A Generalized Threats Search Algorithm

Tristan Cazenave

Labo IA, Université Paris 8, 2 rue de la Liberté, 93526, St-Denis, France
cazenave@ai.univ-paris8.fr

Abstract. A new algorithm based on threat analysis is proposed. It can model existing related algorithms such as Lambda Search and Abstract Proof Search. It solves 6x6 AtariGo much faster than previous algorithms. It can be used in other games. Theoretical and experimental comparisons with other related algorithms are given.

1 Introduction

A new search algorithm based on threats is presented. It is related to other threat based search algorithms. Threat based search algorithm work well in games such as the capture game of Go [1–3] or Go-Moku [4]. Generalized Threats Search (GTS) is based on the notion of generalized threats. GTS is a generalization of previously published algorithms. It is able to model the other existing threat based algorithms. It also solves problems and games faster than the other threat based algorithms when used with appropriate parameters. An analysis of the well formed generalized threats that give good results is given. Experimental comparisons of the different algorithms on the game of 6x6 AtariGo are detailed. The algorithm can also be used to solve problems related to many other games.

The second section explains the game of AtariGo which has been used for the experiments. The third section discusses some related threat search algorithms. The fourth section defines generalized threats. The fifth section describes the search algorithm based on generalised threats as well as how it can model Abstract Proof Search and Lambda Search. The sixth section details experimental results. The seventh section outlines future work and the eighth section concludes.

2 AtariGo

AtariGo is used to teach beginners to play the game of Go. The goal is to be the first player to capture a string. It can be played on any board size. It is usually played on a small board so that games do not take too much time. Teachers also often choose to start with a crosscut in the centre of the board to have an unstable position. We have tested different algorithms on the version with a crosscut in the centre of a 6x6 board.

The rules are similar to Go: Black begins, Black and White alternate playing stones on the intersections of the board, strings of stones are stones of the same

color that are linked by a line on the board. The number of empty intersections adjacent to the string is the number of liberties of the string. A string is captured if it has no liberty. For example in the Figure 6, the two black strings each have four liberties. A string that has only one liberty left is said to be in Atari and can be captured by the other color in one move.

3 Related Work

Search algorithms that develop trees based on threats work well in games such as Go-Moku [4], or the capture search in the game of Go [1–3].

In this section we give an overview of search algorithms based on threats. The first subsection deals with Threat Space Search that has been used by V. Allis to solve Go-Moku. The second subsection briefly summarizes Abstract Proof Search (APS) which has been used to solve capturing problems in the game of Go. The third subsection is about Lambda Search (LS) which deals with similar problems. The fourth subsection exposes Iterative Widening (IW), an improvement over APS. The fifth subsection describes Gradual Abstract Proof Search (GAPS) which introduce some graduality in IW and APS.

3.1 Threat Space Search

Go-Moku has been solved by V. Allis and coworkers using a selective proof search algorithm based on threats and proof number search for the main search when no threats are available [4]. 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.

3.2 Abstract Proof Search

Abstract Proof Search [3] 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 the win in one move are considered and tried at this node. The search of order one consists in developing small 'trees' consisting of only one Left move at each Min node. A search of order N consists in developing trees with N Left moves (depth $2N-1$ plies search trees) at each Min node.

3.3 Lambda Search

Lambda Search [1] 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 order 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 ideas.

3.4 Iterative Widening

The Iterative Widening algorithm [2] 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.

3.5 Gradual Abstract Proof Search

Gradual Abstract Proof Search (GAPS) [5] is based on gradual games. A gradual game is defined as the shape of a search tree. Gradual games can be deeper for a lower order than the games used in APS. They can also be more shallow for a higher order than the λ-trees used in LS. GAPS consists in iteratively widening the scope of the gradual games, instead of widening the games based on the depth of the games as in [2]. It is a generalization of Iterative Widening.

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

4 Generalized Threats

GTS is based on the idea of generalized threats. In the first subsection, generalized threats are defined. The second subsection is about the comparison of generalized threats. The third subsection gives the composition operator that enables to build all the relevant generalized threats. The last subsection gives information on the matching of generalized threats.

