Partial Order Evaluation in Game Tree Search, and its Application to Analyzing Semeai in the Game of Go

Martin Müller
ETL Complex Games Lab
Tsukuba, Japan
mueller@etl.go.jp

Abstract

We revisit the problem of constructing an evaluation function for game tree search. While the standard model assumes a numeric evaluation function, partial orders have some desirable properties for constructing a more meaningful evaluation. However, previous partial order tree search algorithms have been quite complex. We introduce *partial order bounding (POB)*, a simple new method that uses partial order evaluations in game tree search. We also discuss the use of partial order move evaluation for move ordering and for pruning dominated moves during search. We show an application of the method to capturing races called *semeai* in Go. We show that liberty and eye counts of blocks in a semeai can be used to construct a partial order evaluation. In experiments, we show the resulting speedups compared to standard minimax search methods.

Evaluation in Game Tree Search

Position evaluation in game tree search is usually computed as a weighted sum of feature values, $v = \sum w_i v_i$.

The main drawback of this approach is that it loses information: it adds and compares features for which the weighted addition and comparison might not make much sense.

For example, a simple, stable position and a highly dynamic, unstable one might end up with the same static evaluation, though they look very different to a human player. The traditional remedy is to not evaluate unstable positions and perform a quiescence search to try to reach a stable position. Stable positions are considered 'easy' to evaluate, while unstable ones are considered 'hard'.

The loss of information is severe in positions that can 'almost' be evaluated statically, where some distinguished features exist that allow a strong focusing of the search. Standard evaluation methods cannot use such partial information. This was our motivation to look for a different approach.

The structure of this paper is as follows: We survey previous search methods that use partially ordered evaluation, and then introduce a simple new method called *partial order bounding (POB)* for search in game trees using partial order evaluations. As a second application of partial order evaluations, we discuss exact and heuristic pruning and move ordering.

We show an application of partial order evaluation methods to the problem of analyzing semeai in Go. After a brief explanation of the problem, we introduce a partial order evaluation of semeai using the number of liberties and the eye status of stones. In the experiments, we show the resulting speedups compared to traditional alpha-beta search, and test the performance of the evaluation method against leading Go programs in full board semeai problems.

A final section on future work discusses development of similar methods in A*-like search, and possible applications to other games or other phases of Go.

Multiobjective Search Methods

All the literature on search with partial ordered evaluation that I was able to find stems from the same origin in multiobjective search. In the multiobjective approach to decision making, multiple potentially conflicting and noncommensurate objectives are investigated simultaneously. Each objective is evaluated by a scalar, and all evaluations are aggregated into a vector instead of combining them into a single scalar-valued evaluation. The first multiobjective search method, multiobjective A* (MOA*), was developed by Stewart and White (Stewart and White 1991). In MOA*, evaluation is a m-dimensional vector of scalar values. For minimization problems, a partial order on these vectors is defined by

$$y \leq y' \Leftrightarrow y_i \leq y'_i \; \forall i \in 1, \ldots, m.$$

Given an OR graph with such vector-valued edge costs, MOA* finds all nondominated paths from a start node to a given set of goal nodes.

Harikumar and Kumar introduced an iterative deepening version called IDMOA* (Harikumar and Kumar 1996). Dasgupta, Chakrabarti and DeSarkar defined multiobjective heuristic search in AND/OR graphs and partial order game tree search (Dasgupta *et al.* 1996b; 1996a). All these algorithms were developed only for the special partial orders of vector dominance defined above, not for general partial orders.

Partial Order Game Tree Search

A game tree search method using a partial order evaluation was developed by (Dasgupta *et al.* 1996b). The method generalizes traditional game tree search by allowing leaf node evaluations from a partially ordered set, in contrast to the usual scalar values.

The main problem with partially ordered evaluations is that min- and max-expressions involving such values are much more complex than on a totally ordered domain. A totally ordered domain such as the integers or reals is closed under the application of the min and max operators: if $x_1, \ldots, x_n$ are values from a totally ordered domain T, then both $\min(x_1, \ldots, x_n)$ and $\max(x_1, \ldots, x_n)$ are also elements of T. For values from a partially ordered set this no longer holds.

