Analysis of pruned minimax trees

Daniel S. Abdi
dshawul@yahoo.com

June 6, 2013
Modified: July 24, 2015

Abstract

This paper examines the number of nodes examined by different pruning algorithms of the Minimax tree. An extensive literature review on analysis of alpha-beta pruning is done. The lessons learned are used to analyze un-sound pruning methods, late move reductions and null move pruning, that are commonly used in computer chess programming. Formulas are derived to calculate the number of leaf nodes of a pruned tree and also classify the nodes according to Knuth's scheme. An asymptotic analysis of late move reduction heuristics is attempted using the derived formulas.

1 Introduction

Computers play two-player games by constructing decision trees that minimizes the loss for a worst case scenario or simply 'Minimaxing'. The Minimax trees are never fully explored and some kind of pruning is used in practice. There are many methods to prune the tree with or without loss of information. The most well known sound pruning method for two-player zero sum games is alpha-beta pruning. This method has received a lot of attention since early computer days and is proven to be asymptotically optimal over the class of directional algorithms [6]. Rarely is alpha-beta used alone in game playing programs. Other pruning methods that are un-sound are used to further reduce the game tree size. The question of a pruning method being unsound is practically irrelevant because the aim is to out-search the opponent and make the best possible move with in the alloted time. Searching deeper usually results in better play even though there are pathological cases where worse moves can be made [5]. In general play gets better by searching deeper though with diminishing returns. Two domain independent pruning heuristics that have proven to be successful in chess playing programs are the null-move pruning and the recently popular late move reductions. Unlike alpha-beta pruning, these methods did not enjoy much theoretical analysis. This paper attempts to make analysis of these pruning methods following methodologies used for analyzing alpha-beta.

The pruning algorithms are studied by enumerating the number of leaf nodes explored by the algorithms.

1. **perft(d)** : The number of games that can be played from a given position to a certain depth. Discussion on calculating the exact perft(d) and estimation of it using monte-carlo methods can be found in Abdi [1].

2. **perft(d,type)** : Nodes are classified into three groups commonly known as PV (type = 1), CUT (type = 2) and ALL (type = 3). The formulas we will develop will give a break down of the number of leaf nodes in to these groups.

The lower the number of leaf nodes examined the better the algorithm is. However this definition is not precise for the un-sound pruning methods to be discussed later. First the number of leaf nodes examined by some pruning methods is not always representative of the total number of nodes explored. For example, late move pruning examines quite few nodes at the last ply. Therefore we will use the total number of nodes as a way of comparing algorithmic complexity of different algorithms. The second problem is that the end result i.e. the score and main line obtained at the root are different when un-sound prunings are used. Since alpha-beta pruning returns exactly same result as that of minimax, comparing those two algorithms is acceptable. However this requirement can not be satisfied by LMR and null-move pruning that give different results depending on many factors. The effective branching factor ($\Re$) defined as follows will be used to compare different algorithms.

$$\Re = \lim_{d=\infty} N^{1/d} \tag{1}$$

Other ratios to compare search efficiency have been used in literature such as the 'depth ratio' of Slagel and Dixon [8].

$$DR = \frac{\log(N)}{\log(N_{MM})} \tag{2}$$

This ratio is however dependent on a reference algorithm i.e. minimax for N_{MM}. Another parameter commonly used in computer chess programming is the ratio of total nodes searched in two consecutive iterations within the iterative deepening framework. However we believe this definition is inadequate for two reasons. First the value of the branching factor for the very first iteration is undefined. Second two consecutive iterations can be performed with a depth increment other than one so the definition has a strict adherence to iterative deepening with depth increment of one.

In the following sections $\Re$ is sought after for best-case, worst-case and average-case behavior of different pruning algorithms. The average case performance is usually the most important, but for modern game tree search where different heuristics help to bring down $\Re$ close to optimal, the best-case performance is the most important.

2 Literature review of analysis of alpha-beta

This section focus on an extensive literature review of analysis of the alpha-beta procedure by several researchers [2, 3, 4, 6, 7, 8, 9]. It will serve as a basis for analysis of other pruning methods discussed in this work and also to highlight open problems for future research. However it is mostly because the literature is interesting for us that are accustomed to the dynamic approach of process description. Knuth and Moore pointed out this can potentially hide details of alpha-beta pruning. If the reader is familiar with the literature it is recommended to go directly to section 3.

2.1 The algorithm

The minimax algorithm is described in Knuth and Moore [4] in negamax form as follows

$$F(p) = \begin{cases} f(p), & \text{if } d = 0 \\ max(-F(p_1), \cdots, -F(p_b)), & \text{if } d > 0 \end{cases} \tag{3}$$

where p_i's are children positions, $f(p)$ is static evaluation of a position and $F(p)$ is the minimaxed value of the parent position. Two improvements to this naive 'brute force' algorithm are possible by using branch-and-bound strategies.

$$F1(p, beta) \leftarrow \begin{cases} = F(p), \text{if } F(p) < beta \\ \geq F(p), \text{if } F(p) \geq beta \end{cases} \tag{4}$$

The first procedure so called F1 uses one bound beta to prune moves that can't possibly improve the evaluation of a position. This procedure is successfully used in single agent search and combinatorial optimization problems. However for the special case of two-player zero sum games a better algorithm is available so called F2.

$$F2(p, alpha, beta) \leftarrow \begin{cases} \leq alpha, \text{if } F(p) \leq beta \\ = F(p), \text{if } alpha < F(p) < beta \\ \geq beta, \text{if } F(p) \geq beta \end{cases} \tag{5}$$

This later procedure is the alpha-beta algorithm which is able to cause 'deep cutoffs' due to the additional bound alpha. However it is not as flexible as the F1 procedure in other areas of game tree search such as multi-player games. The use of alpha-beta there necessitates an assumption that all the players collude to attack the root player, which is a very unrealistic assumption. Even then only shallow cutoffs are possible thus making alpha-beta not so powerful as its use in two player games. Knuth and Moore ask an intriguing question as to why there isn't a better procedure F3 (alpha-beta-gamma) that uses the 2nd best score or other gimmick. Then they went on to prove optimality of alpha-beta for the best case search.

To visualize the algorithm better, let us look at a portion of a tree shown in Baudet [2] and calculate the backed up value $v(J)$ and bounds $c(J), \alpha(J)$, and $\beta(J)$ at any node J. We can represent the path to a give node J using Dewey decimal notation as $J = j_1 \cdots j_d$ as shown in figure 1. Any other node along this path, i distance away from node J, is represented as $J_i = j_1 \cdots j_{d-i}$. The children of node J are represented as $J.j$.

