# Minimum Proof Graphs and Fastest-Cut-First Search Heuristics

**Timothy Furtak and Michael Buro**
Department of Computing Science, University of Alberta, Edmonton, Canada.
email: {furtak,mburo}@cs.ualberta.ca

## Abstract

Alpha-Beta is the most common game tree search algorithm, due to its high-performance and straightforward implementation. In practice one must find the best trade-off between heuristic evaluation time and bringing the subset of nodes explored closer to a minimum proof graph. In this paper we present a series of structural properties of minimum proof graphs that help us to prove that finding such graphs is NP-hard for arbitrary DAG inputs, but can be done in linear time for trees. We then introduce the class of fastest-cut-first search heuristics that aim to approximate minimum proof graphs by sorting moves based on approximations of sub-DAG values *and* sizes. To explore how various aspects of the game tree (such as branching factor and distribution of move values) affect the performance of Alpha-Beta we introduce the class of "Prefix Value Game Trees" that allows us to label interior nodes with true minimax values on the fly without search. Using these trees we show that by explicitly attempting to approximate a minimum game tree we are able to achieve performance gains over Alpha-Beta with common extensions.

## 1 Introduction

Alpha-Beta [Knuth and Moore, 1975] search is the classic heuristic search algorithm for two-player games with perfect information. In past decades AI researchers have found numerous enhancements that, for instance, enable today's chess programs to defeat human World-champions at chess running on ordinary computers. Alpha-Beta improvements usually aim at decreasing the search effort given a fixed task such as searching up to a certain depth, or shaping the search tree to reduce heuristic evaluation errors. In this paper we focus on the *Minimum Proof Graph* problem that asks, given a DAG $G$ corresponding to the states in a 2-player zero-sum game, a player labeling for each state denoting which player is to act, and a score function for each terminal state, what is the minimum number of vertices needed in a subgraph $H \subseteq G$ in order to prove one of the following properties:

(i) Can the first player achieve a score of at least $t$, for some integer $t$? This assumes that the first player only seeks to achieve a score $\geq t$, and the second player tries to prevent this.

(ii) What is the minimax value of the game? This assumes both players seek to maximize their numerical score.

[Knuth and Moore, 1975] answered question (ii) above in the context of homogeneous trees with constant branching factor $b$ and depth $d$: searching $b^{\lceil d/2 \rceil} + b^{\lfloor d/2 \rfloor} - 1$ leaves is necessary and sufficient for establishing the minimax value of such trees. But as [Plaat *et al.*, 1996a] note, trees are rarely homogeneous in practice, and in fact, inputs usually are not even trees, but DAGs. For the purpose of judging how close programs for popular games — such as chess, checkers, and Othello — come to searching minimum proof graphs, they introduce the concepts of Left-First-, Real-, and Approximate-Real-Minimal-Graphs, and estimate upper bounds on minimal proof graphs from game data.

In this paper we approach the problem more formally by first proving some fundamental properties of proof graphs. We then proceed to show that computing minimum proof graphs for DAGs is NP-hard and minimum proof trees for trees can be constructed in linear time. In the second half of the paper we will describe fastest-cut-first search heuristics which we evaluate using a novel synthetic game tree model.

## 2 Proof Graph Definitions

We shall assume throughout that $G = (V, E)$ is a DAG with vertex set $V = \{v_0, v_1, \ldots, v_{n-1}\}$ corresponding to states in the original game, and directed edge set $E$ corresponding to legal moves from each state. Let $p$ define the player function which maps vertices in $V$ to {*white,black*}, and let $f$ be the score function which maps terminal vertices to an integer.

Corresponding to the two properties being proven, we define two classes of decision problems. The first class is a tuple $\langle G, p, f, s, t \rangle$ which asks whether a proof graph exists using at most $s$ nodes to prove that the first player can achieve a score of at least $t$. The second class is a tuple $\langle G, p, f, s \rangle$ which asks whether a proof graph exists using at most $s$ nodes to prove the minimax value of the game. To clarify which type of proof graph we are referring to, we shall use the terms *target proof graph* and *minimax proof graph* respectively.

In both cases we require that accepted instances of the decision problems satisfy the condition that only one vertex of $G$, which we shall call the root, has in-degree 0. Without

loss of generality we shall assume that $v_0$ is the root and that $p(v_0) = white$, such that $white$ is the first player, and that the player to move alternates.

For convenience we introduce the indicator function

$$I(v_i, v_j) = \begin{cases} +1 & : \text{if } p(v_i) = p(v_j), \\ -1 & : \text{otherwise.} \end{cases}$$

We also denote the children of a vertex using $c(v_i) = \{v_j : (v_i, v_j) \in E\}$.

Note that the minimax value may be computed recursively in time linear in the size of $G$. As such, we extend $f$ to be defined over all vertices, rather than only the leaves, where $f$ is now the negamax value of a vertex. Due to the zero-sum nature of the problem we may simplify our analysis by using a negamax perspective (where score is given w.r.t. the player to move) rather than minimax. Specifically, if $c(v_i) \neq \emptyset$ then $f(v_i) = \max_{v_j \in c(v_i)} I(v_i, v_j) \cdot f(v_j)$. For brevity we will use $f_u(v) := I(u, v) \cdot f(v)$ to refer to the value of a vertex $v$ with respect to some other vertex $u$, usually $v$'s parent.

