# THREAT ANALYSIS TO REDUCE THE EFFECTS OF THE HORIZON PROBLEM IN SHOGI

*Reijer Grimbergen*

Department of Information Science, Saga University
Honjo-machi 1, Saga-shi, 840-8502 Japan
E-mail: grimbergen@fu.is.saga-u.ac.jp

## ABSTRACT

In two-player perfect information games, a combination of pruning and extension techniques is used to deal with the horizon problem. These techniques must be tuned carefully for each individual game. In this paper we define a general framework from which pruning and extension decisions can be derived. This framework is based on the notion of threats, i.e. the problems that an intelligent game-playing agent has to handle during search. We have used the framework for pruning and extension decisions in shogi (Japanese chess). Results for a partial implementation of this framework indicate that it will improve the playing strength of a shogi program.

## 1. INTRODUCTION

Since the start of artificial intelligence research, intelligent agents for game-playing have been studied. Specifically, the class of two-player perfect information games has been analyzed extensively. Two player perfect information games are games where only two players are involved and in which the game state is fully accessible to both players. A well-known example of a two-player perfect information game is chess. In contrast, in most card games there is no perfect information as the cards which the opponent(s) hold are usually unknown.

In 1950, Claude Shannon published a paper [11] in which he outlined the two basic approaches for intelligent agents to play games: search and knowledge. In the 1950s and 1960s knowledge about a specific game was the most important component of an intelligent game-playing agent [6]. Since the success of "search-only" chess programs in the 1970s [12], attention has focused on search as the main method for intelligent game playing. Recently, research into the use of knowledge rather than search has made a come-back. This has been caused by research into games like *Go*, in which deep search is almost impossible [3]. Despite this, it is safe to say that search is the basic method for intelligent game-playing agents.

Most game-playing agents use a combination of minimax search and alpha-beta pruning to select the best move in the current game position. Because a game-playing agent is searching under time constraints, searching for a perfect solution (i.e. a certain win against all possible replies of the opponent) is in general not possible. In this case, the only thing a game-playing agent can do is to search as deep as possible until time runs out and return the best result based on the evaluation of the positions searched. The search depth at which the search was terminated is called the *nominal search depth*. There are two reasons why there are errors in the result of search terminated in this way: 1) the evaluation is only an estimate of the true value of each position; 2) the evaluation can change dramatically at the next move beyond the nominal search depth (however, there was no time to search this move). This second problem, where vital information is beyond the "horizon" of the search is called the *horizon effect*.

The horizon effect is usually dealt with by a combination of *pruning* and *extension* techniques. A pruning technique is a heuristic that assesses whether searching from a certain position is likely to change the overall search result. If not, the search is terminated at that point, i.e. before the nominal search depth is reached. Time saved by not searching this part of the search tree can be used to increase the nominal search depth. An extension technique is a heuristic that selects a small number of moves at the nominal search depth for deeper search.

Manipulating the nominal search depth in this way has important risks. Mistakenly pruning moves that (if searched) would have lead to a change in the overall search result is clearly unwanted. Also, if too many insignificant moves are selected for extensions, this will lead to a serious search overhead. In the worst case, this can even result in a search that never terminates. For each game, careful tuning of pruning and extension techniques is necessary to avoid these problems.

The aim of our research is to define a general framework for two-player perfect information games from which pruning and search extensions can be derived. As an application, we will use *shogi* (Japanese chess). In Section 2 we will explain why common pruning and extension techniques do not work very well in shogi. In Section 3 we present a decision framework for two-player perfect information games based on the concept of threats. In Section 4 we will apply this to shogi. Some implementation issues for this application are given in Section 5. Section 6 gives some preliminary results of our method and we end with some conclusions and ideas for future work in Section 7.

## 2. PRUNING AND EXTENSIONS IN SHOGI

In chess, careful combination of pruning and extension techniques have worked well and resulted in very strong chess playing programs [5]. An example of a pruning technique is *futility pruning* [4], where search is terminated if it is unlikely to recover from a material loss in the remaining search until the nominal search depth. An example of extending the search is *quiescence search* [2], where captures are played out beyond the nominal search depth to reach stable ("quiet") positions that have a more reliable evaluation value.

