A theory of game trees, based on solution trees

Wim Pijls, Arie de Bruin, Aske Plaat
Erasmus University Rotterdam, P.O.Box 1738, 3000 DR Rotterdam,
The Netherlands, {*pijls,adebruin,plaat*}*@few.eur.nl*

Abstract

In this paper a complete theory of game tree algorithms is presented, entirely based upon the notion of a solution tree. Two types of solution trees are distinguished: max and min solution trees respctively. We show that most game tree algorithms construct a superposition of a max and a min solution tree. Moreover, we formulate a general cut-off criterion in terms of solution trees. In the second half of this paper four well known algorithms, viz., alphabeta, SSS*, MTD and Scout are studied extensively. We show how solution trees feature in these algorithms and how the cut-off criterion is applied.
Keywords: Game tree search, Minimax search, Solution trees, Alpha-beta, SSS*, MTD, (Nega)Scout.

1 Introduction

A game tree models the behavior of a two-player game. The nodes in such a tree represent positions of a game, whereas edges represent moves. Given a payoff in the leaves of a game tree, the best move in each position can be determined by means of the so-called *minimax* function. Over the years, many algorithms have been designed, which determine the so-called minimax value of a game tree, given a payoff value in the leaves. Although the minimax function is defined using all nodes of the tree, every algorithm tries to inspect a number of nodes as small as possible. The computation of the minimax function has also theoretical interest, because this problem is known to be PSPACE-complete. In this paper, it turns out that the notion of a *solution tree* is a key notion in the theory of game tree algorithms. There are two types of solution trees: max and min solution trees respectively. First, we show that every algorithm computing the minimax value of a game tree constructs a max and a min solution tree. Next, we present a general cutoff criterion for searching game trees. This criterion will be explained in terms of solution trees. In the remainder of this paper, we discuss the role of solution trees in four well-known algorithms, viz. alphabeta, SSS*, MTD and Scout.
We deal only with fixed depth algorithms. In practice, mostly iterative deepening is used. But even with iterative deepening, at one given level, still a fixed depth method is needed.

This paper is organized as follows. The sections 2 and 3 discuss basic concepts. In Section 2, the concept of a solution tree is introduced, and its relation to the game tree value is explained. In Section 3, we show that there is a one-to-one correspondence between a solution tree and a bound to a game value. An important result in this section is, that almost every algorithm must construct a superposition of a max and a min solution tree. In the sections 4 and 5 a general cut-off criterion, used in almost every game tree algorithm, is presented. In the subsequent sections, we focus on four major algorithms, and the interest of solution trees therein. In section 6 the alphabeta procedure and the so-called *global alphabeta* algorithm are studied. In section 7, the working of the complex SSS* is explained. Section 8 presents a framework of algorithms, called *MTD*, closely related to the alphabeta procedure. In Section 9, we discuss some instances of this framework. One instance is MTD(f), which has turned out to be very powerful. [24]. Another one is AB-SSS, which has been proven equivalent to SSS*. Section 9 addresses the Scout algorithm.

We now give some preliminaries. A game tree is denoted by G; its root is mostly called r. The two game players are Max and Min. We assume that Max plays the opening move. As said before, the algorithms to be studied here, aim at computing the minimax function f, which is defined on every game tree. The value $f(r)$, the minimax value in the root, is written alternatively as $f(G)$.
For any tree R (for example $R = G$) containing a node n, $R(n)$ stands for the subtree in R that is rooted in n. The set of children of a node n is denoted by $C(n)$. Every node is regarded to be one of its own descendants.

2 Solution trees

The purpose of the minimax function
We first recall the definition of the well-known minimax function and its raison d'etre. In every terminal p of a game tree, a function value $f(p)$ is assumed to be defined, representing the profit or the payoff for Max, or equivalently, the loss for Min. Max aims at maximizing this payoff, whereas Min aims at minimizing it. Given an arbitrary node n, we are interested in the guaranteed payoff for the Max player, i.e., the highest attainable value for the Max player, under the condition that Min minimizes Max's profit. This guaranteed payoff is determined by the familiar minimax function f. The definition of this function is based on the idea that the guaranteed payoff for Max in a max node is equal to the highest payoff among the children. To achieve this payoff, Max must move to a child with maximal payoff. In a min node the highest attainable payoff for Max is given by the minimum value among the children, assuming optimal play of min.

Strategies or solution trees
To gain more insight into the minimax function with related properties, we consider the concept of a *strategy*. This notion is equivalent to the notion of a solution tree, introduced in [28]. The idea of viewing a solution tree as a *strategy* originates from [11]. A strategy of Max consists of a subtree, including in each Max position exactly one continuation and in each min node all continuations (all countermoves to Max). Since the choice of Max in each position is known in a max strategy, Min is able to calculate the outcome for each series of choices

that he can make. In this paper, a subtree with exactly one child in an internal max node and all children in a min node, which we called a Max strategy above, will be also referred to as a min solution tree, or briefly, a min tree. Dually a max tree or a MIn strategy can be defined.

Definition 2.1 *A max solution tree T^+ is a subtree of a game tree G with the properties:*

- *if an inner max node $n \in G$ is included in T^+, then also all children of n are included in T^+;*
- *if an inner min node $n \in G$ is included in T^+, then exactly one child is included in T^+.*

A min solution tree T^- is a subtree of G with the properties:

- *if an inner min node $n \in G$ is included in T^-, then all children of n are included in T^-;*
- *if an inner max node $n \in G$ is included in T^-, then exactly one child is included in T^-.*

For clarity, we emphasize that the root of a solution tree T is not necessarily the root of G.
Given a min solution tree, the most beneficial choice for Min in each min position is a move towards a terminal with minimal value. Consequently, in a given Max strategy T, the profit of Max under optimal play of Min is equal to the minimum of all payoff values of T. Therefore, we define a function g on the set of solution trees. For a min solution tree T^-, $g(T^-)$ is defined as the minimal value in the terminals of T^-. On a max solution tree, a dual definition holds. Alternatively, on either solution tree type, g can be viewed as the minimax-value of the solution tree, where the computation of the minimax function is restricted to the solution tree.

Definition 2.2 *For a given max tree T^+ and a min tree T^-, the g-value is defined as:*

$$g(T^+) = \max\{f(p) \mid p \textit{ is a terminal in } T^+\}$$
$$g(T^-) = \min\{f(p) \mid p \textit{ is a terminal in } T^-\}$$

In Figure 1, an example of a game tree is shown labeled with its f-values. The bold lines in this figure generate a max solution tree.

An optimal solution tree.
In the foregoing, we have seen that in every max node of a game tree G, the most beneficial move of Max is the transition to a child with the highest f-value. From this fact, we draw the following conclusion. A min solution tree T^-, that contains in each max node the best move of Max in G, is the most profitable strategy for Max. This entails that T^- has a g-value equal to $f(n)$ and every other Max strategy T gives a payoff for Max at most equal to $f(n)$. This property together with its counterpart for min solution trees is expressed formally in the following theorem, which will be referred to as Stockman's theorem. [28].

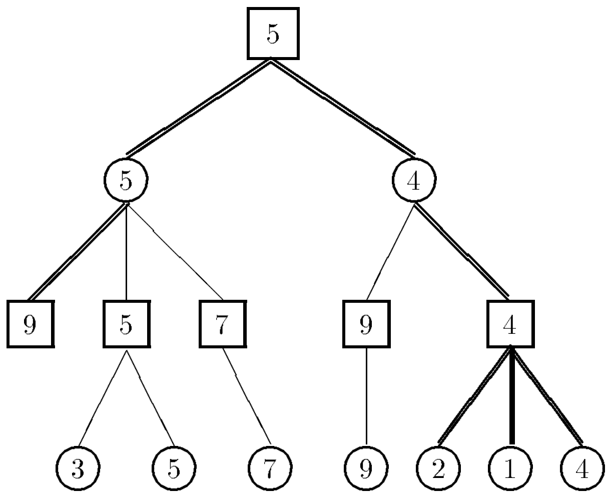


Figure 1: A max solution tree in a game tree with f-values.

Theorem 2.1 *Let n be a node in a game tree G.*

$$f(n) = \min\{g(T^+) \mid T^+ \text{ is a max solution tree with root } n\} \tag{2.1}$$
$$= \max\{g(T^-) \mid T^- \text{ is a min solution tree with root } n\} \tag{2.2}$$

In the remainder of this paper we will speak about the max (solution) tree and min (solution) tree version of Stockman's theorem. A formal proof of the theorem uses induction on the height of n. A solution tree T with $g(T) = f(G)$ is called an *optimal* solution tree. Notice that, when constructing an optimal strategy, it is not necessary to choose an optimal move in every node. Consider for example a max node n with exactly two children, say c_1 and c_2, such that $f(c_1) \leq f(c_2) = f(n)$. Then every optimal max solution tree T^+ has $g(T^+(c_2)) = f(c_2)$ and $f(c_1) \leq g(T^+(c_1)) \leq f(c_2)$. The continuation from c_1 in T^+ needs not to be optimal.

Critical tree

The union of an optimal max and an optimal min solution tree is called a *critical* tree. As soon as a critical tree is obtained, the game value of the root is established and the algorithm may stop.

In [13] the notion of a critical tree is introduced as a minimal tree that has to be searched by alpha-beta in order to find the minimax value in a best first game tree. In that paper, the set of nodes in a critical tree is divided into three subsets. Viewing a critical tree as a superposition of two solution trees, we can give an alternative definition for each of the three types, replacing the (quite complicated) definition in [13]. Type 1 nodes are in the intersection of the optimal solution

trees for the player and its opponent—the critical path. Outside the critical path, there are type 2 and type 3 nodes. Type 2 nodes are either min nodes of the max solution tree or max nodes of the min solution tree. Type 3 nodes are max nodes in the max solution tree or min nodes in the min solution tree.
The observation that a minimal set of nodes, to be visited by any game tree algorithm, consists of the superposition of two strategies was made before in [18].