4.1 Definition of Generalized Threats

A move of order n is a move that wins if it is followed by n-1 moves in a row by the same player. Each move in a game can be associated to an order. The order of a move M is noted $\omega(M)$. The last move of a game directly wins the game, it is always a move of order one.

A generalized tree represents search trees where the players have the possibility to play multiple moves in a row. The two players are named Left and Right. A branch that goes on the left represents some Left moves, and a branch that goes on the right represents some Right moves. A generalized tree is a binary tree. The usual MiniMax algorithm can be represented by a simple generalized tree as the first tree of the Figure 1 which represents a depth 7 usual MiniMax tree. A Null-move Minimax with a reduction factor of 2 stops searching when the result of a search tree of a depth equal to the depth of the current node minus 2, starting with another move of the same color, does not have an evaluation greater than beta at a max node, and less than alpha at a min node. The second generalized tree of the Figure 1 give the tree developed with a depth 5 Null-Move Minimax with a reduction factor of 2. Null-move search speeds up Alpha Beta search but does not preserves the correctness of the results of a search. Whereas Generalized Threats Search is even faster, and moreover preserves the correctness.

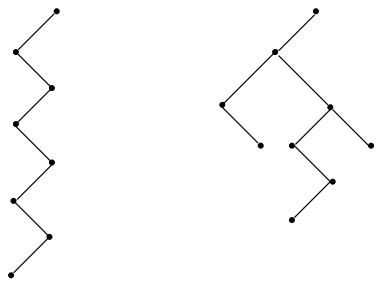

Fig. 1. Generalized trees for a depth 7 MiniMax and a depth 5 Null-Move Minimax.

A node of order n in a generalized tree is a node where the number of left branches in a row after the node is n. For example the root node of the (1,0) generalized threat in the figure 2 is of order one. The root node of the (6,3,2,0) generalized threat is of order 3.

A generalized threat is a set of generalized trees that have some special properties. It is represented by a vector of integers. The first element of the vector is the number of order one nodes that are allowed in the verification of

the threat. The second element is the number of allowed order two nodes, and the nth element gives the maximum number of order n nodes that can be used to verify the threat.

A generalized threat is defined as $g = (o_1, o_2, ..., o_n, 0)$ where o_i is the maximum number of order i nodes that can be visited during the verification of the threat. It always ends with a zero. For example, in order to verify that a winning move is available, the threat (1,0) has to be verified.

In order for a generalized tree to be a generalized threat, it has to fulfill a special property: at each node of the tree that has a left and a right branch, the left subtree has to be included in the left subtree following the right branch.

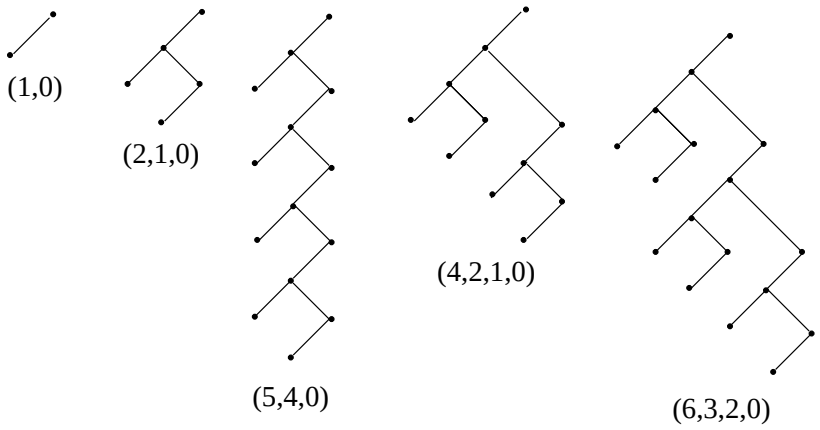

Fig. 2. Some trees representing generalized threats.

The Figure 2 gives some examples of generalized trees representing different generalized threats. Left tries to win the game and Right tries to prevent Left from winning. 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 threat to be verified, all Left moves have to be winning moves, and all Right moves have to be refuted by Left.

