A Quantitative Analysis of Minimal Window Search

Alexander Reinefeld
FB Informatik, Universitat Hamburg
Bodenstedtstr. 16, D-2000 Hamburg 50
<reinefeldOrz.informatik.uni-hamburg.dbp.de>

T.A. Marsland
Computing Science, University of Alberta
Edmonton, Canada T6G 2H1
<tonyOalberta.cdn>

Abstract

Use of minimal windows enhances the aB algorithm in practical applications as well as in the search of artificially constructed game trees. Nevertheless, there exists no theoretical model to measure the strengths and weaknesses of minimal window search. In particular, it is not known which tree ordering properties are favorable for minimal window search. This paper presents a quantitative analysis of minimal window search based on recursive equations which assess the influence of static node values on the dynamic search process. The analytical model is computationally simple, easily extendible and gives a realistic estimate of the expected search time for averagely ordered game trees.

1 Introduction

The structure most frequently used for evaluating the performance of tree searching algorithms consists of a uniform tree of depth *d* and width *w.* In such regular trees the nodes can be categorized into type classes according to whether all descendants must be searched or only some of them (the so-called *cut-nodes* [4]). This was first done by Knuth and Moore [3], who classified the nodes of optimally ordered game trees into three types (Figure 1):

- Nodes of type 1 are examined with the full-width window $(-\infty, +\infty)$. The leftmost descendant is again of type 1 and the remaining $w - 1$ descendants are of type 2.
- Nodes of type 2 are examined with the window $(-\infty, \beta)$, which allows cut-offs. Hence, in optimally ordered trees, there exists only one descendant, which is of type 3.
- Nodes of type 3 are examined with the window $(\alpha, +\infty)$. All *w* descendants are of type 2.

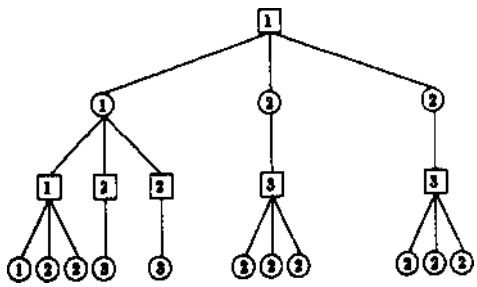

Figure 1: Node types in an optimally ordered tree

In the following, we extend this node classification scheme to include unordered game trees and derive recursive equations to compute the average performance of both the aB-algorithm and minimal window search.

2 Analysis of Minimal Window Search

Minimal window search is based on the assumption that most of the subtrees will prove inferior to the best subtree searched so far. Having traversed the first subtree with a normal window (α, β), the remaining subtrees are searched with the minimal window $(v, v+1)$, where *v* represents the best available minimax value (e.g., the value obtained from the first search). If the value returned is $\leq v$, the subtree is indeed inferior. Only when the value returned is $> v$, is the initial assumption wrong and the current subtree must be re-searched with a wider window to determine its correct value. The re-search can be done either with an aB search in Scout [6,1] and *CoB)* [2], or by performing a recursive self-call, as in NegaScout [7] and PVS [4].

```
integer procedure NS (position p, integer α, β, d);
begin integer i, w, t, lo, hi;
    if d = 0 then
        return Value (p);                          (* horizon cost*)
    determine the successor positions: p.1,...,p.w;
    lo ← α;
    hi ← β;
    for i ← 1 to w do begin
        t ← - NS (p.i, -hi, -lo, d - 1);
        if t > lo and t < β and i > 1 then
            t ← - NS (p.i, -β, -t, d - 1);    (* re-search *)
        lo ← max (lo, t);
        if lo ≥ β then
            return lo;                              (* cut-off *)
        hi ← lo + 1                          (* minimal window *)
    end;
    return lo
end;
```

Figure 2: Basic NegaScout Algorithm (NS)

