

A Study of Decision Error in Selective Game Tree Search

Keh-Hsun Chen

Department of Computer Science
University of North Carolina at Charlotte
Charlotte, NC 28223, USA

E-mail: chen@uncc.edu

Abstract

In this paper, we study decision errors caused by the omission of part of the legal candidate moves and the inaccuracy of static evaluation in a selective minimax game tree search. Error upper bounds are presented in Section 2. A simple game tree model, which captures some basic characteristics of the Go game tree, is introduced in Section 3 for a decision error simulation study. Section 4 presents the result of this simulation study, which shows that a global selective search can be effective for game trees similar to this model. The result also reveals the existence of pathology in selective minimax game tree search.

Keywords: selective game tree search, heuristic evaluation, mini-max backup, decision error.

1. Introduction

The decision error in a full-width minimax game tree search comes from an inaccuracy of the static evaluation function, which is studied in Beal [1], Bratko & Gams [2], and Pearl [5]. A selective minimax game tree search only explores part of the successors for each node. It reduces the branching factor and allows the search to go deeper for an equal amount of effort, but it introduces another major source of decision error by not exploring some of the successors at each node. The best move might be omitted or might not be recognized as the best choice because of the omission of descendants.

In the next section, we shall prove the upper bounds of decision error caused by the evaluation error, the omission error, and their combination. In Section 3, we introduce a simple game tree model, which possesses some basic characteristics of a Go game tree. A compact linear representation of game trees with a uniform branching factor is used to implement a simulation study. The result of the simulation is presented in Section 4, which shows that a selective game tree search is an effective method for our game tree model and also reveals the existence of pathology in a selective minimax game tree search. The pathology of a full-width minimax game tree search was investigated in Nau [3, 4] and Pearl [5].

2. Evaluation error and omission error

In this section, we investigate upper bounds of the decision error caused by an inaccurate static evaluation function and a selective node expansion strategy. We assume there is a pay-off value associated with each game ending position, such as number of points won by Black in a Go game. Let n be any node in a game tree. We use $v(n)$ to denote the game theoretic value of n , i.e. the complete mini-max backup of the pay-off values from the terminal nodes that represent game ending positions. Thus, $v(\text{move } i \text{ at node } n)$ is defined to be $v(n_i)$, where n_i is the successor of n generated by the move i . All values are from max-player's point of view. $\text{Eval}(n)$ shall denote the static evaluation function value on node n . $\text{Backup}(n, d)$ shall denote the mini-max backup value at node n for full-width minimax game tree search, from a predetermined fixed depth d , where nodes are evaluated by the static evaluation function Eval .

Proposition 1. Let e be a constant denoting an error bound of the evaluation function Eval , i.e. $v(n)-e \leq \text{Eval}(n) \leq v(n)+e$ for any node n at search depth d where the evaluation function is applied. Then

$$v(n_0)-e \leq \text{Backup}(n_0, d) \leq v(n_0)+e$$

where n_0 is the root node of the search tree, i.e. the node at depth 0.

Proof. By induction on the search depth d .

Thus, if the evaluation function has an error bound e at the search frontier, then the backup value, through the search tree, also has the same error upper bound e . A full-width minimax search using this evaluation function may select a move worse than the optimal move by a value no more than $2e$. We call the difference between the value of an optimal move and the value of the selected move by the search algorithm the decision error of the search algorithm on the given node. Therefore, a full-width minimax search algorithm has a decision error bounded above by $2e$ under the assumption of Proposition 1.

Now we consider propagation of the omission error, i.e. error due to omitting the expansion of some of the successors at each node. Let $S(n)$ be the set of all successors of node n and $S'(n)$ be the subset of $S(n)$ that the selective search algorithm selects to explore. Let e_0 be a constant. We say that a selective search algorithm on a game tree has an omission error upper bound e_0 if

$|M\{v(m) : m \in S(n)\} - M\{v(m) : m \in S'(n)\}| \leq e_0$ for all interior nodes n , where M is the Max(Min) operator, if n is a max(min) node. Let the current node be n_0 . $\text{Backup}'(n_0, d)$ denotes the mini-max backup value of n_0 by the selective minimax game tree search algorithm from depth d . We find that the backup omission error upper bound grows linearly to the search depth:

Proposition 2. If there is no evaluation error, i.e. $\text{Eval}(n) = v(n)$ for all nodes at search frontier(depth d) then

$$v(n_0) - d * e_0 \leq \text{Backup}'(n_0, d) \leq v(n_0) + d * e_0$$

Proof. By induction on search depth d . Let children of n_0 be $n_1, n_2, n_3, \dots, n_b$. By induction hypothesis, $v(n_i) - (d-1) * e_0 \leq \text{Backup}'(n_i, d-1) \leq v(n_i) + (d-1) * e_0$

for $i = 1, 2, \dots, b$. Hence,

$$\begin{aligned} & v(n_0) - d * e_0 \\ &= M\{v(n_i) \mid i = 1, 2, \dots, b\} - e_0 - (d-1) * e_0 \\ &\leq M\{v(n) \mid n \text{ in the selected set } S'\} - (d-1) * e_0 \\ &= M\{v(n) - (d-1) * e_0 \mid n \text{ in the selected set } S'\} \\ &\leq M\{\text{Backup}'(n, d-1) \mid n \text{ in } S'\} \\ &= \text{Backup}'(n_0, d) \\ &\leq M\{v(n) + (d-1) * e_0 \mid n \text{ in the selected set } S'\} \\ &= M\{v(n) \mid n \text{ in the selected set } S'\} + (d-1) * e_0 \\ &\leq M\{v(n_i) \mid i = 1, 2, \dots, b\} + e_0 + (d-1) * e_0 \\ &= v(n_0) + d * e_0 \end{aligned}$$

Therefore, we have

Corollary 2.1. If the evaluation function is accurate at the search frontier and S' contains an optimal successor of n , then the decision error of a depth d selective search is no more than $2(d-1)e_0$.

Proof. Assuming the current node is a max node,

$$\begin{aligned} & v(\text{optimal child}) - v(\text{selected child}) \\ &\leq (\text{Backup}'(\text{optimal child}, d-1) + (d-1)e_0) - \\ &\quad (\text{Backup}'(\text{selected child}, d-1) - (d-1)e_0) \\ &= 2(d-1)e_0 + (\text{Backup}'(\text{optimal child}, d-1) - \\ &\quad \text{Backup}'(\text{selected child}, d-1)) \\ &\leq 2(d-1)e_0 \end{aligned}$$

(selected child has highest backup value)

The case that the current node is a min node can be proved similarly.

If S' does not contain an optimal successor of n , then the decision error bound is worsened by another e_0 :

Corollary 2.2. If the evaluation function is accurate at the search frontier and S' may or may not contain an optimal successor of n , then the decision error of a depth d selective search is no more than $(2d-1)e_0$.

Proof. Without loss of generality, assume the current node is a max node.

$$\begin{aligned} v(\text{selected child}) &\geq \text{Backup}'(\text{selected child}, d-1) - (d-1)e_0 \\ &= \text{Backup}'(n_0, d) - (d-1)e_0 \\ &\geq v(n_0) - d e_0 - (d-1)e_0 \\ &= v(n_0) - (2d-1) e_0 \end{aligned}$$

Taking both evaluation error and omission error into consideration, we have

Proposition 3. $v(n_0) - e - d * e_0 \leq \text{Backup}'(n_0, d) \leq v(n_0) + e + d * e_0$

Proof. By induction on the search depth d .

Corollary 3.1. The decision error of a selective alpha-beta search to depth d is bounded by $2e + (2d-1)e_0$ where e is the evaluation error bound and e_0 is the omission error bound.

The error bounds show the worst case behavior. In the next two sections, we shall study the result of a Monte Carlo simulation of a selective minimax search on a simple game tree model to gain some insight on selective minimax search's average behavior on decision error.

3. A simple game tree model

In this section, we introduce a simple game tree model, which possesses some basic characteristics of the Go game tree. The game tree has a uniform branching factor b . The height of the tree is h . Each arc in the tree is associated with a random integer between 0 and m (when you make the move represented by the arc, you can gain that many points). The pay-off of a terminal node at level h equals $v_1 - v_2 + v_3 - v_4 \dots v_h$ where $v_1, v_2, v_3, \dots, v_h$ are the random integers associated with arcs along the path from the root to the terminal node. We evaluate the game theoretic value of each node by performing mini-max backup from the terminal nodes at level h . For the static evaluation function $\text{Eval}(n)$, we use $v_1 - v_2 + v_3 - v_4 \dots v_l$ of the path from the root to the node at level l . We note that $\text{Eval}(n)$ and $v(n)$ usually have different values and our evaluation function Eval tends to be more accurate when the node is deeper in the game tree (closer to terminal nodes). On game ending terminal nodes, the evaluation function Eval is accurate. Our selective search at each level will drop the s ($\leq b-2$) worst successor nodes based on the evaluation function values. The s is called the skip number. The move selection at the root is determined by the backup values of the surviving children from the selective search. The game theoretic value of the selected move is compared with the game theoretic value of the optimal move to determine the decision error.

