JMLR: Workshop and Conference Proceedings 24:155–166, 2012 10th European Workshop on Reinforcement Learning

Rollout-based Game-tree Search Outprunes Traditional Alpha-beta

Ari Weinstein AWEINST@CS.RUTGERS.EDU
Michael L. Littman MLITTMAN@CS.RUTGERS.EDU
Sergiu Goschin SGOSCHIN@CS.RUTGERS.EDU
[*]

Editor: Marc Peter Deisenroth, Csaba Szepesvári, Jan Peters

Abstract

Recently, rollout-based planning and search methods have emerged as an alternative to traditional tree-search methods. The fundamental operation in rollout-based tree search is the generation of trajectories in the search tree from root to leaf. Game-playing programs based on Monte-Carlo rollouts methods such as "UCT" have proven remarkably effective at using information from trajectories to make state-of-the-art decisions at the root. In this paper, we show that trajectories can be used to prune more aggressively than classical alpha-beta search. We modify a rollout-based method, FSSS, to allow for use in game-tree search and show it outprunes alpha-beta both empirically and formally.

1. Introduction

Rollout-based planning is an approach that has been highly successful in game-tree search. This class of methods attempts to build estimates of the quality of available actions by sampling values from the reward and transition functions of a generative model, repeatedly beginning execution at some start state and ending at a terminal state. UCT [Kocsis and Szepesvári, 2006] is an example of this method that was initially designed for planning in standard stochastic Markov decision processes (MDPs), but was later extended to find approximate solutions in minimax trees. Since its introduction, UCT has seen significant success in the game of Go, and was the first computer agent to achieve master level play in the 9x9 variant [Gelly and Silver, 2008].

While UCT does produce state-of-the-art results in Go, it has been proven to perform very poorly when searching trees with particular properties, due to highly aggressive pruning [Coquelin and Munos, 2007; Walsh et al., 2010]. Likewise, UCT has not been successful in chess, a result attributed to poor performance of the algorithm in the presence of "search traps," which are common in chess but not in Go [Ramanujan et al., 2011].

Exact minimax search algorithms do not suffer from this behavior. Alpha-beta [Edwards and Hart, 1961] is one such branch-and-bound algorithm which computes the exact minimax value of a search tree. While it has been shown that alpha-beta is optimal in terms of the number of leaves explored [Pearl, 1982], the proof holds only for directional, or depth-first algorithms.

[*] This research was supported in part by National Science Foundation DGE-0549115.

One of the earliest best-first algorithms provably better than alpha-beta in terms of state expansions was the *best-first* SSS* [Stockman, 1979]. This algorithm, however, requires exponential time operations. Framing the algorithm differently resolved this issue: MTD($+\infty$) performs state expansions in the same order as SSS*, but is computationally tractable [Plaat et al., 1996].

In this paper, forward-search sparse sampling [Walsh et al., 2010], or FSSS, is extended for use in minimax search, resulting in the algorithm FSSS-Minimax. In its original formulation for MDPs, FSSS conducts rollouts and prunes subtrees that cannot contribute to the optimal solution. Additionally, it does so in a principled manner that allows it to avoid the previously mentioned problems of UCT. In the setting of game-tree search, we will prove that FSSS-Minimax dominates (is guaranteed to expand a subset of the states expanded by) alpha-beta, while still computing the correct value at the root. Even though the guarantees of FSSS-Minimax, SSS*, and MTD($+\infty$) are the same with respect to alpha-beta, there are important distinctions. SSS* and MTD($+\infty$) are two different approaches that produce identical policies, while FSSS-Minimax follows a different policy in terms of state expansions.

Next, the minimax setting where planning takes place, as well as the original formulation of FSSS, is discussed. From there, the extension of FSSS to minimax trees is presented. After its introduction, FSSS-Minimax is empirically compared to alpha-beta and MTD($+\infty$)$\dot{\text{A}}$ formal comparison of the two algorithms follows, proving FSSS-Minimax expands a subset of states expanded by alpha-beta. The discussion section addresses shortcomings of FSSS, and proposes future work.