Unfortunately, these techniques are not generally applicable. For example, in *shogi* (Japanese chess), futility pruning and quiescence search do not work very well. In shogi it is possible to re-use pieces that were captured from the opponent. Instead of playing a move with one of the pieces on the board, an alternative move is to put one of the pieces that was captured from the opponent on an empty square on the board. Because of this rule, shogi is a game in which the combined total of the pieces for both players remains constant during the entire game. The only way a shogi game can finish is when the king of one of the players is captured. Even though the goal of chess is also to capture the king of the opponent, the simplification resulting from piece captures make gaining material a vital subgoal. In chess, it is almost impossible to recover from losing important pieces like a knight or a rook. On the other hand, position evaluation in shogi is a combination of material, the strength of the attack on the opponent king and the strength of the defense of one's own king.

The problem of both futility pruning and quiescence search is that they are based on the vital importance of material in chess. Futility pruning estimates the probability of recovering from a material deficit in the remainder of the search. In shogi, this should be a combination of material, attack and defense. This will significantly reduce the number of nodes that can be pruned.

Quiescence search is based on the assumption that favorable piece captures are almost always carried out. However, in shogi piece captures are often suspended indefinitely when attack or defense of the king is given a higher priority. If these moves would be included in the quiescence search, this part of the search would explode. The possibility of drops in shogi can lead to long exchanges of attack moves, defense moves and piece captures.

## 3. THREAT ANALYSIS IN TWO-PLAYER GAMES

Therefore, in shogi (and probably in other games as well), we need a different approach to deal with the horizon problem. Our idea is to make the decision for pruning and extensions of the search based on the problem (or problems) that the search has to deal with. We will use the term *threat* for problems that a game-playing agent is facing. First, we will define our notion of threats for two-player perfect information games. Second, we will describe how these definitions can be used for pruning and extension decisions.

### 3.1. Definitions of Threats

- $\Pi = \{T_1, T_2, ..., T_n\}$ is a set of threats.
- $\Delta$ is a partial order on $\Pi$.
- $\Sigma = \{\Sigma_B, \Sigma_W\}$ where $\Sigma_B$ is defined as
  $\Sigma_B = [\{\delta_{11}, \delta_{12}, \ldots, \delta_{1q_1}\}, \ldots, \{\delta_{x1}, \ldots, \delta_{xq_v}\}]$
  and $\Sigma_W$ is defined as
  $\Sigma_W = [\{\mu_{11}, \mu_{12}, \ldots, \mu_{1r_1}\}, \ldots, \{\mu_{y1}, \ldots, \mu_{yr_w}\}]$
  Furthermore $\forall_{i,j}\ \delta_{ij}, \mu_{ij} \in \Pi$.

These definitions can be interpreted as follows. In two-player perfect information games there are two players called $B$ (=*black*) and $W$ (=*white*). $B$ is assumed to move first in the current game position. For each game, $\Pi$ defines the set of all possible threats. On this set of threats a partial order $\Delta$ is defined that gives a relative priority to the threats in $\Pi$. If $\Delta(T_i) > \Delta(T_j)$ then threat $T_i$ has a higher priority than threat $T_j$.

During search we store the unresolved threats against each player at each move in the ordered sets $\Sigma_B$ and $\Sigma_W$. $\delta_{ij}$ and $\mu_{ij}$ are threats unresolved at depth $i$ of the search against the black and the white player respectively. The set of threats against $B$ at search depth $i$ will be called $\Sigma_{B_i}$ and the set of threats against $W$ will be called $\Sigma_{W_i}$.

Furthermore, we can define a maximum operator $\top$ on the set of threats at search depth $i$:

$$\forall_{U \in \Sigma_{B_i}} \Delta(\top(\Sigma_{B_i})) \geq \Delta(U)$$

(The operator for $W$ is similar).