### 2.1 Target Proof Graphs

A subgraph $H \subseteq G$ is a valid target proof graph if and only if all the following hold:

- $v_0 \in H$
- $\big(v_i \in H \text{ and } p(v_i) = p(v_0) \text{ and } c(v_i) \neq \emptyset\big) \Rightarrow$ $\exists v_j \in c(v_i)$ such that $v_j \in H$
- $\big(v_i \in H \text{ and } p(v_i) \neq p(v_0)\big) \Rightarrow c(v_i) \subset H$

For a given target proof graph $H$, let

$$f^{(H)}(v_i) = \begin{cases} f(v_i) & : \text{if } v_i \text{ is a leaf,} \\ \max\limits_{v_j \in c(v_i) \cap H} -f^{(H)}(v_j) & : \text{otherwise.} \end{cases}$$

This function corresponds to the proven value of each vertex.

**Definition 1. MinProofGraph-1** *is a set of succinct encodings of $\langle G, p, f, s, t\rangle$ where there exists a subgraph $H \subseteq G$ corresponding to a valid target proof graph such that $f^{(H)}(v_0) \geq t$, and such that $|H| \leq s$.*

### 2.2 Minimax Proof Graphs

The evaluation of a minimax proof graph is slightly more involved than for a target proof graph as it depends on the concept of Alpha-Beta pruning. Specifically, the order in which a vertex's children are evaluated can affect the size of their proof graphs. This is due to more effective search cutoffs as the range of potential values for a node (and thus its ancestors) is reduced. It should be noted that verifying a minimax proof graph can be done in polynomial time even without explicitly giving the order in which to evaluate a node's children. This follows from results which will be presented in Sec. 3, specifically that an optimal traversal will only examine vertices using one of three possible $\alpha$-$\beta$ pairs.

For concreteness we will now define an indicator function to specify whether, for a vertex in a given minimax proof graph, the proof is accepted. First, for a vertex sequence $\sigma$ of children, let $\alpha_j$ be the highest possible alpha bound resulting from examining the first $j$ nodes in $\sigma$, given the (potential) proof graph $H \subseteq G$. Now let $\mu^{(H)}_{(\alpha,\beta)}(v_i) = \texttt{true}$ iff:

(i) $v_i \in H$ is a leaf, or

(ii) $\exists \sigma$, a permutation of $c(v_i)$, such that $\forall_{1 \leq j \leq |c(v_i)|}$ either:

(a) $\sigma_j \in H, \quad \mu^{(H)}_{(-\beta,-\alpha_j)}(\sigma_j) = \texttt{true}$ or

(b) $\exists k < j$ such that $\beta \leq \alpha_k$.

These conditions correspond to having an ordering to explore a node's children such that the $\alpha$-$\beta$ bounds at any point are sufficient to prove the next child's minimax value using the vertices in $H$, or for the bounds to short-circuit further child evaluations by causing a cutoff, which occurs when beta $\leq$ alpha indicating that the values of a parent's remaining children will provably not affect the game's minimax value. A subgraph $H \subseteq G$ is a valid minimax proof graph if and only if $v_0 \in H$ and $\mu^{(H)}_{(-\infty,\infty)}(v_0) = \texttt{true}$.

**Definition 2. MinProofGraph-2** *is a set of succinct encodings of $\langle G, p, f, s\rangle$ where there exists a subgraph $H \subseteq G$ corresponding to a valid minimax proof graph such that $|H| \leq s$.*

## 3 Minimum Proof Graph Properties

We will now proceed to make some observations which will simplify the computation of a minimax proof graph, assuming that we know the true values of each vertex.

**Theorem 3.1.** *For an Alpha-Beta examination of a vertex $v$ with $\alpha < \beta \leq f(v)$ or $f(v) \leq \alpha' < \beta'$, the size of the minimum proof tree rooted at $v$ is independent of $\alpha$ and $\beta'$.*

*Proof.* We proceed by induction on the height of $v$, the maximum path length starting from $v$. Clearly the observation holds for a height of 0, where $v$ is a leaf and has no children. Now assume the observation holds for all heights less than or equal to some $n$, and that $v$ has height $n+1$ (clearly all of $v$'s children have height $\leq n$).

First consider the case where $\alpha < \beta \leq f(v)$. The examination of $v$ must end with a cutoff where a child is proven to have a value of at least $\beta$. Call such a child *cutting*. Note that examining a child with value $< \beta$ (so as to improve the alpha bound) cannot reduce the effort required to prove a cutting child $w$. To see this observe that $\alpha < \beta \leq f_v(w) \leq f(v)$ and so $w$ will be examined with $f(w) \leq -\beta < -\tilde{\alpha}$, where $\tilde{\alpha}$ is some (potentially) improved alpha bound.

By the induction hypothesis, in neither case does the minimum proof graph size for $w$ depend on $\alpha$. As such, the minimum proof graph for $v$ will consist of examining only one cutting node, and therefore also does not depend on $\alpha$.

Now consider the case where $f(v) \leq \alpha < \beta$. Since no child can have a value greater than $f(v)$, the alpha bound can never be improved, and all children must be examined. The argument that the induction property holds for $n+1$ is the same as in the first case, except now $\tilde{\alpha} = \alpha$.