In these trees, the number of leaves is the number of order one threats, as each leaf is a won position for Left. The number of order two nodes is the number of Left branches that are followed by an order one node for Left. More generally, the number of order n nodes is the number of Left branches that are followed by an order n-1 node for Left.

Generalized threats are a generalization of the gradual games used in GAPS [5]. In GAPS, the representation of the gradual games is less general. A generalized threat can represent multiple GAPS trees. Moreover, programming generalized threats is easier than programming gradual games as they have a simple definition and nice properties as we will see in the following subsections.

Each order n+1 node is followed by at least one order n node. It is easy to see that $\forall i : o_i < o_{i+1}$. Therefore all the values after a zero in a vector representing a threat are also zero. This is why only the first zero of the vector is written in the vector representing a threat.

4.2 Comparison of Generalized Threats

Let g be a generalized game, ω(g) is the value of the first null element of the vector representing g. For example, we have ω(1,0) = 2, ω(2,1,0) = 3 and ω(6,3,2,0) = 4. ω(g) is the value of the maximum order node of the threat plus one.

Let $g_k = (o_{k,1}, o_{k,2}, ..., o_{k,n}, 0)$. We have $g_a \leq g_b$ if $\omega(g_a) \leq \omega(g_b)$ and $\forall i < \omega(g_b) : o_{a,i} \leq o_{b,i}$.

The Figure 3 gives different possible generalized threats. An arrow between two threats means that the pointed threat is less than the other threat. Some of the threats are incomparable.

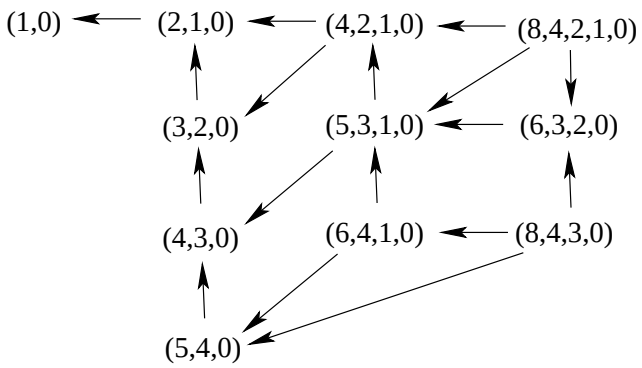

Fig. 3. Order between some generalized threats.

When a generalized threat is greater than another one, it means that all the generalized trees that can be built with the lowest one can also be built with the greatest one. This partial order between threats is particularly useful for building and verifying threats, because the generalized threat following a Right move is always greater than the generalized threat the Right move tries to prevent.

4.3 Composition of Generalized Threats

The basic threat is (1,0). All other threats can be built from this (1,0) threat using a composition operator. Let's name T the operator used to compose two generalized threats.

Let g_l and g_r be two games with $g_l \leq g_r$. We can define the T operator as: $g_t = g_l T g_r$ and $\forall k \neq \omega(g_l) : o_{t,k} = o_{l,k} + o_{r,k}$ and for $k = \omega(g_l) : o_{t,k} = o_{r,k} + 1$.

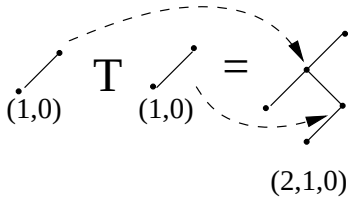

Fig. 4. Composition of the two most simple generalized threats gives (2,1,0).

For example, we have (1,0) T (1,0) = (2,1,0) as explained graphically in the Figure 4. Another example is given in the Figure 5 with the operation (2,1,0) T (4,2,1,0) = (6,3,2,0).

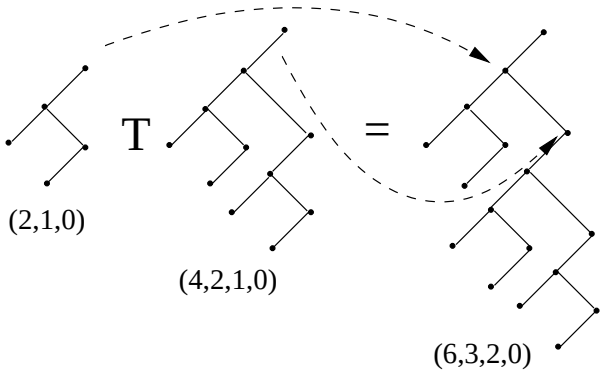

Fig. 5. Composition of (2,1,0) and (4,2,1,0) gives (6,3,2,0).