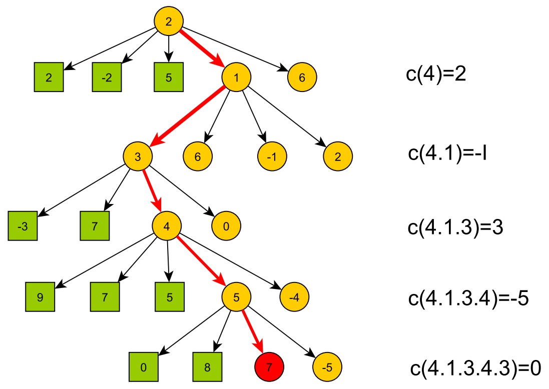

Figure 1: Portion of game tree to node 4.1.3.4.3 courtesy of Baudet [2]

$$\begin{aligned} v(J) &= \max\{-v(J.j) \mid 1 \le j \le b\} \\ c(J.j) &= \max\{-v(J.i) \mid 1 \le i \le j-1\} \\ \alpha(J) &= \max\{-c(J_i) \mid \text{i is } \textit{even}, 0 \le i \le d-1\} \\ \beta(J) &= -\max\{-c(J_i) \mid \text{i is } \textit{odd}, 0 \le i \le d-1\} \end{aligned} \tag{6}$$

2.2 Worst-case behavior

In the worst case the alpha-beta procedure examines all nodes of the minimax tree. Thus the number of nodes explored is $O(b^d)$.

$$N = b^d \tag{7}$$

Knuth and Moore notes that there are game trees where the alpha-beta procedure can not avoid to make cutoffs. This property is not enjoyed by the F1 procedure.

2.3 Best-case behavior

Analysis of the best-case behavior started with Michael Levin (1961) who described the following formula for the total number of nodes examined by alpha-beta with the best move ordering possible. Thus alpha-beta reduces tree size to the order of $O(b^{d/2})$. Knuth and Moore note perfect move ordering is not always the best, but that there always exist an order that produces the minimal tree.

$$N = b^{\lceil d/2 \rceil} + b^{\lfloor d/2 \rfloor} - 1 \tag{8}$$

The formal proof of this formula is first given by Slagel and Dixon [8]. A simple 'hand waving' proof can be made by noting that the alpha-beta procedures examines exactly the critical positions where one move is examined at odd or even plies. Thus the total number of nodes is the sum of the first two cases minus the last one which is counted twice as shown in figure 2. A generalized corollary is given by Knuth and Moore for sub-optimal alpha-beta analysis by replacing the 1's by any other value.

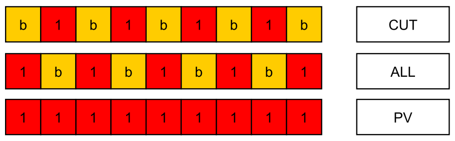

Figure 2: Number of moves for best-case alpha-beta

Knuth and Moore also analyze the F1 procedure for the best case behavior as follows. The F1 procedure does only shallow pruning as shown in figure 3 for a binary tree case. As a result it is suitable to form recurrence relations to study the best and average case behaviors. For this particular case, only the last move can get pruned by the the F1 procedure, thus all the other moves are searched with a $F1(p, \infty)$. If the number of nodes in the first and second moves at at depth d have sub-tree sizes of A_d and B_d respectively then, the recurrence relations are

$$\begin{aligned} A_{d+1} &= A_d + B_d \\ B_{d+1} &= A_d + pB_d \\ A_0 &= B_0 = 1 \\ A_1 &= 2 \end{aligned} \tag{9}$$

For the best case, p = 0 so that the last move is not searched. In that case the total number of nodes searched by the F1 procedure can be obtained by solving

$$\begin{aligned} A_{d+1} &= A_d + A_{d-1} \\ r^{d+1} &= r^d + r^{d-1} \\ r^2 &= r + 1 \end{aligned} \tag{10}$$

The solution of the quadratic equation are the eigen values of the characteristics equation. Equations for the general case with width b can be derived similarly and the solution is

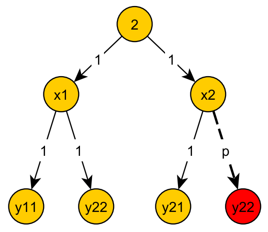

Figure 3: Binary tree with F1 pruning

$$A_d = \frac{1}{\sqrt{4b-3}}\left[\left(\sqrt{b-3/4}+1/2\right)^{d+2} - \left(-\sqrt{b-3/4}+1/2\right)^{d+2}\right] \tag{11}$$

Therefore the F1 procedure has a branching factor of $\sqrt{b-3/4}+1/2$ which is only slightly higher than $\sqrt{b}$.

2.4 Average-case behavior

This is the most challenging part of analysis of alpha-beta literature. It has taken decades to prove asymptotic optimality of the algorithm. The analysis for the average case behavior started with the work of Fuller et al. [3] who obtained formulas for computing average number of nodes examined. The formulas were too complex for full asymptotic analysis of alpha-beta, hence they carried out monte-carlo simulations instead to arrive at $\Re \sim b^{0.72}$. We will briefly discuss the approach and contribution of various researchers in the following sections.

Definition 2.1. *A **RUG** tree is a random uniform game tree with the following properties*

1. *A uniform game tree of width b and height d*

2. *The values assigned to the leaf nodes $v(\vec{i}_d)$ are independent, identically distributed (i.i.d) random variables.*

3. *The cumulative distribution function $V_D(x)$ is continuous. In other words the probability that the value of a leaf node is precisely x is vanishingly small.*

2.4.1 Knuth and Moore's analysis

The approach they took is relatively simple to understand, therefore we will start with a discussion of this particular work. They basically used the F1 procedure for analysis and then predicted the behavior of the F2 procedure from it. Deep cutoffs can not happen in the F1 procedure and this was in fact not well understood by practitioners at the time. This work is attributed to clearing up the matter in this regard. They conjectured that deep cutoffs have only a second order effect and that the conclusions on asymptotic behavior of F1 hold for F2 as well. This was later proven to be the case by Baudet [2].

Using the binary tree in figure 3 again, we can see that the last node is visited if and only if

$$-y_{21} < -min(y_{11}, y_{12}) \tag{12}$$

Hence the probability is p = 2/3 for random values of the leaf nodes. While this is simple for a level-2 tree, the formula for computing for level-3 and more is a bit more complicated. Knuth and Moore found out they made an error in their original calculation of probabilities while comparing their results against Fuller et al. [3]. After correcting their mistake and making further assumptions on the upper bounds of F1, effectively replacing B_n by A_n, they arrived at recurrence relation of the following form

$$\begin{array}{rcl} A_{d+1} & = & A_d + S_d A_{d-1} \\ S_d & = & \sum_i p_{ij} \\ A_0 & = & B_0 = 1 \\ A_1 & = & 2 \end{array} \tag{13}$$

The solution to the above equations gives an upper bound on the performance of F1 as $\Re = O(b/\sqrt{\log(b)})$. Later they were able to tighten the upper bound to just $\Re = O(b/\log(b))$ by taking advantage of some characteristics of F1. For detailed discussion of proof of the asymptotic behavior of the F1 procedure, the reader is referred to their paper. They also noted that the estimate of $b/log(b)$ is not a good approximation for practical search depths of $d < 30$.

