Using Partial Order Bounding in Shogi

Reijer Grimbergen, Kenji Hadano and Masanao Suetsugu
Department of Information Science, Saga University
Honjo-machi 1, Saga-shi, 840-8502 Japan
E-mail: grimbergen@fu.is.saga-u.ac.jp

Abstract

Using a weighted sum of feature values as evaluation function has some important drawbacks. If an evaluation function with a partial order of features is used, information which is otherwise lost by using a weighted sum can be preserved and better move decisions can be made. Partial Order Bounding (POB) is a new search method that promises to be more efficient than earlier attempts to search with partial order evaluations. Thus far, POB has only be applied to Go endgames. In this paper, we discuss the use of POB in shogi. We outline the implementation decisions that have to made when using POB. These concern the choice of the partial order evaluation, how to set the search targets in POB and how to manage multiple iterations of POB. We compare the results of a number of different implementation schemes of POB in shogi. The best scheme is able to find the correct move in 27 out of 50 next move shogi problems, using an average time of 48 seconds per problem. We consider this a good starting point for further research in Partial Order Bounding.

Keywords: Game tree search, partial order bounding, evaluation functions, shogi.

1 Introduction

The mini-max search strategy gives a simple theoretical answer for perfectly playing a two-player, perfect information game. For each position in the search, first check if the game-theoretical value (e.g. win, loss or draw) for this position is known. If so, return this value. If not, generate all legal moves from the position and generate the positions from these moves. For each generated position, this process of returning a value or generating new positions is continued. If the game-theoretical value can be backed up to the initial position, the search terminates and it is known what the result will be of playing the initial position when neither player makes a mistake.

For most interesting games, the general mini-max search strategy is infeasible. The number of positions that must be generated to end the search is prohibitively large. Therefore, rather than searching all positions until the game-theoretical value can be established, the search is terminated according to some criteria (usually a depth or time constraint) and an *evaluation function* is used to estimate the probability of winning from that position. The value returned by the evaluation function is backed up and used to decide which move gives the best chances of winning the game from the current position.

Many search methods carry over to different games and even to different research areas, but the evaluation function usually contains most of the domain-dependent knowledge of a particular game. For different games, a different evaluation function needs to be designed. This is a very time-consuming process, taking years rather than months. Despite the time-consuming nature of designing a good evaluation function, most evaluation functions have a rather simple design. For each position a number of features $f_1, \ldots, f_n$ is established. Often used features are the number of pieces of different types and the safety of the king. Complex game programs like the chess program DEEP BLUE can have thousands of features [1].

As a next step, to each of the features a weight is attached and the evaluation function is now the weighted sum of the feature values:

$$eval = \sum_{i=1}^{n} w_i f_i$$

This approach has been very successful. It is simple and easy to use in combination with efficient mini-max algorithms. However, there are some important drawbacks of using a weighted sum for the evaluation of a position [3]:

Unstable positions Let's assume a simple evaluation function for a chess-like game with only two features, material and attack. Furthermore, the weights of these features are both

Position	*Material*	*Attack*	*Eval*
P_1	0	0	0
P_2	500	-500	0
P_3	-500	500	0

Table 1: Three positions and their weighted sum evaluation

set to 1. Now consider the three positions P_1, P_2 and P_3 with the feature values given in Table 1. From the feature values it can be inferred that position P_1 is very stable and the player to move probably has a lot of freedom to choose the next move. In contrast, positions P_2 and P_3 are very unstable. In P_2 the player to move has sacrificed material for attack and can not afford to let the attack lose steam or the material deficit will be decisive. It is clear that these three positions are completely different despite having identical evaluations.

Long term strategic features In games where the evaluation function is a combination of short term tactical features and long term strategic features, it is hard to control the long term features. Giving large weights to long term strategic features will lead to tactical problems, while giving to little weight to strategic features will make it impossible for a game program to follow long term plans.