Thus, by induction, the observation holds for all $n$.

□

**Theorem 3.2.** *For an Alpha-Beta examination of a vertex $v$ the following properties hold:*

*(i)* $\alpha \leq f(v) < \beta \Rightarrow$ *the MPG size does not depend on $\beta$.*

*(ii)* $\alpha < f(v) \leq \beta \Rightarrow$ *the MPG size does not depend on $\alpha$.*

*(iii)* $\alpha \leq f(v) \leq \beta, \alpha < \beta \Rightarrow \exists\, w \in c(v)$ *to examine first which minimizes the proof graph size and* $f_v(w) = f(v)$.

*Proof.* We again proceed by induction on the height of $v$, for which the listed properties clearly hold for a height of 0 i.e., $v$ being a leaf. Now assume the properties holds for all heights less than or equal to some $n$, and that $v$ has height $n+1$.

*Property (i)*: Since $f(v) < \beta$ all children of $v$ must be examined. Note that $\alpha$ can never become larger than $f(v)$ so the initial constraint that $\alpha \leq f(v) < \beta$ holds at each step. For any child $w \in c(v)$ we have that $f_v(w) \leq f(v)$. Now, either $f_v(w) \leq \alpha < \beta$ or $\alpha < f_v(w) < \beta$. In the first case the MPG size is independent of $\beta$ by Theorem 3.1. In the second case the induction hypothesis gives the desired result, using either (i) or (ii) depending on if $p(v) = p(w)$.

*Property (ii)*: If we can show that the MPG size to examine any child $w$ with $f_v(w) = f(v)$ does not depend on the initial value of $\alpha$ (as long as $\alpha < f(v)$) then we are done, since all other nodes can be examined afterwards, and the tighter bound can only decrease the size of those nodes' MPG. Let $w \in c(v)$ be such that $f_v(w) = f(v)$ and $\alpha < f(v)$, which implies $\alpha < f_v(w) \leq \beta$. By the induction hypothesis (part (i) or (ii), depending on ownership) the MPG size of $w$ does not depend on $\alpha$.

*Property (iii)*: The preconditions of property (iii) match those of (i) or (ii) (or both). In the case of (i), as noted in its proof, the child ordering does not affect the MPG size, so simply take $w$ to have $f_v(w) = f(v)$, as at least one such $w$ exists. In the case of (ii) it is never suboptimal to examine such a $w$ first, and as at least one child must minimize the proof graph, take $w$ to be that one. Hence (iii) holds. Thus, by induction, the properties hold for all $n$.

□

Based on Theorem 3.2 we may assume that the initial path from the root to a leaf follows an optimal line of play, the *principle variation* (PV), such that all nodes $w$ along the path have value $f_{v_0}(w) = f(v_0)$. Clearly this initial path will be explored using bounds $\langle \alpha, \beta \rangle = \langle \infty, -\infty \rangle$. There may be many optimal lines of play, but for clarity we shall assume that the PV refers to some fixed initial path in an MPG.

**Theorem 3.3.** *For an Alpha-Beta examination of a DAG $G$ that minimizes the proof graph size, it will be the case that for each $v \in G$ not on the PV, with initial bounds $\alpha$ and $\beta$:*

$$\begin{array}{lllll}
\textit{(i)} & \langle \alpha, \beta \rangle \in \{\langle \pm f(v_0), \infty \rangle, \langle -\infty, \pm f(v_0) \rangle\} & & & \\
\textit{(ii)} & \langle \alpha, \beta \rangle = \langle -\infty, \ f(v_0) \rangle & \Rightarrow & f(v) \geq & f(v_0) \\
\textit{(iii)} & \langle \alpha, \beta \rangle = \langle -\infty, -f(v_0) \rangle & \Rightarrow & f(v) \geq & -f(v_0) \\
\textit{(iv)} & \langle \alpha, \beta \rangle = \langle \ f(v_0), \ \infty \rangle & \Rightarrow & f(v) \leq & f(v_0) \\
\textit{(v)} & \langle \alpha, \beta \rangle = \langle -f(v_0), \ \infty \rangle & \Rightarrow & f(v) \leq & -f(v_0)
\end{array}$$

The proof is omitted for space, but is a straight-forward case-based analysis which proceeds by induction on the maximum distance, $n$, from the principle variation that a vertex may be.

## 4 NP-Completeness Results

In this section we show that determining the inclusion of an element in MinSetCover, which is NP-complete, can be reduced to determining the inclusion of an element in MinProofGraph-1 or to determining the inclusion of an element in MinProofGraph-2.

Sahni has shown that computing a minimum proof for an AND/OR tree (which is equivalent to MinProofGraph-1) is NP-Complete[Sahni, 1974]. To help illustrate the source of the complexity we present a slightly different reduction.

To see that MinProofGraph-1 is in NP, observe that a valid proof graph $H \subseteq G$ is a sufficient certificate. $H$ is clearly polynomial in the size of the original input, and all constraints listed in Section 2 corresponding to $H$ being valid are easily checked in polynomial time along with computing $f^{(H)}(v_0)$. By a similar argument MinProofGraph-2 is in NP.