To keep the complexity of solutions under some control, the method of (Dasgupta *et al.* 1996b) crucially relies on the use of players' private *preferences*. A player's preference ϕ is a many-to-one mapping from the partially ordered domain P to a number, which preserves the partial order on P. Sets of possible outcomes which are incomparable in P are compared using the following procedure:

Compare(S_1, S_2, ϕ)
To compare sets of outcomes S_1 and S_2
on the basis of preferences ϕ
1. If only S_1 is empty, declare S_2 as better.
   Likewise, if only S_2 is empty, declare S_1 as better.
   If both S_1 and S_2 are empty then
   select S_1 or S_2 randomly and declare it to be better
2. let x_1 be the worst outcome in S_1 and
   x_2 be the worst outcome in S_2 based on ϕ.
3. if x_1 and x_2 are of equal preference then
   3.1 Drop all outcomes from S_1 and S_2 that are of
   equal preference to x_1
   3.2 Goto [Step 1]
4. If x_1 is better than x_2 based on ϕ, then declare S_1 as better
   else declare S_2 as better.

When using partially ordered evaluations, search cannot return a single value only. In general, the minimax backup of values results in a potentially very large set of *non-inferior sets of outcomes.*

It is easy to see that the approach using player's preferences ϕ only works when a reasonably strong ordering is provided by ϕ. If too many values are mapped to the same preference the comparison breaks down. In the extreme (but relevant) case where no a priori preferences between incomparable values in the partial order can be made by a player, a *Compare()* of any two nonempty sets of outcomes degenerates to the final line of step 1: random selection.

Partial Order Bounding: A New Approach to Partial Order Search

As we have seen, existing partial order search methods are burdened with restrictions on the type of partial order, and with complexity problems when the size of non-inferior sets of outcomes backed up through the search tree is large. Our new, simple approach to partial order evaluation in search is based on Pearl's SCOUT method (Pearl 1984), which seeks to establish an inequality between a given bound and the unknown evaluation. In the case of a partially ordered set P, a bound $B \subset P$ can be given by an antichain in P that describes the minimal outcomes we want to achieve. Each bound B partitions P into two sets: The *success set* $S(B) = \{x \in P : \exists b \in B : x \geq b\}$ is the set of 'good enough' values, the *failure set* $F(B) = P - S(B)$ contains all other values. We use search to decide the following: can we achieve a result $x \in S(B)$, or can the opponent prevent this from happening? Figure 1 shows a partially ordered set $\{A, B, \ldots, K\}$, an antichain consisting of the two elements E and G and the resulting partition into success and failure sets.

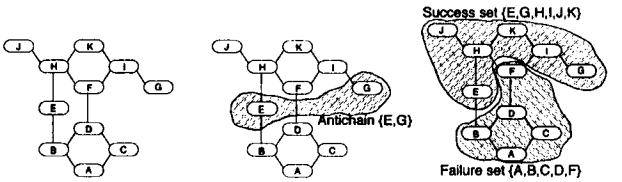

Figure 1: Antichains in partial order bounding

Search with partially ordered evaluation works as follows: in each leaf node, a partial order evaluation is attempted. If the node can be statically evaluated, the outcome is *win* or *loss* depending on whether the evaluation lies in the success set or failure set. If the node cannot be evaluated, the outcome is *unknown*. This approach completely separates the comparison of partially ordered values from the tree backup. Only the three values *win*, *loss* and *unknown* are backed up through the tree. This makes it possible to use partial order evaluations in conjunction with any minimax-based search method, such as alpha-beta or proof number search (Allis 1994).

To summarize, the following are the main differences between POB and (Dasgupta *et al.* 1996b):

- POB allows any partial order as an evaluation, not just ones based on vector dominance.
- Comparison of partial order evaluations occurs only at leaf nodes. The evaluation, if it exists, is compared to a prespecified bound, resulting in one of three outcomes: *win*, *loss* and *unknown*.
- POB needs an input parameter: a bound in the form of an antichain in P that divides the acceptable outcomes, called the *success set*, from the unacceptable ones, the *failure set*.