2.4.2 Fuller et al.'s analysis

This work started the average case analysis of alpha-beta by deriving exact but complex formulas for the number of leaf nodes visited on average. The score of a node at any level in a rug tree is independent of the values of the other nodes at the same level. Also all the nodes at any level are iid, because the leaf nodes are iid random variables. From principles in order statistics of the maximum and minimum of random variables we can state the following

$$\begin{array}{rcll} V_0(x) & = & [V_1(x)]^b & \text{at MAX node.} \\ V_1(x) & = & 1 - [1 - V_2(x)]^b & \text{at MIN node.} \end{array} \tag{14}$$

Using the survivor function $\overline{F}(x) = 1 - F(x)$ and generalizing for any depth

$$\begin{array}{rcll} V_k(x) & = & V_{k+1}(x)^b & \text{if k is even.} \\ \overline{V}_k(x) & = & \overline{V}_{k+1}(x)^b & \text{if k is odd.} \end{array} \tag{15}$$

The next step towards determining the number of nodes examined by alpha-beta procedure is to calculate the probability $P(\vec{i}_d)$ of visiting node $\vec{i}_d$. If alpha and beta are specified for **all** nodes in the tree, and not just to those nodes actually examined, then node $\vec{i}_d$ is examined if and only if

$$\alpha(\vec{i}_d) < \beta(\vec{i}_d) \tag{16}$$

Then given cumulative distributions of alpha and beta, that are independent and continuous, $\underset{\sim}{A}_{\vec{i}_j}(x)$ and $\underset{\sim}{B}_{\vec{i}_j}(x)$ respectively, then $P(\alpha(\vec{i}_d) < \beta(\vec{i}_d))$

$$P(\vec{i}_d) = \begin{cases} \int_{-\infty}^{\infty} \overline{\underset{\sim}{B}}_{\vec{i}_d}(z) d\underset{\sim}{A}_{\vec{i}_d}(z), & \text{for } i_j > 1 \text{ and } j \in \textit{even} \\ \int_{-\infty}^{\infty} \overline{\underset{\sim}{A}}_{\vec{i}_d}(z) d\underset{\sim}{B}_{\vec{i}_d}(z), & \text{for } i_j > 1 \text{ and } j \in \textit{odd} \\ 1, & \text{for } i_j = 1, \forall j \end{cases} \tag{17}$$

They were able to solve the above integral equations for smaller depths using a symbolic manipulation program called MACSYMA. However it was not possible to complete the proof of optimality of alpha-beta due to the resulting complexity of the equations. Other researchers followed up on this work to complete the proof.

For large values of d, the analytical calculations are not feasible thus they opted for experimental simulation using their chess program 'Tech' to arrive at an empirical formula for N

$$N \sim K_d * b^{0.72\boldsymbol{d}+0.277} \tag{18}$$

where K_d is a function of the depth of search.

2.4.3 Baudet, Pearl, Tarsi's contributions

Knuth and Moore arrived at O$(b/log(b))$ for the average case of the branch-and-bound F1 procedure, and speculated this behavior will not be altered much for the F2 procedure because deep cutoffs have 2nd order effects. This conjecture is first confirmed by Baudet [2] that used the F2 procedure with deep cutoffs. Baudet also gave equation 19 as the lower bound for the branching factor of alpha-beta, and derived a tight upper bound much better than the one given by Knuth. It is somewhat of a surprise that he was not able to finish the job of calculating the exact value. That was finally done by Pearl [6] who showed

$$\Re = \frac{\xi_b}{1-\xi_b} \sim b^{3/4} \tag{19}$$

where ξ_b is the positive root of $x^b + x - 1 = 0$. This is exactly the same lower bound Baudet derived, hence Pearl proved asymptotic optimality of alpha-beta over all directional algorithms. However a non-directional algorithm can perform better than alpha-beta and indeed the SSS* algorithm is found to consistently search fewer nodes than alpha-beta. This gave hope that it could result in lower branching factor than alpha-beta. However Tarsi [9] showed that considering a bi-valued game tree,*any* algorithm (directional or not) examines at least $(\frac{\xi_b}{1-\xi_b})^d$ nodes. This task is same as evaluating a continuous rug tree, hence alpha-beta is finally proven to be asymptotically optimal among all game searching algorithms.

The derivation of the asymptotic branching factor of alpha-beta (equation 19) is interesting, so we give a glimpse of the major steps here. Equations for number of leaf nodes examined by alpha-beta in a rug tree were first derived by Baudet starting with discrete terminal values and then refining quantization. Let $f_0(x) = x$ be a function mapping [0,1] on to itself ,and, for i=1,2,...,define

$$\begin{aligned}
f_i(x) &= 1-(1-f_{i-1}(x)^d)^d \\
r_i(x) &= \tfrac{1-f_{i-1}(x)^d}{1-f_{i-1}(x)} \\
s_i(x) &= \tfrac{f_i(x)^d}{f_{i-1}(x)^d} \\
R_i(x) &= r_1(x) * \cdots * r_{\lceil i/2 \rceil}(x) \\
S_i(x) &= s_1(x) * \cdots * s_{\lfloor i/2 \rfloor}(x)
\end{aligned} \tag{20}$$

Then the average number of leaf nodes examined by alpha-beta given distributions of alpha and beta (F_A and F_B) is as follows

$$\begin{aligned}
N &= \text{Non-critical nodes } (P(\alpha < \beta)) + \text{Critical nodes (equation 2)} \\
&= \textstyle\int_{-\infty}^{\infty} \left[\sum F_A(x) F_B'(x)\right] dx + \left(b^{\lceil d/2 \rceil} + b^{\lfloor d/2 \rfloor} - 1\right) \\
&= b^{\lfloor d/2 \rfloor} + \textstyle\int_0^1 R_d'(t) S_d(t) dt
\end{aligned} \tag{21}$$

The solution of the above integral is very difficult due to the recursive nature of $f_i(x)$. Pearl [6] introduced the Poincare equation shown below to replace $f_0(x) = x$ by $f_0(x) = \phi(x)$, hence rendering the distribution of every node in minimax identical.

$$\begin{gathered}
\phi(x) = g(\phi(ax)) \\
g(\phi) = 1-(1-\phi^d)^d
\end{gathered} \tag{22}$$

3 Alpha-beta pruning

Now that we have done extensive review of alpha-beta literature, let us summarize the results before we attempt to analysis other pruning methods.