The T operator ensures by construction that for all the nodes in a tree representing a generalized threat, if the node has a left and a right branch, then the left subtree is always smaller than the subtree at the left of its right branch. This property is very important for finding forced moves and for finding won states even if Right is to play. The right branch represents the set of Right moves that can possibly prevent a Left threat. Because we expect Left to have a harder job winning after a Right move than before, the right branch has to be followed by a generalized tree greater than the Left threat tree in order to ensure that all the Right moves fail.

4.4 Verification of Generalized Threats

Verifying a generalized threat consists in verifying that for each left branch, there is a winning Left move, and that for each right branch, there are no Right moves that prevent Left from winning. It is possible to verify threats without optimizations. However, we use several optimizations that are described in this subsection.

An optimization used to verify generalized threats is iterative widening on the maximum order of the threat at nodes that have only one left branch. At these nodes, the program starts with trying an order one move. If it does not work, it tries an order 2 moves, decrements the number of order 2 nodes in the threat at hand, update the threat at hand so that every value in the threat vector is less or equal than the following value, and tries to verify the updated threat. If the threat is not verified, it continues to increase the order of the threat until a threat is verified or the maximal order of the threat at hand is reached.

At nodes that contain both a left and a right branch, we can make another nice optimization. We know that the left threat is lower than the threat at the left of the right branch. Therefore, as we know the threat that has to be verified at the current node, we can define a new threat which is the current threat divided by two (dividing by two all the integers in the vector representing the current threat). This new threat is the maximum threat that has to be verified for the left subtree. For example, if the program has to verify a (4,2,1,0) threat, it will only try the (2,1,0) threat for the left subtree. Because the right subtree is greater than the left subtree, therefore the left subtree is at most the half of the overall tree (this is why all the integers representing the overall tree are divided by two in order to find the maximum left subtree).

At every node of the tree, the verified threat can be smaller than the maximal threat that was to be verified. The program always memorizes the verified threat. At nodes that contains a left and a right branch, it computes the maximal right threat that has to be tried as the substraction of the vector of the threat to verify minus the vector of the verified left threat.

Another optimization which gives very good results is to use sets of abstract moves as they are defined in [3] during the verification of the search. For example, an order 2 search is bound to fail in AtariGo if all the Right strings have strictly more than 2 liberties. In this case and in similar cases, the search returns fail without even being tried.

5 Generalized Threats Search

This section starts with giving the optimizations used in the Alpha-Beta algorithm for solving AtariGo, which are the same as the optimizations used in the main Alpha-Beta used to perform a GTS. Then the second subsection describes how to select the moves at the Min nodes of the Alpha-Beta so as to perform a GTS. The third subsection is about the selection of moves at the Max nodes of the Alpha-Beta in order to perform a GTS. The fourth subsection shows that it is possible to model the Abstract Proof Search and the Iterative Widening algorithms with the Generalized Threats Search algorithm. The fifth subsection shows how to model Lambda Search with Generalized Threats.

5.1 Alpha-Beta

An optimized Alpha-Beta is used as the core algorithm of GTS. Generalized threats are used in different ways at Max and at Min nodes of the Alpha-Beta. At Max nodes, generalized threats are used to find Left moves that prevent Left from losing if Right plays first, and if no threat is verified, all relevant moves are tried. At Min nodes, generalized threats are used to find Right moves that prevent Left from winning if Left plays first, and if no threat is verified, the node is cut.

The optimizations used are the use of transposition tables, containing the score and the best move. The memorization and 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 least liberties and the number of liberties of the white string that has the least liberties. The number of liberties of strings are updated incrementally too.

These optimizations are similar to the optimizations used in [6] to solve AtariGo with Alpha-Beta.

5.2 Forced moves for Right

Right is the player that tries to prevent Left from winning. Right moves take place at the Min nodes of the Alpha-Beta.