At the start of the search $\Sigma_B$ and $\Sigma_W$ will be initialized with the threats against black and white in the current game position. These initial sets will be called $\Sigma'_B$ and $\Sigma'_W$.

Threats at a certain search depth are an unordered. The sets $\Sigma_B$ and $\Sigma_W$ are ordered to be able to analyze the threat history from the initial position to the position at search depth $i$.

### 3.2. Using Threats for Pruning and Extension Decisions

One of the obvious desired search results is $\Sigma_B = \emptyset$, i.e. all threats against player $B$ resolved. Another desired search result is that there is at least one unresolved threat $T$ in $\Sigma_W$ with $\Delta(T) > \Delta(U)$ for all $U$ in $\Sigma_B$, i.e. there is an unresolved threat against white with a higher priority than all of the unresolved threats against black. The problem is that in general threats cannot statically be judged resolved or unresolved. Search is needed to draw this conclusion and search in game playing has the limitation of the nominal search depth $N$. However, we can use the sets of unresolved sets for a number of pruning and extension decisions (these rules all assume that it is $B$ to play at the decision point; the rules for $W$ to play are straightforward):

- *Pruning rule 1*:

  $$|\Sigma_{B_{N-1}}| = |\Sigma_{B_N}| \wedge \forall_{T \in \Sigma_{B_{N-1}}} \exists_{U \in \Sigma_{B_N}} T = U$$

  Prune all moves at search depth $N$-1 that do not resolve any threats.

- *Pruning rule 2*:

  $$\forall_{T \in \Sigma_{W_i}} \Delta(T) < \Delta(\top(\Sigma_{B_i}))$$

  Prune all moves at depth $i$ that introduce threats against $W$ with a lower priority than the highest priority threat against $B$.

- *Extension condition 1*:

  $$\top(\Sigma'_B) \in \Sigma_{B_N}$$

  Extend the search if the highest priority threat of the initial set is still unresolved.

- *Extension condition 2*:

  $$\Delta(\top(\Sigma_{B_N})) > \beta$$

  Extend the search if the highest priority threat exceeds a certain threshold.

To avoid a search explosion in the search extensions, the aim of the extension should be to empty $\Sigma_{B_N}$ and $\Sigma_{W_N}$ as quickly as possible. This can be implemented by comparing $\top(\Sigma_{B_N})$ and $\top(\Sigma_{W_N})$. Emptying the threat sets can now be done by the following rules (again only the case in which it is $B$ to move is given):

1. IF $\Delta(\top(\Sigma_{B_N})) \geq \Delta(\top(\Sigma_{W_N}))$
   THEN execute the threat $\top(\Sigma_{B_N})$
2. IF $\Delta(\top(\Sigma_{W_N})) > \Delta(\top(\Sigma_{B_N}))$
   THEN defend against the threat $\top(\Sigma_{W_N})$

## 4. THREAT ANALYSIS IN SHOGI

The definitions given in Section 3 are general for all two-player games. For each specific game, we need to define the set of threats and the partial order on these threats. As explained, our application domain is shogi. In this section we will define a set of threats and a partial order on these threats for shogi.

### 4.1. Set of Threats in Shogi

First, we need to define which threats are significant in shogi. We have chosen the following set of threats:

$$\Pi = \{T_{min}, M_1, \ldots, M_7, K_1, \ldots, K_4, T_{max}\}$$

These threats have the following meaning:

- $T_{min}$: The null threat, i.e. no threat.
- $M_1 \ldots M_7$: Material threats. Threats to capture a pawn ($M_1$), lance ($M_2$), knight ($M_3$), silver ($M_4$), gold ($M_5$), bishop ($M_6$) and rook ($M_7$).
- $K_1 \ldots K_4$: Threats against the king. These four threats are based on the evaluation of the attack of the opponent on the eight squares adjacent the king. Let $S = \{S_1, \ldots, S_8\}$ be the set of squares adjacent to the king. Suppose the number of opponent pieces attacking a square $S_i \in S$ is represented as $A(S_i)$ and the number of pieces defending this square is represented as $D(S_i)$ (note that $D(S_i)$ is at least 1 for all $S_i$, as the king defends all adjacent squares). Then the threats against the king can be described as follows:

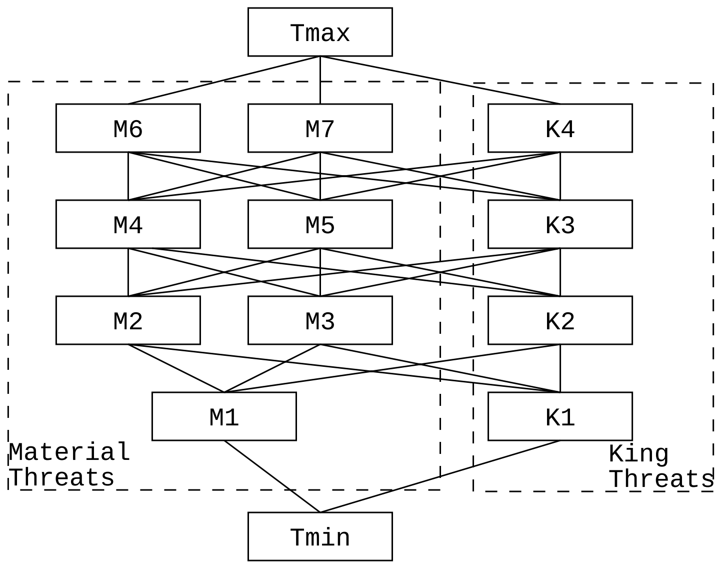

Figure 1: Partial order of threats in shogi

- $K_4$: This is the most severe threat. It is very likely that the opponent can mate the king on the next move [1].
- $K_3$:
$$\exists_{S_i \in S} A(S_i) > D(S_i)$$
There is at least one square adjacent to the king that is controlled by the attacker.
- $K_2$:
$$\forall_{S_i \in S} \neg(A(S_i) > D(S_i)) \wedge \\ \exists_{S_i \in S} A(S_i) = D(S_i)$$
There is no square which is controlled by the attacker, but there is at least one square where the number of pieces attacking this square is the same as the number of pieces defending this square.
- $K_1$:
$$\forall_{S_i \in S} \neg(A(S_i) \geq D(S_i)) \wedge \exists_{S_i \in S} A(S_i) \neq 0$$
There is at least one square that is being attacked.

- $T_{max}$: The maximum threat, i.e. a threat to capture the king (check).

[1]For those familiar with shogi: an example is a drop of a gold on the head of a king on the bottom rank

### 4.2. Partial Order of Threats in Shogi

The partial order $\Delta$ on $\Pi$ for shogi is defined as follows:

- $\forall_{U \in \Pi} \Delta(T_{min}) < \Delta(U)$

  The null threat has a lower priority than any other threat.

- $\forall_{U \in \Pi} \Delta(T_{max}) > \Delta(U)$

  The maximum threat has a higher priority than all other threats.

- $\Delta(M_1) = \Delta(K_1) < \Delta(U)$ with $U$ all threats in $\Pi$ except $T_{min}$

  Attacking a pawn has the same value as a weak king attack and has a lower priority than any other threat except the null threat.

- $\Delta(M_2) = \Delta(M_3) = \Delta(K_2) < \Delta(U)$ with $U$ all threats in $\Pi$ except $T_{min}$, $M_1$ and $K_1$

  Attacking a knight or a lance has the same priority as $K_2$. These threats have a higher priority than $T_{min}$, $M_1$ and $K_1$, but a lower priority than the other threats.

- $\Delta(M_4) = \Delta(M_5) = \Delta(K_3) < \Delta(U)$ with $U$ any threat from the set $\{T_{min}, M_1, M_2, M_3, K_1, K_2\}$

  Attacking a gold or a silver has the same priority as $K_3$.These threats have a lower priority than $M_6$, $M_7$ and $T_{max}$.