The selection of a suitable bound depends on both the application domain and the specific problem. In some applications such as the semeai we will discuss later, a single bound is sufficient to solve a problem. In other, less constrained circumstances, an iterative process could be used to test different bounds.

Application to Adversarial Planning

An adversarial planning problem with many subgoals naturally defines a partial ordering of possible outcomes. As an example, Figure 2 shows an ordering of three incomparable subgoals A, B and C as well as the simplest conjunctions and disjunctions of these subgoals. Figure 3 shows a stronger order in the case where achieving subgoal C dominates achieving both subgoals A and B.

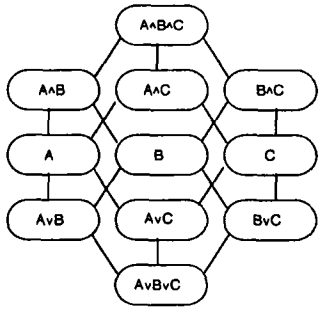

Figure 2: Partial ordering of subgoals

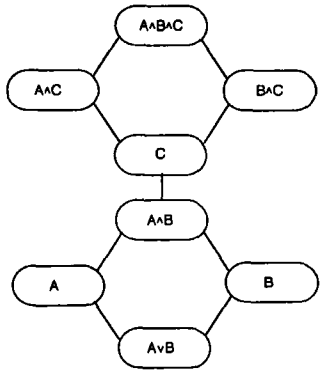

Figure 3: C dominates $A \wedge B$

Partial Orders for Move Ordering and Pruning

From experts commenting on their own games, one can frequently hear judgments such as the following:

> Whatever happens next, 'a' is the only move in this kind of position.

Such a comment seems to indicate a situation in which the expert does not search to resolve the outcome, but is nevertheless certain about the right way to play. Partial orders for move evaluation can capture this kind of knowledge. As an example, Figure 4 shows such a partial move ordering for semeai. For move ordering, a partial order can be extended to a total order consistent with it. For example, in the figure the two top values 'make 2 eyes' and 'capture' are incomparable. However, for move ordering capturing moves should probably rank higher, since capturing often leads to a quicker overall decision.

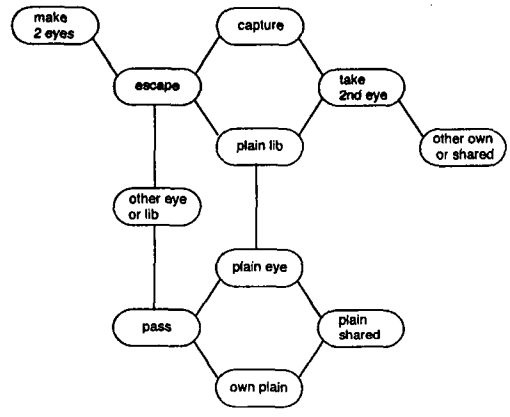

Figure 4: Partial move ordering in semeai

In highly constrained game situations, it may be possible to construct an exact partial move ordering, which can be used for pruning. If a move with a certain value exists, then all moves with dominated value can be pruned. Furthermore, if the value completely specifies the effect of the move, all but one move with this value can be pruned as well.

Application of POB and Partial Move Ordering to Semeai in Go

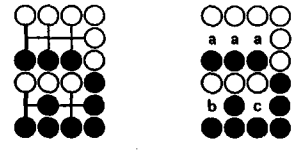

Figure 5: Simple semeai with 3 vs 2 liberties

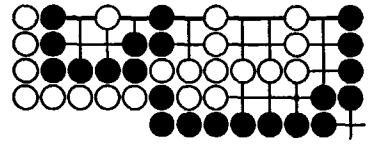

Figure 6: Complex semeai with big eyes and many liberties

Remark: for the benefit of readers who are not so familiar with Go terminology, the *emphasized* technical terms are explained in the appendix.

A *semeai* is a race to capture between non-alive stones of both players. Figure 5 shows a semeai where Black has three liberties marked 'a', and White has two liberties in different *regions* of the board marked 'b' and 'c' respectively. Figure 6 shows a more complex semeai where both players have one large *eye*. Figure 7 shows a semeai at an earlier stage, where the *eye status* of the involved stones is not yet clear.