3.1 Minimal alpha-beta

Given uniform tree of width b and height d:

$$\begin{aligned} perft(d,1) &= 1 \\ perft(d,2) &= b^{\lceil d/2 \rceil} - 1 \\ perft(d,3) &= b^{\lfloor d/2 \rfloor} - 1 \\ perft(d) &= b^{\lceil d/2 \rceil} + b^{\lfloor d/2 \rfloor} - 1 \end{aligned} \tag{23}$$

The type-2 and type-3 nodes are equally divided on even plies i.e CUT/ALL = 1 , while CUT nodes dominate on odd plies because the ratio is $\sim b^{\lceil d/2 \rceil}/b^{\lfloor d/2 \rfloor} = b$. This significant difference suggests the location of the terminal nodes affect the ratio more than any other factor. With more pruning added on top of alpha-beta, the ratio is expected to lie somewhere within this limits. In any case, the number of type-2 nodes should always be greater than type-3 nodes. When internal nodes are included, the ratio will be slightly smaller but the same observations should apply.

3.2 Suboptimal alpha-beta

For a sub-optimal move ordering where the best move is not necessarily placed first i.e.k moves are examined on average, then the equations are

$$\begin{aligned} perft(d,1) &= k^d \\ perft(d,2) &= b^{\lceil d/2 \rceil} k^{\lfloor d/2 \rfloor} - k^d \\ perft(d,3) &= b^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil} - k^d \\ perft(d) &= b^{\lceil d/2 \rceil} k^{\lfloor d/2 \rfloor} + b^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil} - k^d \end{aligned} \tag{24}$$

Again the ratio CUT/ALL=1 on even plies but on odd plies CUT/ALL$< b$ depending on the ratio b/k. For chess the branching factor is around 35, and if we assume $k = 2$ moves are examined on average, then we can see that type-2 nodes will still dominate. In general when $b > k$ we can expect to have more CUT nodes. In the worst case when $k = b$ where all moves are searched with open windows the formula converges to CUT=ALL=0 and PV=b^d as expected.

4 Late move reduction heuristics

The definition of $perft(d)$ for reduced depth search is ambiguous without specifying the initial search depth d^* to be specified at the root. For LMR and null move analysis we assume a $perft(ply = d, depth = d^*)$ where $d = d^* - r$ is implied even though it is simply referred as $perft(d)$. The depth will be reduced down the tree and a node is taken to be terminal, i.e. becomes part of $perft(d)$, when the depth becomes 0 as shown in figure 4.

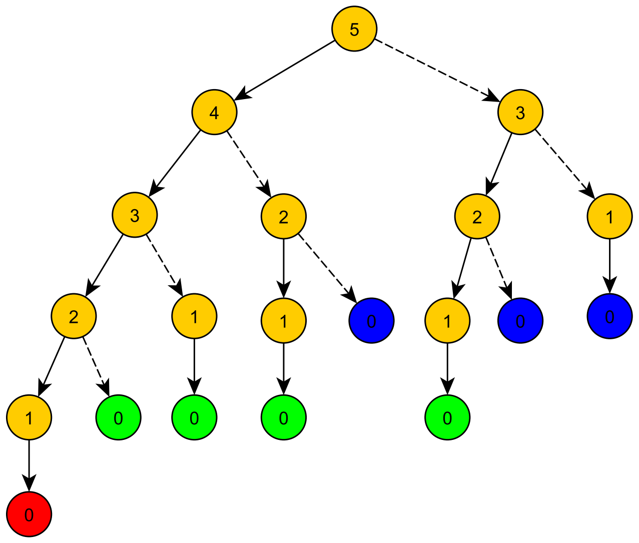

Figure 4: LMR pruned minimax tree with $b_0 = b_1 = 1$

4.1 Case 1

The standard LMR reduces the search depth of b_1 moves out of b by one, and the rest of the moves i.e b_0 are unreduced.

$$b = b_0 + b_1 \tag{25}$$

We assume at least one move is searched with full depth that is $b_0 \geq 1$, which is also the case in practice.

4.1.1 Minimax LMR

The number of leaf nodes at depth d follows a *binomial* distribution.

$$perft(d) = g(r; d, b_1, b) = \binom{d}{r} {b_1}^r (b - b_1)^{d-r} \tag{26}$$

We can rewrite it in terms of probability of reducing a node $p = b_1/b$. Using the probability mass function

$$f(r; d, p) = g(r; d, p, 1) = \binom{d}{r} p^r (1-p)^{d-r} \tag{27}$$

the number of leaf nodes is

$$perft(d) = f(r; d, p) * b^d \tag{28}$$

The binomial term is the difference with minimax tree which has b^d leaf nodes. Hence when no moves are reduced i.e. p = 0, the probability mass function f = 1 due to the property that $0^0 = 1$, hence the number of nodes becomes b^d.

Example: Take the LMR reduced tree in Figure 4 with 2 moves at each node. The dotted edges indicate moves where LMR reduction occurred, i.e. a reduction besides the necessary one ply depth decrement. Since exactly one move is reduced and one move searched, the formula reduces to $perft(d) = \binom{d}{r}$. Thus the number of leaf nodes are $\binom{5}{0} = 1$, $\binom{4}{1} = 4$, and $\binom{3}{2} = 3$ indicated by the red,green and blue nodes respectively.

4.1.1.1 Asymptotic analysis

The total number of nodes N can be calculated as

$$N = \sum_{r=0}^{\lfloor d*/2 \rfloor} \sum_{d=r}^{d^*-r} f(r; d, p) * b^d \tag{29}$$

or in a more suitable format for asymptotic analysis

$$N = \sum_{d=0}^{d^*} \sum_{r=0}^{min(d^*-d,d)} f(r; d, p) * b^d \tag{30}$$

Using the second equation for N

$$\begin{aligned}
N &= \sum_{d=0}^{d^*} \sum_{r=0}^{min(d^*-d,d)} f(r;d,p) * b^d \\
&= \sum_{d=0}^{d^*} F(min(d^*-d,d);d,p) * b^d \\
&= \sum_{d=0}^{\lfloor d^*/2 \rfloor} F(d;d,p) * b^d + \sum_{d=\lfloor d^*/2 \rfloor+1}^{d^*} F(d^*-d;d,p) * b^d \\
&= \sum_{d=0}^{\lfloor d^*/2 \rfloor} b^d + \sum_{d=\lfloor d^*/2 \rfloor+1}^{d^*} F(d^*-d;d,p) * b^d
\end{aligned} \tag{31}$$