When the generalized threat is not verified for Left at a node of the Alpha-Beta where Right is to move, a cut is performed, Right has prevented Left from winning with this threat.

If the generalized threat is verified, all the right moves that may prevent the threat are tried.

For example, the White move number 2 at E4 in the Figure 6 is found by a (4,3,0) generalized threat (the principal variation of the threat is B(E4), W(D5), B(E5), W(D6), B(D6), W(C6), B(B6) captures the white string). Once this threat is verified, all the relevant White moves are tried, and after each White move, the same threat is checked (in this case the (4,3,0) threat). The only White moves that are kept are the moves that prevent the threat to be verified.

5.3 Forced moves for Left

Left moves take place at the Max nodes of the Alpha-Beta.

In some positions, Left has a limited number of moves if he does not want to lose the game. The generalized threat is tried for Right at each node of the Alpha-Beta where Left is to play. If the generalized threat is verified for Right, the only moves to be tried for Left are the forced moves of the threat.

In positions where there are no forced moves for Left, all the possible moves for Left are tried.

For example, the move number 5 at D2 in the Figure 6 is a forced move for Left (=Black). If Left does not play at move 5, Right can win with a (3,2,0) generalized threat for White (the principal variation for this threat is W(D2), B(F3), W(F4), B(F2), W(F1) capturing a black string).

5.4 Modeling Abstract Proof Search and Iterative Widening

It is possible to model Abstract Proof Search (APS) with GTS. We have ip1 = (1,0), ip2 = (2,1,0), ip3 = (4,2,1,0), ip4 = (8,4,2,1,0), and so on. An APS of order one is a GTS with the (1,0) generalized threat. An APS of order three as described in [3] is a GTS with the (4,2,1,0) generalized threat.

The Iterative Widening algorithm [2] consists in performing a GTS (1,0), and if it fails to perform a GTS (2,1,0), and if it fails a GTS (4,2,1,0) and so on...

5.5 Modeling Lambda Search

λ-search can be modeled with generalized threats. For example, developing a λ1-tree is equivalent to verifying a (∞,∞,0) generalized threat. The different λ-trees can be modeled as follow: λ1-tree = (∞,∞,0). λ2-tree = (∞,∞,∞,0). λ3-tree = (∞,∞,∞,∞,0), λ4-tree = (∞,∞,∞,∞,∞,0).

6 Experimental results

The computer used for these experiments is a 600 MHz Pentium III with 256 MB of RAM running Linux. We have tested four different algorithms.

Alpha-Beta solves 6x6 Atari-Go at Depth 14 in 2793s, results are in the Table 1. All the optimizations described in the Alpha-Beta subsection are used.

Gradual Abstract Proof Search solves 6x6 AtariGo in 62s at depth 10 with the ip4221 gradual game, results are in the Table 2. The Alpha-Beta used in the GAPS is the same as the Alpha-Beta used for the experiments in the Table 1. At each node of the Alpha-Beta, all the gradual ip games are tested and the set of forced moves is the intersection of all the sets of moves sent back by the gradual games.

Lambda Search solves 6x6 AtariGo in 3555s at depth 15 and order 3, results are in the Table 3. We did not use the optimization of Alpha-Beta in LS, we have simply reused the code given by Thomas Thomsen on his web page associated

Table 1. Solving 6x6 Atari-Go with Alpha-Beta.

Depth	*Value*	*Move*	*Time*	*Nodes*
1	1	D5	0.00	33
2	0	D5	0.00	143
3	1	D5	0.01	1234
4	0	C5	0.01	3177
5	1	C5	0.09	25662
6	0	C5	0.29	71265
7	1	C5	2.04	563k
8	0	C5	2.49	604k
9	1	C5	27.91	7442k
10	0	C5	44.04	10375k
11	1	C5	168.06	43034k
12	0	C5	303.21	69300k
13	1	C5	2094.46	518016k
14	500	C5	150.39	34178k
Total			2793.00	

Table 2. Solving 6x6 Atari-Go with GAPS.