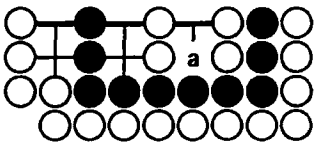

Figure 7: Earlier stage of semeai

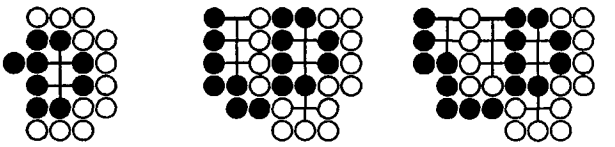

Figure 9: Liberty and eye count (2,0) or (1,1)

Semeai Evaluation: Liberty and Eye Count

The contribution of a region to the strength of a block in a semeai can be measured by the *liberty and eye count*.

Definition: given a block b and an adjacent region r, the *liberty and eye count* of b in r is an ordered pair (l, e), where l is the number of liberties of b in r, and e is the *eye status* of r.

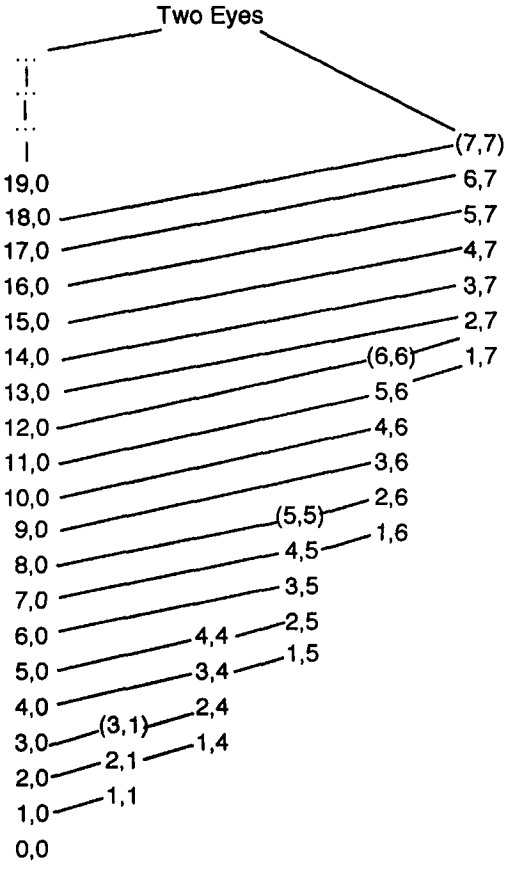

Figure 8: Partial order evaluation of (liberty, eye) pairs

In simple cases, the liberty and eye count can be evaluated statically. In general, a unique count need not exist. The leftmost picture in Figure 9 shows an example where Black has surrounded two empty points. In this region, the large black block can achieve an evaluation of either (2,0) or (1,1). The other two pictures in the figure show situations where one option is successful while the other option would lose. Since (2,0) and (1,1) are incomparable, it is not possible to assign a unique liberty and eye count to the original position. In such cases POB can be used to efficiently search the semeai.

For more details on the Go-specific aspects of semeai evaluation and the mapping of semeai to an instance of POB, we refer the reader to our forthcoming technical report (Müller 1999).

Experimental Results

Comparison with Alpha-beta

We measure the performance of POB and partial order pruning by comparing four kinds of search: Plain alpha-beta (abbreviated $\alpha\beta$ in the table) uses partial ordering for move ordering only, and evaluates wins and losses only by capture. Alpha-beta search with partial order pruning ($\alpha\beta^{POP}$) uses knowledge about partial ordering to prune moves. Non-pruning POB (POB^-) uses partial order leaf evaluation but no pruning. Full POB uses both partial order leaf evaluation and partial order pruning. However, all four types of search apply the pruning rule of using only one of several equivalent moves, such as outside liberties. Without this rule, the non-pruning variants would be much worse.