2. Setting

Search is performed on a tree $G = \langle S, A, R, T\rangle$. This tuple is composed of a set of states S, a set of actions A, a reward function R, and a transition function T. All functions in G are deterministic. Each state s has a value $V(s)$, which is conditioned on the rules dictated by the three types of states in the tree. The state types are: max where the agent acts, min where the opponent chooses, and leaves where no further transitions are allowed. The goal of a max state s is to select actions that maximize $V(s)$ by choosing a path to a leaf l that maximizes $R(l)$, and min states must minimize this value. The value of the tree, $V(G)$ is equal to the value of the root of G, $V(G.root)$

The following functions over states $s \in S$ are defined: $\min(s)$, $\max(s)$, and $\text{leaf}(s)$ which return Boolean values corresponding to membership of the state to a class of states. Because it simplifies some of the descriptions, min and max states have a field defined as f. For a max state s, $s.f = \max$, and likewise for min states.

The transition function, T, is defined only for min and max states, and actions $a \in A$, where $T(s, a) \rightarrow s' \in S$. For leaves, T is undefined. The reward function $R(s) \rightarrow \mathbb{R}$ is defined only for leaves.

2.1. FSSS-Expectimax

The original formulation of FSSS, which we call FSSS-Expectimax, is designed for searching a tree that consists of expect (an additional type of state where a stochastic transition occurs) and max states [Walsh et al., 2010]. Like all FSSS algorithms, FSSS-Expectimax explores the tree while conducting rollouts from the root to a leaf, maintaining upper and lower bounds (U and L) on the value of each state in the tree. The ultimate goal is to have the the lower bound L and upper bounds U meet at the root. Rollouts in FSSS-Expectimax proceed by checking the function to choose a next state. In max states, the child is the one with the highest upper bound. In expect states, the outcome which contributes most to the difference between L and U is chosen. Previously published results show that this approach computes the correct value of the tree in finite time.

After the rollout, information from the expanded leaf is propagated up in the following manner: when a leaf is expanded, its upper and lower bounds are set to its reward. From there, for a max parent s, and updated child s', if $L(s')$ or $U(s')$ are maximal values amongst its siblings, the bounds of the parent are updated. For an expectation parent s, L and U are updated based on the weighted averages of value over all observed children. This process continues up the tree until either an update fails to modify a state value, or $L(G.root)$ and $U(G.root)$ have been updated, at which point a new rollout begins.

3. FSSS-Minimax

The most commonly used method to solve minimax trees is the alpha-beta algorithm, which prunes subtrees of the currently visited state s, based on whether further information from s can contribute to the solution. It does so by maintaining variables α and β, which correspond to the range of values at the current state that could impact the value at the root, based on the expansions done to that point. Alpha-beta search is described in Algorithm 1, but is written to highlight its similarities to Algorithm 2 (our main contribution) so storage of V is not technically necessary.

When exploring a state where the inequality $\alpha < \beta$ is not maintained, search is terminated from that state, as any further information from the subtree rooted at that state could not possibly influence $V(G)$. This simple rule is quite effective, and there are cases where the algorithm visits the minimum number of possible states [Knuth and Moore, 1975]. As mentioned, however, alpha-beta's optimality in terms of pruning is only over the class of depth-first search algorithms [Pearl, 1982]. Because rewards only exist in the leaves the setting described in this paper, we define performance in terms of the number of leaves expanded. The algorithm introduced next is a best-first algorithm that never expands more leaves than alpha-beta.

In Algorithm 2, we introduce the extension of FSSS to minimax trees, FSSS-Minimax. For the sake of brevity, beyond this point we will use FSSS to refer to FSSS-Minimax. The description that follows is for max states, but the behavior in min states is symmetric. Like alpha-beta, FSSS uses top-down variables α and β to guide pruning, but unlike alpha-beta, it also computes lower and upper bounds on the minimax value bottom up, which are maintained in the variables L and U which are maintained and updated during all rollouts on the game tree. Children effectively have L and U clipped by the α and β values of the parent. The bounds are constrained in this manner because, just like in alpha-beta, values outside the range of (α, β) cannot influence the value at the root.