In preparation for proving that all problems in NP can be polynomial-time reduced to MinProofGraph-1 and MinProofGraph-2 we introduce the following definition for the Minimum Set Cover problem:

**Definition 3. MinSetCover** *is a set of succinct encodings of $\langle X, S, k \rangle$ where $X$ is a finite set, and $S$ is a collection of subsets of $X$ such that there exists $S' \subseteq S$ where $|S'| \leq k$ and $X = \bigcup_{T \in S'} T$.*

**Theorem 4.1.** *MinSetCover $\leq_p$ MinProofGraph-1 and MinSetCover $\leq_p$ MinProofGraph-2.*

**Corollary 4.2.** *MinProofGraph-1 and MinProofGraph-2 are NP-complete.*

*Proof.* We describe a function $\mathcal{F}$ that maps arbitrary code words $w$ into the encoding of an associated MinProofGraph-1 instance with the following properties:

(i) If $w$ encodes a MinSetCover tuple $\langle X, S, k \rangle$, then $\langle X, S, k \rangle \in$ MinSetCover if and only if $\mathcal{F}(w) \in$ MinProofGraph-1.

(ii) Otherwise, $\mathcal{F}(w)$ encodes a fixed tuple not in MinProofGraph-1.

(iii) There exists a Turing machine **F** and a polynomial $p$ such that for all $w$, **F** started on input $w$ computes $\mathcal{F}(w)$ in at most $p(|w|)$ steps.

Given an element of Minimum Set Cover $\langle X, S, k \rangle$ consisting of $X = \{x_0, \ldots, x_{n-1}\}$ and sets $S = \{S_0, \ldots, S_{m-1}\}$, let $\mathcal{F}(\langle X, S, k \rangle) = \langle G, p, f, s, t \rangle$. $\mathcal{F}$ constructs a graph with four levels corresponding to the root node $t$, a dummy node $u$, the elements of $X$ ($v_0, \ldots, v_{n-1}$), and the elements of $S$ ($w_0, \ldots, w_{m-1}$), as in Figure 1. Let the edge set be $\{(t,u), (u,v_i) : 0 \leq i < n\} \cup \{(v_i, w_j) : x_i \in S_j\}$. Let $p(t) = p(v_i) = white$ and $p(u) = p(w_i) = black$, with $white$ nodes having value 1 and $black$ nodes having value -1. Finally, let $t = 1$ and $s = 2 + n + k$.

We now show that $\langle G, p, f, s, t \rangle$ is accepted if and only if $\langle X, S, k \rangle$ is accepted. Any solution to the MinProofGraph-1 instance will include the first three levels and some fourth level nodes. By definition the minimum number of fourth level nodes will be included, and as each third level $v_i$ node has at least one associated fourth level $w_j$ node, the fourth level nodes chosen correspond to a subset of $S$ which forms a minimum set cover of $X$.

The proof for MinProofGraph-2 being NP-complete is almost identical, except that two new vertices are required as

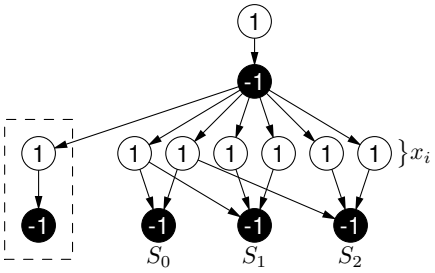


Figure 1: Sample construction of a DAG corresponding to an instance of Minimum Set Cover, with vertices labeled with their minimax value. Vertices $x_i$ correspond to elements in the MinSetCover universe and $S_i$ to the potential covering sets. Vertices in the dashed box are used when reducing to MinProofGraph-2.

shown in Figure 1. In the minimax case at least one third level node must have all its children examined, but after that, due to the given node values, all other third level nodes need only examine any one of their children to produce a cut. As the new vertices must necessarily be included in any proof subgraph, we can assume that they are examined before the other vertices on their levels.

□

## 5 Minimum Proof Graphs for Trees

Intuitively, the computational complexity of finding minimum proof graphs in the presence of nodes with in-degree $\geq 2$ arises from deciding whether to choose the proof graph rooted at a "larger" child over the proof graph rooted at a smaller child, such that the proof for the larger child may be reused in a different portion of the graph, resulting in a smaller overall proof graph. As we will see in this section, for trees the minimum proof graph is itself a tree which can be computed bottom-up in linear time. We distinguish between computing target and minimax proof trees.

### 5.1 Computing Target Proof Trees

For a MinProofGraph-1 instance $\langle G, p, f, s, t\rangle$ we may restrict the vertices of $G$ that need to be considered by removing all $white$ vertices $v_i$ such that $f(v_i) < t$ and then removing components disconnected from the root as a preprocessing step in $O(|V| + |E|)$ time. If $f(v_0) < t$ then the instance is rejected.

To compute the minimum target proof graph size one need only recursively determine this value for the subgraph rooted at each vertex. For first player nodes this is the minimum over the sizes of the child proof graphs plus 1. For second player nodes this is the sum of the sizes of the child proof graphs plus 1. Extracting an associated minimum target proof graph is clearly trivial.

