimum number of liberties of opponent strings is greater than two, else to send back the liberty of a friend string with only one liberty, else to send back the liberties of the opponent strings with two liberties, and if this last condition is not verified to send back an empty set.

The order two abstract knowledge for the defender move generator is almost the symmetric of the attacker move generator, except when there is a defender string to save with two liberties. In this case the possible moves are the liberties of the defender string, the empty neighbors of the liberties of the defender string, the liberties and the empty neighbors of the liberties of the strings that have two liberties and which are also adjacent to the defender string, the liberties of the strings that have three liberties and which are also adjacent to the defender string and the liberties of the attacker strings that have two liberties.

The order three abstract knowledge for the attacker move generator is programmed in a similar way as the order two abstract knowledge, except that all the empty intersections that can be connected in two moves to the defender string are sent back when the defender string has only two liberties. Only the liberties are sent back when the defender string has three liberties.

There is no order three abstract knowledge for the defender move generator, it sends back all possible moves.

7. EXPERIMENTAL RESULTS

7.1 Phutball

We have tested our algorithm on an Athlon 1.7 GHz. It has solved 9x9 and 11x11 Phutball. The shortest win for the 9x9 board with the gradual search algorithm is given in the Figure 6. Placement of white stones are noted 'p(Intersection)'. Ball moves are noted 'b(Intersection)', Intersection is the location of the intersection where the ball finishes its last jump. With this notation, the solution of the 9x9 board is: p(F5), p(D6), p(H6), b(C7), p(D6), p(E5), p(G5). At this point a depth 4 won gradual game is verified (for example p(F4), p(H2), p(B8), b(J1)), and the game is won for Left. The length of the solution is 11, and the depth of the search to find this solution is 7. The table 1 gives the time and the number of nodes used to search at a given depth, starting from the initial 9x9 board. The nodes count is the number of nodes of the main Alpha-Beta search, not counting the nodes of the gradual games. Solving 9x9 Phutball with Gradual Abstract Proof Search takes 0.15 seconds.

When solving the 11x11 board, only one Left move is tried at the top of the tree (putting a stone at the intersection G7 in the Figure 7). The complete search to find the solution of the 11x11 board takes 188s. It consists in trying all the Right responses to the Left move at G7. For each response, a gradual search is

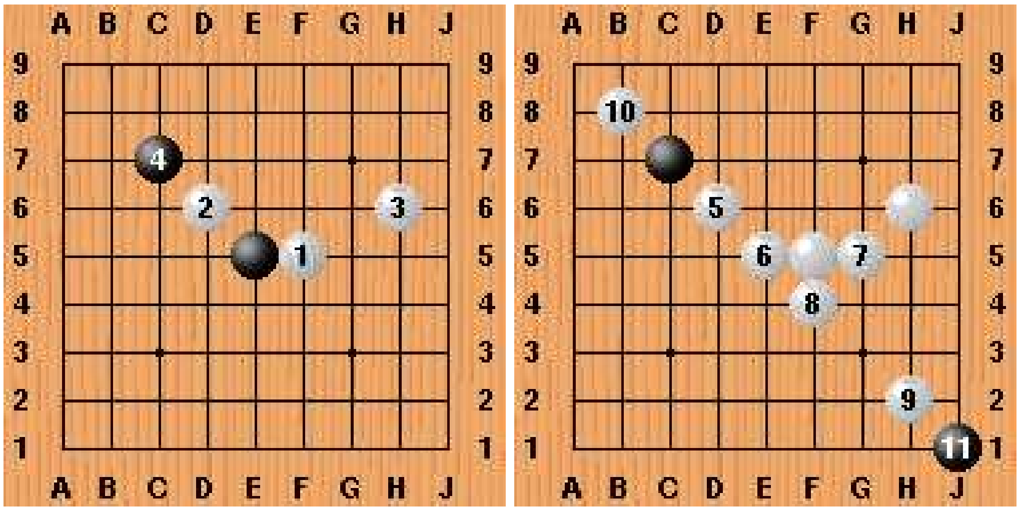

Figure 6: The 9x9 Phutball solution.

Table 1: Time and nodes used to find the 9x9 Phutball solution.