- $\Delta(M_6) = \Delta(M_7) = \Delta(K_4) > \Delta(U)$ with $U$ any of the threats in $\Pi$ except $T_{max}$

$\Delta$ is graphically represented in Figure 1.

## 5. IMPLEMENTATION

We have partially implemented our framework in the shogi program SPEAR. SPEAR has participated in the Computer Shogi World Championship every year since 1997 and its playing strength has steadily improved. In the 2001 Championship it reached 13th place in a field of 55 programs.

SPEAR has the following features, which it has in common with most of the strong shogi programs:

- Iterative alpha-beta search.
- Principal variation search [7].
- Quiescence search [2].
- History heuristic and killer moves [8].
- Null-move pruning [1].
- Hash tables for transposition and domination [10].
- Specialized mating search [9].

We have made a new version of our original program in which we implemented the framework presented in this paper. At the moment, this is only a partial implementation with the following features:

- Both of the pruning rules of Section 3.2 have been added.
- The extension decisions of Section 3.2 have not been implemented yet.
- We have replaced the quiescence search of the original program with a search extension that aims at emptying $\Sigma_{B_N}$ and $\Sigma_{W_N}$ as quickly as possible. This search extension does a static evaluation of $\Sigma_{B_N}$ and $\Sigma_{W_N}$, i.e. it does not generate actual moves to deal with the threats.

To make the search extension work, threats can no longer be defined relative to each other, but need actual values. Let $V(T)$ be the value of threat $T$ and $\gamma$ the value returned by the static search extension[2]. The search extension is then implemented as follows:

[2]Of course $V(T) > V(U)$ iff $\Delta(T) > \Delta(U)$

1. $\gamma = 0$

   Initialization of the return value.

2. IF $ToMove(B) \wedge \Delta(\top(\Sigma_{W_N})) \geq \Delta(\top(\Sigma_{B_N}))$
   THEN $\gamma = \gamma + V(\top(\Sigma_{W_N}))$
   $Remove(\Sigma_{W_N}, \top(\Sigma_{W_N}))$

   *Threat execution*: If $B$ to move and the highest priority threat against $W$ has a higher priority than the highest priority threat against $B$ then add this value to $\gamma$ and remove the threat from $\Sigma_{W_N}$.

3. IF $ToMove(B) \wedge \Delta(\top(\Sigma_{B_N})) > \Delta(\top(\Sigma_{W_N}))$
   THEN $Remove(\Sigma_{B_N}, \top(\Sigma_{B_N}))$

   *Threat defense*: If $B$ to move and the highest priority threat against $B$ has a higher priority than the highest priority threat against $W$, then remove the threat from $\Sigma_{B_N}$.

4. IF $ToMove(W) \wedge \Delta(\top(\Sigma_{B_N})) \geq \Delta(\top(\Sigma_{W_N}))$
   THEN $\gamma = \gamma - V(\top(\Sigma_{B_N}))$
   $Remove(\Sigma_{B_N}, \top(\Sigma_{B_N}))$

   *Threat execution*: Similar to 2), but in this case the value of the threat is subtracted from $\gamma$.

5. IF $ToMove(W) \wedge \Delta(\top(\Sigma_{W_N})) > \Delta(\top(\Sigma_{B_N}))$
   THEN $Remove(\Sigma_{W_N}, \top(\Sigma_{W_N}))$

   *Threat defense*: Similar to 3).

6. IF $ToMove(B)$
   THEN $ToMove(W)$
   ELSIF $ToMove(W)$
   THEN $ToMove(B)$

   If it was $B$ to move, set $W$ to move and vice versa.

7. IF $|\Sigma_{B_N}| > 0 \wedge |\Sigma_{W_N}| > 0$
   THEN GOTO 2
   ELSE RETURN $\gamma$

   Loop until both $\Sigma_{B_N}$ and $\Sigma_{W_N}$ are empty.

