Bounded Suboptimal Game Tree Search

Dor Atzmon and **Roni Stern**
Ben Gurion University of the Negev
Be'er Sheva, Israel

Abdallah Saffidine
Australian National University
Canberra, Australia

Abstract

Finding the minimax value of a game is an important problem in a variety of fields, including game theory, decision theory, statistics, philosophy, economics, robotics, and security. Classical algorithms such as the Minimax algorithm can be used to find the minimax value, but require iterating over the entire game tree, which is in many cases too large. Alpha-Beta pruning identifies portions of the game tree that are not necessary for finding the minimax value, but in many cases the remaining part of the game tree is still too large to search in reasonable time. For such cases, we propose a class of algorithms that accepts a parameter ϵ and returns a value that is guaranteed to be at most ϵ away from the true minimax value. We lay the theoretical foundation for building such algorithms and present one such algorithm based on Alpha-Beta. Experimentally, we show that our algorithm allows controlling this runtime/solution quality tradeoff effectively.

Introduction

Minimax is a fundamental decision-making rule in zero-sum games (Wald 1945), in which a player chooses an action that maximizes its utility assuming the other player will choose the action that will minimize its utility. The *minimax value* of a game is the expected utility for the first player obtained assuming both players follow the minimax rule. Finding the *minimax value* of a game is a challenging problem with applications in a variety of fields, including game theory, decision theory, statistics, philosophy, economics, robotics, and security. In fact, when the minimax value of a game is found the game is said to be *solved* (Schaeffer et al. 2007; Irving, Donkers, and Uiterwijk 2000; Saffidine, Jouandeau, and Cazenave 2011).

Consequently, many algorithms have been proposed for finding the minimax value of a game, such as the Minimax algorithm, Alpha-Beta, Proof Number Search (Allis, van der Meulen, and Van Den Herik 1994), Monte Carlo Tree Search (Coulom 2006; Kocsis and Szepesvári 2006; Lanctot et al. 2013), and others. Nonetheless, finding the minimax value of many games is still a challenge due to the size of the game tree they define.

This complexity challenge also exists when solving hard single-agent search problems, where finding optimal solutions is often not feasible. A popular way to address this in single-agent search is to relax the requirement of finding an optimal solution, and allow returning suboptimal solutions. This often raises a runtime versus solution quality tradeoff. Bounded-suboptimal single-agent search algorithms control this tradeoff with a user-defined parameter that bounds the allowed suboptimality of the returned solution. Weighted A* (Pohl 1970), Explicit Estimation Search (Thayer and Ruml 2011), and Dynamic Potential Search (Gilon, Felner, and Stern 2016) are examples of such algorithms.

In this work, we present a theory for building "bounded-suboptimal" algorithms for finding the minimax value of game trees. Such algorithms, which we refer to as *bounded-suboptimal game tree* algorithms, are guaranteed to return a value that is at most ϵ from the true minimax value, where ϵ is a user-provided parameter.

We provide a general theory for extending the well-known Alpha-Beta pruning technique to provide bounded-suboptimal solutions. This includes concrete pruning rules for both deterministic games and games with chance, based on *-Minimax (Ballard 1983; Hauk, Buro, and Schaeffer 2004). These pruning rules can be applied to a wide range of game tree search algorithms. We provide details of a concrete game tree search algorithm that uses these rules, called Bounded Alpha-Beta (BAB). Experimentally, BAB shows that it indeed provides the desired tradeoff of runtime for optimality on two domains. While the focus of this work is not on developing a new AI game playing algorithm, we also provide preliminary evidence that suggests the pruning done by a player using bounded-suboptimal search algorithm can be effective also for game-playing.

Background

In this work, we focus on finite, two-player, zero-sum game with chance and perfect information, defined as follows.

Definition 1 (Zero-Sum, Perfect Information Game)**.** *A finite, two-player, zero-sum game with chance and perfect information is defined by a tuple $\langle \mathcal{S}, \mathcal{T}, \mathcal{A}, \mathcal{P}, \tau, u, s_1 \rangle$, where: (1) $\mathcal{S}$ is a finite, non-empty, set of states; (2) $\mathcal{T} \subseteq \mathcal{S}$ is the set of terminal states; (3) $\mathcal{A}$ is a finite, non-empty, set of actions; (4) $\mathcal{P}$ is a transition probability function where $\mathcal{P}(s, a, s')$ is the probability of getting to state s' after performing a in state s; (5) $\tau : \mathcal{S} \rightarrow \{1, 2\}$ is a player index function, returning the player whose turn is to act in a given state, (6) u*

is a utility function $u : \mathcal{T} \to [v_{min}, v_{max}]$, mapping terminal states to their utility to player 1, where v_{min} and v_{max} denote the minimum and maximum possible utility; and (7) s_1 is the initial state.

Hereinafter, we use the term *game* to refer to the type of games defined in Def. 1. A game is called *deterministic* if performing an action on a state can lead to exactly one state, i.e., for every state action pair (s, a) there exists a single state s' for which $\mathcal{P}(s, a, s') = 1$.

We refer to player 1 and player 2 as the MAX player and the MIN player, respectively. A game starts with the MAX player ($\tau(s_1) = 1$). In each step the turn player chooses an action from $\mathcal{A}$ to apply to the current state, generating a new state according to the transition probability function $\mathcal{P}$. This process continues until a player reaches a terminal state, at which point the MAX player gets the utility of that state.

A game defines a tree, referred to as the *game tree*. There are three types of nodes in a game tree: MAX, MIN, and CHANCE. A MAX node and a MIN node represent a state s in which it is the MAX player's turn ($\tau(s) = 1$) and the MIN player's turn ($\tau(s) = 2$), respectively. A CHANCE node represents a state-action pair (s, a). A MAX or MIN node that represents a state s has a CHANCE node child for every applicable action in a. The corresponding CHANCE node has a MAX or MIN child node for every state s' such that $\mathcal{P}(s, a, s') > 0$. For every such child node n' we denote by $\pi(n')$ the probability of reaching s' after doing a in state s, i.e., $\mathcal{P}(s, a, s') = \pi(n')$. n_1 denotes the node corresponding to the initial state s_1. For a given node n in a game tree, we denote by $p(n)$ and $C(n)$ the parent and set of children of n.

The *minimax* value of a node n, denoted $V(n)$, is defined by

$$V(n) = \begin{cases} \max\limits_{c \in C(n)} V(c) & n \text{ is a MAX node} \\ \min\limits_{c \in C(n)} V(c) & n \text{ is a MIN node} \\ \sum\limits_{c \in C(n)} \pi(c)\, V(c) & n \text{ is a CHANCE node} \\ u(s) & n \text{ is a leaf, representing state } s \end{cases} \tag{1}$$

The minimax value of n_1 is also referred to as the minimax value of the game.