Depth	*Value*	*Move*	*Time*	*Nodes*
1	1	D5	0.00	33
2	0	D5	1.84	47
3	1	D5	2.33	235
4	0	E3	4.72	325
5	1	E3	5.76	594
6	0	E3	11.52	861
7	1	E3	11.98	2557
8	1	E3	20.58	1982
9	2	E3	2.78	1838
10	500	E3	0.46	53
Total			61.97	

Table 3. Solving 6x6 Atari-Go with Lambda Search.

Depth	*Res Order*1		*Res Order*2		*Res Order*3	
3	0	0.01	0	0.00	0	0.01
5	0	0.00	0	0.03	0	0.13
7	0	0.00	0	0.10	0	1.29
9	0	0.00	0	0.96	0	12.34
11	0	0.00	0	6.13	0	106.38
13	0	0.00	0	41.29	0	1521.08
15	0	0.00	0	108.10	500	1914.07
Total		0.01		156.61		3555.30

Table 4. Solving 6x6 Atari-Go with Generalized Threats Search.

Depth	R	$(1,0)$	R	$(2,1,0)$	R	$(5,4,0)$	R	$(4,2,1,0)$	R	$(6,3,2,0)$
1	1	0.00	1	0.00	1	0.00	1	0.00	1	0.00
2	0	0.00	0	0.00	0	0.02	0	0.01	0	0.14
3	1	0.00	1	0.00	1	0.02	1	0.12	2	0.23
4	-1	0.00	-1	0.01	0	0.05	0	0.07	0	0.30
5	0	0.00	0	0.00	1	0.02	1	0.17	1	0.56
6	-500	0.00	-500	0.00	-1	0.09	-1	0.17	0	1.00
7					0	0.11	0	0.35	1	1.56
8					-1	0.20	-1	0.69	1	4.69
9					0	0.06	0	0.51	2	1.27
10					-1	0.08	-1	0.14	500	0.08
11					0	0.04	0	0.13		
12					-1	0.08	-1	1.29		
13					0	0.04	-500	0.01		
14					-500	0.09				
Total		0.00		0.01		0.90		3.66		9.83

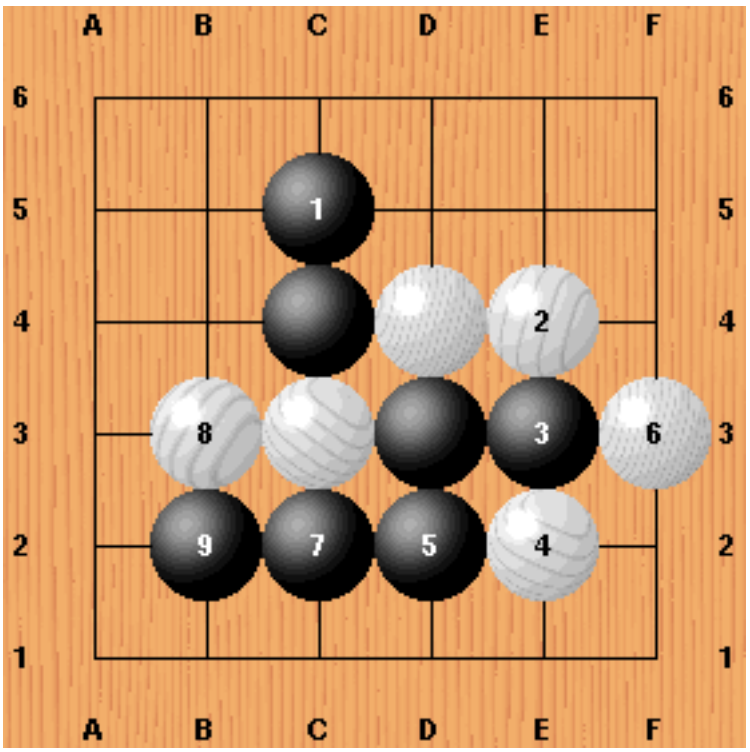

Fig. 6. The solution to 6x6 Atari-Go found by GTS(6,3,2,0).

to his paper. In order to have a better basis for comparison between the relative merits of LS and GTS, we turned off the Alpha-Beta optimizations in GTS. GTS only takes 115s to solve 6x6 AtariGo when all the Alpha-Beta optimizations are turned off, using the (6,3,2,0) generalized threat. GTS also solves 6x6 AtariGo in 57s without Alpha-Beta optimizations, using the (12,6,4,0) generalized threat.