The first half of the equation represents the un-pruned part of the tree

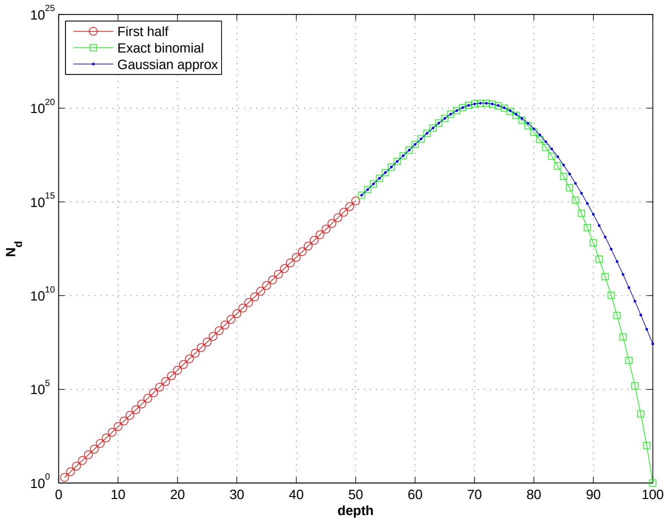

Figure 5: Gaussian approximation of N

and is exactly the same as an un-reduced minimax tree. The second half represents part of the tree which gets pruned by LMR. This part may be approximated by a Gaussian curve for large d as shown in figure 5. However $d^* - d$ starts from $d^*/2$ and goes down to 0, thus the approximation towards the end is not that good. For preliminary analysis, we can assume N to be of a simple form as $N = O(b^{kd})$ as is common for alpha-beta studies. Therefore k is bounded from above by k=1 for p=0, and from below by $k = 0.5$ for p=1. The lower bound is exactly the same as that of alpha-beta's, so we can conclude from this alpha-beta pruning is always better than LMR even when all moves are reduced. Let us assume that $k = 2^{-p}$ so that both limits are exactly satisfied. This model is an exponential fit

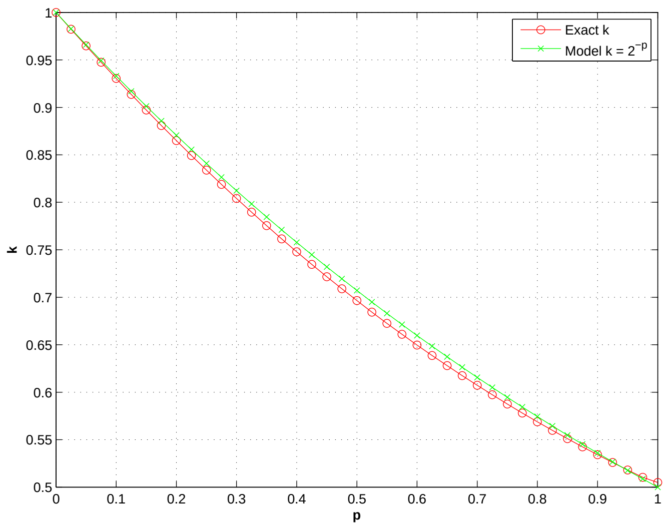

Figure 6: Asymptotic estimate of k for LMR

of form $exp(ax+b)$. For b=2 and d=100, this model is plotted along with the actual value of k in figure 6. We can observe that there is a good fit at least for this particular set of values. Hence the simulation results seem to suggest that $\boldsymbol{O(b^{(2^{-p})d})}$ nodes are visited by LMR, and that the effective branching factor $\Re = \boldsymbol{O(\sqrt[2^p]{b})}$. For higher values of $b > 2$ the fit may not be so good. The simple b^{kd} assumption is not accurate for large b, because k itself depends on b implying the model is not correct. This can be checked by solving equation 31 using a Gaussian approximation. The result so obtained using Mathematica (a symbolic manipulation program) is very complex with terms that couple p, d and b in an intricate way thus making further simplification very difficult.

$$\begin{aligned}
N &= \boldsymbol{\frac{1}{2ln(b)}}\left[a_0\boldsymbol{b^d} + a_1\boldsymbol{b^{\frac{d}{2}}} + a_2\boldsymbol{b^{\frac{d}{2}(1-p)(2+pln(b))}}\right] \\
a_0 &= \text{erfc}\left(\frac{dp}{\sqrt{2dp(1-p)}}\right) \\
a_1 &= \text{erfc}\left(\frac{d/2-dp}{\sqrt{2dp(1-p)}}\right) \\
a_2 &= \text{erfc}\left(\frac{\sqrt{d}(-1+2p+2(p-1)ln(b))}{2\sqrt{2p(1-p)}}\right) - \text{erfc}\left(\frac{\sqrt{dp}(1+(p-1)ln(b))}{\sqrt{2(1-p)}}\right)
\end{aligned} \tag{32}$$

4.1.2 Minimal alpha-beta with LMR

$$\begin{aligned} perft(d,1) &= 1 \\ perft(d,2) &= g(r; \lceil d/2 \rceil, b_1, b) - 1 \\ perft(d,3) &= g(r; \lfloor d/2 \rfloor, b_1, b) - 1 \\ perft(d) &= g(r; \lceil d/2 \rceil, b_1, b) + g(r; \lfloor d/2 \rfloor, b_1, b) - 1 \end{aligned} \quad (33)$$

4.1.3 Suboptimal alpha-beta with LMR

Analysis of a suboptimal alpha-beta with LMR that searches at least k moves at critical nodes is done for two cases.

4.1.3.1 Case 1 The first case is when all the k moves are not reduced i.e. $k \leq b_0$.

$$\begin{aligned} perft(d,1) &= k^d \\ perft(d,2) &= g(r; \lceil d/2 \rceil, b_1, b) k^{\lfloor d/2 \rfloor} - k^d \\ perft(d,3) &= g(r; \lfloor d/2 \rfloor, b_1, b) k^{\lceil d/2 \rceil} - k^d \\ perft(d) &= g(r; \lceil d/2 \rceil, b_1, b) k^{\lfloor d/2 \rfloor} + \\ & \quad g(r; \lfloor d/2 \rfloor, b_1, b) k^{\lceil d/2 \rceil} - k^d \end{aligned} \quad (34)$$

4.1.3.2 Case 2 The second case where $k > b_0$ is more difficult than the previous case because $k - b_0$ moves are reduced and the rest b_0 movers are not. In plies where the b moves have to be examined $b - b_0$ moves are reduced and b_0 are not. The difference in the number of moves reduced requires separate treatment of the two cases. Assuming the reduction r is distributed for the two groups as r_0 and r_1 and all enumerations are considered, then $perft(d)$ can be calculated as follows