Deterministic Games

The Minimax algorithm is a classical algorithm for finding the minimax value of a deterministic game. It performs a depth-first search over the game tree and propagates the utility of the terminal nodes according to the computation of the minimax value (Eq. 1).

The size of the game tree is often too large to search, and many improvements and variants of the Minimax algorithm have been proposed. A common approach is to limit the depth of the search, and to estimate "leaves" that are not terminal states by a heuristic function. Using a heuristic function to evaluate non-terminal states is common when developing AI players. When using a heuristic, however, the value obtained when running the Minimax algorithm can be different from the minimax value of the game.

```
function ALPHA-BETA-SEARCH(state) returns an action
  $v \leftarrow$ MAX-VALUE(state, $-\infty, +\infty$)
  return the action in ACTIONS(state) with value $v$

function MAX-VALUE(state, $\alpha, \beta$) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  $v \leftarrow -\infty$
  for each $a$ in ACTIONS(state) do
    $v \leftarrow$ MAX($v$, MIN-VALUE(RESULT(s,a), $\alpha, \beta$))
    if $v \geq \beta$ then return $v$
    $\alpha \leftarrow$ MAX($\alpha$, $v$)
  return $v$

function MIN-VALUE(state, $\alpha, \beta$) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  $v \leftarrow +\infty$
  for each $a$ in ACTIONS(state) do
    $v \leftarrow$ MIN($v$, MAX-VALUE(RESULT(s,a) , $\alpha, \beta$))
    if $v \leq \alpha$ then return $v$
    $\beta \leftarrow$ MIN($\beta$, $v$)
  return $v$
```

Figure 1: The Alpha-Beta pseudo code, from Russell and Norvig's AI textbook (Russell and Norvig 2016).

An orthogonal improvement that is the focus of this work is the famous Alpha-Beta pruning method. Figure 1 lists the pseudo-code of Alpha-Beta as given by the standard Russell and Norvig AI textbook (2016). When using Alpha-Beta pruning, every searched node n maintains two values $\alpha(n)$ and $\beta(n)$. Informally, the Alpha-Beta pruning rule says that if $V(n)$ is smaller than $\alpha(n)$ or larger than $\beta(n)$ then n can be pruned, i.e., the minimax value of the game can be found without expanding n. In more details, $\alpha(n)$ is initialized to v_{min} for n_1 and to $\alpha(p(n))$ for all other nodes, and $\beta(n)$ is initialized to v_{max} for n_1 and to $\beta(p(n))$ for all other nodes. If a MAX node has a child that returns a value larger than $\alpha(n)$ then $\alpha(n)$ is updated to that value. Similarly, if n is a MIN node and has a child that returns a value smaller than $\beta(n)$ then $\beta(n)$ is updated to that value. The Alpha-Beta pruning rules states that a state is pruned if $\alpha(n) \geq \beta(n)$.

Many improvements and variants of Alpha Beta have been proposed throughout the years, including searching with narrower (α, β) window and changing it dynamically (Pearl 1980; Reinefeld 1983), using transposition tables, and using sophisticated child ordering. See Schaeffer and Plaat (1996) for a survey.

Games with Chance

Game tree search algorithms such as Minimax and Alpha-Beta have well-known counterparts for games with chance. The ExpectiMax algorithm (Michie 1966) performs a depth-first search to find the minimax value of games with chance. The *-Minimax (Ballard 1983) family of algorithms adapt Alpha-Beta pruning to games with chance. The high-level idea of these pruning techniques is to consider the value of unexplored children of CHANCE nodes as any value between v_{min} and v_{max} and prune accordingly.

In more details, *-Minimax algorithms maintain two values for every explored node n, $L(n)$ and $U(n)$, which maintain lower and upper bounds, respectively, on the minimax value of n. $L(n)$ is initialized to v_{min} and $U(n)$ to v_{max}. MIN nodes update $U(n)$ so that $U(n)$ is the minimum over the U values of all its explored children. Analogously, MAX nodes update $L(n)$. CHANCE nodes update $U(n)$ to be the expectation over the $U(n)$ value of its children, assuming $v_{\max}$ for every unexplored child, and update $L(n)$ to be the expectation over the $L(n)$ value of its children, assuming $v_{\min}$ for every unexplored child. The α and β values of a node n are then derived from $U(n)$ and $L(n)$. For more details see (Hauk, Buro, and Schaeffer 2004).

Finding a Bounded-Suboptimal Solution

The pruning done by Alpha-Beta and *-Minimax are *safe*, in the sense that the minimax value the search ends up with is exactly the same as the minimax value obtained without the pruning. In the best case, one can obtain a square root reduction in the number of generated states for uniform game trees (Knuth and Moore 1975; Ballard 1983). In the worst case, however, no pruning occurs. Even with these pruning techniques, computing the exact minimax value of many games is still not feasible.

When this occurs in single-agent search, a common approach is to trade-off solution quality for runtime (Pohl 1970; Thayer and Ruml 2011; Gilon, Felner, and Stern 2016). In this work, we propose a method for doing this for game tree search, when searching for the minimax value. To define solution quality in this context, we say that an *optimal* solution to a game corresponds to finding its exact minimax value, and the *suboptimality* of any other solution is its absolute difference from the minimax value. Thus, a *bounded-suboptimal* game tree search algorithm is defined as a game tree search algorithm that accepts $\epsilon \geq 0$ as input and outputs a solution with suboptimality at most ϵ. Naturally an effective bounded-suboptimal game tree search algorithm is also expected to provide stronger pruning for larger values of ϵ. To the best of our knowledge, there are no bounded-suboptimal game tree search algorithms in the literature.

Naïve Approach

Consider first only deterministic games, and the following straightforward approach to adapt Alpha-Beta to be a bounded suboptimal game tree search algorithm. Instead of pruning a node n only when $\beta(n) \leq \alpha(n)$ (see the 5th line the `Max-Value` and `Min-Value` functions in Figure 1), prune n when $\beta(n) \leq \alpha(n) + \epsilon$.

While this simple Alpha-Beta adaption may prune larger parts of the game tree compared to regular Alpha-Beta, the resulting suboptimality, however, is not bounded by ϵ. To see this, consider the following example.