The simulation is implemented via a compact array representation of the game tree as follows. We represent a tree of height h and uniform branching factor b as an array $a[0..s-1]$, where $s = b^0 + b^1 + b^2 + \dots + b^h$ is the size of the tree. The nodes are level ordered, with the root, level 0, at $a[0]$, then its children, level 1 nodes, at $a[1..b]$, ... etc. The parent of node $a[k]$ is $a[(k-1)/b]$ ($/$ truncates the quotient to an integer). The children of $a[k]$ are $a[k*b+1], a[k*b+2], \dots, a[k*b+b]$. Level of $a[k] =$ the smallest m such that $b^0 + b^1 + b^2 + \dots + b^m > k$ and the height of $k = h - \text{level of } k$. Nodes at level L are $a[(b^1 + b^2 + \dots + b^{L-1} + 1) .. (b^1 + b^2 + \dots + b^{L-1} + b^L)]$. The whole tree is packed compactly into a linear array. A dynamic programming approach is used to compute the Eval function values from the root of the whole tree down towards the leaves, where the Eval value also becomes the pay-off value. Another pass computes game theoretic values for the game tree nodes bottom-up. The selective minimax tree search is performed in normal depth first fashion with $b-s$ successors with highest Eval values expanded and the remaining s successors ignored. When search depth d is reached, the Eval value of the frontier node is used for back-up.

4. Result of simulation

Simulation studies were done for each of the branching factors $b = 2, 3, \dots, 14$. The tree height was set so that the tree size did not exceed main memory size too much (so that there would not be too much slow down by OS virtual memory system). Arc values were random numbers between 0 and 9 inclusive. For each b , each skip value $s = 0, 1, \dots, b-2$ are simulated in turn. The decision error for each (b, h, h_1, d, s) -tuple is averaged over at least 1000 simulations, where b is the branching factor, h is the height of the game tree, h_1 is the height of the node, d is the search depth, and s is the skip number. Results of the two cases $(5, 10, 9, d, s)$ and $(11, 7, 7, d, s)$ are representative and summarized in Tables 1~4 and Figures 1 and 2.

Table 1. Summary result of a simulation study, where branching factor = 5, tree height = 10, node height = 9, search depth = d, #moves skipped at each internal node = s. Each entry shows an average decision error in points. For comparison, the average game theoretic node value is 4.3.

d \ s	0	1	2	3
1	0.949	0.949	0.943	0.940
2	0.618	0.617	0.611	0.687
3	0.450	0.459	0.482	0.530
4	0.348	0.352	0.378	0.537
5	0.302	0.300	0.320	0.459
6	0.240	0.252	0.319	0.493
7	0.196	0.207	0.272	0.481
8	0.136	0.157	0.246	0.457
9	0.000	0.035	0.155	0.473

Table 2. Same parameter values as in Table 1. Each entry indicates number of nodes generated per search.

d \ s	0	1	2	3
1	6	5	4	3
2	31	21	13	7
3	156	85	40	15
4	781	341	121	31
5	3906	1365	364	63
6	19531	5461	1093	127
7	97656	21845	3280	255
8	488281	87381	9841	511
9	2441406	349525	29524	1023

Ave. decision error

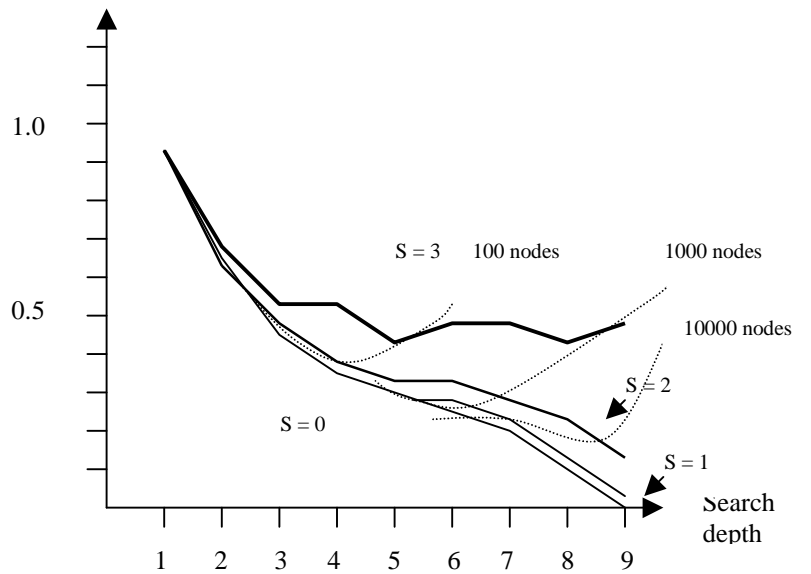


Fig. 1 Performance comparison for selective searches on game trees with branching factor = 5, tree height = 10, & node height = 9.