Assuming k moves are examined at $\lceil d/2 \rceil$ plies, then the number of ways a reduction of r_0 can be applied is

$$g(r_0; \lceil d/2 \rceil, k - b_0, k) = \binom{\lceil d/2 \rceil}{r_0} (k - b_0)^{r_0} {b_0}^{\lceil d/2 \rceil - r_0} \quad (35)$$

In the rest of the plies $\lfloor d/2 \rfloor$, b moves are examined

$$g(r_1; \lfloor d/2 \rfloor, b - b_0, b) = \binom{\lfloor d/2 \rfloor}{r_1} (b - b_0)^{r_1} {b_0}^{\lfloor d/2 \rfloor - r_1} \quad (36)$$

The number of type-1 nodes can be calculated by noting that k moves are searched at all plies

$$perft(d,1) = g(r; d, k - b_0, k) = \binom{d}{r} (k - b_0)^r {b_0}^{d-r} \quad (37)$$

Therefore the total number of ways a reduction of r can be applied with this arrangement is the product of the above two terms. We also sum for the total number of ways that r can be distributed in two groups

$$\begin{aligned} perft(d)_2 &= \textstyle\sum_{r_0=0}^{r} \left[g(r_0; \lceil d/2 \rceil, k-b_0, k) * g(r_1; \lfloor d/2 \rfloor, b-b_0, b)\right] \\ &= \textstyle\sum_{r_0=0}^{r} \left[\binom{\lceil d/2 \rceil}{r_0}(k-b_0)^{r_0} {b_0}^{\lceil d/2 \rceil - r_0} * \binom{\lfloor d/2 \rfloor}{r_1}(b-b_0)^{r_1} {b_0}^{\lfloor d/2 \rfloor - r_1}\right] \\ &= \textstyle\sum_{r_0=0}^{r} \left[\binom{\lceil d/2 \rceil}{r_0}\binom{\lfloor d/2 \rfloor}{r_1}(k-b_0)^{r_0}(b-b_0)^{r_1} {b_0}^{d-r}\right] \end{aligned} \tag{38}$$

Let us define a function t as follows

$$t(d, d_u, d_l, k, b) = \sum_{r_0=0}^{r} \left[\binom{d_u}{r_0}\binom{d_l}{r_1}(k-b_0)^{r_0}(b-b_0)^{r_1} {b_0}^{d-r}\right] \tag{39}$$

Then $perft(d)_2$ can be written in terms of t by substituting the appropriate values of d_u and d_l. After that the number of type-2 nodes $perft(d, 2)$ is calculated by removing the type-1 nodes. The calculation for the number of type-3 nodes follow the same procedure except that the number of moves at even and odd plies is reversed. In summary.

$$\begin{aligned} perft(d, 1) &= g(r; d, k-b_0, k) \\ perft(d, 2) &= t(d, \lceil d/2 \rceil, \lfloor d/2 \rfloor, k, b) - g(r; d, k-b_0, k) \\ perft(d, 3) &= t(d, \lfloor d/2 \rfloor, \lceil d/2 \rceil, k, b) - g(r; d, k-b_0, k) \\ perft(d) &= t(d, \lceil d/2 \rceil, \lfloor d/2 \rfloor, k, b) + \\ & \quad\; t(d, \lfloor d/2 \rfloor, \lceil d/2 \rceil, k, b) - g(r; d, k-b_0, k) \end{aligned} \tag{40}$$

That is $perft(d)$ for a suboptimal alpha-beta with LMR. For the worst case scenario i.e k=b, the formula should converge to minimax LMR case.

$$\begin{aligned} t(d, d_u, d_l, k, b) &= \textstyle\sum_{r_0=0}^{r} \left[\binom{d_u}{r_0}\binom{d_l}{r_1}(b-b_0)^{r_0}(b-b_0)^{r_1} {b_0}^{d-r}\right] \\ &= \textstyle\sum_{r_0=0}^{r} \left[\binom{d_u}{r_0}\binom{d_l}{r_1}(b-b_0)^{r} {b_0}^{d-r}\right] \\ &= \left[\textstyle\sum_{r_0=0}^{r} \binom{d_u}{r_0}\binom{d_l}{r_1}\right](b-b_0)^{r} {b_0}^{d-r} \\ &= \binom{d}{r}(b-b_0)^{r} {b_0}^{d-r} \end{aligned} \tag{41}$$

The last step of the proof uses Vandermonde's identity. Since t = g for this case $perft(d)$ converges to the minimax LMR case.

4.2 Case 2

If moves are reduced by variable amounts , then the perft counts at different depth follow a *multinomial* distribution. This distribution converges to binomial distribution in case of standard LMR, when the number of groups is two i.e moves are either reduced by one ply or not at all. Let us define b_r and d_r as follows

$$\begin{aligned} b_r &= \text{number of moves reduced by an amount of r out of b moves} \\ d_r &= \text{number of times reduction of r is applied along depth d} \end{aligned} \tag{42}$$

We can then formulate the following relations

$$\begin{aligned} b &= \textstyle\sum_{i=0}^{k} b_i \\ d &= \textstyle\sum_{i=0}^{k} d_i \\ r &= \textstyle\sum_{i=0}^{k} d_i * i \end{aligned} \tag{43}$$

4.2.1 Minmax LMR

The number of nodes at a specific depth d can be calculated as follows

$$\begin{aligned} h(d_0 \cdots d_k; d, b_0 \cdots b_k) &= \tfrac{d!}{d_0! \cdots d_k!} b_0^{d_0} \cdots b_k^{d_k} \\ &= \left[\tfrac{d!}{d_0! \cdots d_k!} p_0^{d_0} \cdots p_k^{d_k}\right] * b^d \end{aligned} \tag{44}$$

The probability mass function f is a multinomial distribution.

$$f(d_0 \cdots d_k; d, p_0 \cdots p_k) = \frac{d!}{d_0! \cdots d_k!} p_0^{d_0} \cdots p_k^{d_k} \tag{45}$$

The total number of leaf nodes can be calculated by summing all combinations of reductions that lead to a total reduction of r.

$$\begin{aligned} perft(d) &= \textstyle\sum_{i=0}^{p(r)} h(d_0 \cdots d_k; d, b_0 \cdots b_k) \\ &= g(d_0 \cdots d_k; d, r, b_0 \cdots b_k) \end{aligned} \tag{46}$$