Considering we execute the standard alpha-beta algorithm with a pruning rule of $\beta(n) \leq \alpha(n) + \epsilon$ on the tree presented in Figure 2 (squares represent MAX nodes and circles represent MIN nodes). We explore first the root and then its leftmost child, update $\alpha(A) = X$, and transmit it to node B. We explore the left child of node B and update $\beta(B) = X + \epsilon$, then we prune its right child, return its value of $X+\epsilon$ up to A, update $\alpha(A)$ to $X + \epsilon$, and transmits it to node C. After we explored the left child of node C, we update $\beta(C) = X+2\epsilon$, prune its right child and return $X + 2\epsilon$ to the root. This procedure can go on and on until $X + m\epsilon$ (node D) for every possible positive value of m. At the end of the algorithm execution, the value of the root is $X + m\epsilon$ despite the fact that

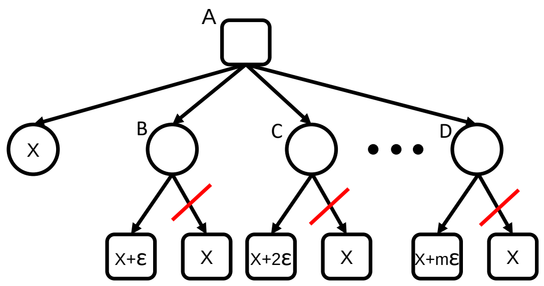


Figure 2: Example of adding ϵ to the standard alpha-beta

the minimax value in this example is X and that this gap is greater than the value of ϵ.

Next, we present our pruning rule, which is guaranteed to preserves the desired suboptimality bound and is applicable to both deterministic games and games with chance.

Bounded-Suboptimal Pruning Rule

Let G be a game tree of some game, A be a game tree search algorithm, and $A(G)$ be the subtree of G that includes all the nodes generated by A when run on G. We assume that $A(G)$ is connected and has the same root as G. $C_A(n)$ denotes the children of n that are in $A(G)$. Of course $C_A(n) \subseteq C(n)$. The nodes in $A(G)$ can be divided to three types: *leaves*, *partially expanded*, and *fully expanded*. A node is a *leaf* in $A(G)$ if none of its children are in $A(G)$. Note that a leaf in $A(G)$ may or may not be a terminal node (leaf) in G. A node in $A(G)$ is *fully expanded* if all its children are in $A(G)$. Every node in $A(G)$ that is not a leaf node and is not fully expanded is called a *partially expanded* node. Partially expanded nodes correspond to nodes with pruned children. In order to prune children of chance nodes, Lanctot et al. (2013) used lower and upper bounds. Below, we use these bounds to achieve a bounded suboptimal minimax value. Following Lanctot et al., we define L, U, α, and β for nodes in $A(G)$, as follows. Each of these four maps associates nodes in $A(G)$ to values in $[v_{\min}, v_{\max}]$. Roughly, $L(n)$ and $U(n)$ represent the tightest lower and upper bounds, respectively, on the minimax value of n obtained by running A on G, and if $V(n)$ is not in the range $[\alpha(n), \beta(n)]$ then it will not impact the minimax value of its parent. Formally, L, U, α, and β must satisfy the following conditions.

C1. For every non-leaf MAX node n:
- (1) $L(n) = \max_{c \in C_A(n)} L(c)$
- (2) $U(n) = v_{\max}$ if n is partially expanded
- (3) $U(n) = \max_{c \in C_A(n)} U(c)$ if n is fully expanded.

C2. For every non-leaf MIN node n:
- (1) $U(n) = \min_{c \in C_A(n)} U(c)$
- (2) $L(n) = v_{\min}$ if n is partially expanded
- (3) $L(n) = \min_{c \in C_A(n)} L(c)$ if n is fully expanded

C3. For every non-leaf CHANCE node n:
- (1) $L(n) = \sum_{c \in C_A(n)} \pi(c) L(c) + (1 - \sum_{c \in C_A(n)} \pi(c)) v_{\min}$
- (2) $U(n) = \sum_{c \in C_A(n)} \pi(c) U(c) + (1 - \sum_{c \in C_A(n)} \pi(c)) v_{\max}$

C4. For the root n_1, $\alpha(n_1) = L(n_1)$ and $\beta(n_1) = U(n_1)$

C5. For every MAX or MIN node n that is not the root and every child node $c \in C_A(n)$:
(1) $\alpha(c) = \max(\alpha(n), L(c))$
(2) $\beta(c) = \min(\beta(n), U(c))$

C6. For every CHANCE node n that is not the root and every child node $c \in C_A(n)$:
(1) $\alpha(c) = \max(L(c), \frac{\alpha(n)-U(n)}{\pi(c)} + U(c))$
(2) $\beta(c) = \min(U(c), \frac{\beta(n)-L(n)}{\pi(c)} + L(c))$

First, we establish that L and U constitute admissible bounds for all nodes of the tree.

Lemma 1. *If for every leaf node n it holds that $V(n) \in [L(n), U(n)]$, then it also holds for all other nodes in $A(G)$.*

Proof. We prove Lemma 1 by induction on the height of the nodes. For the base case, nodes of height 0 are leaves, and the statement is an assumption of the lemma.

For the induction step, let us assume that for any node n of height $\leq H$, we have $L(n) \leq V(n) \leq U(n)$ and let us prove that this also holds for nodes of height $H+1$. For a node n of height $H+1$, any child $c \in C_A(n)$ has height $\leq H$, so in particular $L(c) \leq V(c) \leq U(c)$ (induction hypothesis).

Case #1: n is a MAX node. $C_A(n) \subseteq C(n)$, so we have

$$L(n) = \max_{c \in C_A(n)} L(c) \leq \max_{c \in C_A(n)} V(c) \leq \max_{c \in C(n)} V(c) = V(n) \quad (2)$$

If n is fully expanded, then $C_A(n) = C(n)$ and so

$$V(n) = \max_{c \in C(n)} V(c) = \max_{c \in C_A(n)} V(c) \leq \max_{c \in C_A(n)} U(c) = U(n) \quad (3)$$

Else, n is partially expanded and so $V(n) \leq v_{\text{max}} = U(c)$. Thus, in all cases it holds that $L(n) \leq V(n) \leq U(n)$.

Case #2: n is a MIN node. $C_A(n) \subseteq C(n)$, so we have

$$V(n) = \min_{c \in C(n)} V(c) \leq \min_{c \in C(n)} U(c) \leq \min_{c \in C_A(n)} U(c) = U(n) \quad (4)$$

If n is fully expanded, then $C_A(n) = C(n)$ and so

$$L(n) = \min_{c \in C_A(n)} L(c) \leq \min_{c \in C_A(n)} V(c) = \min_{c \in C(n)} V(c) = V(n) \quad (5)$$

Else, n is partially expanded and so $L(c) = v_{\text{min}} \leq V(n)$. Thus, in all cases it holds that $L(n) \leq V(n) \leq U(n)$.