Depth	$Time$	$Nodes$
1	0.00	11
2	0.01	49
3	0.01	131
4	0.01	277
5	0.03	475
6	0.03	876
7	0.06	1180

performed. The game is won for Left. The longest defense for Right the search has found for the 11x11 board is: p(G7), p(G5), p(J7), p(G3), p(K6), b(F2), p(G3), p(G2), p(J5), b(H2), b(F4), p(E3), p(G4), b(D2), p(E3), p(F4), p(H4), p(G3), p(K2), p(G5), p(L9). At this point, a Won-211 game is verified, so the position is won for Left (for example b(F8), p(G8), p(H8), b(L6)). The depth of the search is 21, and the length of the solution is 25. The gradual game used to find this solution is Forced-4111 after p(G7), p(G5). However the Forced-52111 game is needed to find a depth 15 solution after p(G7), p(E7). The beginning of the 11x11 solution is given in the Figure 7. The table 2 gives the time and the number of nodes used to search the solution at a given depth after the p(G7), p(G5) moves.

7.2 Atari-Go

A basic Alpha-Beta with no optimizations (without transpositions tables and killer moves) is not able to solve 6x6 Atari-Go in a realistic time (depth 10 already takes 1923 seconds and the solution is at depth 14), whereas GAPS on top of such an unoptimized Alpha-Beta solves 6x6 Atari-Go in 182 seconds.

6x6 Atari-Go with a crosscut is solved with GAPS on top of an optimized Alpha-Beta in 24.28 seconds and 11,395 Kmoves on an Athlon 1.7 GHz as shown in the table 4. The solution is found at depth 10 where a gradual game of depth 5 is verified. The Alpha-beta that uses the same optimizations, but no gradual games goes depth 14 and takes 1208 seconds and 793081 Kmoves as shown in the table 3. In these two tables, we count all the moves played during the search in the NbMoves column, even the moves played in the gradual games for GAPS.

The maximum game used to solve 6x6 Atari-Go is the Forced-4221 game. The solution found by GAPS is depicted in the Figure 8.

Another experiment was run adding a quiescence search for 6x6 Atari-Go. The quiescence search consists in

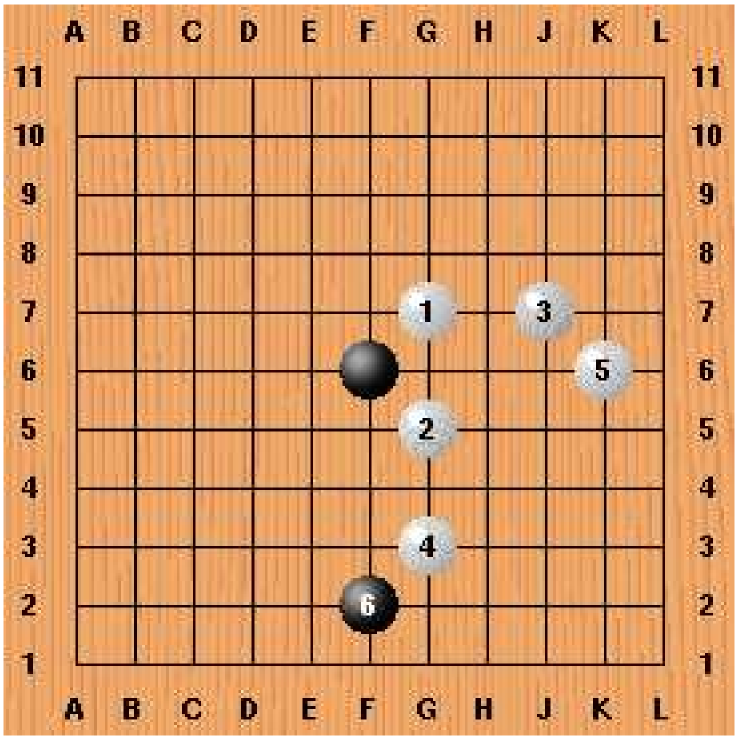

Figure 7: The beginning of the 11x11 Phutball solution.

Table 2: Searching the 11x11 Phutball board after p(G7), p(G5).

Depth	$Time$	$Nodes$	$Depth$	$Time$	$Nodes$
1	0.14	28	11	2.21	11352
2	0.14	48	12	1.13	10755
3	0.48	161	13	21.90	114150
4	0.32	324	14	1.68	15894
5	0.37	392	15	5.15	27660
6	0.37	1059	16	2.09	18104
7	0.46	1376	17	4.53	24379
8	0.48	3265	18	3.50	32832
9	1.41	7199	19	5.98	28942
10	0.79	6249			

Table 3: Solving 6x6 Atari-Go with Alpha-Beta.