Table 3. Summary result of another simulation study, where branching factor = 11, tree height = 7, node height = 7, search depth = d, #moves skipped at each internal node = s. Each entry shows an average decision error in points. For comparison, the average game theoretic node value is 8.9.

d\s	0	1	2	3	4	5	6	7	8	9
1	0.535	0.535	0.535	0.535	0.535	0.535	0.535	0.540	0.550	0.515
2	0.370	0.370	0.370	0.370	0.370	0.360	0.360	0.390	0.390	0.430
3	0.245	0.245	0.245	0.245	0.245	0.245	0.250	0.285	0.295	0.320
4	0.250	0.250	0.250	0.250	0.250	0.250	0.240	0.245	0.270	0.305
5	0.180	0.180	0.180	0.180	0.180	0.180	0.185	0.195	0.250	0.255
6	0.095	0.095	0.095	0.095	0.095	0.095	0.100	0.095	0.155	0.285
7	0.000	0.000	0.000	0.000	0.000	0.005	0.005	0.025	0.090	0.260

Table 4. Same parameter values as in Table 3. Each entry indicates number of nodes generated per search.

d\s	0	1	2	3	4	5	6	7	8	9
1	12	11	10	9	8	7	6	5	4	3
2	133	111	91	73	57	43	31	21	13	7
3	1464	1111	820	585	400	259	156	85	40	15
4	16105	11111	7381	4681	2801	1555	781	341	121	31
5	177156	111111	66430	37449	19608	9331	3906	1365	364	63
6	1948717	1111111	597871	299593	137257	55987	19531	5461	1093	127
7	21435888	11111111	5380840	2396745	960800	335923	97656	21845	3280	255

Ave. decision error

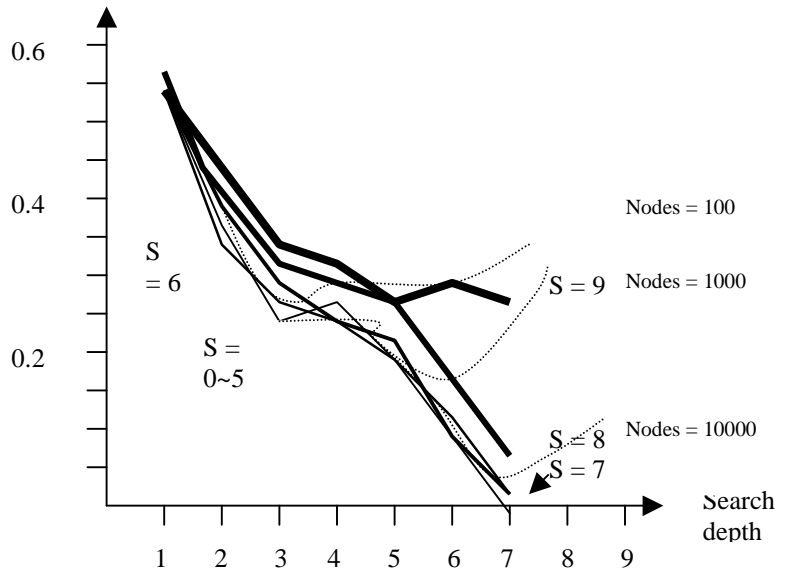


Fig. 2 Performance comparison for selective searches on game trees with branching factor = 11, tree height = 7, & node height = 7.

The dotted lines in Figs. 1 & 2 show equal search efforts in terms of number of nodes generated. The lower the vertical value the better (smaller decision error) the result. Figure 1 shows that the selective search skipping 2 moves (i.e. exploring only the 3 moves with best evaluation function values at each node) works the best for the studied cases in the (5, 10, 9, d, s) situations. Figure 2 favors skipping 7 for 100 or 10000-node searches, and favors skipping 8 for 1000-node search in the (11, 7, 7, d, s) situations. All the simulation results, including those not shown here, point to the conclusion that selective search can be very effective for this simple game tree model, which suggests global selective search may also be an effective method for Go.

A downward curve in the Figures 1 and 2 means that the deeper you search, the smaller decision error you get. The fact that the curves in Figures 1 and 2 do not always go downwards reveals the existence of pathology in selective minimax search. This comes as no surprise since the omission error propagates much worse than the evaluation error.

References

- [1] D.F. Beal, An analysis of minimax, in: M.R.B. Clarke (Ed.), *Advances in Computer Chess 2*, Edinburgh University Press, 1980, pp. 103-109.
- [2] I. Bratko and M. Gams, Error analysis of the minimax principle, in: M.R.B. Clarke (Ed.), *Advances in Computer Chess 3*, Pergamon Press, London, 1982, pp. 1-15.
- [3] D. S. Nau, An investigation of the causes of pathology in games, *Artificial Intelligence* 19, 1982, pp. 257-278.
- [4] D. S. Nau, Pathology on Game Trees Revisited, and an Alternative to Maximizing, *Artificial Intelligence* 21, 1983, pp. 221-244.
- [5] J. Pearl, On the Nature of Pathology in Game Searching, *Artificial Intelligence* 20, 1983, pp. 427-453.