To simplify presentation, the self-call minimal window version of NegaScout, as shown in Figure 2, is referred to as NS. A variation, designated NS^{aB}, invokes $\alpha\beta$ to do the re-search. It is formed by replacing the corresponding line of NS with

$$t \leftarrow -\alpha\beta\ (p.i,\ -\beta,\ -t,\ d-1).$$

The nodes visited when NegaScout traverses an averagely ordered tree are shown in Figure 3 There the node types are prefixed with an "JV", and are changed from Figure 1 because the tree is not optimal. For simplicity we initially assume that a re-search is never necessary, so Figure 3 is valid for both NS and NS^{ab}. NegaScout starts searching the root node of type *NX* with the full window $(-\infty, \infty)$. Since there is not yet any better information, the root's left descendant is searched with the same full window, hence it is also of type *Nl.* The remaining descendants of type *NA* are searched with the minimal window $(v, v+1)$. According to our average tree assumption, these cut-nodes have only 1 -f *g* descendants. The first *g* descendants are again of type A4, whereas the (1 + y)-th descendant is of type N5, since all *w* descendants of this *N5* node must be searched to cause the cut-off one level above.

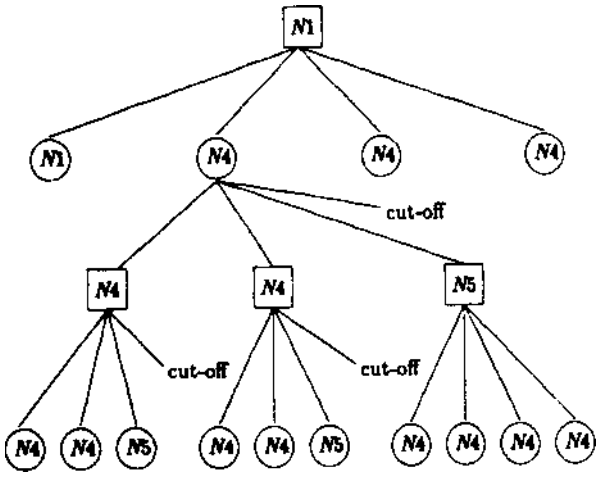

Figure 3: An averagely ordered NegaScout tree $(w = 4, g = 2)$

Based on Figure 3, the node dependencies can be formulated mathematically. Following an established notation [4], let k be the processing cost of an internal node and let e be the cost of evaluating a terminal node. Then $N1_d$, the cost for searching a uniform tree of depth d and width w with $1 + g$ descendants at cut-nodes is given by:

$$\begin{aligned} N1_d &= k + N1_{d-1} + (w-1)N4_{d-1} && \text{for } d > 0 \\ N1_0 &= e \end{aligned}$$

$$\begin{aligned} N4_d &= 2k + gN4_{d-1} + wN4_{d-2} && \text{for } d > 1 \text{ and } 0 \le g < w \\ N4_1 &= k + (1+g)e \\ N4_0 &= e \end{aligned}$$

The consistency of these equations may be checked by observing the special case of optimally ordered game trees (i.e., the case $g = 0$). By setting k - 0 (interior nodes have no cost) and e - 1 (all leaf nodes have equal cost) it is possible to show that

$$\begin{aligned} N1_{2m+1} &= (w-1)(w^0 + w^0 + \cdots + w^{m-1} + w^{m-1} + w^m) + 1 \\ &= 2(w-1) + 2(w^2 - w) + \cdots \\ &\quad \cdots + 2(w^m - w^{m-1}) + (w^{m+1} - w^m) + 1 \\ &= w^m + w^{m+1} - 1, \end{aligned}$$

which is equivalent to abest case in odd-depth trees, as it should be [3]. Similarly, the consistency can be checked for even search depths.