The function $p(r)$ is the *integer partition* of r. For example, a reduction of 4 can be obtained in 5 ways viz a viz, 4, 3 + 1, 2 + 2, 1 + 1 + 2, 1 + 1 + 1 + 1. All of the them lead to the same tree with a depth of d-4, thus the total number of leaf nodes is obtained by summing all contributions from p(r) possibilities to reach the same reduced depth of d. Determining integer partition is extremely hard and is an active research topic in number theory. The partition function p(r) can be computed using recursive defintion of Leonhard Euler

$$p(r) = \sum_i (-1)^{i-1} p\left(r - \frac{i(3i-1)}{2}\right) \tag{47}$$

An approximate formula that maybe useful for asymptotic studies is given by Ramanujan and Hardy

$$p(r) \sim \frac{1}{4r\sqrt{3}} \exp\left(\pi\sqrt{\frac{2r}{3}}\right) \text{ as } r \to \infty. \tag{48}$$

4.2.2 Minimal alpha-beta with LMR

$$\begin{aligned} perft(d,1) &= 1 \\ perft(d,2) &= g(d_0 \cdots d_k; \lceil d/2 \rceil, r, b_0 \cdots b_k) - 1 \\ perft(d,3) &= g(d_0 \cdots d_k; \lfloor d/2 \rfloor, r, b_0 \cdots b_k) - 1 \\ perft(d) &= g(d_0 \cdots d_k; \lceil d/2 \rceil, r, b_0 \cdots b_k) + \\ & \quad g(y_0 \cdots d_k; \lfloor d/2 \rfloor, r, b_0 \cdots b_k) - 1 \end{aligned} \tag{49}$$

Use of partition function $p(r)$ in the previous case assumes reductions can be applied at any ply along depth d. For best case analysis , reductions can not be applied at about half of depth because exactly one move is analyzed there. Hence to use the same partition function $p(r)$ in equation 46 for this case as well $r < \lceil d/2 \rceil$. Otherwise $p(r)$ should be replaced by an *intermediate integer function* $p(m,r)$ that represent the number of partitions of r using natural numbers $\leq m$. The function becomes same as standard partition function $p(r) = p(d,r)$ for $r < d$.

4.2.3 Suboptimal alpha-beta with LMR

Analysis of a suboptimal alpha-beta with variable LMR follows the same analysis as the suboptimal standard LMR case.

4.2.3.1 Case 1 For $k \leq b_0$

$$\begin{aligned} perft(d,1) &= k^d \\ perft(d,2) &= g(d_0 \cdots d_k; \lceil d/2 \rceil, r, b_0 \cdots b_k) - k^d \\ perft(d,3) &= g(d_0 \cdots d_k; \lfloor d/2 \rfloor, r, b_0 \cdots b_k) - k^d \\ perft(d) &= g(d_0 \cdots d_k; \lceil d/2 \rceil, r, b_0 \cdots b_k) + \\ & \quad g(y_0 \cdots d_k; \lfloor d/2 \rfloor, r, b_0 \cdots b_k) - k^d \end{aligned} \tag{50}$$

4.2.3.2 Case 2 For $k > b_0$

$$\begin{aligned}
perft(d,1) &= g(d_0 \cdots d_j; d, r, b_0 \cdots b_{j-1} b_j^*) \\
perft(d,2) &= t(d_0 \cdots d_j, \lceil d/2 \rceil, \lfloor d/2 \rfloor, b_0 \cdots b_{j-1} b_j^*) - \\
&\quad g(d_0 \cdots d_j; d, r, b_0 \cdots b_{j-1} b_j^*) \\
perft(d,3) &= t(d_0 \cdots d_j, \lfloor d/2 \rfloor, \lceil d/2 \rceil, b_0 \cdots b_{j-1} b_j^*) - \\
&\quad g(d_0 \cdots d_j; d, r, b_0 \cdots b_{j-1} b_j^*) \\
perft(d) &= t(d_0 \cdots d_j, \lceil d/2 \rceil, \lfloor d/2 \rfloor, b_0 \cdots b_{j-1} b_j^*) + \\
&\quad t(d_0 \cdots d_j, \lfloor d/2 \rfloor, \lceil d/2 \rceil, b_0 \cdots b_{j-1} b_j^*) - \\
&\quad g(d_0 \cdots d_j; d, r, b_0 \cdots b_{j-1} b_j^*)
\end{aligned} \tag{51}$$

Here k is also greater than $j-1$ groups of reductions i.e. $k > \sum_{i=0}^{j-1} b_i$, thus $b_j^* = k - \sum_{i=0}^{j-1} b_i$ moves fall inside group j. The derivation of the functions g and t follow similar procedures but the results are not shown here. The function t can be defined as a product of two multinomial functions instead of binomials.

5 Null move pruning heuristics

Null move pruning heuristics works by trying a null move (a pass move) first to see if the opponent can damage us enough to lower our score below beta. If our score is still good enough after the pass move, it means we can stop searching this line because it is very unlikely we won't fail high. It assumes that a pass move is the worst move possible which is mostly true in chess. But there are some positions known as *zugzwang* where a null move is the best move. Aside from that it is a very effective heuristic that reduces the branching factor significantly. After one side makes a null move, the other side is not allowed to make a null move right away.

5.1 $R = 0$ case

First we make a second simplifying assumption of no reduction (R = 0) for the null move. This is of course a simplistic assumption that makes the null move be just like any other move. Later we will consider a realistic reduction of R = 3 or more that is a bit complicated. In figure 7 are shown three cases where null move is applied in different ways. We can see that the number of moves in a given ply is not uniform for all the three cases. Therefore the exact formulas for the number of leaf nodes will be complicated no matter which scheme we use. We will consider the first case where null move is tried everywhere , and the second where we don't try it at PV nodes from here