### 5.2 Computing Minimax Proof Trees

The Alpha-Beta algorithm has been well studied in the case of trees [Knuth and Moore, 1975], and it has been stated that the best-case move ordering is to examine children in order of decreasing score. This does not, however, necessarily minimize the total proof tree size in the case of multiple best moves, where "best" refers to either absolute numerical score or ability to generate a cut. Using Theorem 3.3 we can see that there are at most three possible incoming bounds with which a given vertex can be explored in a minimal proof tree (MPT). We may thus compute, bottom-up, a table containing the MPT size for each vertex given the incoming $\alpha$-$\beta$ bounds. This process therefore runs in linear time and space in the graph size. Moreover, the table entries need only be computed as needed, and the set of potential best moves is easily constrained using the observations from Theorem 3.2's proof. The algorithm is listed in Figure 2, and the size of the minimum minimax proof tree is the result of `MPT_size`$(v_0, -\infty, \infty)$. The runtime and space requirement is linear in the input graph size and adjusting the algorithm to actually contruct an MPT is straight-forward.

**Function** `MPT_size`$(v, \alpha, \beta)$

**Data**: subtree root vertex $v \in V(G)$, and $\alpha$-$\beta$ bounds.
**Result**: Min. proof tree size for subtree rooted at $v$, given $\alpha, \beta$.
**begin**
  **if** $\alpha \geq \beta$ **then return** 0
  **if** $c(v) = \emptyset$ **then return** 1 `// leaf node`
  **alias** $T_w[a, b] :=$ `MPT_size`$(w, -b, -a)$
  *Assume* `MPT_size` *values are memoized across calls.*
  **if** $f(v) \leq \alpha$ **then return** $1 + \sum_{w \in c(v)} T_w[\alpha, \beta]$
  **if** $f(v) \geq \beta$ **then return** $1 + \min_{\substack{w \in c(v) \\ f_v(w) \geq \beta}} T_w[\alpha, \beta]$
  $t \leftarrow \sum_{x \in c(v)} T_x[f(v), \beta]$ `// compute total`
  **return** $1 + \min_{\substack{w \in c(v) \\ f_v(w) = f(v)}} \Big(T_w[\alpha, \beta] + t - T_w[f(v), \beta]\Big)$
**end**

Figure 2: Algorithm for computing the minimum minimax proof tree size for a subtree of $G$ rooted at $v$ and bounds. Minimax value function $f$ and tree $G$ are assumed to be global.

## 6 Fastest-Cut-First Heuristics

In this section we develop a move sorting heuristic for Alpha-Beta search based on the results on minimum proof graphs we have presented. In real-world game applications, usually none of the quantities used in `MPT_size` are known at the time when the decision about the next move to be searched is to be made. One therefore has to resort to move ordering heuristics. In addition, search is often performed in DAGs, which complicates the computation of minimum proof graphs. Even so, if estimates for values *and* sizes are available, we could base move decisions on those, and hope to approximate minimum proof graphs sufficiently well. This idea is not new. Plaat et al. [1996], for instance, suggest to use enhanced transposition table look-ups and bias moves towards those with small subtrees, to exploit properties of inhomogeneous DAGs, such as varying out-degrees and leaf depths. Closer in terms of using both value and size estimates comes what is known since the mid-1990s as

"fastest-first" search in the Othello programming community. The idea is to search moves first that have a considerable chance to produce a cutoff, and among those to prefer moves with small sub-DAGs. Employing such heuristics has improved Othello endgame solvers considerably (`http://radagast.se/othello`).

Because both move values and DAG sizes are taken into account we prefer to call heuristics of this type "fastest-cut-first" heuristics. Implementations we have used successfully in the past include sorting moves according to $(V_i - C \cdot S_i / \max_j S_j)$, where $V_i$ and $S_i$ are the estimated move values and sub-DAG sizes, and $C > 0$ is a constant. Alternatively, moves can be restricted to those with $V_i \geq \beta - C$ and from this set a move with minimal size estimate is selected.

Here, we propose a new fastest-cut-first heuristic which is motivated by the following theorem:

**Theorem 6.1.** *Consider a naive version of Alpha-Beta search that does not adjust $\alpha$ based on returned values and searches a game tree starting at cut-node $v$ with successors $v_1, \ldots, v_n$. Let $S_i$ be the expected size of the subtree rooted at $v_i$, and $P_i$ the probability of move $i$ leading to a $\beta$-cut. Then, visiting $v_i$ in ascending order of $S_i/P_i$ minimizes the expected number of visited nodes starting at $v$.*

*Proof.* For move ordering $(1, \ldots, n)$ the expected number of visited nodes is

$$1 + \sum_{i=1}^{n} \Big[ S_i \prod_{j=1}^{i-1} (1 - P_j) \Big] =$$

$$A + (S_k \cdot \pi) + (S_{k+1} \cdot \pi(1 - P_k)) + C,$$

for $\pi := \prod_{j=1}^{k-1} (1 - P_j)$ and suitable $A > 0$ and $C \geq 0$.