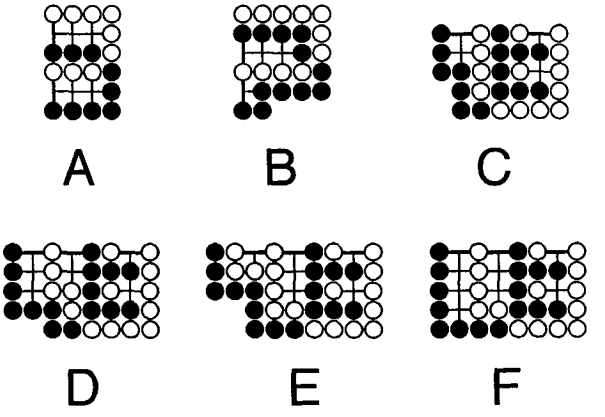

Figure 10: Semeai test problems

Figure 10 shows six test problems, and Table 1 the number of nodes expanded by $\alpha\beta$, $\alpha\beta^{POP}$, POB^- and POB on these problems. In this and the following table, the column BW indicates whether Black or White moved first from the starting position.

Test Against Other Programs

The partial order semeai evaluation module has been integrated into our Go program *Explorer* (Müller 1995).

	BW	POB	POB^-	$\alpha\beta^{POP}$	$\alpha\beta$
A	B	1	1	20	50
A	W	1	1	20	50
B	B	1	1	210	441
B	W	1	1	250	358
C	B	20	26	23	33
C	W	6	21	12	21
D	B	18	27	433	1443
D	W	18	27	134	284
E	B	304	333	1339	1523
E	W	208	333	993	1367
F	B	18	27	1066	4199
F	W	18	27	272	662

Table 1: Comparison of POB and alpha-beta

Probl.	Opponent	BW	Result	Gain
Full 1	Explorer	B	B+36	-
Full 1	Many Faces (B)	B	W+103	+139
Full 1	Many Faces (W)	B	B+53	+17
Full 1	Go 4++ (B)	B	W+40	+76
Full 1	Go 4++ (W)	B	B+116	+80
Full 1	Explorer	W	B+6	-
Full 1	Many Faces (B)	W	W+85	+91
Full 1	Many Faces (W)	W	B+117	+111
Full 1	Go 4++ (B)	W	W+40	+46
Full 1	Go 4++ (W)	W	B+129	+123
Full 2	Explorer	B	W+9	-
Full 2	Many Faces (B)	B	W+93	+84
Full 2	Many Faces (W)	B	B+95	+104
Full 2	Go 4++ (B)	B	W+99	+90
Full 2	Go 4++ (W)	B	B+55	+64
Full 2	Explorer	W	W+69	-
Full 2	Many Faces (B)	W	W+73	+4
Full 2	Many Faces (W)	W	B+139	+208
Full 2	Go 4++ (B)	W	W+99	+30
Full 2	Go 4++ (W)	W	B+51	+120

Table 2: Results of full board semeai tests

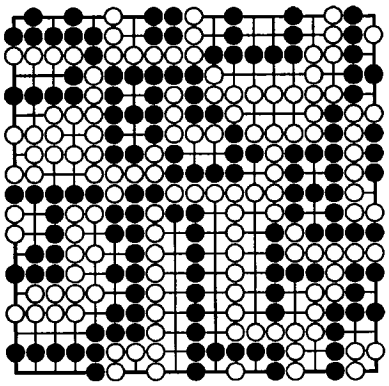

Figure 11: Full board semeai 1

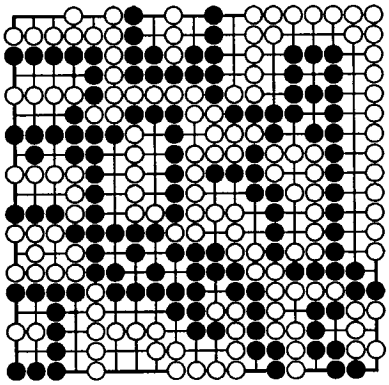

Figure 12: Full board semeai 2

For testing, we constructed two full board Go positions that contain a large variety of semeai positions, as shown in Figures 11 and 12. Starting from these positions, we played Explorer against two of the leading Go programs: current world champion *Many Faces* and many times runner-up *Go 4++*. Explorer won most test games by large margins. Detailed results, including the game-theoretic result determined by self-play, are listed in Table 2. The test positions and game records in SGF format are accessible through the internet at http://www.etl.go.jp/etl/suiron/~mueller/cgo/semeai.html.