After a child is identified to be visited, the α and β bounds must be adjusted based on the bounds of its siblings to avoid unnecessary calculation further down in the tree. In alpha-beta, α'—the α value used at a child state—is set to be the maximum value amongst

Algorithm 1: Alpha-Beta

```
begin Solve($G$)
1   return Search ($G.root, -\infty, \infty$)
end
begin Search($s$, $\alpha$, $\beta$)
2   if $leaf(s)$ then
3       $V(s) \leftarrow R(s)$
4       return
    end
5   $\alpha' \leftarrow \alpha$
6   $\beta' \leftarrow \beta$
7   for $i^* \leftarrow 1...|A|$ do
8       if $\max(s)$ then
9           $v \leftarrow \max_{i<i^*} V(T(s, a_i))$
10          $\alpha' \leftarrow \max(\alpha, v)$
        end
11      if $\min(s)$ then
12          $v \leftarrow \min_{i<i^*} V(T(s, a_i))$
13          $\beta' \leftarrow \min(\beta, v)$
        end
14      if $\alpha' \geq \beta'$ then
15          break
        end
16      Search ($T(s, a_{i^*}), \alpha', \beta'$)
    end
17  $V(s) \leftarrow (s.f)_i V(T(s, a_i))$
end
```

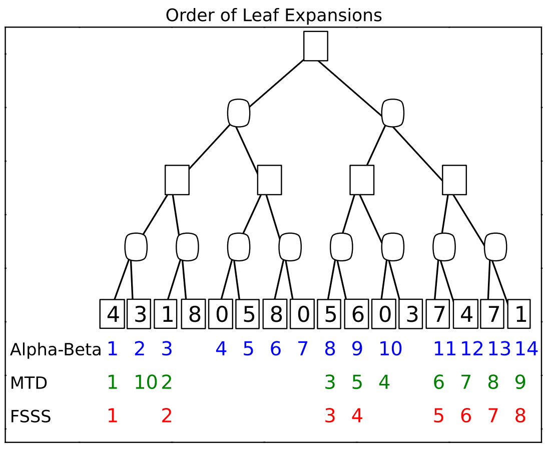


Figure 3.1: Order of expansions performed by alpha-beta, MTD($+\infty$), and FSSS. Rewards are rendered in the leaves. Max states are square, min states are circular.

α and the siblings that have been searched (those to its left). Because FSSS has partial information about all siblings, α' is set to the maximum of α and the *upper bounds* of the siblings of the child to be visited.

The modification of α' and β' by ϵ (which must be a value smaller than the possible differences in leaf values) on lines 18 and 24 of Algorithm 2, has no direct parallel in alpha-beta. As will be proved later, there is always an unexpanded leaf at the end of the trajectory followed by FSSS. For this to hold, it is necessary to have both a non-point range of possible values, as well as overlap between the ranges (α, β) and (L, U). Therefore, if $\alpha = U$, or $\beta = L$, the range of values considered becomes a point value of the state, making it look prematurely closed. Because of this, slightly incrementing β (or decrementing α) allows FSSS to still consider a range of values, without impacting the correctness.

In Figure 3.1, an illustration of the leaf expansions performed by alpha-beta, MTD($+\infty$) and FSSS for a small tree is presented. Leaves have integer rewards as labeled, and actions are ordered left to right. In this small example, alpha-beta expands 14 leaves, MTD($+\infty$) expands 10, and FSSS expands 8. The minimax value is 4, which comes from the leaf third from the right. As a depth-first method, alpha-beta can only choose to ignore leaves, but must proceed strictly from left to right. MTD($+\infty$) and FSSS, as best-first methods, can explore the leaves "out of order," which helps to grant them more effective pruning. (The numbers beneath the leaves denote the expansion order of the leaves for each algorithm.)

This example demonstrates the difference between MTD($+\infty$)and FSSS, which occurs at the fourth leaf expansion. Because of the initialization of the null-window in MTD($+\infty$), the first iteration performs what is called a "max-expansion" of the tree. In a max-expansion, max states expand all their children, but min states only expand the first (leftmost) child. Therefore, in a tree of this height, MTD($+\infty$) will always perform the first 4 expansions in this manner independent of the values encountered at the leaves.