Case #3: n is a CHANCE node. For every child c it holds that $v_{\text{min}} \leq V(c)$. Also, $1 - \sum_{c \in C_A(n)} \pi(c) = \sum_{c \in C(n) \setminus C_A(n)} \pi(c)$. Following C3(1) we have

$$L(n) = \sum_{c \in C_A(n)} \pi(c) L(c) + (1 - \sum_{c \in C_A(n)} \pi(c)) v_{\text{min}} \quad (6)$$

$$L(n) \leq \sum_{c \in C_A(n)} \pi(c) V(c) + \sum_{c \in C(n) \setminus C_A(n)} \pi(c) V(c) = V(n) \quad (7)$$

Similarly

$$V(n) = \sum_{c \in C_A(n)} \pi(c) V(c) + \sum_{c \in C(n) \setminus C_A(n)} \pi(c) V(c) \leq U(n) \quad (8)$$

Thus, $L(n) \leq V(n) \leq U(n)$ as required. □

Second, we make the following straightforward observation, based on conditions C4–C6.

Lemma 2. *For any node n, $L(n) \leq \alpha(n)$ and $\beta(n) \leq U(n)$.*

Third, we prove that ϵ bounds coming from leaves and pruned nodes propagate through the tree without increasing.

Lemma 3. *If $\beta(n) \leq \alpha(n) + \epsilon$ holds for all leaf nodes and all partially expanded nodes, then it also holds for all other nodes in $A(G)$.*

Proof. We prove Lemma 3 by induction on the height of the nodes. For the base case, nodes of height 0 are leaves, and the statement is an assumption of the lemma.

For the induction step, let us assume that for any node n of height $\leq H$, we have $\beta(n) \leq \alpha(n) + \epsilon$ and let us prove that this also holds for nodes of height $H+1$. For a node n of height $H+1$, any child $c \in C_A(n)$ has height $\leq H$, so in particular $\beta(c) \leq \alpha(c) + \epsilon$ (the induction hypothesis).

Case #1: n is a MAX node. For every child c, applying condition C5 to the induction hypothesis gives the inequality $\min(\beta(n), U(c)) \leq \max(\alpha(n), L(c)) + \epsilon$. Therefore

$$\max_{c \in C(n)} \min(\beta(n), U(c)) \leq \max_{c \in C(n)} \max(\alpha(n), L(c)) + \epsilon \quad (9)$$

$$\min(\beta(n), \max_{c \in C(n)} U(c)) \leq \max(\alpha(n), \max_{c \in C(n)} L(c)) + \epsilon \quad (10)$$

$$\min(\beta(n), U(n)) \leq \max(\alpha(n), L(n)) + \epsilon \quad (11)$$

From Lemma 2 conclude that $\beta(n) \leq \alpha(n) + \epsilon$.

Case #2: n is a MIN node. Similarly to Case #1 we have

$$\min_{c \in C(n)} \min(\beta(n), U(c)) \leq \min_{c \in C(n)} \max(\alpha(n), L(c)) + \epsilon \quad (12)$$

$$\min(\beta(n), \min_{c \in C(n)} U(c)) \leq \max(\alpha(n), \min_{c \in C(n)} L(c)) + \epsilon \quad (13)$$

$$\min(\beta(n), U(n)) \leq \max(\alpha(n), L(n)) + \epsilon \quad (14)$$

Leading to the desired result, $\beta(n) \leq \alpha(n) + \epsilon$.

Case #3: n is a CHANCE node. For every child node c, using condition C6 gives the two equalities $\alpha(c) = \max(L(c), \frac{\alpha(n)-U(n)}{\pi(c)} + U(c))$ and $\beta(c) = \min(U(c), \frac{\beta(n)-L(n)}{\pi(c)} + L(c))$. Consider each of the four possible assignments for $\alpha(c)$ and $\beta(c)$.
Case #3.1: there is a child $c \in C_A(n)$ such that

$$\alpha(c) = \frac{\alpha(n) - U(n)}{\pi(c)} + U(c) \textbf{ and } \beta(c) = U(c)$$

The induction hypothesis gives

$$U(c) \leq \frac{\alpha(n) - U(n)}{\pi(c)} + U(c) + \epsilon \quad (15)$$

$$U(n) \leq \alpha(n) + \pi(c)\epsilon \quad (16)$$

Since $\beta(n) \leq U(n)$ and $0 \leq \pi(c) \leq 1$, we get $\beta(n) \leq \alpha(n) + \epsilon$.

Case #3.2: there is a child $c \in C_A(n)$ such that

$$\alpha(c) = L(c) \textbf{ and } \beta(c) = \frac{\beta(n) - L(n)}{\pi(c)} + L(c)$$

The induction hypothesis gives

$$\frac{\beta(n) - L(n)}{\pi(c)} + L(c) \leq L(c) + \epsilon \quad (17)$$

$$\beta(n) \leq L(n) + \pi(c)\epsilon \quad (18)$$

As $L(n) \leq \alpha(n)$ and $\pi(c) \in [0, 1]$, we get $\beta(n) \leq \alpha(n) + \epsilon$.

Case #3.3: there is a child $c \in C_A(n)$ such that

$$\alpha(c) = \frac{\alpha(n) - U(n)}{\pi(c)} + U(c) \textbf{ and } \beta(c) = \frac{\beta(n) - L(n)}{\pi(c)} + L(c)$$

The induction hypothesis gives

$$\frac{\beta(n) - L(n)}{\pi(c)} + L(c) \leq \frac{\alpha(n) - U(n)}{\pi(c)} + U(c) + \epsilon \quad (19)$$

Since $c \in C_A(n)$, then we have

$$\pi(c)(U(c) - L(c)) \leq \sum_{c' \in C_A(n)} \pi(c')(U(c') - L(c')) \quad (20)$$

$$\pi(c)(U(c) - L(c)) \leq U(n) - L(n) \quad (21)$$

$$L(n) - \pi(c)L(c) \leq U(n) - \pi(c)U(c) \quad (22)$$

Adding $\pi(c)\cdot$ Eq. (19) to Eq. (22) gives $\beta(n) \leq \alpha(n) + \pi(c)\epsilon$.

Case #3.4: for each child $c \in C_A(n)$ it holds that

$$\alpha(c) = L(c) \textbf{ and } \beta(c) = U(c)$$

Since $\beta(c) \leq \alpha(c) + \epsilon$, we have that

$$\sum_{c \in C_A(n)} \pi(c)\beta(c) \leq \sum_{c \in C_A(n)} \pi(c)(\alpha(c) + \epsilon) \quad (23)$$

$$\sum_{c \in C_A(n)} \pi(c)U(c) \leq \sum_{c \in C_A(n)} \pi(c)L(c) + \sum_{c \in C_A(n)} \pi(c)\epsilon \quad (24)$$

Since n is fully expanded, $\sum_{c \in C_A(n)} \pi(c) = 1$ and C3 gives

$$U(n) \leq L(n) + \epsilon \quad (25)$$

from Lemma 2 conclude that $\beta(n) \leq \alpha(n) + \epsilon$. □

We are now fully equipped to prove the main theoretical contribution of this paper.