From this experiment, it is clear that the partial order evaluation is able to evaluate semeai much earlier than the standard numerical alpha-beta searches used by the other programs. While static partial order evaluation is possible only for a restricted class of semeai, such as those shown in Figures 11 and 12, search using POB will allow a much broader class of semeai to be handled correctly as part of Explorer's position evaluation. Integration of POB in Explorer is currently underway.

Conclusion and Future Work

We have defined a simple but powerful new partial order search method called POB, which applies Pearl's SCOUT idea for testing if a 'good enough' evaluation

can be achieved, in domains with partially ordered values. We also discussed the use of partial orders for move ordering and pruning.

We used POB to solve semeai problems in Go and validated the method using a comparison to standard alpha-beta search. We also tested a partial order semeai evaluation against two of the world's top Go programs, with excellent results.

There are many open questions indicating promising directions for future work: Can our approach be used for algorithms similar to IDMOA*, to make this kind of search methods work for arbitrary partial orders? Can work on evaluation function learning be extended to learn partial order evaluations? How can POB be applied to other games, especially with a heuristic partial order evaluation? Is it effective in games with less structure and fewer decomposition opportunities than Go? What are good partial order evaluations for other phases of a Go game? In semeai, can stronger partial orders be developed, for example for semeai where one player has split the opponent's stones into two parts?

References

L.V. Allis. *Searching for Solutions in Games and Artificial Intelligence.* PhD thesis, University of Limburg, Maastricht, 1994.

P. Dasgupta, P. Chakrabarti, and S. DeSarkar. Multiobjective heuristic search in AND/OR graphs. *Journal of Algorithms*, 20(2):282–311, 1996.

P. Dasgupta, P. Chakrabarti, and S. DeSarkar. Searching game trees under a partial order. *Artificial Intelligence*, 82(1–2):237–257, 1996.

S. Harikumar and S. Kumar. Iterative deepening multiobjective A*. *Information Processing Letters*, 58(1):11–15, 1996.

M. Müller. *Computer Go as a Sum of Local Games: An Application of Combinatorial Game Theory.* PhD thesis, ETH Zürich, 1995. Diss.Nr.11.006.

M. Müller. A program for fighting semeai in Go. Technical report, Electrotechnical Laboratory, Tsukuba, Japan, 1999. to appear.

J. Pearl. *Heuristics: Intelligent search strategies for computer problem solving.* Addison-Wesley Publishing Company, 1984.

B. Stewart and C. White. Multiobjective A*. *Journal of the ACM*, 38(4):775–814, 1991.

Appendix: Go Terms used in this Paper

Block A connected component of stones of the same color. Blocks have *liberties*, which can be distributed over several *regions*. A block that loses its last liberty is captured. In Figure 5 there is a black block and a white block consisting of 3 stones each. The black block has three liberties marked 'a', while the white block has two liberties in two different regions of the board (one in each) marked 'b' and 'c' respectively.

Liberty An empty point adjacent to a stone is a liberty for the *block* containing that stone. In semeai, liberties can be classified into outside liberties, eye space and common liberties. See the *semeai* entry for some examples.

Region An area surrounded by stones of one or both colors. In Figure 5 there are three regions marked 'a', 'b' and 'c'.

Eye A small area surrounded by stones of one color, possibly containing some opponent stones.

Eye status A measure of the strength of an eye in semeai, depending on the size of the eye. The eye status is 0 for regions which are not eyes, 1 for 'small' eyes of size 1, 2 or 3, and n for 'large' eyes of size $n, 4 \leq n \leq 7$.

Semeai A race to capture between non-alive stones of both players. Figure 5 shows a simple semeai between three black and three white stones. Figure 6 shows a semeai where black has a five point *eye* containing one white stone. White has a four point eye, and six external *liberties* distributed over two regions. There are two shared liberties between the black and white stones.