Switching move $k$ and $k+1$ leads to expected search effort

$$A + (S_{k+1} \cdot \pi) + (S_k \cdot \pi(1 - P_{k+1})) + C.$$

Thus, if $\pi > 0$ searching move $k+1$ before $k$ while keeping the other moves' ordering constant is not detrimental iff

$$\begin{aligned} & S_{k+1} + S_k(1 - P_{k+1}) \leq S_k + S_{k+1}(1 - P_k) \\ \Leftrightarrow \quad & \frac{S_{k+1}}{P_{k+1}} \leq \frac{S_k}{P_k}, \end{aligned} \tag{1}$$

where we define $c/0 := \infty$ for $c > 0$ to cover the cases $P_k = 0$ and/or $P_{k+1} = 0$. If $\pi = 0$, switching moves $k$ and $k+1$ does not change the expected search effort. This shows that when starting with an arbitrary move ordering we can continue switching moves $k$ and $k+1$ for which (1) holds and not increase the expected search effort. Therefore, the expected number of searched nodes is minimized when visiting nodes $v_i$ in ascending order of $S_i/P_i$. □

Picking the move with the lowest $S_i/P_i$ ratio first accomplishes the goal of producing a cut with high probability, while keeping the node count low, in a principled way, unlike the ad-hoc fastest-cut-first heuristics presented earlier. The expected number of searched nodes established in the theorem is an upper bound on the actual number of nodes the true Alpha-Beta algorithm would visit, because with rising $\alpha$ values, the expected tree sizes $S_i$ do not increase. So, by minimizing this number we can hope to decrease the Alpha-Beta search effort — also when searching DAGs. We will present experimental results on the performance of the novel fastest-cut-first heuristic in Section 8.

## 7 Prefix Value Game Trees

In this section we present a synthetic game tree model that we will use to gauge the performance of the fastest-first heuristic we proposed in the previous section.

Search algorithm improvements are often driven by specific applications. A classic example is the development of forward pruning and parallelization in computer chess. When working in particular search domains, state space properties — such as the average branching factor, the density of terminal nodes, state value distributions and correlations — are fixed. To gain insight into how such properties generally affect search algorithms, several synthetic game tree models have been proposed in the literature, e.g., [Fuller *et al.*, 1973; Newborn, 1977; Nau, 1982; Pearl, 1983; Kaindl and Scheucher, 1992; Scheucher and Kaindl, 1998], with emphasis on studying pathology in alpha-beta search.

In [Fuller *et al.*, 1973] Fuller et al. describe a game tree model in which node value correlations are introduced by assigning values from $\{1, \ldots, N\}$ to edges and assigning a value $p(v)$ to leaves by summing up edge values along the unique path from the root node. One drawback of this so-called incremental game tree model is that values increase with depth and therefore don't have the usual interpretation as representing the expected payoff. In [Scheucher and Kaindl, 1998] this bias is offset first by considering move values from $\{-N, \ldots, N\}$, and then in a second step by realizing that playing a move can never increase a position's value, i.e.,

$$p(v_i) = -p(v) - d_i(v), \tag{2}$$

where $p(x)$ denotes the accumulated value along the unique

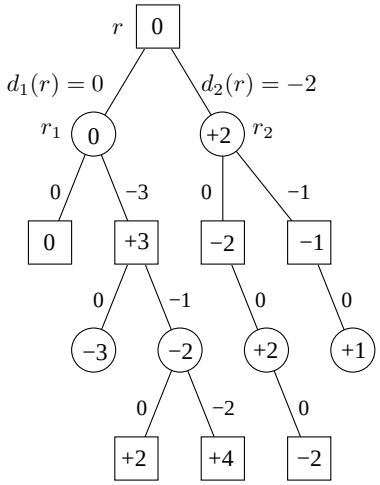


Figure 3: An incremental game tree. Node values indicate the current path value $p(v)$ in the view of the player to move. Edge values represent the damage $d_i(v)$ self-inflicted by playing potentially non-optimal moves.

path when reaching node $x$ in view of the player to move, $v_i$ is the $i$-th successor of $v$, and $-D \leq d_i(v) \leq 0$ is the bounded damage self-inflicted by playing move $i$ in $v$. Figure 3 shows an example. To evaluate search performance it is useful to know exact values of intermediate states. However, in the above incremental model — when $d_i$ is chosen randomly from $[-D, 0]$ — searching entire subtrees is required to compute the true negamax value of interior nodes, which can be very costly in terms of space and time requirements. With a simple yet effective refinement of the incremental tree model, we are able to determine exact node values *on the fly* while generating trees. Being able to leak true state evaluations in interior nodes at no additional cost greatly helps analyzing search algorithms in general settings because no time has to be spent on engineering domain dependent heuristics. For instance, noise can be added when testing heuristics that depend on heuristic state evaluations of interior nodes, such as ProbCut [Buro, 1995], e.g., $\hat{f}_C(v) = f(v) + C \cdot e$, where $f(v)$ denotes the true negamax value of $v$ and $e$ is $\mathcal{N}(0, 1)$ normally distributed. In addition, move preferences can be leaked probabilistically by means of the parameterized softmax function

$$\text{Prob(move } k \text{ is best in state } v) = \frac{\exp(C \cdot f(v_k))}{\sum_i \exp(C \cdot f(v_i))},$$

which spans the range from uniform random ($C = 0$) to perfectly informed choices ($C \to \infty$) and can be used to design synthetic move sorting routines for alpha-beta search or playout policies for UCT [Kocsis and Szepesvari, 2006].