Depth	$Evaluation$	$BestMove$	$Time$	$NbMoves$
1	1	D5	0.00	32
2	0	D5	0.00	142
3	1	D4	0.00	1233
4	0	D4	0.01	3176
5	1	D4	0.04	25661
6	0	D4	0.13	71249
7	1	D4	0.75	536k
8	0	D4	1.15	704k
9	1	D4	8.41	5938k
10	0	D4	14.89	9342k
11	1	D4	73.38	49436k
12	0	D4	134.59	85830k
13	1	D4	821.52	549841k
14	500	D4	153.18	91347k

Table 4: Solving 6x6 Atari-Go with GAPS.

Depth	$Evaluation$	$BestMove$	$Time$	$NbMoves$
1	1	D5	0.00	638
2	0	D5	0.72	359k
3	1	D5	0.92	461k
4	0	E3	1.85	899k
5	1	E3	2.28	1083k
6	0	E3	4.55	2088k
7	1	E3	4.67	2208k
8	1	E3	8.05	3710k
9	2	E3	1.08	501k
10	500	E3	0.20	83262

reading ladders. It continues to search if the opponent has less than two liberties on any of his strings. The quiescence search only considers one or two move at each node. The quiescence search has to be tried at every node of the search tree in order to have its maximum efficiency (even at interior nodes). When the quiescence search returns won, the search is stopped and returns won. With this optimization, the Alpha-Beta is much better. It solves 6x6 AtariGo in 37 seconds and 4349k nodes. The optimization does not help much GAPS, as the corresponding Win-51111 game is already verified and corresponds to a depth 10 ladder. GAPS with ladders takes 21 seconds. The gain of GAPS over Alpha-Beta is much less important with this optimization but still exists.

8. FUTURE WORK

Solving Phutball on a larger size such as 13x13 seems feasible in light of the results presented in this paper. Solving the original 19x15 Phutball may be much harder, but looks closer now with this new family of algorithms. Future work includes testing similar selective proof search algorithms on other games that are combinatorially too hard for usual basic search algorithms.

Another possible source for optimizations is the development of an opening book for Max node moves at the top of the search tree. Currently, for the 11x11 board, the book is composed of only one position: the initial position associated to the p(G7) move. It enables to avoid searching the symetrical p(G5) move that has the same result. The other possible Max node move advised by our Max nodes heuristic is the move p(G6), but it leads to a deep and unsolved search. 11x11 Phutball can be solved without an opening book.

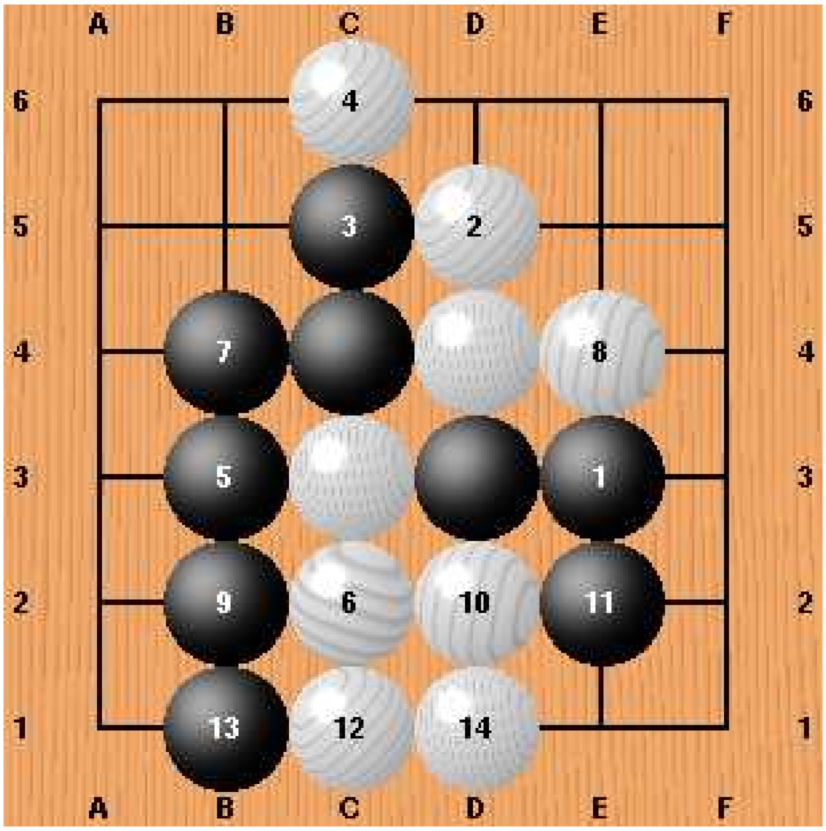

Figure 8: A solution to 6x6 AtariGo.