3 Analysis of $\alpha\beta$

The quantitative analysis of aB is more complex, because < $\alpha\beta$'s search process uses a window that may take any size from minimal to full-width. As before, we first present Figure 4, the average tree searched by *aB.* Again, the root node of type Al is searched with the full window $(-\infty, \infty)$ and has descendants of type Al and A2. The latter are searched with a reduced window $(-\infty, \beta)$ which allows cut-offs. So, only 1 + *g* descendants need to be searched. The leftmost *At* descendant is of type A3 and is searched with the window $(-\beta, \infty)$. None of its descendants can be pruned, because the upper bound of the window is still infinite. However, the leaf node values gathered in this subtree establish a tight B-bound for the search of the next $g - 1$ nodes of type *AA.* Therefore, only $1 + g$ descendants need to be searched at this point The rightmost A2 descendant of type A5 finally returns the cut-off value, but only after all its descendants have

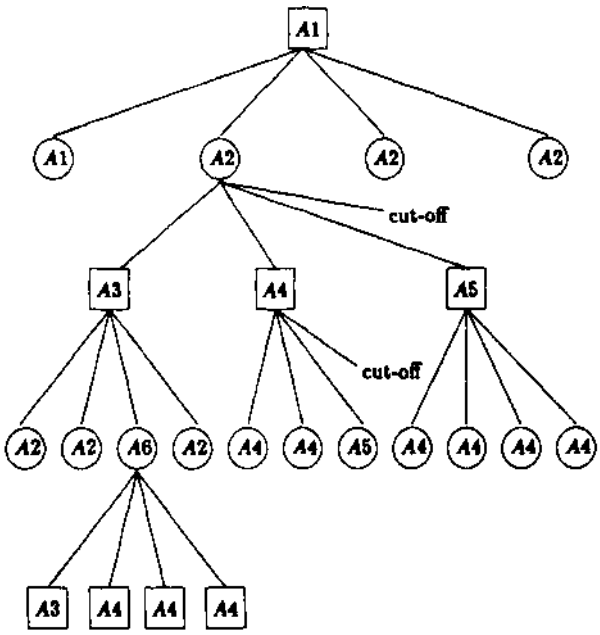

Figure 4: An averagely ordered $\alpha\beta$-tree $(w = 4, g = 2)$

been searched. Although they have the same number of descendants, it is necessary to distinguish between nodes of type *A2* and *AA*. The reason is that the *A2* nodes are expanded with $\alpha = -\infty$, so that all of its left descendant's descendants must be searched. Nodes of type A4, in contrast, are searched with a narrow window (α, β), and that allows cut-offs in all deeper tree levels.

Comparing both sample trees (Figures 3 and 4), clearly $\alpha\beta$'s lack of a finite B-bound at the *A3* nodes causes the degraded performance. NegaScout's minimal window search, in contrast, achieves a cut-off after $1 + g$ descendants have been searched at the *N4* nodes that correspond to aB's *A3* nodes To make things worse for *aB,* each *A3* node has a descendant of type A6, which in turn has an A3 descendant (see Figure 4).

As for $N1_d$, $A1_d$ (the cost for searching a uniform tree of depth d and width w with the $\alpha\beta$-function) may be estimated by

$$\begin{aligned} A1_d &= k + A1_{d-1} + (w-1)A2_{d-1} && \text{for } d > 0 \\ A1_0 &= e \end{aligned}$$

$$\begin{aligned} A2_d &= 2k + A3_{d-1} + (g-1)A4_{d-1} + wA4_{d-2} && \text{for } d > 1 \\ A2_1 &= k + (1+g)e && \text{and } 0 < g < w \\ A2_0 &= e \end{aligned}$$

$$\begin{aligned} A3_d &= 2k + A3_{d-2} + (w-1)A4_{d-2} + (w-1)A2_{d-1} && \text{for } d > 1 \\ A3_1 &= k + we \\ A3_0 &= e \end{aligned}$$

$$\begin{aligned} A4_d &= 2k + gA4_{d-1} + wA4_{d-2} && \text{for } d > 1 \\ A4_1 &= k + (1+g)e && \text{and } 0 < g < w \\ A4_0 &= e \end{aligned}$$

It is now clear that the equations for $A4_d$ and $N4_d$ have the same form, although $\alpha\beta$ uses only a narrow window (α, β) rather than a minimal window $(v, v+1)$.