Close to terminal positions There are many positions where only a single feature is enough to determine whether the position is good or bad. An evaluation function with a weighted sum needs to calculate many unnecessary features. This problem is usually attacked by using a fast and a slow evaluation (see for example [1]), but this ad-hoc method can not be expected to lead to a general solution to the problem.

Partial Order Bounding (POB) is a search method that tries to deal with these problems by maintaining the different feature values throughout the search instead of taking them together by a weighted sum. This method has been applied to Go *semeai* [3]. Because the problems mentioned above also occur in shogi, we decided to try and apply POB to shogi. We will first explain Partial Order Bounding in Section 2. Then we discuss the problems of having an evaluation function with a weighted sum of features in shogi and explain how POB can be applied to shogi in Section 3. We will give some preliminary test results of applying POB to shogi in Section 4 and end with conclusions and suggestions for future work in Section 5.

2 Partial Order Bounding

2.1 Partial Order Evaluation

In Section 1 we described the potential problems of using a weighted sum of feature values as evaluation function. If possible, we would like to keep the complete vector of feature values and only make a decision based on these feature values after the search has finished. This should improve the quality of the move decision.

The aim of a mini-max search is to find the best move from the current position, i.e. find the move which is better than all the other moves. Therefore, it is not necessary to know what the exact probability of winning is for a move. We only need to know that this probability is higher than the probability of all the alternatives. Therefore, we do not need a weighted sum of features values as long as we can compare the evaluations of different positions.

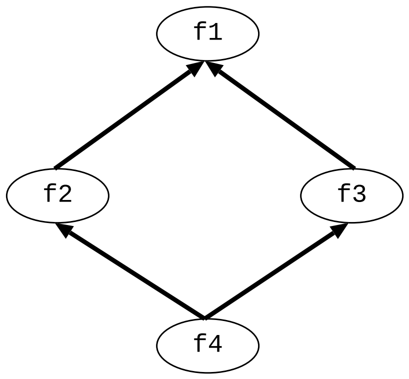

Figure 1: Partial order evaluation example

This is the main idea of having a *partial order evaluation*. Evaluation features are ordered according to their importance and positions are compared by comparing the feature values according to the partial order. An example of a partial order evaluation is given in Figure 1. This partial order can be interpreted as follows: feature f_1 is more important than feature f_2 and feature f_3, which are both more important than feature f_4. Using this partial order evaluation it is easy to compare two positions P_1 and P_2 if the feature values of these positions are known. Some examples of different feature values are given in Table 2.

In general, to compare two positions P_1 and P_2 according to the partial order of Figure 1, we first compare the value of feature f_1. The position with

P_1				P_2				*Comp*
f_1	f_2	f_3	f_4	f_1	f_2	f_3	f_4	
10	25	25	25	10	20	20	50	$P_1 > P_2$
10	25	25	25	20	20	20	20	$P_1 < P_2$
10	10	10	25	10	10	10	30	$P_1 < P_2$

Table 2: Using a partial order evaluation to compare positions

the higher value is better. If both values are the same, we compare the values of feature f_2 and f_3. If the values of both these features for P_1 (or P_2) are higher than those for P_2 (P_1) than P_1 (P_2) is the better position. If the values of f_1, f_2 and f_3 are identical, we compare the values of f_4.

If is was always possible to compare positions perfectly according to the partial order, there would be no reason to use a weighted sum of feature values. A partial order evaluation would be efficient and always provide the answer about which position has the highest evaluation. The problem of using a partial order evaluation is clear when we try to compare the two positions of Table 3.

Position	f_1	f_2	f_3	f_4
P_1	10	50	20	0
P_2	10	15	30	0

Table 3: How to compare these two positions?

For both P_1 and P_2 the value of feature f_1 is identical. However, for P_1 the value of f_2 is higher than for P_2, while for P_2 the value of feature f_3 is higher than for P_1. Using the partial order evaluation, it is not possible to decide which position is better. The reason for this is that the set $\{f_2, f_3\}$ is an *antichain* of the partial order. In general, an antichain of a partial order is a subset of the partial order for which all pairs of distinct elements are incomparable.