Theorem 1. *For a game tree G, a game tree search algorithm A, and any $\epsilon \geq 0$, if all the following conditions hold*

1. *for every leaf node n*

$$L(n) \leq V(n) \leq U(n) \textit{ and } U(n) - L(n) \leq \epsilon$$

Algorithm 1: BoundedAB

```
Input: $n$ - a game tree node
Input: $\epsilon$ - an error margin
1  if $n$ is a terminal node then
2      return $(V(n), V(n))$
3  $L(n) \leftarrow v_{\min}$; $U(n) \leftarrow v_{\max}$
4  if $n$ is a MAX node then  $best_U \leftarrow v_{\min}$
5  else if $n$ is a MIN node then  $best_L \leftarrow v_{\max}$
6  if $n$ is $n_1$ then  $\alpha(n) = v_{\min}$; $\beta(n) = v_{\max}$
7  else if $p(n)$ is a MAX node or a MIN node then
     $\alpha(n) = \alpha(p(n))$; $\beta(n) = \beta(p(n))$
8  else
9      $\alpha(n) \leftarrow \max(v_{\min}, \frac{\alpha(p(n)) - U(p(n))}{\pi(n)} + U(n))$
10     $\beta(n) \leftarrow \min(v_{\max}, \frac{\beta(p(n)) - L(p(n))}{\pi(n)} + L(n))$
11 foreach child $c$ of $n$ do
12     $(L(c), U(c)) \leftarrow$ BoundedAB($c$, $\epsilon$)
13     if $n$ is a MAX node then
14         $best_U \leftarrow \max(best_U, U(c))$
15         $L(n) \leftarrow \max(L(n), L(c))$
16     else if $n$ is a MIN node then
17         $best_L \leftarrow \min(best_L, L(c))$
18         $U(n) \leftarrow \min(U(n), U(c))$
19     else
20         $L(n) \leftarrow L(n) + \pi(c)(L(c) - v_{\min})$
21         $U(n) \leftarrow U(n) - \pi(c)(v_{\max} - U(c))$
22     $\alpha(n) \leftarrow \max(L(n), \alpha(n))$
23     $\beta(n) \leftarrow \min(U(n), \beta(n))$
24     if $\beta(n) \leq \alpha(n) + \epsilon$ and c is not the last child of $n$
        then return $(L(n), U(n))$
25 if $n$ is a MAX node then  $U(n) \leftarrow best_U$
26 else if $n$ is a MIN node then  $L(n) \leftarrow best_L$
27 return $(L(n), U(n))$
```

2. *for every partially expanded node n, $\beta(n) \leq \alpha(n) + \epsilon$*

then

$$V(n_1) \in [L(n_1), L(n_1) + \epsilon]$$

Proof. The first condition lets us apply Lemma 1 and obtain for the root $L(n_1) \leq V(n_1) \leq U(n_1)$, and to apply Lemma 2 to obtain $\beta(n) \leq \alpha(n) + \epsilon$ for every leaf node n. Together with the second condition we can apply Lemma 3 for all nodes in $A(G)$. In particular, we obtain for the root $\beta(n_1) \leq \alpha(n_1) + \epsilon$. From condition C4, we conclude that $U(n_1) \leq L(n_1) + \epsilon$. Therefore, $L(n_1) \leq V(n_1) \leq L(n_1) + \epsilon$ as intended. □

Note that in the above proof we prove that $U(n_1) \leq L(n_1) + \epsilon$. Therefore, if the conditions in Theorem 1 hold, then it is also true that $V(n_1) \in [U(n_1) - \epsilon, U(n_1)]$.

The Bounded Alpha-Beta (BAB) Algorithm

Next, we described Bounded Alpha-Beta (BAB), which is a bounded-suboptimal game tree search algorithm that is based on the classical Alpha-Beta algorithm, *-Minimax algorithm, and our bounded-suboptimal pruning rule (Theorem 1). BAB performs a depth-first search on the game tree, maintains in each node its U and L values, computes its α and β values based on the U and L values of its generated children, and prunes states whenever $\beta \leq \alpha + \epsilon$. Note that $\epsilon = 0$ correspond to the standard alpha-beta algorithm.

Algorithm 1 lists the pseudo code of BAB. Lines 1–2 describe the case where n is a leaf node and hence it returns $V(n)$. Lines 3–10 initialize the L, U, α, and β values. In addition, we initialize $best_L$ and $best_U$ to v_{min} for MIN nodes and v_{max} for MAX nodes, respectively. These values keep track of the highest L and the lowest U in the generated children, are used later to update $L(n)$ and $U(n)$ (see lines 25-26). Lines 7-10 update α and β according to the type of n. Line 11 starts a transition over each child of the current node, and recursive call to BAB on that node is in line 12. Lines 13–21 updates the values of the relevant variables, and the actual pruning is done in line 24. Note that the computation of $L(n)$ and $U(n)$ for CHANCE nodes is done incrementally (lines 20-21) by adding the new $L(c)$ and $U(c)$ values and subtracting the initial v_{min} and v_{max} values.

Choosing a Bounded-Suboptimal Action BAB (Algorithm 1) returns a range $[L(n_1), U(n_1)]$ for a given game tree and ϵ. As proven in Theorem 1, the minimax value is in that range, and the size of the range is at most ϵ. Thus, any value chosen in that range can serve as a bounded-suboptimal solution for the given ϵ.

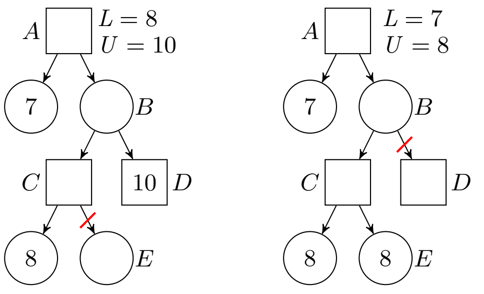
To obtain a concrete policy from the output of BAB the MAX player needs to choose the action that leads to the highest L values. This policy is guaranteed (on expectation) to provide at least a bounded-suboptimal minimax value, for any action the MIN player may choose. By contrast, choosing the action that leads to the highest U values may end up with a strategy whose expected value is smaller than the minimax value by more than ϵ.

Pruning Analysis Consider the amount of pruning that can be obtained when using BAB. Like Alpha-Beta and *-Minimax, in the worst case we still may need to generate all nodes in the game tree before halting, as long as $\epsilon < v_{\text{max}} - v_{\text{min}}$. Such a worst case occurs, for example, when the children of MAX nodes all lead to v_{min} except one node and the children of MIN nodes all lead to v_{max} except one node, and this individual node is always generated last.