4 Re-Search Overhead

The model discussed so far is still too optimistic since it assumes that the principal variation always lies in the leftmost path. However, it is not the location of the principal variation that is of prime importance but rather the frequency with which it changes. Assuming r principal variation changes, $\alpha\beta$'s root node has $r + 1$ descendants of type $A1$: One lies to the left and there are r additional $A1$ descendants, one for each change of location. Thus for $d > 0$ and $0 \le r < w$,

$$\begin{aligned} A1_d &= k + (r+1)A1_{d-1} + (w-r-1)A2_{d-1} \\ A1_0 &= e. \end{aligned}$$

Whenever there is a change in the principal variation, NegaScout must do a re-search. This can be achieved either with a recursive self-call (as in NS) or with a call to $\alpha\beta$ (as in $NS^{\alpha\beta}$). Hence for $d > 0$ and $0 \le r < w$ we obtain two equations:

$$\begin{aligned} N1_d &= k + (r+1)N1_{d-1} + (w-r-1)N4_{d-1} + rN5_{d-1} \\ N1_d^{\alpha\beta} &= k + N1_{d-1}^{\alpha\beta} + rA1_{d-1} + (w-r-1)N4_{d-1} + rN5_{d-1} \end{aligned}$$

with

$$N1_0 = N1_0^{\alpha\beta} = e.$$

From Figure 3, $N5_d$ (the cost of a minimal window search that fails) may be estimated as:

$$\begin{aligned} N5_d &= k + wN4_{d-1} \\ N5_0 &= e. \end{aligned}$$

For simplicity we omitted in the above equations NegaScout's "depth-2"-improvement [7], which ensures that no re-search occurs when the remaining search depth is ≤ 2. More precisely, for the case $0 < d \le 2$,

$$N1_d = k + N1_{d-1} + (w-r-1)N4_{d-1} + rN5_{d-1}$$

A similar equation can be derived for $N1_d^{\alpha\beta}$.

5 Numerical Comparison

The equations just established are complex, but yield to numerical evaluation. Because there are many free variables (d, w, g, r), and also (e, k), a variety of graphical presentations is conceivable. For example, the data in Figure 5 are obtained from trees of fixed depth d - 7, in which the principal variation changes twice at each type-1 node (r - 2). The data-points show the search complexity as a function of g (the number of extra-expansions in cut-nodes) for various tree widths. The search complexity is given in terms of leaf-node evaluations (hence $e = 1, k = 0$) normalized to $\alpha\beta$.

Despite being degraded by two re-searches in each *NI* node, the data in Figure 5 shows that both NegaScout variants outperform *aB* in well ordered trees with low g-values. This illustrates how severely a/? is hurt by its missing B-bound in the left part of all *A2* subtrees. NegaScout, in contrast, searches these parts with a minimal window at the risk of eventually being forced to do a re-search later. The advantages of minimal window search become even more apparent in wide trees, because the overhead of the two re-searches has less influence on the overall performance.

From Figure 5 one can see that NS is always better than NS^{ab} if fewer than 20% of the descendants at type 2 nodes must be expanded, even though a second re-search occurs. If more than 20% of the cut-node descendants are searched, both NegaScout variants are consistently poorer than aB. So we can do without since either pure *aB* or pure NS has the advantage.