Partial Order Bounding is a search method that can find the best move in a given position for a partial order evaluation with antichains. With this search method, it is possible to keep a vector of feature values throughout the search, which solves the problems that occur when using a weighted sum evaluation function.

2.2 Partial Order Bounding

As outlined in Section 2.1, the problem of how to search if the evaluation function is not a weighted sum of feature values, but a vector of feature values that are partially ordered can be reduced to the problem of how to handle antichains in the partial order evaluation. A simple approach to this problem is to keep partially ordered values in every node of the search tree. However, in general this will lead to large sets of incomparable options. There have been attempts to reduce the size of the feature sets by some method, but this leads to a loss of information. Keeping the information about all the evaluation function features was the exact reason for using partial order evaluations, so this is not a desirable solution to the problem.

Partial Order Bounding [3] is a search method that avoids the problems of large sets of partially ordered values by separating the comparison of the evaluations from the process of backing up the values. The idea is to define a target vector with targets for each of the feature values in the antichain. Search is then used to determine if the player who is to move in the current position can reach this target against all possible moves by the opponent.

As an example, consider the antichain $\{f_2, f_3\}$ of Figure 1. Let's define two targets for the values in this antichain: $T_1 = \{5, 3\}$ and $T_2 = \{6, 4\}$. A simple example of a search tree to depth 2 is given in Figure 2. Comparison between the bound and the targets are only made at the leaf nodes of the tree. For example, in node E, feature f_2 has value 5 and f_3 has value 7. If we compare these values to the targets, we see that target T_1 is met, because $5 \geq 5$ and $7 \geq 3$. However, target T_2 is not met, since the feature value of f_2 (which is 5) is lower than the target value for f_2 in T_2 (which is 6). If we look one level higher in the three at node B, it is the opponent to select a move. Of course the opponent will select the move where as many targets as possible will not be met. From B, the opponent will select E, and this means that in B it is possible to reach T_1, but not T_2. In the same way, the result in node C is that neither T_1 nor T_2 can be reached. If we now move to the root of the tree (node A), the player to move will select the move that leads to B over the move that leads to C, because this guarantees him that at least target T_1 will be reached. Therefore, from this search tree we can conclude that the player to move can reach target T_1, but not T_2.

3 Partial Order Bounding in Shogi

The evaluation function in shogi suffers from the same problems that were explained in Section 1. Therefore, we wondered if Partial Order Bounding could be a viable alternative in shogi as well. To apply Partial Order Bounding in shogi, a number of important decisions have to be made [4]:

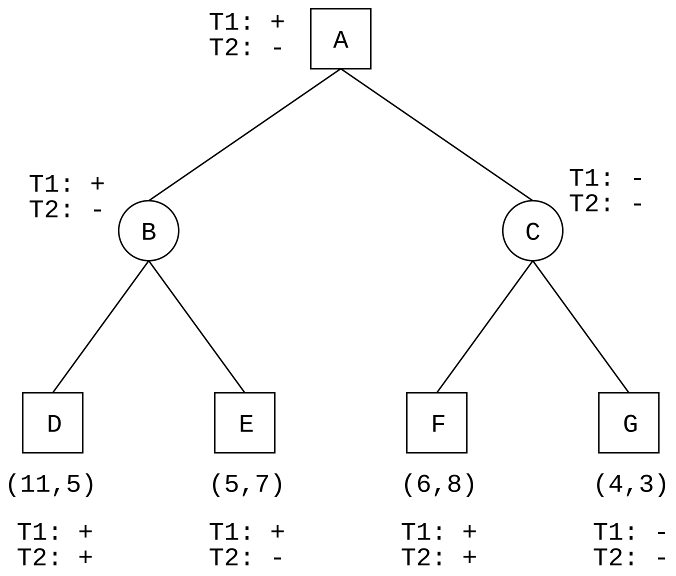

Figure 2: Partial order bounding example

- Which partial order evaluation to use, or more precisely, which antichain(s)?
- How to set the search targets?
- What to do if the search target is met or fails?
- To what depth should be searched?