Another experiment was run with all Alpha-Beta optimizations turned off, and with all the abstract knowledge removed in order to have an algorithm that is even less optimized than Lambda Search (all the forced moves are computed in the threats in GTS even if they do not need to be, whereas Lambda Search stops after the first working forced move without looking for the others). To compare it to LS, we have summed the times used at even depth for this algorithm less optimized than LS, and it solve 6x6 AtariGo in 1731s with the (6,3,2,0) generalized threat. Therefore even with less optimizations than LS, generalized threats still solve 6x6 AtariGo twice as fast.

Another thing that can be noted about LS, is that in T. Thomsen code, the iterative deepening LS is called with orders ranging from 1 to (depth-1)/2. We did not use these settings because for AtariGo it would spend a very long time trying to solve order 4 and higher order Lambda Search unsuccessfully. We have voluntarily restricted LS to the order 3 which is the order needed to solve AtariGo.

From a more general point of view, we think it is better to be more cautious about the increasing of order in LS. An heuristic such as Iterative Widening [2] is more appropriate for LS: start with fully searching at order 1, and if it does not work, search at order 2 and so on.

Using the Alpha-Beta optimizations, Generalized Threats Search solves 6x6 AtariGo in 10s at depth 10 with the (6,3,2,0) generalized threat. The results are in the Table 4. The columns with an R are the result of the GTS. The following columns give the time used to search each depth. The solution found by GTS(6,3,2,0) is given in the Figure 6.

7 Future Work

GTS works for AtariGo. It has good chances to work in other games. We plan to test it for capture, connection and life and death in the game of Go, Lines of Action, Phutball, Hex, Shogi and Chess.

A special attention has to be given to the order in which the generalized threats have to be tried. We only have a partial order between generalized threats, so there is room for choice in the order in which generalized threats can be tried. This might be game dependent. However some heuristics on the order of the threats are also probably game independent. It might be possible to find a good game independent order between generalized threats.

An optimization used in LS and not yet used in GTS is to incrementally find the forced moves. GTS searches for all the forced moves before trying them in the Alpha-Beta. It would improve the response time to incrementally search for the forced moves and to stop as soon as one of them prevents the win.

Using transposition tables and killer moves for the verification of the generalized threats could certainly speed up GTS. These optimizations are currently only used in the main Alpha-Beta search. There are opportunities to integrate more closely Alpha-Beta and GTS.

8 Conclusion

We have described GTS, a search algorithm based on the notion of generalized threats. We have given a constructive definition of generalized threats, and we have unveiled some properties of generalized threats that enable to optimize their verification. We have also defined a partial order between generalized threats. Generalized threats can easily be inserted in an existing Alpha-Beta to speed it up, as it is described in the GTS section. We have also shown that GTS is a generalization of previous related algorithms such as Abstract Proof Search and Lambda Search. Experimental results for solving the game of 6x6 AtariGo show that it solves the game faster than other related search algorithms. Some further optimizations are still possible, and the algorithm can be used in other games.

References

1. Thomsen, T.: Lambda-search in game trees - with application to go. ICGA Journal **23(4)** (2000) 203–217
2. Cazenave, T.: Iterative widening. In: Proceedings of IJCAI-01, Vol. 1, Seattle (2001) 523–528
3. Cazenave, T.: Abstract proof search. In Marsland, T.A., Frank, I., eds.: Computers and Games. Volume 2063 of Lecture Notes in Computer Science., Springer (2002) 39–54
4. Allis, L.V., van den Herik, H.J., Huntjens, M.P.H.: Go-moku solved by new search techniques. Computational Intelligence **12** (1996) 7–23
5. Cazenave, T.: La recherche abstraite graduelle de preuves. In: Proceedings of RFIA 02, Angers, France (2002)
6. van der Werf, E.: Message to the computer go mailing list. (2002)