In the best case, however, significant pruning can be achieved. We conjecture that the best case cannot be better than that of Alpha Beta, except for the case where $\epsilon = v_{\text{max}} - v_{\text{min}}$, in which only a single branch in the game tree will be expanded, but the resulting value is not helpful.

In general, we expect that BAB with ϵ_2 will prune more nodes than BAB with ϵ_1, if $\epsilon_2 > \epsilon_1$. We verify experimentally later in this paper that this occurs in practice in most cases. However, in some special cases, BAB with ϵ_2 actually prunes fewer nodes than BAB with ϵ_1.

Figure 3 demonstrate this phenomenon. The right tree represent an example of BAB with $\epsilon = 1$ and the left one represent BAB with $\epsilon = 2$. In both cases $v_{\text{min}} = 0$ and $v_{\text{max}} = 10$. In both cases, after we explored the left child of node A, we



(a) $\epsilon = 2$ (b) $\epsilon = 1$

Figure 3: Example of BAB pruning with different suboptimality values. Depending on the relative size of the subtrees rooted in D and E, $\epsilon = 2$ may lead to more or less pruning than $\epsilon = 1$.

update $\alpha = 7$ and $\beta = 10$. These values transmitted down to B and from there to C. After we explored the left child of node C, $\alpha = 8$ and $\beta = 10$. In this point, the left tree prune the right branch and the values that return to B are $(8, 10)$, we update the β value of node B to 10 and explore the right child. In the right tree, the right child of C does not get pruned and the values that return to B are $(8, 8)$, we update the β value of node B to 8 and the right child get pruned. Although these two examples explored the same amount of nodes, the branch of the right child of B and the branch of the right child of C could contain large sub-trees and hence a greater ϵ value might explore less, more or equal amount of nodes.

Experimental Results

In this section, we evaluate BAB experimentally. The main purpose of this evaluation is to measure the advantage and disadvantages of using our bounded pruning compared to the standard pruning provided by Alpha-Beta pruning and *-Minimax algorithm. We focus on two metrics: *Expanded Ratio (ER)* and *Actual Minimax Bound (AMB)*. *ER* is the amount of expansions divided by the amount of expansions done by the corresponding safe pruning method (i.e., BAB with $\epsilon = 0$). Smaller *ER* corresponds to a faster search. *AMB* is the absolute difference between the U value and the L value of the root. Smaller *AMB* means higher solution quality, as the bound around the true minimax value is smaller. Theorem 1 ensures that *AMB* is no more than ϵ.

Domains

We evaluated BAB on two domains—random trees (RT) and a cops and robbers (C&R) scenario (Moldenhauer and Sturtevant 2009). The games in the RT domain are randomly generated trees with a fixed depth and branching factor. We experimented with fixed depths of 3, 5, 7, and 9, and fixed branching factors of 2, 3, 4, and 5. The utility of leaves of these trees were drawn uniformly in the range $[0, 100]$.

In the C&R games, we randomly allocated n cops and one robber on a 4-connected 6×6 grid. The cops and the robber can move in their turn to one of their neighboring grid cells or stay in their current location. The robber is a MAX player and the cops are the MIN player, and all cops move simultaneously. The game ends when one of the cops reaches the robber. The score is the time until the robber is caught. In our experiments, we limited the depth of the C&R games, and used the average Manhattan distance between all cops to the robber as a heuristic for the leaf nodes. We experimented with fixed depth of 1, 3, 5, and 7.

For the experiments on games with chance, we modified both domains to include a stochastic element, as follows. In the stochastic RT games, after a player chooses a move the move occurs with probability p, and every other move occurs with probability $\frac{1-p}{b-1}$, where p is a parameter that we set to 0.8 and b is the branching factor. Here also, the utility of leaves of these trees were drawn uniformly in the range $[0, 100]$. The stochastic C&R game introduces stochasticity in the same manner: the chosen move is performed with probability p and each other moves with probability $\frac{1-p}{b-1}$. p we set to 0.8 in these experiments as well.

Results for Deterministic Games

Figure 4(a) shows the average *ER* as a function of ϵ for RT instances with branching factor of 4 and depths of 3, 5, 7, and 9. Each data point is an average over 50 instances. As expected, increasing ϵ decreases the *ER* value, as more pruning occurs and consequently less nodes are being expanded. In addition, the impact of our bounded pruning increases for larger game trees: running BAB with the same ϵ for deeper trees yielded a smaller *ER*. This occurs because increasing the size of the game tree creates larger sub-trees that can be exploited by BAB and get pruned.

Figure 4(b) shows the average *AMB* as a function of the ϵ over the same instances. As can be seen, for all depths, as we increase the ϵ value, the *AMB* is increased as well, but never beyond ϵ, as expected. For example for depth of 9, the *AMB* for $\epsilon = 0$ is zero, the *AMB* for $\epsilon = 32$ is 30.97, and the *AMB* for $\epsilon = 64$ is 61.88.

The impact of the depth of the tree on the average *AMB* is not as dramatic as on the average *ER*. Still, the *AMB* for deeper trees is clearly larger, for the same value of ϵ, than for shallower trees. For example for ϵ of 32, for depth of 3, 5, 7, and 9, the *ER* is 0.67, 0.37, 0.18, and 0.05, respectively. This trend can be explained by the fact that increasing the size of the tree creates more pruning opportunities, and consequently more opportunities to increase the minimax bound. Figure 4(c) shows the relation between the *ER* and *AMB* values. Here, we divided all instances with branching factor of 4 and depth of 5, into 5 bins. The first bin calculate the average over all instances with *ER* of 0 to 0.2, the second bin over *ER* of 0.2 to 0.4 and so on. It is easy to see that smaller *ER* correspond with larger *AMB*, as expected, extensive pruning causes smaller *ER* and wider gap between the values of the L and the U. Same trend was observed in all couples of branching factor and depth that we tested.

Figures 5(a) and (b) show the average *ER* and *AMB*, respectively, as a function of ϵ for 50 random instances of the C&R game on 6×6 open grid with 3 cops and depths of 3, 5, and 7. Here, $v_{\min} = 0$ and $v_{\max} = 12$. The cops can not catch the robber before $v_{\min} = 0$ because here the score is the time in which the cops catch the robber, and $v_{\max} = 12$ is the maximum manhattan distance of the grid. Here too, we see that increasing ϵ results in lower *ER* and higher *AMB*, as expected. Also, we see the same trend with the impact of the depth of the tree on the average *ER*, however the impact of the depth on the average *AMB* here is not as conclusive. 5(c) shows the same division of all instances with depth 3 into bins as in figure 4(c) and the same trend that observed in previous experiment.