We will now discuss each of these in detail.

3.1 Partial Order Evaluation in Shogi

An evaluation function for shogi is quite complex and can have many incomparable features. Each of these incomparable features is a candidate for an antichain. In our research, we decided to start with the following simple antichain:

- Material
- Strength of attack
- Strength of defense

The reason for choosing this simple antichain is that we wanted to have a partial order evaluation that was representative for shogi. The three features given here are part of every evaluation function for shogi. Furthermore, it is expected that these three features will dominate most other features, so that it will be quite difficult to find an antichain which has only a subset of these three features.

3.2 Setting the Search Targets

Setting the search targets is an important problem in POB. If the search target is set too low, then there will be many moves for which the target is reached and it will be impossible to distinguish between these moves. In contrast, if the target is set too high, there will be no moves for which the target is reached and no decision can be made about the next move to play.

Our solution to this problem is to perform a shallow alpha-beta search before the actual Partial Order Bounding. This alpha-beta search is done with a scalar evaluation function, but the values of the three features of the highest evaluation are remembered and this becomes the first target of the Partial Order Bounding phase [4].

3.3 Success and Failure

As mentioned in Section 3.2, setting a target too high or too low is an important problem for Partial Order Bounding. POB will not give good results if only a single target is searched. If such a target is reached, it is unknown whether or not an even better target can be reached. More importantly, there can still be a number of moves for which the target can be reached, which means that no decision can be made about the next move to play. Therefore, in this case the target should be increased and a new POB search performed. On the other hand, if a target is not reached, the target should be lowered until the target is met and a move can be selected.

Therefore, POB is actually a series of searches

Move	*POB iteration*				
	1	2	3	4	5
M_1	T	F			
M_2		F			
M_3		T	F	F	T
M_4			F	F	F

Table 4: Several POB iterations to find the best move. *T* means the target has been reached, while *F* stands for failure to reach the target.

with different bounds that is continued until the best move is found or time runs out. Table 4 illustrates this idea. In Table 4 the search target of the initial search is set too low. From the initial position there are four moves that can be played, but already the first move satisfies the target. In this case, none of the other moves needs to be searched and POB ends successfully. However, we still don't know which target can be reached and which move is best. Therefore, we set a higher target and search again. This time M_1 and M_2 fail to meet the target, but by playing M_3, the target is reached. Now we know that M_3 is better than M_1 and M_2, but we still can make no difference between M_3 and M_4. Therefore we raise the target again and now both M_3 and M_4 fail to meet the target (note that it is no longer necessary to search M_1 and M_2). Since the target was set too high we lower it (but not as low as in iteration 2) and search both M_3 and M_4 again. Again for both moves the target fails, so we need to lower the target again. This time only M_3 meets the target and we know that M_3 is the best move.

As can be concluded from this example, it can take a number of POB iterations before the optimal move is found. Each iteration takes an amount of time, so for POB to be useful in practice, it is important that the targets are set in such a way that the number of iterations is kept to a minimum. An additional problem is which search targets to increase when there are multiple features in the antichain. For example, if a target with values for material, attack and defense fails, should all three feature targets be increased, only one or a combination of the three?

Therefore, one of the most important problems of applying POB is to find a good scheme for lowering and increasing search targets. We have not been able to come up with a general method for doing this. Although we will continue to investigate this problem, we believe that this is a tuning problem of POB. Therefore, it is highly likely that it is only possible to empirically compare different schemes for increasing and decreasing search targets. Just like tuning evaluation functions to a certain game and even to certain situations in a game (e.g. opening, middle game and endgame). POB target settings should also be tuned to fit a certain game best. We will describe some of the schemes we tried in detail in the next section.

3.4 Search Depth

Another problem of POB is the depth of the search. In many applications with specific search targets (e.g. *tsume shogi*) it is easy to test if in a certain position a target is met or not. However, in POB this is not the case. Even when a search target is met, a deeper search could reveal an opponent move that makes the target fail. One could argue that it is safe to say that a target can be reached when the target is reached in a certain position where it is the player to move (or the target can not be reached if it fails in the current position and the opponent is to play). Although we did not investigate this, we feel that this type of nullmove condition is rare and will not avoid a search explosion.