3 The search tree

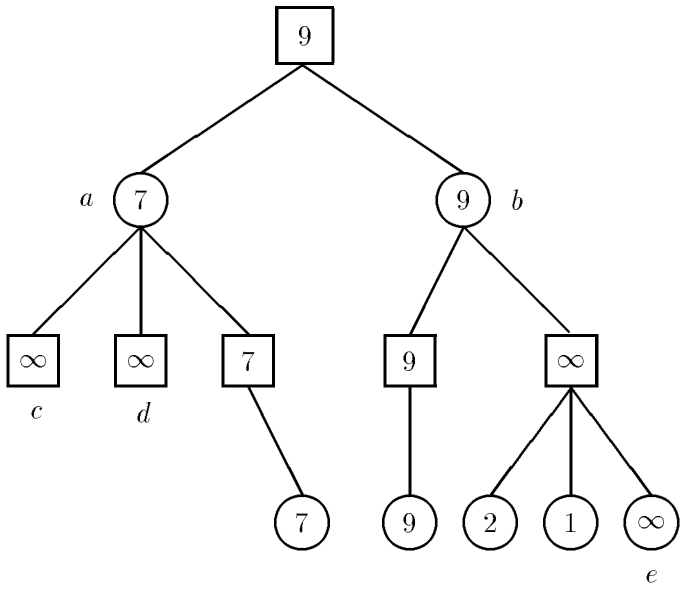
Definition of a search tree
First of all, we recapitulate the notion of a *search tree*, introduced in [10]. In all game tree algorithms, the game tree is explored step by step. So, at each moment during execution of a game tree algorithm, a subtree has been visited. This subtree of the game tree is called the *search tree*. We assume that, as soon as at least one child of a node n is generated or visited, all other children of n are also added to the search tree. Hence, a search tree S has the property, that for every node $n \in S$ either all children are included in S or none. An interior node n of a game tree, whose children have been generated during execution of the game tree algorithm under consideration, is called *expanded*. Hence, an interior node in the search tree is always expanded. If an interior node n of game tree is a leaf of the search tree, or, in other words, if the children of n have not been generated yet, then n is called *open*. A terminal in a game tree is necessarily a leaf in the search tree, but is not always called open. A terminal is called *open* or *expanded*, according whether its game value has been computed or not. In summary, *expanding* an open node n consists of appending the children to n, if it is a non-terminal, or computing the payoff of n, if it a terminal. Instead of *expanding* a terminal, we will also speak of *closing* a terminal.

Bounds to the game value
In the *open* nodes of a search tree we can define two provisional payoff values. The most optimistic payoff for Max is equal to $+\infty$, the most pessimistic $-\infty$. By introducing these provisional values, we also introduce two game trees S^+ and S^-, which are derived from S. In the open nodes p of S^+, $f^+(p) = +\infty$ by definition, and in the inner nodes of S^+, f^+ is determined according to the minimax function. For f^+ and f^- as provisional optimistic and pessimistic payoff values, we have

$$f^-(n) \leq f(n) \leq f^+(n)$$

A formal proof is by induction on the height. We see that f^+ and f^- are bounds to the game value.
An example of a search tree, related to the game tree of Figure 1 can be found in Figure 2, where c, d and e are open nodes. In the trees of Figure 2, the f^+- and f^--values respectively are shown.

The relation between bounds and solution trees
Like a game tree, a search tree S also contains solution trees. The leaves of such a solution tree are leaves in S. For each type (min and max) of solution trees in



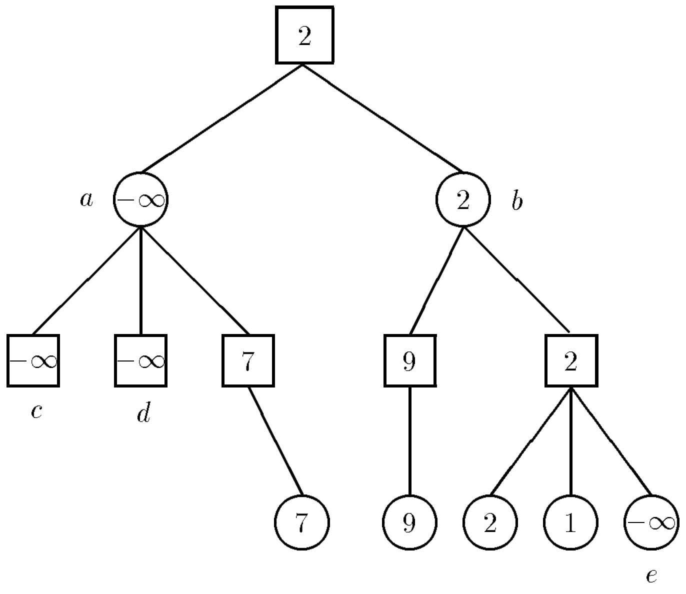


Figure 2: A search tree with f^+- (top) and f^-- (bottom) values.

S, we have a division into two subtypes.

Definition 3.1 *A solution tree T in S is called open, if T has at least one open leaf in S. A solution tree T is called closed, if T has solely closed leaves.*

Notice that a closed solution tree in a search tree S is also a solution tree in the entire game tree.
Stockman's theorem can be invoked in S^+ as well as in S^-. First, the max tree version of Stockmans's theorem, expressed in (2.2), is applied to S^+. This gives the statement, that $f^+(n)$ is equal to the minimum of all values $g(T)$ with g computed in S^+ and T a max solution tree in S. Every open max tree in S^+ has g-value equal to $+\infty$. Therefore, when applying the max tree version to S^+, we only need to take closed max solution trees into account. This results into equality (3.1). The second equality (3.2) is obtained, when the min tree version of Stockman's theorem is applied to S^-.

$$f^+(x) = \min\{g(T) \mid T \text{ \textit{is a closed max solution tree with root} } x\} \tag{3.1}$$

$$f^-(x) = \max\{g(T) \mid T \text{ \textit{is a closed min solution tree with root} } x\} \tag{3.2}$$

We immediately see that $f^+(n)$ has a finite value if and only if n is the root of closed max tree T^+. So, to every finite value $f^+(n)$ a max solution tree T^+ is associated with $g(T^+) = f^+(n)$.

An obvious termination criterion

Almost every game algorithm builds a search tree and stops when the equality $f^+(r) = f^-(r)$ is achieved. At that time, there are optimal solution trees T^+ and T^- satisfying $g(T^+) = g(T^-) = f(G)$. A union of an optimal max and an optimal min solution tree, has been called a critical tree in the Section 2. We conclude that, in order to compute the game value at the root, a critical tree is needed.

If we are interested primarily in the best move from r instead of the value of r, an algorithm may stop without constructing a critical tree. For any search tree, for at least one child c_0: $f^-(r) = f^-(c_0)$. If moreover $f^+(c) \le f^-(c_0)$ for every other child $c \ne c_0$, and hence, $f^+(c_0) = f^+(r)$, then c_0 is guaranteed to be the best continuation. However, in this situation, the equality $f^+(c_0) = f^-(c_0)$ or equivalently $f^-(r) = f^+(r)$, does not hold necessarily. Here, we have a search tree, where the best move is found, but not the game value.

The intersecting path in a critical tree

A max and a min tree always have exactly one path in common. We focus on the intersecting path of a pair of optimal trees Let two trees be given with $g(T^+) = f(G) = g(T^-)$. Let $n_0 = r, n_1, \ldots n_k$ be the nodes of the intersection path of two optimal tree T^+ and T^-. In T^+ and T^- respectively, we have:

$$f^+(n_i) \le g(T^+(n_i)) \le g(T^+), \quad i = 0, 1, \ldots k, \tag{3.3}$$

$$f^-(n_i) \ge g(T^-((n_i)) \ge g(T^-), \quad i = 0, 1, \ldots k. \tag{3.4}$$

Since $g(T^+) = f(G) = g(T^-)$, also $f^+(n_i) = f^-(n_i) = f(G)$ for $i = 0, 1, \ldots k$. We conclude that the intersection of T^+ and T^- is a path with $f(n_i) = f(n)$.

A path in a game tree from the root r to a terminal such that the f-value is constant on this path (and thus $= f(r)$) is called a *critical path.*

4 Open solution trees

Speaking about solution trees in Section 3, we distinguished between open and closed solution trees. We mentioned, that every open max solution tree T has at least one leaf p with $f^+(p) = +\infty$ and hence $g(T) = +\infty$. Such a solution tree can be given another provisional value, defined as the maximum of the game values of the closed leaves. Following the notation in [11], this value is denoted by $c(T)$ for a max solution tree T. So a new function on the set of max solution trees is defined. Of course, this function has a dual counterpart.

Definition 4.1 *For a max tree T^+ and a min tree T^- in a search tree S, a function c is defined as:*

$$\begin{aligned} c(T^+) &= \max\{f(p) \mid p \ \ \textit{a closed terminal in } T^+\}, \\ c(T^-) &= \min\{f(p) \mid p \ \ \textit{a closed terminal in } T^-\}. \end{aligned}$$

Obviously, if T is a closed (max or min) solution tree, then $c(T) = g(T)$. The c-value of an open max solution tree T^+ can be viewed as the minimax value of T^+ with $f(p) = -\infty$ in the open nodes p of T^+, i.e., as the minimax value of T^+ in S^-. In Section 3, we invoked the max tree version of Stockman's theorem in S^+, see (3.1). (Recall that the minimax value of an open max solution tree in S^+ is $= +\infty$.) Stockman's max tree version applied to S^-, says that the smallest c-value of the max solution trees with root x is equal to $f^-(x)$. This result, extended to its dual counterpart, is expressed by the formulas:

$$\begin{aligned} f^-(x) &= \min\{c(T^+) \mid T^+ \ \textit{max solution tree in } S \textit{ with root } x\} && (4.1) \\ f^+(x) &= \max\{c(T^-) \mid T^- \ \textit{min solution tree in } S \textit{ with root } x\} && (4.2) \end{aligned}$$

We stated at the end of Section 3, that any game tree algorithm must visit at least a critical tree, defined as the superposition of an optimal max and an optimal min solution tree. Consider the following problem: *given a node n in a search tree S, does an extension of S to a game tree $\bar{S}$ exist, such that n is in a critical tree?* Suppose S is a search tree for a game tree $G = \bar{S}$ and a node n in S belongs to an optimal max solution tree $\bar{T}^+$ of $\bar{S}$. Let T^+ denote the subtree of $\bar{T}^+$ included in S, in other words T^+ is the intersection of $\bar{T}^+$ and S. Since T^+ is a subtree of $\bar{T}^+$, $g(\bar{T}^+) \geq c(T^+)$. It follows that, if a node n in a search tree S becomes into an optimal max solution tree when extending S to a full game tree, the corresponding optimal g-value is larger than or equal to:

$$\min\{c(T^+) \mid T^+ \ \textit{a max solution tree in } S \textit{ through } r \textit{ and } n\}$$

Of course, this quantity has a dual form. For these quantities, we will introduce a special notation. In the remainder of this section, we will derive some properties of them In the next section, they will be used to infer a general cutoff criterion for game tree algorithms.