Before we formulate the observation our model refinement is based on, we define the *prefix negamax value* of a node and show how to compute it. W.l.o.g., we assume that moves are alternated from now on.

**Definition 4.** *In the incremental game tree model the prefix negamax value $p(v)$ of node $v$ is defined recursively based on (2): If $v$ is the root $r$ of the tree, then $p(v) = f(r)$. Otherwise, $p(v) = -p(v') - d_i(v')$, where $v'$ is the predecessor of $v$ on the path to the root, $v = v'_i$, and $d_i(v') \leq 0$.*

**Lemma 7.1.** *For the incremental game tree model described above*

$$p(v) = (-1)^n f(r) + \sum_{k=0}^{n-1} (-1)^{n-k} d_{i_k}(u_k) \qquad (3)$$

*holds, where $r$ is the root of the tree and $u_0, \ldots, u_n$ is the unique path from $r$ to $v$, i.e., $u_0 = r, u_n = v$, and $u_{k+1}$ is the $i_k$-th successor of $u_k$.*

*Proof.* (Induction on depth $n$ of $v$). For $n = 0$, we have $v = r$, so the statement trivially holds. Assuming (3) holds for nodes with depth $n$, consider the value of a node $v'$ at depth $n + 1$ with predecessor $v$ and $v' = u_{i_n}$. Then:

$$\begin{aligned} p(v') &= -p(v) - d_{i_n}(v) \\ &= -\Big((-1)^n f(r) + \sum_{k=1}^{n-1} (-1)^{n-k} d_{i_k}(u_k)\Big) - d_{i_n}(v) \\ &= (-1)^{n+1} f(r) + \sum_{k=1}^{n} (-1)^{n+1-k} d_{i_k}(u_k) \end{aligned}$$

□

Our refinement of the incremental game tree model is based on the following observation:

**Theorem 7.1.** *If for all interior nodes $v$ in trees generated by the incremental model there exists a move $i$ with $d_i(v) = 0$, then for all $v$, $f(v) = p(v)$.*

*Proof.* (Induction on node height $h(v)$). If $h(v) = 0$, then $v$ is a leaf and $f(v) = p(v)$ holds by definition. Now consider a node $v$ with height $n + 1$ and suppose $f(v') = p(v')$ holds for all $v'$ with $h(v') \leq n$. W.l.o.g., we assume $d_1(v) = 0$. Among the $p(v_i)$ values, $p(v_1)$ is one of the smallest, because $p(v_i) = -p(v) - d_i \geq -p(v) = p(v_1) + d_1(v) = p(v_1)$. By the induction hypothesis we also know $f(v_i) = p(v_i)$ for all $i$. Therefore, $f(v) = \max_i(-f(v_i)) = \max_i(-p(v_i)) = -p(v_1) = p(v) + d_1 = p(v)$. □

With this result, evaluating nodes while generating trees is trivial because (3) can be maintained incrementally. This property in invaluable for analysing search performance in large trees based on true node values. We call our refined model the "Prefix Value Game Tree Model". As usual, the number of successors and move values can either be fixed or sampled from distributions to better model real-world games. We will use prefix value trees to compare the performance of alpha-beta and fastest-first search in the next section.

## 8 Experiments

To investigate the effect on search effort resulting from the FCF heuristics, we constructed synthetic trees using various branching factors, edge value ranges, and heuristic evaluation errors. For each set of parameters we generated 500 trees and performed an iterative deepening (ID) search up to depth 10 using plain Alpha-Beta, NegaScout, and MTD(f) [Plaat *et al.*, 1996b]. The expanded tree nodes were annotated with search information, such as estimated value and treesize, which was used for ordering the next ID iteration.

Our move ordering function is a weighted combination of the estimated node value, the $S_i/P_i$ value, and $S_i$. For a given node we took $S_i$ to be its out-degree. Looking deeper in the node's subtree was actually slightly detrimental since, ideally, large portions of that tree will never actually be expanded during search. Moreover the out-degree is 'free' to compute, given that we have expanded the node anyway. The value of $P_i$ is initially a hard threshold based on whether the node value (from the last ID pass) is $\geq$ beta. Computing $P_i$ empirically by "training" on trees from the same generating family produced a slight ($\sim$3%) improvement. The ordering of child nodes which have not yet been expanded is random.

In addition to synthetic trees we used FCF heuristics in solving open-handed cardplay for the trick-taking German card game *Skat* described in [Buro *et al.*, 2009]. Our search implementation sorts moves by way of a linear combination of features corresponding to the current value of a trick and the ability of teammates to contribute points. Our search also employs various pruning techniques and a transposition table — no iterative deepening is performed. Our lack of a forward-looking state evaluation function for Skat precluded an easy way to measure expected value or $P_i$. As such, we