Algorithm 2: FSSS-Minimax

```
begin Solve($G$)
1      while $L(G.root) \neq U(G.root)$ do
2          Search $(G.root, -\infty, \infty)$
       end
3      return $L(G.root)$
end
begin Search($s, \alpha, \beta$)
4      if $leaf(s)$ then
5          $L(s) \leftarrow U(s) \leftarrow R(s)$
6          return
       end
7      $i^*$, $\alpha'$, $\beta' \leftarrow$ Traverse $(s, \alpha, \beta)$
8      Search $(T(s, a_{i^*}), \alpha', \beta')$
9      $L(s) \leftarrow (s.f)_i \; L(T(s, a_i))$, $U(s) \leftarrow (s.f)_i \; U(T(s, a_i))$
end
begin Traverse($s, \alpha, \beta$)
10     for $i \leftarrow 1...|A|$ do
11         $U'(s, a_i) \leftarrow \min(\beta, U(T(s, a_i)))$, $L'(s, a_i) \leftarrow \max(\alpha, L(T(s, a_i)))$
       end
12     $\alpha' \leftarrow \alpha$, $\beta' \leftarrow \beta$
13     if $\max(s)$ then
14         $i^* \leftarrow \text{argmax}_i \, U'(s, a_i)$
15         $v \leftarrow \max_{i \neq i^*} U'(s, a_i)$
16         $\alpha' \leftarrow \max(\alpha, v)$
17         if $\alpha' = U'(s, a_{i^*})$ then
18             $\alpha' \leftarrow \alpha' - \epsilon$
           end
       end
19     if $\min(s)$ then
20         $i^* \leftarrow \text{argmin}_i \, L'(s, a_i)$
21         $v \leftarrow \min_{i \neq i^*} L'(s, a_i)$
22         $\beta' \leftarrow \min(\beta, v)$
23         if $\beta' = L'(s, a_{i^*})$ then
24             $\beta' \leftarrow \beta' + \epsilon$
           end
       end
25     return $i^*$, $\alpha'$, $\beta'$
end
```

FSSS, on the other hand, is only forced to perform the first 3 expansions regardless of the observed rewards. In all cases, the algorithm will also traverse to the grandparent of the third leaf to be expanded on the following trajectory. From that state, FSSS traverses based on the results of the first two expansions. Here, because neither had a reward greater than 5, FSSS need not expand the fourth leaf expanded by MTD($+\infty$).

4. Empirical Comparison

In this section, we compare alpha-beta, MTD($+\infty$), and FSSS in two classes of synthetic game trees. All trees have binary decisions and randomly selected rewards in the leaves. Because rewards only exist in the leaf states, the metric we report is the number of leaf states expanded, and in the experiments the number of leaves (and therefore the height of the trees) varies. Children of a state are initially explored in fixed left-to-right order in all cases, and ties are broken in favor of the left child. The height h of the tree is counted starting at 0, which is just the root, so there are 2^h leaves in a tree.

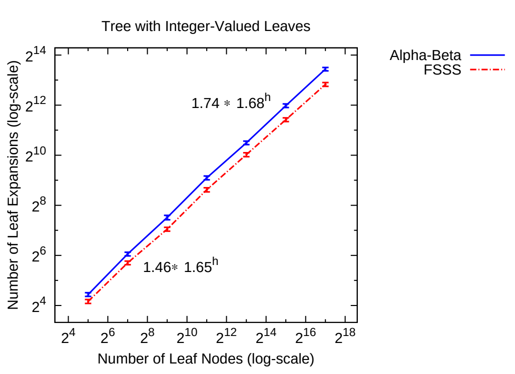
In the first experiment, rewards are integers sampled uniformly from roughly -2^{31} to 2^{31}. The results presented in Figure 4.1(a) show the number of leaves expanded by alpha-beta and FSSS when searching trees of a varying size. Note that both axes are log-scale, so the slight difference in the slope of the two lines indicates a difference in the base determining exponential growth. Using the height h, generating fit curves to the data yielded the functions $1.74 \cdot 1.68^h$ for alpha-beta and $1.46 \cdot 1.65^h$ for FSSS, which fit within the error bounds of the empirical data at almost all points. In this case, the number of expansions of FSSS and MTD($+\infty$) are not statistically significantly different, so only the expansions of FSSS are rendered. Between FSSS and MTD($+\infty$), the algorithm that expands the fewest leaves varies from tree to tree, so neither algorithm dominates the other.

We performed a second experiment to provide a clearer picture of the relative strengths of FSSS and MTD($+\infty$) in the context of trees with binary (0 or 1) valued leaves. Specifically, we measured the proportion of such trees where FSSS or MTD($+\infty$) expanded strictly fewer leaves than alpha-beta. (Neither algorithm ever expands *more* leaves than alpha-beta.)