Definition 4.2 *Given a search tree S with root r and a node n included. The set of all max solution trees in S through r and n is denoted by $\mathcal{M}(n)$ and the set of all min solution trees in S through r and n will be denoted by $\mathcal{N}(n)$.*

Definition 4.3

$$h^-(n) = \min\{c(T^+) \mid T^+ \in \mathcal{M}(n)\} \tag{4.3}$$

$$h^+(n) = \max\{c(T^-) \mid T^- \in \mathcal{N}(n)\} \tag{4.4}$$

Later on in this section, we will give a simple formula to compute the h-functions. The h-functions will play an essential role in the cut off criterion for nodes, to be presented in Section 5.

Theorem 4.1 *Let a game tree G with root r including a node n be given.*

$$\begin{aligned} h^-(n) &= \max\{f^-(x) \mid x = n \vee x \in ANC(n)\} && (4.5)\\ h^+(n) &= \min\{f^+(x) \mid x = n \vee x \in ANC(n)\} && (4.6) \end{aligned}$$

Proof
Only the first equality is proved.
Let the path from r to n be given by $r = n_0, n_1, \ldots, n = n_k$. We give a proof by induction along the path.
Basic step. The maximal c-value of the max trees in $\mathcal{M}(r)$ is equal to $f^-(r)$, according to (4.1). Hence, (4.5) is correct for $r = n_0$
Induction step. Suppose that n_j is a max node. An optimal tree in $\mathcal{M}(n_j)$ is also optimal in $\mathcal{M}(n_{j+1})$. Since n_j is a max node, $f(n_{j+1}) \leq f(n_j)$. It follows that (4.5) is correct for $n = n_{j+1}$, provided that it is correct for $n = n_j$.
Suppose that n_j is a min node. If $f^-(n_{j+1}) \leq h^-(n_j)$, then, according to (4.1), n_{j+1} is the root of a max solution tree with value $= f^-(n_{j+1}) \leq h^-(n_j)$. Consequently, there is an optimal tree in $\mathcal{M}(n_j)$, which also crosses n_{j+1}. Alternately, if $f(n_{j+1}) > h^-(n_j)$, then every max solution tree rooted in n_{j+1} has c-value $\geq f^-(n_{j+1}) > h^-(n_j)$. Consequently, an optimal tree in $\mathcal{M}(n_j)$ does not cross n_{j+1}. The optimal c-value in $\mathcal{M}(n_{j+1})$ is equal to $f^-(n_{j+1})$. Combining both relations between $f^-(n_{j+1})$ and $h^-(n_j)$, we see that the optimal c-value in $\mathcal{M}(n_{j+1})$ is equal to $\max(f^-(n_{j+1}), h^-(n_j))$. It follows that (4.5) is correct for $n = n_{j+1}$, provided that it is correct for $n = n_j$. $\square$

There are yet other formulas for the h-functions. For these formulas, we need new sets, associated with a node n in a game tree. In Figure 3, these sets are introduced using a picture. The set $AMAX(n)$ is the set of max nodes in $ANC(n)$. The set $AMAX\text{-}C(n)$ is the set of nodes, whose father is in $AMAX(n)$, but which are not ancestors of n. Similar definitions hold for $AMIN(n)$ and $AMIN\text{-}C(n)$ respectively.
Later on, we will also use the sets $AMAX\text{-}C\text{-}L(n)$ and $AMAX\text{-}C\text{-}L(n)$, splitting the set $AMAX\text{-}C(n)$. The set $AMAX\text{-}C\text{-}L(n)$ contains the nodes of $AMAX\text{-}C(n)$ left to n and $AMAX\text{-}C\text{-}R(n)$ contains the nodes right to n.

Every min node $m \in ANC(n)$ has a descendant m', such that $f^-(m') \geq f^-(m)$ and $m' = n$ or m' is a max ancestor of n. It follows, that the maximum in (4.5)

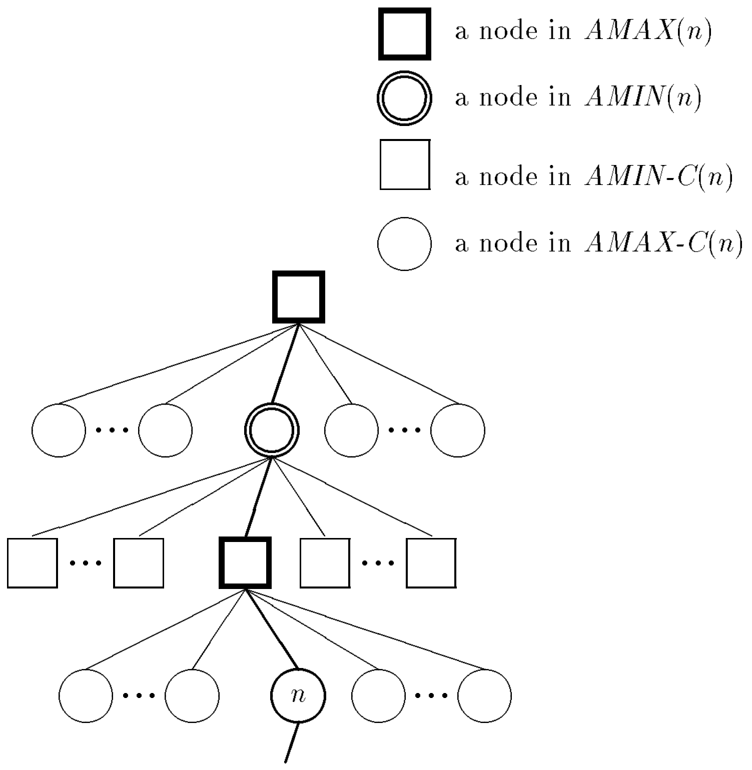


Figure 3: Definition of node sets.

is not achieved in a min node, and consequently, the min nodes can be removed from the set $ANC(n)$ in the right-hand side of (4.5).

$$h^-(n) = \max\{f^-(x) \mid x = n \vee (x \in AMAX(n)\} \tag{4.7}$$
$$h^+(n) = \min\{f^+(x) \mid x = n \vee x \in AMIN(n)\} \tag{4.8}$$

In general for a max node x, $f^-(x)$ is the maximum of all values $f^-(c)$, c a child of x. Replacing the max nodes by their children in (4.7) and removing min ancestors of n yields the following formula for $h^-(n)$, which has also a dual counterpart:

$$h^-(n) = \max\{f^-(x) \mid x = n \vee x \in AMAX\text{-}C(n)\} \tag{4.9}$$
$$h^+(n) = \min\{f^+(x) \mid x = n \vee x \in AMIN\text{-}C(n)\} \tag{4.10}$$

The proof is left to the reader.
As said in Section1, our definition of a game tree does not require the alternation of a max and a min node in any path.

5 A general cut off criterion

In this section, we will formulate a general cut off criterion, to be expressed by Theorem 5.1. In the subsequent sections, we will investigate, how far the best known algorithms meet this criterion. It is important to recall (cf. Section 1) that every node is its own descendant.

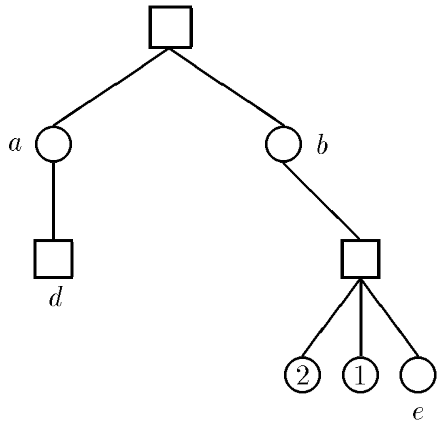
Lemma 5.1 *Suppose n is a node with $h^-(n) < h^+(n)$ in a search tree S. Let P denote the set of open descendants of n in S. Then there is a game tree $G \supset S$, in which every critical tree of G includes at least one node from P.*