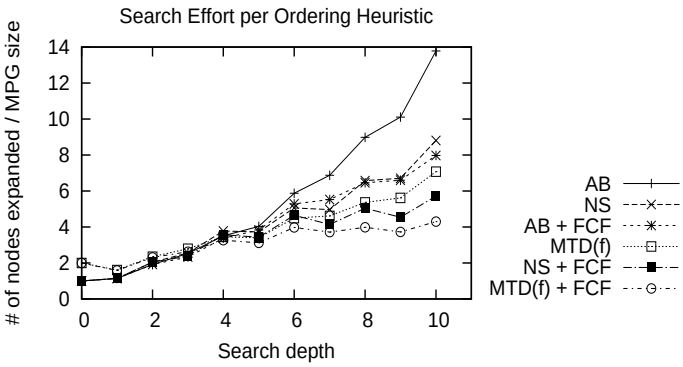


Figure 4: Number of search nodes expanded for different ordering heuristics, relative to the minimum proof graph size.

| Ordering Heuristic | Avg. Time (ms) | Avg. Nodes |
|---|---|---|
| $X$ | 77.0 | 396386 |
| $X - C \cdot S_i$ | 44.7 | 243090 |

Table 1: Effect of FCF heuristics on the search effort of open-hand Skat initial cardplay positions.

include an FCF term in our linear combination by subtracting $S_i$ := the branching factor for each move, times $C$, a weighting term optimized independently. Results are shown in Table 1, with $X$ being the portion of the move evaluation which does not consider tree size. Results are computed from 4000 initial cardplay positions from human games.

Performance results were consistent across the tree generation parameters, although the weights for combining heuristic components needed to be tuned. We made a best effort to determine the optimal weights in each case. In all cases, even when the node evaluation functions were given perfect accuracy, we observed a significant reduction in nodes expanded, compared to only using the estimated node value. Representative results are presented in Figure 4, which shows the *cumulative* number of search nodes expanded, including previous ID passes, relative to the MPG size. Values are averaged over 500 trees. The results shown are produced by trees with branching factor uniformly drawn from $[4..12]$ at each node, evaluations errors drawn from $[-4..4]$, and edge values drawn from $[-6..0]$. There was no significant difference in performance between using $S_i/P_i$ and $S_i$ as the FCF weighting term, with node reductions of 35% and 39% for NegaScout and MTD(f) respectively at depth 10. Note that the baseline MPG size grows exponentially with depth.

## 9 Conclusions and Future Work

In this paper we have presented properties of minimum proof graphs, NP-completeness results, and a linear-time algorithm for computing minimum proof graphs for trees. We also introduced the class of fastest-cut-first heuristics which sort moves dependent on sub-DAG size and value estimates. Our experimental results using a novel synthetic game tree model indicate that standard Alpha-Beta search algorithm gain exponentially when using fastest-cut-first heuristics for move ordering, but an exponential gap between the minimum proof tree size and the size of the trees visited by Alpha-Beta search remains. In future work we plan to investigate how to combine fastest-cut-first heuristics with other enhancements such as information stored in transposition tables.

## Acknowledgments

Financial support was provided by NSERC.

## References

[Buro *et al.*, 2009] M. Buro, J.R. Long, T. Furtak, and N. Sturtevant. Improving state evaluation, inference, and search in trick-based card games. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI2009)*, 2009.

[Buro, 1995] M. Buro. ProbCut: An effective selective extension of the alpha–beta algorithm. In *ICCA Journal*, pages 71–76, 1995.

[Fuller *et al.*, 1973] S.H. Fuller, J.G. Gaschnik, and J.J. Gillogly. An analysis of the alpha-beta pruning algorithm. Technical report, Carnegie Mellon University, 1973.

[Kaindl and Scheucher, 1992] H. Kaindl and A. Scheucher. Reasons for the effects of bounded look-ahead search. *IEEE Trans. Systems Man Cybernet*, 22(5):992–1007, 1992.

[Knuth and Moore, 1975] D. Knuth and R. Moore. An analysis of alphabeta pruning. *Artificial Intelligence*, 6(4):293–326, 1975.

[Kocsis and Szepesvari, 2006] L. Kocsis and C. Szepesvari. Bandit based Monte-Carlo planning. In *Proceedings of the European Conference on Machine Learning*, pages 282–293, 2006.

[Nau, 1982] D.S. Nau. An investigation of the causes of pathology in games. *Artificial Intelligence*, 19(3):257–278, 1982.

[Newborn, 1977] M.M. Newborn. The efficiency of alpha-beta search on trees with branch-dependent terminal node scores. *Artificial Intelligence*, pages 137–153, 1977.

[Pearl, 1983] J. Pearl. On the nature of pathology in game searching. *Artificial Intelligence*, 20(4):427–453, 1983.

[Plaat *et al.*, 1996a] A. Plaat, J. Schaeffer, W. Pijls, and A. De Bruin. Exploiting graph properties of game trees. In *AAAI National Conference 1:234–239*, pages 234–239, 1996.

[Plaat *et al.*, 1996b] A. Plaat, J. Schaeffer, W. Pijls, and A. de Bruin. Best-first fixed-depth minimax algorithms. *Artificial Intelligence*, 87(1-2):255–293, 1996.

[Plaat, 1996] A. Plaat. *Research, Re: search & Re-search*. PhD thesis, Rotterdam, Netherlands, 1996.

[Sahni, 1974] Sartaj Sahni. Computationally related problems. *SIAM J. Comput.*, 3(4):262–279, 1974.

[Scheucher and Kaindl, 1998] A. Scheucher and H. Kaindl. Benefits of using multivalued functions for minimaxing. *Artificial Intelligence*, pages 187–208, 1998.