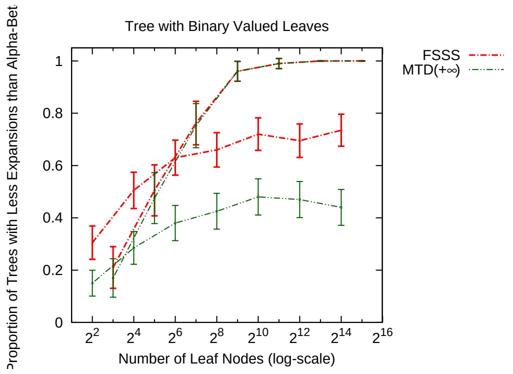
The results of this experiment are shown in Figure 4.1(b) (higher values indicate more pruning). Two things are worth mentioning. First, the results varied greatly based on the parity of the tree height—we plot performance for odd height and even height trees separately in the graph. A similar finding has been reported previously [Plaat et al., 1996]. Second, although the figure shows that FSSS expands less nodes than alpha-beta more often than MTD($+\infty$), in many cases the difference between the actual number of leaves expanded by FSSS and MTD($+\infty$)was extremely small even on the largest trees, partially because of the small range of leaf values.

5. Formal Comparison

The previous section showed that FSSS outprunes alpha-beta on a set of randomly generated game trees. This section proves a stronger result—FSSS will never expand a state that alpha-beta does not. Therefore, the total number of leaves expanded by FSSS is upper bounded by the expansions made by alpha-beta.



(*a*) Difference in number of leaf expansions between alpha-beta and FSSS.



(*b*) Fraction of trees where FSSS and MTD($+\infty$) expand strictly fewer states than Alpha-Beta.

Figure 4.1: Emprical comparison of FSSS and alpha-beta.

First, we argue that FSSS finds $V(G)$ and that it terminates because it never revisits an expanded leaf.

Theorem 1 (FSSS) *FSSS never attempts to enter a closed state, and its bounds are correct.*

This proof will be by induction. When execution begins from the root, if $L(G.root) = U(G.root)$, then calculation is complete. At each stage during a rollout, if s is a leaf, that leaf is expanded and the ranges are updated accordingly. Otherwise, in the rollout, $L(s), U(s)$ are the bounds on the value of state s, k is the branching factor ($k = |A|$), and $s_1, ..., s_k$ are the children of state s, reachable by corresponding actions $\{a_1, ..., a_k\} = A$. $L(s_1), ..., L(s_k)$ and $U(s_1), ..., U(s_k)$ are the lower and upper bounds on the children.

Recall that $L(s) = \max_i L(s_i)$, $U(s) = \max_i U(s_i)$ for max states and $L(s) = \min_i L(s_i)$, $U(s) = \min_i U(s_i)$ for min states. At the start of the rollout, 3 conditions hold:

1. $L(s) \neq U(s)$, because rollouts end when the bounds at the root match.
2. $L(s) < \beta$, because $\beta = \infty$. (If $L(s) = \beta = \infty$, then $U(s) = \infty$, contradicting Condition 1.)
3. $\alpha < U(s)$, because $\alpha = -\infty$. (If $U(s) = \alpha = -\infty$, then $L(s) = -\infty$, contradicting Condition 1.)

We now consider the recursive step when a child s_{i^*} is selected for traversal.

5.1. Condition 1: $L(s_{i^*}) \neq U(s_{i^*})$

Proof Proof by contradiction, for the case when s is a max state. Assume $L(s_{i^*}) = U(s_{i^*})$. Note that $L(s_{i^*}) \leq L(s)$ and $L(s) < \beta$. So, $L(s_{i^*}) < \beta$ and $U(s_{i^*}) < \beta$. Since $U(s_{i^*}) = \max_i \min(\beta, U(s_i))$ and $\beta > U(s_{i^*})$, then $U(s_{i^*}) = \max_i U(s_i) = U(s)$. In addition, $L(s) = \max_i L(s_i)$, so it follows that $L(s) = L(s_{i^*}) = U(s_{i^*}) = U(s)$. But, $L(s) \neq U(s)$, by the inductive hypothesis, so the contradiction implies $L(s_{i^*}) \neq U(s_{i^*})$. The same argument goes through with the appropriate substitutions if s is a min state. ■

5.2. Condition 2: $L(s_{i^*}) < \beta'$

Proof If s is a max state, $\beta' = \beta$. We have $L(s) < \beta$ by our induction hypothesis. We also have $L(s_{i^*}) \leq L(s)$ because $L(s)$ is computed to be the largest of the $L(s_i)$ values.