Proof
There are trees $T^+ \in \mathcal{M}(n)$ and $T^- \in \mathcal{N}(n)$ with $c(T+) = h^-(n) < h^+(n) = c(T^-)$. Let p_0 denote the leaf at the end of the common path. Since n belongs to both solution trees, p_0 is a descendant of n. We first argue that p_0 is open in S. If p_0 was closed, we would have $f(p_0) \leq c(T^+)$ and, dually $f(p_0) \geq c(T^-)$, which relations contradict the inequality $c(T+)) < c(T^-)$. Therefore, we conclude that p_0 is open, and hence $p_0 \in P$.
Choose a value f_0 with $c(T^+) \leq f_0 < c(T^-)$. (An alternate choice is $c(T^+) < f_0 \leq c(T^-)$). We construct G by choosing $f(p_0) = f_0$ and $f(p) \leq f_0$ for any open node $p \neq p_0$ in T^+ and $f(p) > f_0$ for any open node $p \neq p_0$ in T^-. As far as open nodes are left in the extended search tree, they are closed arbitrarily. The extended solution trees $\bar{T}^+$ and $\bar{T}^-$ both have a g-value equal to f_0 and are optimal therefore.
We now show that every optimal max tree of G includes p_0. Notice, that $\bar{T}^-$ has, apart from p_0, solely terminals with f-value $> f_0$. Every node in *AMIN-C*(p_0) is included in $\bar{T}^-$. Since every node $m \in$*AMIN-C*(p_0) has solely descendant terminals with f-value $> f_0$ in $\bar{T}^-$, we have $g(\bar{T}^-(m)) > f_0$ and consequently $f(m) > f_0$ in G. If a max tree T' in G crosses a node $m \in$ *AMIN-C*(p_0), its g-value satisfies $g(T') \geq g(T'(m)) \geq f(m) > f_0$. Since any optimal max solution tree in G has g-value $= f_0$, it cannot cross any node $m \in$*AMIN-C*(p_0). We conclude that every optimal max tree contains the path from r to p_0. (In case of the alternate choice $c(T^+) < f_0 \leq c(T^-)$, we can construct a game tree, where every optimal *min* solution tree contains the path from r to p_0.) $\square$

This lemma is illustrated using the Figures 2 and 4. The latter figure shows some solution trees embedded in the search tree of the former figure. One can derive easily from Figure 2, that $h^-(a) = 2$ and $h^+(a) = 7$. According to the proof Lemma 5.1, it is possible to construct a game tree G with game value $f_0 \in [2, 7)$, such that every optimal max tree crosses a. We have to consider the max tree and the left min tree. Applying the method, indicated in that proof, with $f_0 = 2$ we define $f(d) = 2$, $f(e) \leq 2$ and $f(c) > 2$. It is also possible to get node b into a critical tree. Since $h^-(b) = 2$ and $h^+(b) = 9$, a game tree G' can be built with game value $f_0 \in (2, 9]$, such that b is included is every min tree of G'. To that end, the max tree and the right min tree in Figure 4 have to be considered. According to the proof of Lemma 5.1, we define $f(e) = f_0$ and $f(d) < f_0$.

Lemma 5.2 *Suppose n is a node with $h^-(n) \geq h^+(n)$ in a search tree S. Let P denote the set of open descendants of n in S. Then every game tree $G \supset S$ contains at least one critical tree without any node in P.*

Proof
Let G be any game tree enclosing S. We are going to construct a critical tree consisting of optimal trees T_0^+ and T_0^-, that haven't any node in P.
We choose arbitrarily an optimal max and an optimal min tree in G. These are



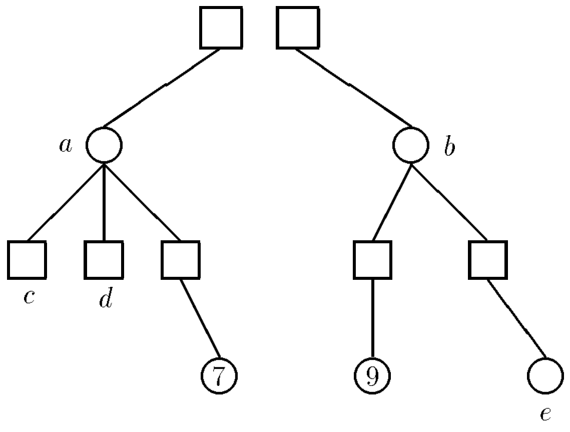


Figure 4: Some open solution trees of the search tree in Figure 2: a max solution tree with c-value 2 and two min solution trees with c-value 7 (left) and 9 (right).

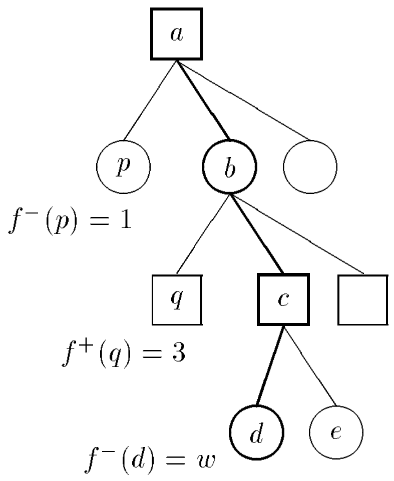


Figure 5: An example.

denoted by $\bar{T}^+$ and $\bar{T}^-$ respectively, where T^+ and T^- denote the restriction of $\bar{T}^+$ and $\bar{T}^-$ respectively to S (or: the intersection of $\bar{T}^+$ and $\bar{T}^-$ with S).
If $\bar{T}^+$ and P haven't any node in common, then $T_0^+ = \bar{T}^+$. In the alternate case, i.e., a node of P is in $\bar{T}^+$, then n is included in T^+, and we have in S that $c(T^+) \geq h^-(n)$. Further, in general, $g(\bar{T}^+) \geq c(T^+)$. By Theorem 4.1, a node $u \in ANC(n) \cup \{n\}$ satisfies $f^+(u) = h^+(n)$ in S. Due to (4.1) and (3.2) respectively we know, that $f^+(u)$ is the c-value of an open min tree with root u as well as the g-value of a closed max tree T', rooted in u. From the latter relation follows that in S: $g(T') = f^+(u) = h^+(n) \leq h^-(n) \leq g(\bar{T}^+)$. Since T' is a closed solution tree in S, P and T' have an empty intersection. Replacing $T^+(u)$ by T' in the optimal tree $\bar{T}^+$ yields T_0^+, which gets around P. Since $g(T') \leq g(T^+)$, the g-value doesn't increase by this replacement, and consequently, T_0^+ is also optimal.
Dually, a min tree T^- is obtained, which gets around P. $\Box$

To illustrate the foregoing lemma, see figure 5. We concentrate on node c; the nodes right to the path from a to d are open. The boundary values are $h^+(c) = 3$ and $h^-(c) = \max(w, 1)$. Any max solution tree T^+ through c, has subtrees, rooted in p and d respectively, with $c(T^+(p)) \geq 1$ and $c(T^+(d)) \geq w$. Suppose $w \geq 3$. Then the open node e needs no longer to be expanded. This node will not belong to an optimal max tree, because the most profitable move for min in b goes to q, the root of a strategy for min (=a max tree) with value 3. So, an optimal max solution tree through c is dominated by one through p. Since $f^+(q) = 3$, any optimal min tree through c has g-value at most equal to 3. Therefore, an optimal min solution through c may include d.
If $w < 3$, then e is still worth expanding.

Note

In [11] the following definition was presented. *A max solution tree* T_1 *dominates*

T_2, if the set of open leaves of T_1 is a subset of T_2's open set and $c(T_1) \leq c(T_2)$. The relevance of *dominance* appears from the following statement: *if T_1 dominates T_2, then every enhancement of the search tree that makes T_2 an optimal solution tree, also makes T_1 optimal.* A dual definition and dual statement holds for min trees. The proof of Lemma 5.2 actually shows that every solution tree through a descendant p of n is dominated by a second tree avoiding p. The lemma itself can be reformulated in this sense.

The assertions in the two previous lemmas can also be expressed, using quantors:

$$(\exists G \supset S \;\; \forall C \subset G \; \exists p \in P, \;\; p \in C) \Leftrightarrow h^-(n) < h^+(n),$$

or equivalently:

$$(\forall G \supset S \;\; \exists C \subset G \; \forall p \in P, \;\; p \notin C) \Leftrightarrow h^-(n) \geq h^+(n),$$

where C denotes a *critical tree.* This formula can be expressed verbally in the following way:

Theorem 5.1 *Given a node n in a search tree S, the set of open descendants of n in S can be ignored, if and only if $h^-(n) \geq h^+(n)$.*

A node n in a search tree with $h^-(n) \geq h^+(n)$ is called *dead*; analogously a node with $h^-(n) < h^+(n)$ is called *alive.* Theorem 4.1 shows that every child of a dead node is also dead. Every max node in a search tree has at least one child c_0 with $f^+(n) = f^+(c_0)$. Since also $f^-(n) \geq f^-(c_0)$, we have $h^+(c_0) = h^+(n)$ and $h^-(c_0) = h^-(n)$. We conclude that every alive max node has at least one alive child. By duality, this also holds for an alive min node.

As mentioned in Section 3, the termination criterion of almost every game tree algorithm is: $f^+(r) = f^-(r)$. The cut off criterion in Theorem 5.1 for the special case $n = r$ is equivalent to this termination criterion. We may make the following statement about the execution of a game tree algorithm: as long as the root is alive, at least one leaf of the search tree is also alive, and the execution must be continued; when the root is dead, everything is dead and the execution can be stopped.

Ibaraki [10] introduced a cut off criterion for so-called *informed* game trees, based upon the top-down functions. In our paper we deal with so-called *static* trees. Since our h-functions are the counterparts of the top-down functions, as shown in the previous section, we have extended this cut off criterion to the static model. It was shown in [11] that cutoffs in the alpha-beta or SSS* algorithm can be explained in terms of dominance of one solution tree over another one.

6 Alpha-beta revisited

An extensive treatment of the alphabeta procedure can be found in [13]. The same paper also includes a historical survey of the rise of this procedure. Figure 6 shows the code of the *alphabeta* procedure. We will present an extended post-condition, related to the boundary functions. The accompanying precondition is: $\alpha < \beta$. The return value is denoted by v.