Games with Chance

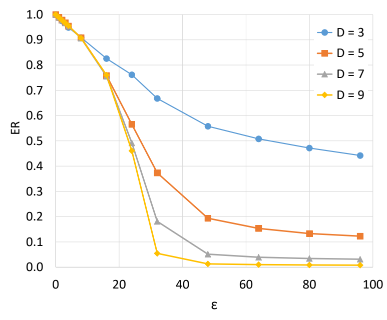
Figures 6(a) and (b) show the average *ER* and *AMB*, respectively, as a function of ϵ for 50 random instances of stochastic RT, for depth=4 and branching factor of 2, 3, 4, and 5. Again, the general trend in which higher ϵ corresponds to lower *ER* and higher *AMB* is clearly observed. For example for branching factor of 4, for ϵ of 0, 8, and 24, the *ER* is 1, 0.79, and 0.4, respectively, and the *AMB* is 0, 6.58, and 20.71, respectively. Note that for the specific case of $\epsilon = 80$ and branching factor=2 there is a slight increase in *ER* compared to $\epsilon = 60$. Such rare events are possible, as shown in Figure 3. Consider now the effect of increasing the branching factor. The results show that increasing the branching factor for the same ϵ are in general that higher branching factor corresponds to lower *ER* and higher *AMB*. The reason is similar to the explanation of how depth impacted *ER* and *AMB* in the deterministic case: larger branching factor corresponds to larger tree allowing more pruning opportunities. In fact, we experimented with different branching factor and tree depth in both deterministic and stochastic cases and observed these trends in both setting.

Finally, Figures 6(c) and (d) show the average *ER* and *AMB*, respectively, as a function of ϵ for 50 random instances of the C&R game with chance on 6x6 grid with 20% randomly allocated obstacles, 3 cops, $v_{\min} = 0$, $v_{\max} = 12$, and depth=3.[1] Here too we see the same general trends: higher ϵ yields lower *ER* and higher *AMB*. For example, for ϵ of 0, 4, and 8, the er is 1, 0.25, and 0.07, respectively, and the *AMB* is 0, 3.17, and 7.19, respectively.

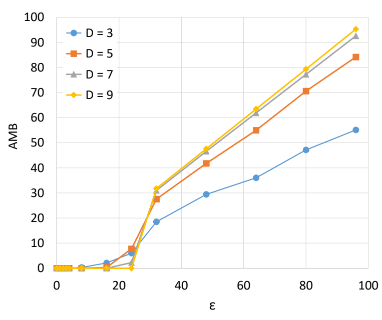
Related Work

There is a large volume of literature on Alpha-Beta and *-Minimax variants and improvements. Work on bounding the minimax value dates back to Pearl's SCOUT algorithm (1980) or even earlier. The main concept used by most such methods is to initialize α and β by some values that are different from $v_{\min}$ and $v_{\max}$, providing stronger pruning. The range between these initial values is called the Alpha Beta window. If the minimax value is not withing the Alpha Beta window, the search fails. Some algorithms, e.g., Nega Scout (Reinefeld 1983), follow with this approach and perform multiple calls to Alpha Beta with different windows. In general, these algorithms either find the minimax value or fail (high or low). By contrast, a bounded-suboptimal algorithm finds a solution that is within ϵ of the minimax value.

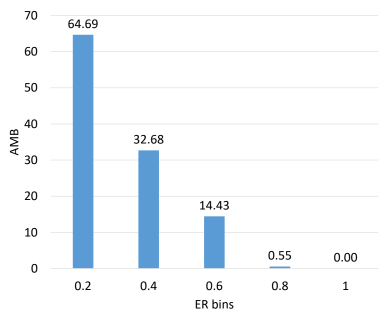
[1]We experimented with a wide range of parameters and settings, and show this setting as a representative example.



(a) *ER* as a function on ϵ

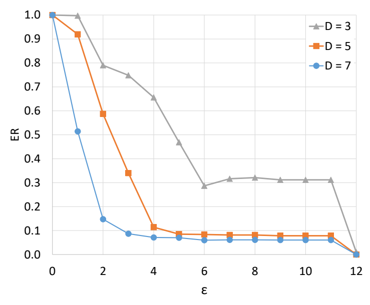


(b) *AMB* as a function on ϵ

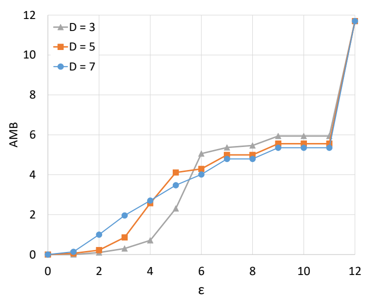


(c) *AMB* as a function on *ER* bins.

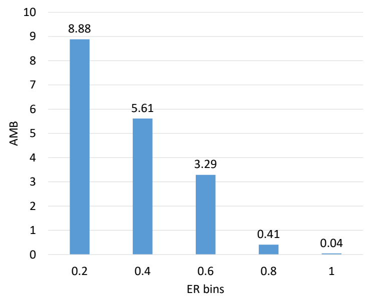
Figure 4: Experimental results for the deterministic RT domain.



(a) *ER* as a function on ϵ



(b) *AMB* as a function on ϵ



(c) *AMB* as a function on *ER* bins.

Figure 5: Experimental results for the deterministic C&R domain.

More recent work also studied improvements for *-Minimax for games with chance, including using transposition tables (Veness and Blair 2007), and performing forward pruning (Schadd, Winands, and Uiterwijk 2009). While the latter also does pruning as we do, it does not provide the suboptimality guarantee that we do.

Conclusion, Discussion, and Future Work

In this work, we proposed the concept of a bounded-suboptimal algorithm for game tree search, as a way for approximately solving two player zero-sum games with full observability. The theoretical framework for building such algorithms is given for games with and without chance, and a concrete bounded-suboptimal algorithm is presented called Bounded Alpha-Beta (BAB). Experimental results on two domains confirm that BAB can tradeoff solution quality for runtime effectively.

This work opens a wide range of directions for future work on bounded suboptimal game tree search.