We also have not found a general solution to the problem of setting the search depth. We have compared different search depths, using standard methods like quiescence search to make the values of the evaluation features more reliable.

4 Results

As pointed out, there are many ways to implement POB. We have tried a number of different settings. In this section we give the implementation schemes that turned out to be the most successful.

4.1 Setting the Targets

We have compared three schemes for setting the targets between POB iterations:

Scheme A Increasing (if the current target succeeded) or decreasing (if the current target failed) all three search targets (material, attack, defense) by the same amount. Several tests showed that for our implementation an increase or decrease of 250 (a pawn is worth 100 points in our evaluation function) gave the best results.

Scheme B Giving more weight to material. This is done by increasing or decreasing the value of the material feature by an amount of 400 and the attack feature and the defense feature by 100.

Scheme C Giving more weight to attack. This is done by increasing or decreasing the value of the attack feature by an amount of 400 and the material feature and defense feature by 100.

We also tested schemes where more weight was given to the defense feature, but were unable to find values that improved results over the use of Scheme A. This could indicate that defense is not actually part of an antichain.

If a target fails or succeeds for all the remaining candidate moves (iteration 3 and 4 in Table 4), the target changes are halved. For example, if Scheme A was used and the target of iteration 2 was {250, 250, 250} and M_3 and M_4 both fail to reach the target {500, 500, 500}, then the target for iteration 4 becomes {375, 375, 375}.

4.2 Search Depth

In our POB implementation, there are two areas where search depth is important. First, the depth of the alpha-beta search that is used to determine the first search target. We used a 3 ply search for this initial alpha-beta search. This turned out to be the only option on the relatively slow machine (Pentium III, 750 MHz) we used to run our experiments on. A 2 ply search was unable to provide a reasonable search target, while a 4 ply search often took too long to finish.

For the actual POB iterations, we compared the results of 3, 4 and 5 ply searches. For our tests, we used 50 next move problems from Shukan Shogi. We only used the first problem of the six problems that are given in each Shukan Shogi issue. The problems we used are from next move sets 750 to 799. The results are given in Table 5, Table 6 and Table 7

	A	B	C
Solved problems	17	17	15
Avg. time per problem	0:07	0:10	0:05

Table 5: 3 ply Partial Order Bounding

	A	B	C
Solved problems	23	19	27
Avg. time per problem	1:00	1:47	0:48

Table 6: 4 ply Partial Order Bounding

From these results we can conclude that 4 ply Partial Order Bounding using Scheme C gives the best results. With this method 27 problems can be solved in only 48 seconds on average. 5 ply Partial Order Bounding is unable to do better and takes a lot more time.

	A	B	C
Solved problems	27	23	25
Avg. time per problem	17:23	26:13	12:19

Table 7: 5 ply Partial Order Bounding

We were surprised that giving more weight to attack leads to better results than giving more weight to material. A majority of the test problems have solutions that aim at winning material and this was expected to give an important advantage to Scheme B. However, for all search depths, Scheme B had the worst results. Further testing is needed to conclude if this bad performance is caused by increasing the material target by 400 or if increasing the attack target by 100 instead of 250 had the side effect of finding ways to gain material too late.

Detailed results of 4 ply Partial Order Bounding for each test problem are given in Table 8. Here it can be seen clearly that setting the weight targets has an important impact on the test results. Even though Scheme A, B and C solve 23, 19 and 27 problems respectively, only 6 problems are solved by all three. This raises the important question of the possibility of searching positions with different kinds of search targets. This could be done in parallel to improve search speed.

5 Conclusions and Future Work

In this paper we have presented the implementation issues and some preliminary results of applying Partial Order

Bounding in shogi. Partial Order Bounding is a relative new search method that has not been investigated thoroughly yet. From our experience with implementing POB in shogi, we can conclude that POB can not be considered a general solution to the problem of using a partial order evaluation instead of a weighted sum of features. POB introduces new problems when it is used in a specific game. The choice of the partial order evaluation, the method of setting search targets and the problem of how to change targets between POB iterations requires careful tuning.