Lemma 6.1 *The following postcondition holds for an alphabeta call.*

$$v \leq \alpha \quad \Rightarrow \quad v = f^+(n) \ \textit{(low failure)}, \tag{6.1}$$

$$\alpha < v < \beta \quad \Rightarrow \quad v = f^+(n) = f^-(n), \tag{6.2}$$

$$v \geq \beta \quad \Rightarrow \quad v = f^-(n) \ \textit{(high failure)}. \tag{6.3}$$

Proof(sketch)
This theorem dealing with a recursive procedure is proved by induction on the depth of the calling tree. We show that the theorem holds for a call, assuming that the theorem holds for any recursive subcall.
For reasons of duality, only the case that n is a max node, is studied. The *while* loop in the alphabeta code has the following invariants:

$$\beta > v = \max\{f^+(x) \mid x \in C(n) \wedge \ x \textit{ left to } c\} \tag{6.4}$$

and

$$(\alpha < \alpha' = v = f^-(x) \ \textit{for at least one such } x) \bigvee (v \leq \alpha' = \alpha) \tag{6.5}$$

The relations (6.1) and (6.2) can be derived easily from these invariants. If the execution ends with $v \geq \beta$, then $v = f^-(c_0) = f^-(n)$ where c_0 denotes the parameter in the last subcall. □

In case of $v \geq \beta$ on termination *high failure*, the child c_0 that was parameter in the last subcall, is called the cutoff child of n, since it caused an cutoff of the other (still open) children. Dually, also in case of a low failure a cutoff is defined. In some literature, the terms β-cutoff and α-cutoff are used instead of high and low failure. Likewise, the return value v is called an α or a β bound respectively.

Suppose that a call in a max node ends with a high failure (β-cutoff). The subcall in the cutoff child also ended with a high failure, as we can see in the proof of Lemma 6.1a). The dual version of that proof shows that a high failure in a min node implies that every child has been visited and every visit has also ended with a high failure. After a high failure in n, we can re-consider top-down the visited descendants of n. Taking all children in a min node and taking the cutoff child in a max node, we generate a min solution tree, which is called the *key* solution tree. Dually, a low failure gives rise to a *key* solution tree. which is of the max type.

A node m is called a *left* child of a max key solution tree T^+, if m is a child of a min node in T^+ and m is left to the single child of n in T^+. Likewise a right child of max solution tree is defined. Dually, a left or right node of a min solution are defined. After a low failure, the postcondition of the alphabeta-procedure can be extended with the following proposition: the left children of the key solution tree T^+ have an f^--value $\geq v$ and the right children are open. Notice that the children left to a cutoff node are dead and can be ignored in the sequel of the search process. This has been called a *left ignore* cutoff in [15].

The exact value of a game tree is computed by a call *alphabeta*$(r, -\infty, +\infty)$. We will show which role is played by the boundary functions during execution of this call.

```
function alphabeta($n, \alpha, \beta$);
if terminal ($n$) then  $v$ :=eval($n$);
else if max($n$) then
        $v := -\infty$;
        $\alpha' := \alpha$;
        $c$ :=first($n$);
        while $v < \beta$ and $c < \perp$ do
              $v'$ :=alphabeta($c, \alpha', \beta$);
              $v := \max(v, v')$;
              $\alpha' := \max(\alpha', v')$;
              $c$ := next($c$);
else if min($n$) then
        $v := +\infty$;
        $\beta' := \beta$;
        $c$ :=first($n$);
        while $\alpha < v$ and $c < \perp$ do
              $v'$ :=alphabeta($c, \alpha, \beta'$);
              $v := \min(v, v')$
              $\beta' := \min(\beta', v')$;
              $c$ := next($c$);
return $v$;
```

Figure 6: The alpha-beta procedure

Lemma 6.2 *Suppose a call alphabeta*$(r, -\infty, +\infty)$ *is performed. Then at every nested call alphabeta*(n, α, β)*:*

- $f^+(x) \leq \alpha$ *for every* $x \in$ *AMAX-C-L*(n) *and* $f^+(x) = f^-(x) = \alpha$ *for at least one such node* x*.*
- $f^-(x) \geq \beta$ *for every* $x \in$ *AMIN-C-L*(n) *and* $f^+(x) = f^-(x) = \beta$ *for at least one such node* x*.*
- *every* x *in AMIN-C-R*(n) *or in AMAX-C-R*(n) *is open.*

Proof
Follows from the invariant relations (6.4) and (6.5). □

Theorem 6.1 *Suppose a call alphabeta*$(r, -\infty, +\infty)$ *is performed. Then at every nested call alphabeta*(n, α, β)*, every node left to* n *is dead and* $h^-(n) = \alpha < \beta = h^+(n)$*.*

Proof
Follows directly from Lemma 6.2. □

The latest theorem actually says that a node n, when expanded by alphabeta, is the leftmost open alive node in the actual search tree.
We might say that, before n is expanded, the algorithm attempts to establish an f^+-value as low as possible in each $x \in$ *AMIN-L-C*(n). To make sure in any x that no lower value $f^+(x)$ can be achieved, an f^--value is established. Of course, a dual statement holds.

It follows immediately from Lemma 6.2, that a node n, expanded by alphabeta satisfies:

$$\begin{aligned} \beta &= \min\{f(x) \mid x \in \mathit{AMIN\text{-}C\text{-}L}(n)\} & (6.6) \\ \alpha &= \max\{f(x) \mid x \in \mathit{AMAX\text{-}C\text{-}L}(n)\} & (6.7) \end{aligned}$$

This result was already presented in [3]. In that paper, the inversion was also proved: if $\alpha < \beta$ for a node n, where α and β are defined according to (6.6) and (6.7) respectively, then, during the alphabeta algorithm, n is parameter in a call $\mathit{alphabeta}(n, \alpha, \beta)$.

7 SSS* revisited

Since the time, when people started to use game-playing programs, the alpha-beta algorithm and, later on, its variants Negascout and PVS have taken a prominent place. Besides, another algorithm is known, which has fascinated many researchers in the past seventeen years. This algorithm is called SSS*, published in 1979 by Stockman [28]. The originating paper is one of the top 50 referenced in the AI Journal[5]. Before 1994, it was never used in actual applications. Anyway, SSS* has drawn considerable attention in literature. An (incomplete) list of the major papers can be found in [5] or [21]. In this section, we will gain a lot of insight into the question, why the algorithm works. The boundary functions will turn out to be crucial in the understanding of SSS*.

The SSS* algorithm manipulates triples of the form $\langle n, s, \hat{h}\rangle$, where n is a node in the game tree, s denotes a status and $\hat{h}$ a so-called merit, a real value. The possible values of the status s are *open* and *closed*; (the original paper uses *live* and *solved* respectively instead.) The triples are included in LIST. In each iteration, one triple is selected and removed from LIST, and this triple is replaced by one or several new triple(s). This replacement is performed according to Figure 7. For one case (Case 1), in addition to putting one triple into LIST, other triples are deleted from LIST. The code of SSS* is:

```
SSS* Scheme
initially, LIST includes the single triple ⟨$r$, open, $\infty$⟩:
repeat
    select a triple $\langle n, s, \hat{h}\rangle$ from LIST with maximal $\hat{h}$;
    remove $\langle n, s, \hat{h}\rangle$ from LIST and insert new triple(s) $\langle n', s', \hat{h}'\rangle$
    into LIST, according to Figure 7;
until a triple ⟨$r$, closed, $\hat{h}$⟩ is included in LIST.
```

This code is more general than that of the seminal paper[28]. When we require that the *leftmost* triple with maximal merit is selected in each iteration, the original version is recovered. In Section 8.4, we will see that SSS* has an alternative description: a sequence of alphabeta calls with a null-window.

Given a search tree S, any leaf of a solution tree in S is also a leaf of S, as mentioned in Section 3. Here, we introduce another, more general type of solution tree, which we call a *truncated solution tree*. In a truncated solution tree T, an interior node of a search tree may be a leaf of T.

Case	Restrictions to triple $\langle n, s, \hat{h} \rangle$	new triple(s) to be inserted
1	s = closed type(parent(n)) = max	$\langle n'$, closed, $\hat{h}\rangle$ with n' =parent(n), Purge LIST of all triples $\langle p, s, \hat{h}" \rangle$ with p an descendant of n'.
2	s = closed type(parent(n)) =min next(n) $\neq$ NIL	$\langle n'$, open , $\hat{h}\rangle$ with n' =next(n)
3	s = closed type(parent(n)) = min next(n) = NIL	$\langle n'$, closed, $\hat{h}\rangle$ with n' =parent(n)
4	s = open n is a terminal	$\langle n$, closed , $\hat{h}'\rangle$ with $\hat{h}' = \min(\hat{h}, f(n))$
5	s = open n is not a terminal type(n) = min	$\langle n'$, open, $\hat{h}\rangle$ with n' =first(n)
6	s = open n is not a terminal type(n) = max	$\langle n'$, open , $\hat{h}\rangle$ for n' := first(n) to last(n)

Figure 7: Replacing a triple in SSS*

Lemma 7.1 *The SSS* algorithm has the following invariant:*

a) the nodes in the set $\{n \mid n$ occurs in a triple of LIST$\}$ *are the leaves of a truncated max solution tree with root r;*

b) for any node m that has a descendant in LIST, *a partition of the set $\mathcal{N}(m)$ is given by: $\{\mathcal{N}(p) \mid p$ occurs in a triple of* LIST *and p is a descendant of $m\}$.*

Proof
a) This can be proved by showing that a) is preserved in each of the six cases.
b) Follows directly from a). □

Theorem 7.1 *SSS* Scheme has the following invariant: every triple $\langle n, s, \hat{h}\rangle \in$* LIST *has the properties:*

a) $\hat{h} = h^+(n)$,

b) every non-open node $x \in$ AMIN-C$(n) \cup \{n\}$ satisfies $f^-(x) \geq \hat{h}$.

Proof
Each part is proved separately. We will prove that each of the six cases preserves either part in every iteration. Notice that the search tree does not change in the cases 1, 2, and 3.