If s is a min state, $i^* = \text{argmin}_i \max(\alpha, L(s_i))$. Then, $\beta' = \min_{i \neq i*} \max(\alpha, L(s_i))$. We know $L(s_{i^*}) \leq \beta'$ because the range of values in the min that defines β' is more restrictive than the one that defines i^*. If $L(s_{i^*}) < \beta'$, the result is proved. If $L(s_{i^*}) = \beta'$, the algorithm increases β' infinitesimally, which also completes the result. ■

5.3. Condition 3: $\alpha' < U(s_{i^*})$

In this case, the arguments here are the same as for Condition 2, with the appropriate substitutions.

5.4. FSSS and Alpha-Beta

Now that we have established that FSSS makes consistent progress toward solving the tree, we show that all of the computation takes place within the parts of the tree alpha-beta has not pruned.

Theorem 2 *FSSS only visits parts of the tree that alpha-beta visits.*

Proof We proceed by showing that, for any state s visited by alpha-beta, FSSS will not traverse to a child state that alpha-beta prunes. Given the base case—that FSSS and alpha-beta both start at the root—it follows that FSSS will remain within the set of states visited by alpha-beta.

Let α and β be the values of the corresponding variables in FSSS when the current state is s. (We assume s is a max state. The reasoning for min states is analogous.) Let $\hat{\alpha}$ and $\hat{\beta}$ be the α and β values in alpha-beta at that same state. The "primed" versions of these variables denote the same values at a child state of s. Note that the following invariant holds: $\hat{\alpha} \leq \alpha$ and $\beta \geq \hat{\beta}$. That is, the α and β values in FSSS lie inside those of alpha-beta. To see why, note that this condition is true at $G.root$. Now, consider the computation of $\hat{\alpha}'$ in alpha-beta. It is the maximum of $\hat{\alpha}$ and all the values of the child states to its left.

Compare this value to that computed in Lines 15 and 16 of Algorithm 2. Here, the value α' passed to the selected child state is the maximum of α (an upper bound on $\hat{\alpha}$ by the inductive hypothesis) and the upper bounds of the values of *all* the other child states (not just those to its left). Because α' is assigned the max over a superset of values compared to in alpha-beta, α' must be an upper bound on $\hat{\alpha}'$.

The only case left to consider is what happens in Lines 17 and 18 of Algorithm 2. In this case, if the α' value has been increased and now matches the clipped upper bound of the selected child state, α' is decremented a tiny amount—one that is guaranteed to be less than the difference between any pair of leaf values. Since this value is increased (almost) up to β, it remains an upper bound on $\hat{\alpha}'$.

For max states, the values of β do not change in either algorithm. Thus, if the α and β range for FSSS lies inside the α and β range for alpha-beta at some state, it will also lie inside for all the states visited along a rollout to a leaf. Therefore, FSSS remains within the subtree not pruned by alpha-beta. ■

Putting the two theorems together, as a rollout proceeds, FSSS always keeps an unexpanded leaf that alpha-beta expands beneath it, and FSSS at most expands the same states expanded by alpha-beta.

6. Discussion

In terms of related work, FSSS is similar to the Score Bounded Monte-Carlo Tree Search algorithm [Cazenave and Saffidine, 2010]. The major difference between it and FSSS is the lack of α and β values to guide search of the game tree, which is significant because without them does not dominate alpha-beta. We also note there are other algorithms provably better than alpha-beta not mentioned, as we are most concerned with the relationship between FSSS and alpha-beta; further comparisons would make valuable future work.

While FSSS is guaranteed to never expand more leaves than alpha-beta, the best-first approach comes at a cost. In terms of memory requirements, alpha-beta only needs to store α, β, and the path from the root to the current state being examined, which has cost logarithmic in the size of the tree. FSSS, on the other hand, must store upper and lower bounds for all states it has encountered, which is linear in the number of states visited. Fortunately, it has been shown that memory is not a limiting factor when performing game-tree search even on "large" games such as chess and checkers [Plaat et al., 1996].