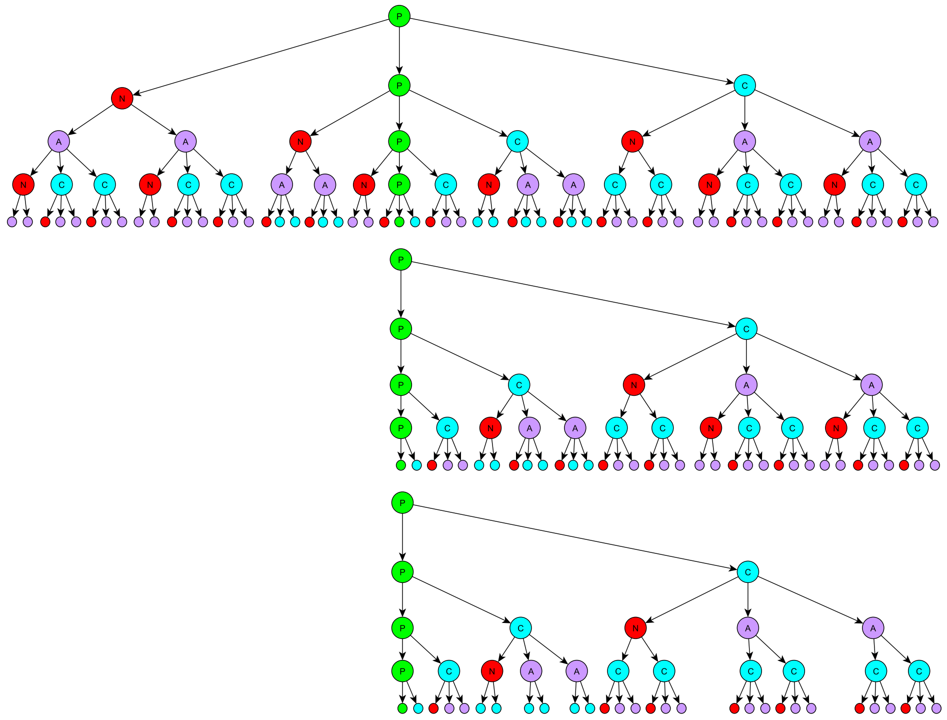

Figure 7: Null move applied at node types a)PV/CUT/ALL - 60 leaves b) CUT/ALL - 35 leaves c) CUT - 29 leaves with R=0.

on. The second case is important when we don't want to have null moves on the principal variation which is especially true for the root move. In both cases double null moves are not allowed.

$$\begin{aligned}
P_{d+1} &= N_d + P_d + (b-1)C_d \\
C_{d+1} &= N_d + bA_d \\
A_{d+1} &= N_d + bC_d \\
N_{d+1} &= bC_d \\
P_0 &= C_0 = A_0 = N_0 = 1
\end{aligned} \tag{52}$$

The addition of the null move to a minimax tree is just a burden since we don't have alpha-beta bounds. To find the exact number of leaf nodes, we form recurrence relations based on node types P=PV, C=CUT, A=ALL, N=NULL. We can drop some of the terms in the equations to solve for the other cases.

5.1.1 Case-1

For the first case where null move is tried everywhere, the number of leaf node is as follows given $k = \sqrt{b(b+4)}$

$$perft(d) = c_1 \left(\frac{b+k}{2}\right)^d + c_2 \left(\frac{b-k}{2}\right)^d \tag{53}$$

where $c_1 = \frac{k+(b+2)}{2k}$ and $c_2 = \frac{k-(b+2)}{2k}$.

We can substitute $b = 2$ and $d = 4$ for the tree in figure 7 to confirm $perft(4) = 60$. The limiting branching factor for this case is found to be $b+1$ for $b >> 1$. This is expected because real moves have a branching factor of $b+1$ and null moves have b.

The minimal tree occurs when null move always succeeds. This is rather unrealistic because we can get a cutoff on the root PV node.

$$perft(d) = b^{\lfloor d/2 \rfloor} \tag{54}$$

5.1.2 Case-2

Solving similarly for the second case where null move is not tried at PV nodes we obtain

$$perft(d) = c_1 \left(\frac{b+k}{2}\right)^d + c_2 \left(\frac{b-k}{2}\right)^d + c_3 \tag{55}$$

where $c_1 = \frac{bk+b^2-k-1}{2bk-k}$,$c_2 = \frac{bk-b^2-k+1}{2bk-k}$ and $c_3 = \frac{k}{2bk-k}$.

Substituting values of $b = 2$ and $d = 4$ we find that $perft(4) = 35$ for the second case in figure 7.

The minimal tree occurs when null move always succeeds. The result is similar to the minimal tree of standard alpha-beta which is better than the previous case where we allowed null move in PV nodes too.

$$perft(d) = b^{\lceil d/2 \rceil} + b^{\lfloor d/2 \rfloor} - 1 \tag{56}$$

5.2 $R > 0$ case

We can try to solve this case by forming recurrence equations as before, but soon we realize the equations get very complex. The case of $R = 1$ requires finding roots of cubic polynomial, and in general the characteristic equation is a polynomial of degree $R + 2$. For larger reductions we can conclude the resulting formula to be practically useless. Therefore we will tackle this problem with a different approach.

5.2.1 Case-1

First we note that a reduction of $R >= 3$ makes the cost of testing null move very small. Nowadays many chess programs use $R = 3$ applied recursively which makes the sub-tree size much smaller than other moves. Some programs also use a depth dependent reduction that can be very aggressive as to make the cost of null move close to 0. Therefore we will assume the cost of null move to be 0 and also that it is tried everywhere. With this scheme the branching factor is b everywhere. If the null move succeeds with a probability p, then on average $b_{eff} = (1-p)b$ moves are searched. We can then substitute b_{eff} instead of b for the case of alpha-beta pruning (section 3), to get formulas for determining the average number of leaf nodes.

5.2.2 Case-2

It is worthwhile to consider the second case where null move is not tried at PV nodes to avoid problems with returning a null move at the root. Relatively easy recurrence relation can be formed as follows

$$\begin{aligned} P_{d+1} &= P_d + (b-1)C_d \\ C_{d+1} &= b_{eff}A_d \\ A_{d+1} &= b_{eff}C_d \\ P_0 &= C_0 = A_0 = 1 \end{aligned} \tag{57}$$

The solution to the above set of equations gives

$$perft(d) = \frac{\left[b_{eff}^d(b-1) + b_{eff} - b\right]}{b_{eff} - 1} \tag{58}$$

When the cost of null-move is significant for some reason then b_{eff} can be modified to $b_{eff} = (\frac{1-p}{1-q})b$. Here q is the cost of null move relative to the total cost of searching the parent node.

6 Conclusions

References

[1] D. Abdi. Monte carlo methods for estimating game tree size. April 2013.

[2] G. Baudet. On the branching factor of the alpha-beta pruning algorithm. Artificial intelligence, 10, 1978.

[3] S. Fuller, J. Gaschnig, and Gillogly. Analysis of the alpha-beta pruning algorithm. Technical report, Carnegie-Mellon Univesity, 6, 1973.

[4] D. Knuth and R. Moore. An analysis of alpha-beta pruning. Artificial intelligence, 6, 1975.

[5] D. Nau. An investigation of the cause of pathology in game trees. Artificial intelligence, 170, 1982.

[6] J. Pearl. Asymptotic properties of minimax trees in game-searching procedures. Artificial intelligence, 14, 1980.

[7] J. Pearl. The solution for the branching factor of the alpha-beta pruning algorithm and its optimality. Communications of ACM, 25, 1982.

[8] J. Slagel and J. Dixon. Experiments with some programs that search game trees. Journal of the association for computing machinery, 14, 1969.

[9] M. Tarsi. Optimal search on some game trees. Journal of the ACM, 30, 1983.