Part a)
Case 1: For n' as the father of of n, $h^+(n') \geq h^+(n)$ and by the invariant, $h^+(n) = \hat{h}$. Before the iteration, we may apply Lemma 7.1b). This tells us, that every tree T in $\mathcal{N}(n')$ belongs to a set $\mathcal{N}(p)$ with p a descendant of n' occurring LIST, and $c(T) \leq h^+(p)$. Since $\hat{h}$ (the merit of n') is maximal in LIST, we conclude that $c(T)$ is at most equal to $\hat{h}$ for any $T \in \mathcal{N}(n')$. We conclude that

$h^+(n') \leq \hat{h}$. Combining this inequality with the above relation $h^+(n') \geq \hat{h}$, yields $h^+(n') = \hat{h}(n)$.
Case 2 and 3. Since, in general $h^+(x) = h^+(c)$ for each child of a min node x, the equality $h^+(n') = h^+(n)$ holds.
Case 4: Here n is closed, and $\hat{h}'$ is the updated value for $h^+(n)$.
Cases 5 and 6 comprise the expansion of n. These cases are almost trivial.

Part b).
Case 1. Since $f^-(n) \geq \hat{h}$ and n' is a max node, also $f^-(n') \geq \hat{h}$. Now the invariant follows from the relation *AMIN-C*$(n) =$*AMIN-C*(n').
Case 2. The sets *AMIN-C*(n) and *AMIN-C*(n') are identical. Since by the invariant $f^-(n) \geq \hat{h}$ and $n \in$ *AMIN-C*(n'), the invariant is preserved.
Case 3. Since $f^-(n) \geq \hat{h}$ and, by the invariant $f^-(b) \geq \hat{h}$ for every brother b of n, we conclude $f^-(n') \geq \hat{h}$. Again, the invariant now follows from the relation *AMIN-C*$(n) =$*AMIN-C*(n').
Case 4. Node n is closed and $f^-(n) \geq \hat{h}$ holds.
Case 5 and 6. The sets of non-open nodes in *AMIN-C*$(n) \cup \{n\}$ and *AMIN-C*$(n') \cup \{n'\}$ respectively are identical. □

Theorem 7.2 *When SSS* terminates with a triple* $\langle r,$ *closed,*$\hat{h}\rangle$ *in* LIST, *then* $f^+(r) = f^-(r) = \hat{h}$.

Proof
Notice that (Lemma 7.1a) implies that $\langle r,$ *closed,*$\hat{h}\rangle$ is the single triple in LIST on termination. Applying Theorem 7.1b) yields $f^-(r) \geq \hat{h} = h^+(r)$. Now, the theorem follows from the general equalities $h^+(r) = f^+(r)$ and $f^+(r) \geq f^-(r)$. □

Note
An extra property can be derived from a) and b) in Theorem 7.1. This theorem 7.1 implies that $f^-(x) \geq \hat{h} = h^+(n)$ for every non-open $x \in$*AMIN-C*$(n) \cup \{n\}$ with n a node in a triple of LIST. At least one such x has $f^+(x) = h^+(n)$, according to (4.10). Taking into account the relation $f^+(x) \geq f^-(x)$, we come to our extra property: at least one node x in *AMIN-C-L*$(n) \cup \{n\}$ satisfies $f^+(x) = f^-(x) = \hat{h}$. Notice that Theorem 7.2 is an instance of the new property.

Theorem 7.1b) can be extended with the following statement: every $x \in$*AMIN-C-L*(n) left to x is expanded (non-open), every $x \in$*AMIN-C-R*(n) is open. This requires a little longer proof. Since $h^+(n) = \hat{h}$ for any triple $\langle n, s, \hat{h}\rangle$ in LIST, there is a min solution tree T_n in $\mathcal{N}(n)$ with $c(T_n) = h^+(n) = \hat{h}$. As a matter of fact, the path from r to n is included in this solution tree. Using the extended Theorem 7.1b), we can describe the shape of one such min tree T_n more precisely. Every node x in *AMIN-C-L*(n) is the root of a closed min tree with g-value $\geq \hat{h}$. If n is clsoed, it is also the root of a closed min solution tree. A tree T_n is obtained by appending these min trees to the path from r to n. The nodes in *AMIN-C-R*(n) are open in S and hence in T_n. See figure 8 for a skeleton of T_n.

We have seen that every triple stands for min solution tree T_n. It is easily derived from its shape, that T_n dominates any other tree in $\mathcal{N}(n)$. Now, it can be argued that SSS* is a branch-and-bound method, looking for the min solution

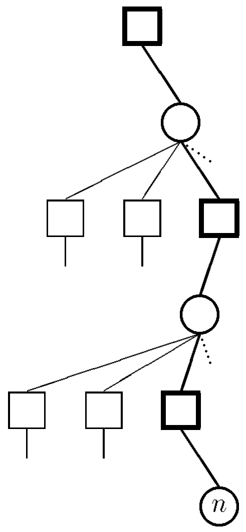


Figure 8: T_n, a min solution tree in S.

tree with maximal g-value. A solution tree T_n represents a subproblem. Branching is equivalent to expanding the leftmost open node in a min solution tree T_n, see the cases 4, 5 and 6. Bounding is computing the c-value. In the cases 5 and 6 this c-value isn't affected. Selecting a triple with maximal merit means selecting a subproblem with optimal c-value. So, the best-first selection criterion is applied.
Suppose that a triple $\langle n, \mathit{closed}, \hat{h}\rangle$ is selected and triples $\langle p, s, \hat{h}'\rangle$, where p is a proper descendant of n' =$\mathit{father}(n)$, are deleted according to Case 1. Then $c(T_n) = \hat{h} \geq \hat{h}' = c(T_p)$ and apart from the descendants of n', T_n and T_p consist of the same nodes. Therefore, T_n dominates T_p. Best-first branch-and-bound terminates, when the subproblem selected includes a solution, i.e., a closed min solution tree. This termination condition is obeyed by SSS*.

8 The MTD-framework

In this section, we discuss the MTD-framework. MTD stands for Memory Test Driver. The root of this algorithm is the *Test* routine, introduced by Pearl[17]. This procedure is equivalent to alphabeta with a so-called null-window, i.e., a window with $\beta - \alpha = 1$. A null-window is represented by one value, the greater parameter $\gamma = \beta = \alpha + 1$. When several *Test* calls are executed successively (each with a different null window), the search tree can be retained in memory and bounds can be stored at each node. The *Test* procedure, which exploits the bounds of former calls and stores new bounds, which may be of use to future calls, is named MT(Memory Test). The code of MT is presented in Figure 9. The bounds to $f(n)$, which are stored into memory, are denoted by $n.f^+$ and $n.f^-$ respectively. Bounds which cannot be retrieved are assumed to be infinite. The code of MTD is the following:

```
function MT($n, \gamma$);
if terminal ($n$) then
        if open($n$) then $v$ :=eval($n$)
        else $v := n.f^+$ or $n.f^-$; /* only in case weak storage */
else
        if open($n$) then generate the children of $n$;
        if max($n$) then
              $v := -\infty$;
              $c$ :=first($n$);
              while $v < \gamma$ and $c < \perp$ do
                   if $c.f^+ \geq \gamma$ then $v'$ :=alphabeta($c, \gamma$) else $v' := c.f^+$ ;
                   $v := \max(v, v')$;
                   $c :=$ next($c$);
        if min($n$) then
              $v := +\infty$;
              $c$ :=first($n$);
              while $v \geq \gamma$ and $c < \perp$ do
                   if $c.f^- < \gamma$ then $v'$ :=alphabeta($c, \gamma$) else $v' := c.f^-$ ;
                   $v := \min(v, v')$;
                   $c :=$ next($c$);

storage($n$);
return $v$;
```

Figure 9: MT

```
function MTD($n$);
$p := -\infty$;
$q := +\infty$;
repeat
      choose a value $\gamma \in [p + 1, q]$;
      $v$ :=MT($n, \gamma$);
      if $v \geq \gamma$ then $p := v$; else $q := v$;;
until $p = q$;
return $q$ $(= p)$;
```

In most actual applications of game tree search, a so-called transposition table, abbreviated as TT, is maintained, recording all positions visited before. This table can also serve to register the search tree. To make the code of MTD independent of the maintenance an TT, two variables p and q are used, intended to represent the values $n.f^+$ and $n.f^-$. Like the *alphabeta* code, the MT procedure also returns one value v. Consistently, one variable, either p or q is updated after a call in the main loop of MTD.

New values for $n.f^+$ or $n.f^-$ are established by the procedure *storage*(n). In most implementations of instances of MTD, e.g. [23, 24, 29], the following code is used, which we call *weak* storage.

```
procedure storage($n$) /* weak storage*/
if $v < \gamma$ (low failure) then  $n.f^+ := v$
else if $v \geq \gamma$ (high failure) $n.f^- := v$,
```

Properties of the search tree

The search tree contains a tree T^+ with $g(T^+) = q$ and a min tree with $g(T^-) = p$. As already mentioned in Section 6, every left child of T^+ as well as every left child of T^- is dead. This implies that left to the common path everything is dead.

Let us consider the situation along the common path more closely. Suppose in the small window algorithm two subsequent calls end with a low and high failure respectively. After the low failure, any node in the key solution tree T^+ has $g(T^+(n)) \leq q$. After the subsequent high failure, a left child c of T^- has $q \geq p > f^+(c)$. If the father of c is on the common path of the two key trees, then after the two calls $q \geq T^+(c)$ (and p needs not to exceed $g(T^+(c))$.

Right to the common path most children are alive in general. Possibly, a right child c is already dead. This is case for instance if $g(T^+(c) \leq p$.