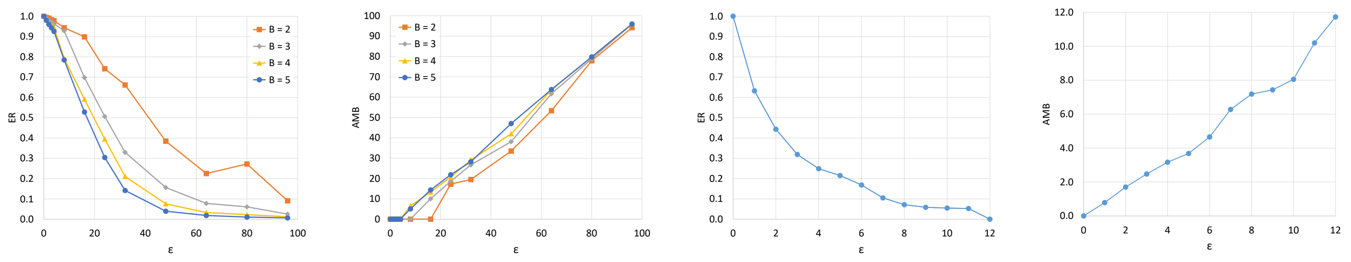
Different game tree search strategies. Our algorithm is based on classical Alpha-Beta and its extension to games of chance, but Theorem 1 applies directly to any other game tree search algorithms, including Monte Carlo Tree Search solver (Lanctot et al. 2013) and Proof Number Search (Allis, van der Meulen, and Van Den Herik 1994). Exploring how suboptimal-bounding affects their behavior.

Dynamic suboptimality bound. A deeper look on the proof of Theorem 1 reveals that it is possible to have more flexibility in how ϵ is distributed between the children of a CHANCE node. Instead of passing $\epsilon(n)$ to each child node c, it is possible to provide different values of ϵ to different children of n. Let $\epsilon(c)$ be the ϵ passed to a child node c in line 12 in Algorithm 1. The following relation between $\epsilon(n)$ and $\epsilon(c)$ is sufficient to maintain the desired suboptimality.

$$\sum_{c\in C(n)} \epsilon(c)\pi(c) \leq \epsilon(n) \qquad (26)$$

Future work will investigate how to distribute ϵ to the children of CHANCE nodes according to Eq. (26) to allow further pruning and faster search.

Game playing with BAB. We performed a very preliminary set of experiments to investigate how a player using BAB with $\epsilon > 0$ fares against Alpha Beta, which is equivalent to BAB with $\epsilon = 0$. We experimented on the deterministic version of C&R with 3 cops and a 6×6 grid and ϵ values of 0, 1, 2, 4, and 8. We limited the number of node expansions allowed in each step to 200,000, and had each algorithm run in an iterative deepening manner until the the total number of node expansions exceeded 200,000, choosing the best move according to the last iteration. Each instance played twice, once with each starting player, and a player is determined as a winner if its average score over the two games is greater than the other's. The results showed that BAB with $\epsilon = 1$ won 11 times, lost 6 times, and had a draw 33 times, but



(a) *ER* as a function on ϵ (b) *AMB* as a function on ϵ (c) *ER* as a function on ϵ (d) *AMB* as a function on ϵ

Figure 6: Experiments on domains with chance.

BAB with $\epsilon = 2$ it won 6 times, lost 12 times, and had a draw 32 times. This shows some potential for BAB but further research is needed.

ϵ-equilibrium. The optimal policy for MAX and MIN players that use classical Alpha-Beta pruning is known to be a Nash equilibruim, that is, no player can gain by unilaterally deviating from its action. Obviously, a the policy proposed in ”The Bounded Alpha-Beta Algorithm” section obtained by BAB in not necessarily a Nash equilibruim. However, we conjucture that this policy reaches an ϵ-equilibrium (Nisan et al. 2007), which is a joint policy in which no player can gain more than ϵ by unilaterally deviating from his strategy. Proving this conjucture is a topic for future research.

Acknowledgements

This research was supported by the Israel Ministry of Science, the Czech Ministry of Education, and by ISF grant #210/17 to Roni Stern.

References

Allis, L. V.; van der Meulen, M.; and Van Den Herik, H. J. 1994. Proof-number search. *Artificial Intelligence* 66(1):91–124.

Ballard, B. W. 1983. The *-minimax search procedure for trees containing chance nodes. *Artificial Intelligence* 21(3):327–350.

Coulom, R. 2006. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, 72–83.

Gilon, D.; Felner, A.; and Stern, R. 2016. Dynamic potential search—a new bounded suboptimal search. In *Symposium on Combinatorial Search (SoCS)*.

Hauk, T.; Buro, M.; and Schaeffer, J. 2004. Rediscovering *-minimax search. In *International Conference on Computers and Games*, 35–50.

Irving, G.; Donkers, J.; and Uiterwijk, J. 2000. Solving kalah. *ICGA Journal* 23(3):139–147.

Knuth, D. E., and Moore, R. W. 1975. An analysis of alpha-beta pruning. *Artificial intelligence* 6(4):293–326.

Kocsis, L., and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*, 282–293.

Lanctot, M.; Saffidine, A.; Veness, J.; Archibald, C.; and Winands, M. H. 2013. Monte Carlo *-minimax search. In *IJCAI*, 580–586.

Michie, D. 1966. Game-playing and game-learning automata. In *Advances in programming and non-numerical computation*. 183–200.

Moldenhauer, C., and Sturtevant, N. R. 2009. Evaluating strategies for running from the cops. In *IJCAI*, volume 9, 584–589.

Nisan, N.; Roughgarden, T.; Tardos, E.; and Vazirani, V. V. 2007. *Algorithmic game theory*. Cambridge university press.

Pearl, J. 1980. Scout: A simple game-searching algorithm with proven optimal properties. In *AAAI*, 143–145.

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *Artificial Intelligence* 1(3-4):193–204.

Reinefeld, A. 1983. An improvement of the scout tree-search algorithm. *ICCA Journal* 6(4):4–14.

Russell, S. J., and Norvig, P. 2016. *Artificial intelligence: a modern approach*. Pearson.

Saffidine, A.; Jouandeau, N.; and Cazenave, T. 2011. Solving breakthrough with race patterns and job-level proof number search. In *Advances in Computer Games*, 196–207.

Schadd, M. P.; Winands, M. H.; and Uiterwijk, J. W. 2009. Chanceprobcut: Forward pruning in chance nodes. In *IEEE Symposium on Computational Intelligence and Games (CIG)*, 178–185.

Schaeffer, J., and Plaat, A. 1996. New advances in alpha-beta searching. In *ACM conference on Computer science*, 124–130.

Schaeffer, J.; Burch, N.; Björnsson, Y.; Kishimoto, A.; Müller, M.; Lake, R.; Lu, P.; and Sutphen, S. 2007. Checkers is solved. *science* 317(5844):1518–1522.

Thayer, J. T., and Ruml, W. 2011. Bounded suboptimal search: A direct approach using inadmissible estimates. In *IJCAI*, volume 2011, 674–679.

Veness, J., and Blair, A. 2007. Effective use of transposition tables in stochastic game tree search. In *IEEE Symposium on Computational Intelligence and Games (CIG)*, 112–116.

Wald, A. 1945. Statistical decision functions which minimize the maximum risk. *Annals of Mathematics* 265–280.