## 6. RESULTS

To evaluate the performance of our partial implementation, we compared its performance on a set of tactical shogi problems to the performance of the program version without threat analysis.

The shogi problems in the test were taken from the weekly magazine *Shukan Shogi*. The test set consists

| Version | Solved | % | TotalTime |
|---|---|---|---|
| NTA | 100 | 34% | 2h52m0s |
| TA | 107 | 36% | 2h54m16s |

Table 1: Results of a version of SPEAR with threat analysis (*TA*) and a version without threat analysis (*NTA*) on 298 tactical shogi problems.

of 300 problems published in issues 762 (November 4th 1998) to 811 (October 20th 1999). The problems in each issue are divided into six classes, ranging from starting level to expert level. It should be noted that the starting level is already quite advanced and is too hard for beginners. Two of the problems in the test set are incorrect and have been removed from the test set.

All versions of the program were given 60 seconds per problem on a Athlon 1.2GHz PC. Except for the changes described in Section 5, both programs were identical. The results of this test are given in Table 1.

In this table we can see that the program version with threat analysis can solve more problems than the program version without threat analysis. There is a price for this accuracy, as the program with threat analysis is slower than the program without threat analysis. However, over 298 problems the difference amounts to less than 1 second per problem.

## 7. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a new framework for making decisions about pruning and extending search in two-player perfect information games. These decisions are an important part of intelligent game-playing agents as they help reduce the effects of the horizon problem.

The framework we presented is based on a set of threats in an arbitrary two-player perfect information game. On this set of threats a partial order is defined, indicating which threats have a higher priority than other threats. During search this partial order bounding and two sets of unresolved threats (one for each player) provide the decisions for pruning or extending the search. This framework has been applied to shogi.

The first results indicate that our framework improves the tactical ability of a shogi program. We expect that this improvement will show more clearly when our method is fully implemented. After the full implementation is finished, self-play experiments are needed to evaluate if the improved tactical ability also leads to an improvement in playing strength of the program. Implementing the full framework and further experiments will be conducted in the near future.

## 8. REFERENCES

[1] D. Beal. Experiments with the Null Move. In D.Beal, editor, *Advances in Computer Chess 5*, pages 65–79. Elsevier Science Publishers: The Netherlands, 1989.

[2] D. Beal. A Generalised Quiescence Search Algorithm. *Artificial Intelligence*, 43:85–98, 1990.

[3] B. Bouzy and T. Cazenave. Computer Go: An AI oriented survey. *Artificial Intelligence*, 132:39–103, 2001.

[4] E. Heinz. Extended Futility Pruning. *ICCA Journal*, 21(2):75–83, June 1998.

[5] M. Newborn. *Kasparov versus Deep Blue: Computer Chess Comes of Age.* Springer Verlag, 1996. ISBN 0-387-94820-1.

[6] A. Newell, C. Shaw, and H. Simon. Chess Playing Programs and the Problem of Complexity. *IBM Journal of Research and Development*, 2:320–335, 1958.

[7] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison Wesley Publishing Company: Reading, Massachusetts, 1984. ISBN 0-201-05594-5.

[8] J. Schaeffer. The History Heuristic and Alpha-Beta Search Enhancements in Practice. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(11):1203–1212, 1989.

[9] M. Seo. The C* Algorithm for AND/OR Tree Search and its Application to a Tsume-Shogi Program. Master's thesis, Faculty of Science, University of Tokyo, 1995.

[10] M. Seo. On Effective Utilization of Dominance Relations in Tsume-Shogi Solving Algorithms. In *Game Programming Workshop in Japan '99*, pages 129–136, Kanagawa, Japan, 1999. (In Japanese).

[11] C.E. Shannon. Programming a Computer for Playing Chess. *Philosophical Magazine*, 41:256–275, 1950.

[12] D. Slate and L. Atkin. Chess 4.5: The Northwestern University Chess Program. In P. Rey, editor, *Chess Skill in Man and Machine*, pages 82–118. Springer Verlag, New York, 1977.