The above procedure *storage* puts a value into the variable $n.f^+$ or $n.f^-$. This contents of this variable needs not to equal $f^+(n)$ or $f^-(n)$ respectively at any time. Consider for instance the tree of Figure 10, where a and c are root of a

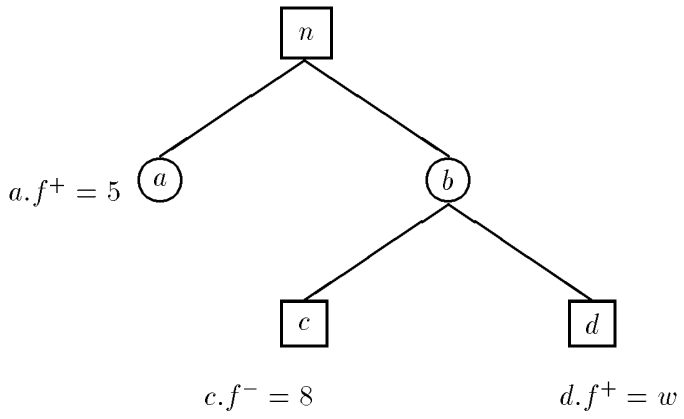


Figure 10: Side effects on boundaries.

subtree and d is a terminal with game value $f(w) \leq 5$. Initially, n is open and hence, $f(n) = -\infty$. Let a call $MT(n, 6)$ be executed. Suppose that the boundary values are stored as shown in the figure. Although $d.f^-$ and $n.f^-$ are not set (and hence are considered $= -\infty$), the actual values $f^-(d)$ and $f^-(n)$ have a finite value w. The next call may be $MT(n, -3)$. If the game tree contains a min solution tree through a with g-value $= -3$, this solution tree is generated and $n.f^-$ is set to -3. Simultaneously, $f^+(a)$ as well as $f^+(n) = \max(f^+(a), w)$ may take a new value. (Consider for instance the situation that a has three children: first two terminals with values 5 and 4 respectively and next a child a' which takes $f^-(a') = -3$.) We conclude that in case of weak storage, the bounds $n.f^+$ or $n.f^-$ stored in the TT, possibly are relaxations of the actual bounds $f^+(n)$ and $f^-(n)$ in the search tree. Since some bounds in the TT are not up-to-date, the algorithm overlooks for some nodes that they are dead. Consequently, dead nodes are re-visited and dead open nodes are expanded possibly. If $w = 5$ in Figure 10, then n is dead after the first call $MT(n, 6)$ and the algorithm may stop.

In order to make sure that all bounds in the TT are up-to-date at any time, we may introduce the so-called *heavy storage*.

```
procedure heavy-storage($n$)
if $v < \gamma$ (low failure) then
   $n.f^+ := v$;
   if terminal($n$) then $n.f^- := v$
                    else  $n.f^- := \max / \min\{c.f^- \mid c \in C(n)\}$
else if  $v \geq \gamma$ (high failure)
   $n.f^- := v$;
   if terminal($n$) then $n.f^+ := v$
                    else  $n.f^+ := \max / \min\{c.f^+ \mid c \in C(n)\}$
```

As a matter of fact, heavy storage has higher requirements with respect to time and space and, in practice, a trade-off has to be made.
In the MTD algorithm as described at the beginning of this section, the variables p are q are intended to represent the values $n.f^-$ and $n.f^+$ of the search tree. To be consistent, both p and q must be updated after each MT call in the main loop of MTD, when using heavy storage. Using this storage method, the call $MT(n, \gamma)$ has the same postcondition as alphabeta has with window $[\gamma - 1, \gamma]$ and the equalities $f^+(x) = x.f^+$ and $f^-(x) = x.f^-$ hold at any node x at any time. It can be derived from the code of MT, that in a search tree, where $x.f^-$ and $x.f^+$ are truthful values for every x, the relation

$$h^-(n) < \gamma \leq h^+(n) \tag{8.1}$$

holds at any nested call $MT(n, \gamma)$ during execution of MTD. This can be shown by induction on the depth of n. Hence, using heavy storage, the algorithm visits solely alive nodes. Especially, a closed terminal is never a parameter in an MT call.
Similarly to Lemma 6.2, we can show that at every nested call MT(n, γ) in the MT-SSS algorithm:

$$f^+(x) < \gamma \;\; \textit{for every } x \in \textit{AMAX-C-L}(n) \tag{8.2}$$

$$f^-(x) \geq \gamma \;\; \textit{for every } x \in \textit{AMIN-C-L}(n) \tag{8.3}$$

Likewise, we have the following counterpart to (6.6) and (6.7) for a node, when expanded by MTD:

$$\min\{f(x) \mid x \in \textit{AMIN-C-L}(n)\} \geq \gamma > \max\{f(x) \mid x \in \textit{AMAX-C-L}(n)\} \tag{8.4}$$

In Section 6 we presented a sufficient condition (due to Baudet), for a node to be visited by alpahabeta. It follows from that condition that every node visited by MTD and hence obeying (8.4) is also visited by alphabeta.

Note

Apart from our concern with storage methods, we will point to another interesting phenomenon. Looking at the code of MT, we see that in a child of a max node only the f^+-value is inspected. This is due to the fact that $h^-(n) = h^-(c)$ for each child c of a max node n. In a min node, we have a dual phenomenon. It turns out that it is useless to store the f^+ in a max node's child or to store f^- in a min node's child, Consequently, only one bound needs to be stored per node, except r. These observations hold regardless which storage procedure (weak or heavy) is applied.

9 Some interesting instances of MTD

The following instance of MTD is called *MTD-f*.

```
**function** MTD-f($n, f$);
$v$ :=MT($n, f$);
**if** $v < f$ then
    **repeat**
        $q := v$;
        $v$ :=MT($n, q$);
    **until** $v = q$;
**if** $v \geq f$ then
    **repeat**
        $p := v$;
        $v$ :=MT($n, p+1$);
    **until** $v = p$;
**return** $v$;
```

An interesting instance of of MTD-f(f)is the instance with $f = +\infty$ and weak storage. In this algorithm q varies and p remains $= -\infty$ throughout execution. This instance is called MT-SSS, because it is equivalent to SSS* in the sense that the sequence of nodes expanded successively is the same in either algorithm. This is proved formally in [22]. A sketch of the proof can be found in [24]. The algorithm SSS-2, presented in [19], is also equivalent to MT-SSS. An extensive analysis of SSS-2 can be found in [20]. Since weak storage is used, MT-SSS as well as SSS* may expand dead nodes. See Figure 10 with $w = 5$. First, MTD executes a call $MT(n, \infty)$. Then the bounds are as shown in the figure. Next a call $MT(n, 5)$ is performed, which expands dead open descendants of a and revisits d. However, if the first loop in the above code is executed, it can be shown, using the results of [22], that $x.f^+$ is up to date for any x throughout execution. Furher, any closed node x, which is parameter in a nested call $MT(x, \gamma)$, has $h^+(x) = f^+(x) = \gamma$. Such a node x has node has a small chance to be dead. This is the case, only if the key solution tree T^+ includes a path P from x to a terminal p, such that P contains the youngest child in each min node and $f(p) = \gamma$. If the second loop in the above code is executed, dual statements hold.

In MT-SSS with heavy storage, (8.1) holds, when an open node n is expanded. The equalities $f^+(r) = \gamma$ and (4.6) imply that $h^+(n) \leq \gamma$. We conclude that $h^+(n) = \gamma$. Furthermore, the inequalities (8.2) and (8.3) hold. This implies that every desendant of a node $x \in AMIN(n)$ is dead and every descendant d of a node $x \in AMAX(n)$ has $h^+(d) < \gamma$. Since $f^+(r) = \gamma$, there is no node in the search tree with h^+-value $> \gamma$. We conclude that, at any call $MT(n, \gamma)$ with n open during MT-SSS, $h^+(n)$ is maximal in the search tree and n is leftmost open alive node with maximal h^+-value.

Extensive tests with MT-SSS, MT-Dual and MTD(f) using weak storage are described in [23] and [24]. It turns out that, combining MTD(f) with iterative deepening and taking the value of each previous iteration as the next f-value, results in a quick and efficient algorithm.

We conclude this section with discussing the conditions for a node to be expanded by SSS* or MT-SSS. When an open node n is expanded in SSS* or MT-SSS, every $x \in \mathit{AMIN}\text{-}C\text{-}R(n)$ is open. Since $f^+(r) \leq \gamma$, we have $h^+(n) = \gamma$. It follows from (8.3) that $f^+(x) = f^-(x) = \gamma$ for at least one $x \in \mathit{AMIN}\text{-}C\text{-}R(n)$. Let x_0 denote the node closest to the root with this property. Then $f^-(x) > \gamma$ and hence also $f^+(x) > \gamma$ for every $x \in \mathit{AMIN}\text{-}C\text{-}L(x_0)$. As mentioned above, every node in $\mathit{AMIN}\text{-}C\text{-}R(n)$ is open. It follows that $f^+(y) \leq \gamma$ for every $y \in \mathit{ANC}(x_0)$, because if we had $f^+(y) > \gamma$, the relation for h^+ in (4.10) would yield $h^+(y) > \gamma$, which is not compatible with $f^+(r) \leq \gamma$.
Now, we are ready to formulate necessary conditions for an open node n, which is expanded by a call $MT(n, \gamma)$.

$$\min\{f(x) \mid x \in \mathit{AMIN}\text{-}C\text{-}L(n)\} = \gamma > \max\{f(x) \mid x \in \mathit{AMAX}\text{-}C\text{-}L(n)\} \quad (9.5)$$

and

$$\max\{f(y) \mid y \in \mathit{ANC}(x_0)\} \leq \gamma, \quad (9.6)$$

where x_0 is the highest node x in $\mathit{AMIN}\text{-}C\text{-}L(n)$ whose f-value achieves the minimum in (9.5).
Similarly to alphabeta (cf. Section 6), the above conditions necessary for a node to be expanded by a call $MT(n, \gamma)$ during MT-SSS, are also sufficient. This is shown in [19] and [20] for SSS-2, an obsolete formulation of MT-SSS with weak storage.
The combination of (9.5) and (6.6) shows that a node n, that is expanded in MT-SSS by a call $MT(n, \gamma)$ is expanded in the alphabeta algorithm with parameter $\beta = \gamma$. In the proof for the equivalence of MT-SSS and SSS* it appears that Case 5 or 6 once applies to a triple $\langle n, \mathit{open}, \hat{h}\rangle$ with $\hat{h} = \gamma$.