There is also a general problem with POB that we have not been able to solve: what to do if time runs out without finding a single best move. Should one of the moves that satisfied the target picked at random? We have started investigating the op-

Pos	A		B		C		*Pos*	A		B		C	
750	1:23	×	0:31	×	1:52	○	775	3:12	×	14:39	×	1:00	○
751	0:53	×	0:07	×	0:10	×	776	0:58	○	0:12	×	0:25	○
752	2:23	×	0:38	○	0:27	×	777	0:52	×	0:19	○	0:15	×
753	0:43	×	0:13	○	0:11	○	778	1:08	○	0:21	×	0:22	×
754	2:31	×	0:19	×	0:42	×	779	8:14	×	0:45	○	1:45	○
755	1:38	×	8:27	×	0:18	×	780	0:23	○	0:04	×	0:03	×
756	1:22	○	0:25	○	0:24	○	781	19:24	×	1:32	×	0:38	×
757	3:17	×	1:05	×	0:18	×	782	0:22	×	0:13	○	0:02	○
758	0:53	○	0:22	○	0:20	○	783	1:10	×	0:17	×	0:16	×
759	13:13	×	4:50	×	7:06	×	784	1:21	○	0:26	○	0:19	×
760	4:24	×	3:02	×	1:05	×	785	2:46	×	5:55	×	0:56	○
761	2:03	○	0:26	○	0:14	○	786	1:08	×	0:36	×	0:18	×
762	0:57	×	12:30	×	0:39	×	787	0:18	×	0:05	○	0:05	○
763	5:38	×	2:38	×	2:04	×	788	1:13	×	0:41	○	0:29	○
764	5:16	×	3:38	×	0:44	×	789	1:57	○	0:27	×	0:13	×
765	7:33	×	3:47	×	1:57	×	790	1:47	×	8:07	×	0:57	○
766	0:47	×	0:22	×	0:14	○	791	6:02	×	4:12	○	0:18	○
767	1:04	×	0:24	×	0:16	×	792	12:04	×	4:57	×	2:05	×
768	1:33	○	0:31	×	0:19	×	793	0:06	○	0:02	○	0:04	×
769	0:15	×	0:05	×	0:08	×	794	3:39	○	1:34	○	0:39	×
770	0:21	○	0:10	○	0:03	○	795	5:52	○	0:57	×	0:28	○
771	1:43	○	0:40	○	0:28	○	796	0:40	○	0:09	○	0:14	○
772	0:26	×	0:02	×	0:04	×	797	3:21	×	4:35	×	0:57	×
773	4:11	×	0:13	×	1:08	×	798	0:26	×	0:08	○	0:02	×
774	10:10	×	1:12	×	0:47	×	799	0:43	×	0:21	○	0:17	×

Table 8: Time and results for 50 next move problems from Shukan Shogi using three different types of POB

tion of reserving a certain amount of search time to perform normal alpha-beta search on the moves for which the target is reached [2]. So far, we have not reached any conclusion about the feasibility of this approach. Therefore, this remains a target for future research.

Our research shows that using conventional search still outperforms POB. However, we also believe that the results show that POB is a promising search method in shogi. For example, using different search targets in parallel can be an important future research topic.

References

[1] M. Campbell, A.J. Hoane Jr., and F-h. Hsu. Deep Blue. *Artificial Intelligence*, 134:57–83, 2002.

[2] K. Hadano. Improving Partial Order Bounding in Shogi. Graduation thesis, Saga University 2003. (In Japanese).

[3] M. Müller. Partial Order Bounding: A new Approach to Evaluation in Game Tree search. *Artificial Intelligence*, 129:279–311, 2001.

[4] M. Suetsugu. Using Partial Order Bounding in Shogi. Graduation thesis, Saga University 2002. (In Japanese).