A more developed opening book that advises top Max node moves could speed up the solving of the 13x13 board. This could also be useful in other games. In order to solve difficult games, a game solving assistant that takes advices from an expert player can be designed by interactively developing the book. The interactive development of the book could be automated asking for Max nodes moves in difficult positions where the solver has failed to solve the game.

The introduction of graduality in the search enables to differentiate between simple and difficult Win and Forced games for a given depth. Whereas previous algorithms only made a difference based on the depth of the search or on the order of the search. It is probably possible to refine the graduality of the algorithm and to explore the different possibilities for defining gradual games, so as to improve the generality and the efficacy of our current implementation.

An example of a generalization of gradual games is to represent games by the different number of moves of the different possible orders. A generalized gradual game is an n-uplet, the element number i is the maximal number of order i moves allowed in a gradual tree for Left. For example a Forced-1 game is represented by (1,0), a Forced-21 game by (2,1,0), a Forced-311 game by (3,2,0) and a Forced-321 game by (4,2,1,0). Using and experimenting with these generalized gradual games, comparing them with lambda search, gradual proof search and testing them with Alpha-Beta and Proof Number search is a source of improvements in algorithms for solving games.

Concerning the integration of GAPS with proof/disproof numbers, two issues can be explored. The first one is of course to replace the Alpha-Beta with a Proof Number Search and compare the results. The second issue is to integrate the notions of proof and disproof numbers deeper in GAPS, for example by using them so as to control the gradual games instead of having a pre-defined tree shape.

Reusing the usual optimizations of Alpha-Beta when verifying the game definitions is also a source of possibly large improvements. As well as refining the abstract knowledge used to select relevant moves. GAPS is currently used on top of an Alpha-Beta, we are working toward a better integration of GAPS and Alpha-Beta so that GAPS can be used directly from the root so as to prove or disprove a position.

The next target games where such improvements will be tested are 8x8 Atari-Go with a crosscut in the center, 13x13 Phutball, the capture, the connection, the eye-making and the life and death subgames of Go.

9. CONCLUSION

We have described GAPS, a selective proof search algorithm that can solve 9x9 Phutball in 0.15 seconds and 11x11 Phutball in 3 minutes on a personal computer. The depth of the solution of 11x11 Phutball is 25, and the average number of possible moves for an 11x11 Phutball position is 120. Solving Phutball on larger boards seems highly possible with this family of algorithms.

GAPS can also solve 6x6 Atari-Go in 21 seconds, using an optimized Alpha-Beta algorithm. It finds a depth 15 solution to the game.

Usual search algorithms that consider all possible moves are much less suited to solve games like Phutball than selective proof search algorithms. Similar algorithms have been used to improve search drastically in the game of Go. The algorithm presented in this paper might well be useful in many other games. It extends previous selective search algorithms such as Abstract Proof Search (Cazenave, 2001b), Iterative Widening (Cazenave, 2001a) and Lambda Search (Thomsen, 2000).

10. REFERENCES

Allis, L. V., Herik, H. J. van den, and Huntjens, M. P. H. (1996). Go-Moku Solved by New Search Techniques. *Computational Intelligence*, Vol. 12, pp. 7–23.

Cazenave, T. (2001a). Iterative Widening. *Proceedings of IJCAI-01, Vol. 1*, pp. 523–528, Seattle.

Cazenave, T. (2002). La Recherche Abstraite Graduelle de Preuves. *Proceedings of RFIA-02*, pp. 615–623, Angers, France.

Cazenave, T. (2001b). Abstract Proof Search. *Computers and Games* (eds. T. A. Marsland and I. Frank), Vol. 2063 of *Lecture Notes in Computer Science*, pp. 39–54, Springer. ISBN 3–540–43080–6.

Conway, J. H., Berlekamp, E., and Guy, R. K. (1982). Philosopher's Football. *Winning Ways*, pp. 688–691. Academic Press.

Demaine, E. D., Demaine, M. L., and Eppstein, D. (2002). Phutball endgames are hard. *More Games of No Chance* (ed. R. J. Nowakowski), MSRI Publications. Cambridge Univ. Press. To appear.

Thomsen, T. (2000). Lambda-search in game trees - with application to Go. *ICGA Journal*, Vol. 23(4), pp. 203–217.

Werf, E. van der (2002). Personal communication.