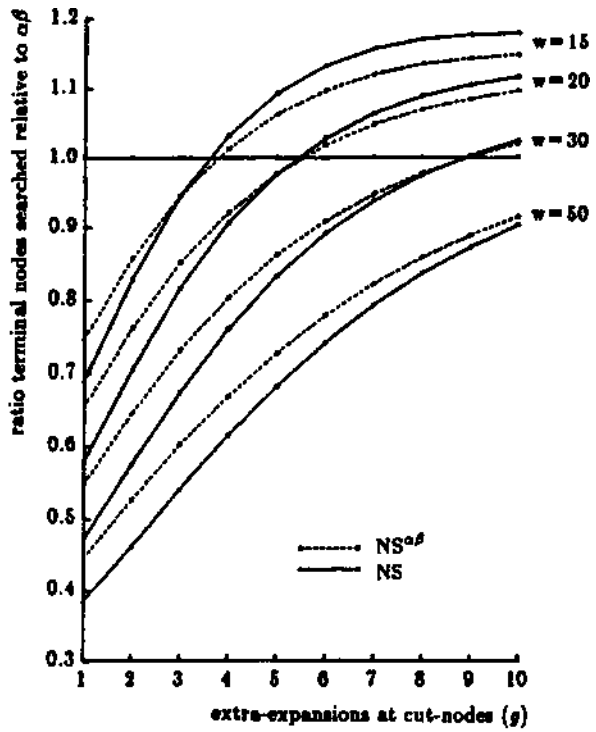

Figure 5: Numerical evaluation of $N1_d$ and $N1_d^{\alpha\beta}$ ($r = 2, d = 7$)

The choice between NS and $\alpha\beta$ depends solely on the tree ordering properties inherent in the given application. As an illustration, consider the search on trees of depth $d = 7$ and width $w = 20$, Figure 6. Clearly, those trees with g/r ratios lying below the thick line should be searched by NS, but if the tree is poorly ordered (i.e., the region above the line) $\alpha\beta$ should be used instead. Thus only if the re-search rate is greater than 15%, or if the principal variation is consistently in the right-half of the tree ($g > w/2$), does $\alpha\beta$ become attractive.

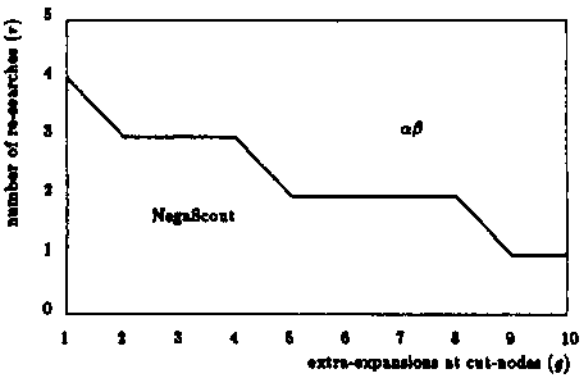

Figure 6: $\alpha\beta$ vs. NegaScout in trees of $w = 20$ and $d = 7$

6 Discussion

In this paper, we presented a simple, flexible and extendible analytical model to assess the expected search time of NegaScout and aB on averagely ordered trees. The model can be used to decide which of the algorithms to use in a given search task, when the ordering of the tree is known in a statistical sense.

The tree ordering properties are described in terms of g, the number of extra-expansions in cut-nodes, and r, the frequency of changes in the principal variation. In practice, r is usually small since the second-best choice prunes almost as effectively as the best. Although *g,* the number of extra-expansions in cut-nodes, might be as large as w/2 in random trees, most often heuristic information is available to sort the descendants before examination. For computer chess, average values of *g* in the vicinity of 1 are observed and in one study an average of 0.5 was quoted [4, p. 449]. Although the mentioned study is not strictly comparable, since it used additional special-purpose pruning methods, it supports the view that the average value of *g* is small. Thus our model provides the theoretical basis to explain why NegaScout and PVS are superior to aB in practice (5).

References

[1] M.S. Campbell, T.A. Marsland. *A comparison of minimax tree search algorithms.* Artificial Intelligence 20,4(1983), 347-367.

(2) J.P. Fishburn. *Analysts of Speedup in Distributed Algorithms.* UMI Research Press, Ann Arbor (1984) (see also Univ. Wisconsin, Ph.D. thesis, 1981).

[3] D.E. Knuth, W Moore. *An analysis of alpha-beta pruning.* Artificial Intelligence 6,4(1975), 293-326.

[4] T.A Marsland, F. Popowich. *Parallel game-tree search.* IEEE PAMI-7,4(1985), 442-452.

[5] A. Muszycka, R. Shinghal. *An empirical comparison of pruning strategies in game trees.* IEEE SMC 15,3(1985), 389-399.

[6] J. Pearl. *Asymptotic properties of minimax trees and game-starching procedures.* Artificial Intelligence 14(1980), 113-138.

[7] A. Reinefeld. *An improvement of the Scout tree search algorithm.* ICCA Journal 6,4(1983), 4-14.