Similarly, alpha-beta is also superior to FSSS in terms of computational cost. In the worst case, nearly all the states in the tree must be expanded to compute $V(G)$. For alpha-beta, processing is done in one depth-first traversal of the tree, which has cost linear in the size of the tree. FSSS, however, must conduct a number of trajectories through the tree, visiting each leaf once. In this case, for a tree of height h, alpha-beta visits 2^{h+1} states, whereas FSSS visits states a total of $h2^h$ times. The costs associated with increased memory use and full rollouts did seem to manifest themselves in practice, as alpha-beta was faster than FSSS in all our experiments.

Although we did observe FSSS to be slower than alpha-beta, this may not hold in all domains. In our experiments, $R(s)$ is quickly queried, so it is not harmful to visit many leaves. In some domains, however, the method that assigns a value to a leaf is expensive to compute. While good heuristic functions are known for evaluating board states in chess, they do not yet exist for the game of Go. As a result, when attempting to evaluate the quality of a leaf (generally not end-game positions in practice), the game is played out to completion using a (potentially randomized) policy and the process may be conducted many times to obtain a reasonable estimate of value. In such cases where evaluating the quality of a leaf is expensive, FSSS may be faster in terms of actual processing time than alpha-beta.

In order to improve computation time, FSSS can benefit from refinements which we have not included in order to simplify the algorithm. In particular, value backups can stop, and rollouts can be restarted from, the deepest state in the tree visited in the last trajectory that did not have a change in its (L, U) bounds instead of from the root of the tree. This does not alter the policy of the algorithm.

Another point worth noting is that, in practice, search algorithms are generally combined with move ordering, iterative deepening, and evaluation functions—speed up techniques that have not been addressed here. While these additions generally improve performance, they also violate assumptions made in terms of guarantees of state expansions between algorithms. For example, when incorporating these methods, it was shown that alpha-beta sometimes expands fewer states than MTD($+\infty$) Plaat et al. [1996]; we believe the same to be true for FSSS.

7. Future Work

There are a number of opportunities to extend FSSS. The original formulation of FSSS incorporates domain knowledge about possible values a state can achieve in G. As long as these values are admissible, pruning would be improved while proofs of correctness as well as dominance of alpha-beta would still be maintained. Additionally, FSSS can be extended for use in minimax domains that involve stochasticity, such as backgammon. Finally, an extension of FSSS to the anytime case would be useful when the game-tree cannot be solved due to time constraints.

Finally, FSSS as presented only terminates once the upper and lower bounds at the root become equal. If we are simply interested in the optimal action, as opposed to the value at the root, execution can be terminated once the lower bound on an action is greater than the upper bound on all other actions at the root. If computation has a hard limitation (in terms of time or number of samples requested), extension of the algorithm to function in an "anytime" manner, such as what UCT performs, also warrants investigation.

References

Tristan Cazenave and Abdallah Saffidine. Score bounded monte-carlo tree search. In *Computers and Games*, pages 93–104, 2010.

Pierre-Arnuad Coquelin and Remi Munos. Bandit algorithms for tree search. In *Uncertainty in Artificial Intelligence*, pages 67–74, 2007.

D.J. Edwards and T.P. Hart. The alpha-beta heuristic. In *AI Memos (1959 - 2004)*, 1961.

Sylvain Gelly and David Silver. Achieving master level play in 9 x 9 computer go. In *Association for the Advancement of Artificial Intelligence*, pages 1537–1540, 2008.

Donald E. Knuth and Ronald W. Moore. An analysis of alpha-beta pruning. In *Artificial Intelligence*, volume 6, pages 293–326, 1975.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European Conference on Machine Learning*, pages 282–293, 2006.

Judea Pearl. The solution for the branching factor of the alpha-beta pruning algorithm and its optimality. *Communications of the ACM*, 25:559–564, August 1982.

Aske Plaat, Jonathan Schaeffer, Wim Pijls, and Arie de Bruin. Best-first fixed-depth minimax algorithms. In *Artificial Intelligence*, volume 87, pages 255 – 293, 1996.

Raghuram Ramanujan, Ashish Sabharwal, and Bart Selman. On the behavior of uct in synthetic search spaces. In *ICAPS 2011 Workshop, Monte-Carlo Tree Search: Theory and Applications*, 2011.

George C. Stockman. A minimax algorithm better than alpha-beta? In *Artifical Intelligence 12*, pages 179–196, 1979.

Thomas J. Walsh, Sergiu Goschin, and Michael L. Littman. Integrating sample-based planning and model-based reinforcement learning. In *Association for the Advancement of Artificial Intelligence*, 2010.