10 Scout

Negascout [25] is another algorithm, which computes the minimax value $f(n)$ of a node n. Negasout is equivalent to PVS[7]. An older version of Negascout is Scout. The idea of this algorithm originates from Pearl[17]. His idea can be paraphrased as follows in terms of solution trees. In order to compute the game value of n, a critical tree with root n must be built. The child of n on the critical path is called th *principal variation.* First, we guess which child of n is the principal variation. For this child the game value is computed, using a recursive call of the scout procedure. Next, in case n is a max node, In each child successively, $c \neq c_0$ a max solution tree T^+ is rooted with $g(T^+) \leq f(c_0)$, if possible. If the construction of such a tree fails for some child c, this child is considered

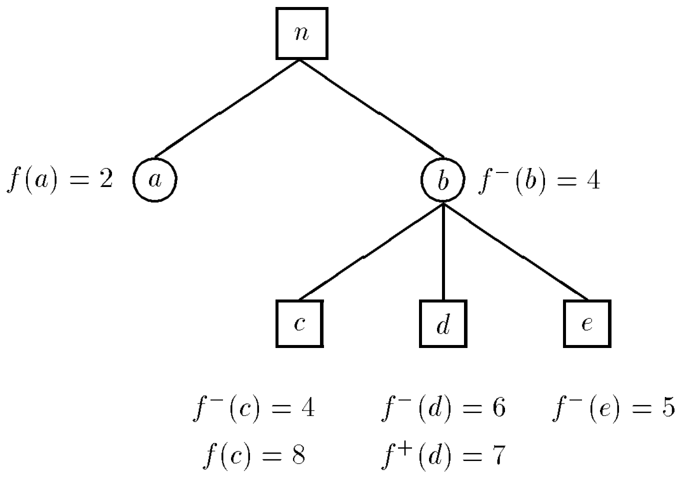


Figure 11: Applying Scout to a game tree.

the principal variation instead of c_0. It is subject to a scout call therefore. The construction of a max solution tree is achieved by a null window call. When a scout call is performed subsequently, the result of the former MT calls can be exploited. We become aware of the fact that a max tree with g-value $\leq f(c_0)$ does not exist, when a min solution is found.

To illustrate Scout, we consider the game tree fragment in Figure 11, where the nodes a, c, d and e are the roots of subtrees in the game tree. We describe a possible run. The principal variation of n and Scout establishes $f(n) = 2$. The subsequent MT call to b ends with $v' = 4$, and hence, b is the root of a min solution tree. In this tree $f^-(c) = 4$, $f^-(d) = 6$ and $f^-(e) = 5$. Next, the scout procedure is invoked with b as parameter. In this scout call, c is chosen as the principal variation and a nested scout call returns 8. As a side effect, $f^-(b)$ has changed to 5. The subsequent call $MT(d, 8)$ ends with $v' = 7$. Then, in the search tree, both boundary values of d as well as of b are finite: $f^+(b) = f^+(d) = 7$, $f^-(b) = 4$ and $f^-(d) = 6$. A scout call to d follows, which returns 6. The call $MT(e, 6)$ ends with a low failure: $v' \geq 6$. After this result, the algorithm finishes.

We will prove some properties of Scout. Each proof uses induction on the depth of the calling tree, as already applied for Theorem 6.1. A result is shown to hold for a call, under the assumption that the same result holds for any recursive subcall.

Theorem 10.1 *The call Scout(n) returns $f(n)$.*

```
function scout($n$);
if terminal ($n$) then
        if open($n$) then $v$ :=eval($n$);
                  else $v := n.f^+$ or $n.f^-$; /*only in case of weak storage */
else
    if open($n$) then generate the children of $n$;
    if max($n$) then select a child $c_0$ with $n.f^- = c_0.f^-$;
              else select a child $c_0$ with $n.f^+ = c_0.f^+$;
    if $c_0.f^+ > c_0.f^-$ then $v$ :=scout($c_0$) else $v := c_0.f^+$;
    $c$ :=first($n$);
    if max($n$) then
          while $v < n.f^+$ and $c < \perp$ do
                if $c.f^+ > v$ then
                    $v'$ :=MT($c, v+1$);
                    if $v' > v$ then if $v' = c.f^+$ then $v := v'$;
                                                   else $v$ :=scout($c$);
                $c$ :=next($c$);
    else if min($n$) then
          while $v > n.f^-$ and $c < \perp$ do
                if $c.f^- < v$ then
                    $v'$ :=MT($c, v$);
                    if $v' < v$ then if $v' = c.f^-$ then $v := v'$;
                                                   else $v$ :=scout($c$);
                $c$ :=next($c$);
return $v$;
```

Figure 12: scout

Proof(for a max node n)
The following loop invariant holds: $v := \max\{f(c) \mid c \text{ is already visited }\}$. When v is set for the first time, the invariant holds. When v is updated, a preceding MT-call with parameters c and $v + 1$ ended with a high failure and therefore showed that $v < f(c)$. □

An obvious choice for the principal variation is the cutoff child (if any) of the most recent MT-call. If this choice is maintained consistently, the Scout algorithm does not visit any dead mode during execution, as we will show now.

Lemma 10.1 *When a call scout(n) is executed with n a max node, the precondition $f^-(c) = f^-(n)$ holds for every inner scout call and the precondition $v = f^-(n)$ holds for every inner MT call.*

Proof
For the first scout call with parameter c_0, the precondition holds as a consequence of the selection criterion $f^-(c_0) = f^-(n)$. On termination of this call $v = f(c_0)$, which value is at least the former values $f^-(c_0) = f^-(n)$. It follows that $v = f^-(n)$ after the first inner scout call, which relation also holds, when the first iteration of the while loop starts.
If $v+1 \leq v'$, then MT has ended with a high failure and $v+1 \leq v' = f^-(c)$ holds. Since the new value value $v' = f^-(c)$ is at least equal to the old value $v = f^-(n)$,

we have $v' = f^-(c) = f^-(n)$, which relation also holds for the subsequent scout call. After the assignment $v := v'$, we obtain $v = v' = f^-(n)$. Similarly to first scout call, the precondition $f^-(c) = f^-(n)$ on call implies $v = f^-(n)$ on termination.
In each next iteration, the precondition of MT and scout is derived in the same way, exploiting the fact that $v = f^-(n)$ holds at the start of each iteration. $\Box$

Suppose the choice of the PV is arbitrary. Then Lemma 10.1 does not hold for the first scout call with $c = c_0$, but it does for the subsequent calls. Further, the precondition for MT becomes: $v \leq f^-(n)$.

Theorem 10.2 *The Scout algorithm only visits alive nodes.*

Proof
Descending in a tree such that in every max node n a child c is chosen with $f^+(c) > f^-(c) = f^-(n)$ and in every min node a dual choice is made, yields a path with solely alive nodes. Since the scout algorithm makes such choices for the recursive scout calls, every parameter in a nested scout call is alive. For each node on the path obtained by the above construction, the relation $f^+(x) = h^+(x) > h^-(x) = f^-(x)$ holds. This relation is preserved, when $f^+(x)$ or $f^-(x)$ change during execution.
Whenever a max node n has a child c, that is parameter in a nested MT-call, then $f^-(c) \leq f^-(n) = v < f^+(c) \leq f^+(n)$, It follows that $h^-(c) = v$ and $v < h^+(c) = f^+(c)$. Since the null window is in the liveness windows, MT solely visits live nodes. $\Box$

The previous theorem assumes that the values $n.f^+$ and $n.f^-$ stored into memory by MT are truthful values at any time at any node.
In practice, the availability of a complete search tree in memory is not guaranteed, since a transposition table acting as a search tree may suffer from collisions.

11 Concluding remarks

In the Sections 2 through 5, we developed a general theory on game tree, based upon solution trees. In this theory, the functions f^+, f^-, h^+ and h^- played an essential role. In the Sections 6 through 9 this theory was applied to some well-known algorithms, viz. alphabeta, SSS*, MTD and Scout. How this theory relates to Proof Number search [1, 2], was shown in [6].

Further, we have given we gave a dynamic characterization of a node, expanded during alphabeta and MT-SSS. So we described the node expanded by alphabeta or MT-SSS in terms of the current search tree, using the h^+ and h^- function. Alphabeta is the algorithm that expands at any time the leftmost open alive node of the current search tree. MT-SSS expands the leftmost open alive node with maximal h^+-value, assuming that the search tree is stored completely (including truthful bounds) in memory. One might state that alphabeta is a depth-first search algorithm, whereas MT-SSS is a best-first search algorithm.

The sets of nodes visited by alphabeta and MT-SSS have been characterized in [3] and [19, 20] respectively. One way of this characterization (the necessary condition for a node to be visited) has also been shown in the present paper.

References

[1] L. Victor Allis, Maarten van der Meulen, and H. Jaap van den Herik, *Proof-number search*, Artificial Intelligence **66** (1994), 91–124.

[2] L. Victor Allis, *Searching for Solutions in Games and Artificial Intelligence*, Ph. D. Thesis, Maastricht, NL 1994.

[3] G. M. Baudet, *On the branching factor of the alpha-beta pruning algorithm.* Artificial Intelligence 10 (1978), pp 173-199.

[4] Subir Bhattacharya and Amitava Bagchi, *A faster alternative to SSS* with extension to variable memory*, Information processing letters 47 (1993), pp. 209-214.

[5] Daniel Bobrow, *Artificial Intelligence in perspective, a retrospective on fifty volumes of the Artificial Intelligene Journal*, Artificial Intelligence, vol. 59 (1993) pp. 5-20.

[6] A. de Bruin, W. Pijls and A.Plaat, *Solution Trees as a Basis for Game Tree Search*, ICCA Journal, 17(4), pp.207-219, December 1994.

[7] Murray S. Campbell and T. A. Marsland, *A comparison of minimax tree search algorithms*, Artificial Intelligence 20 (1983), pp. 347-367.

[8] John P. Fishburn and Raphael A. Finkel, *Parallel alpha-beta search on arachne*, Tech. Report 394, Computer Sciences Dept, University of Wisconsin, Madison, WI, 1980.

[9] C.A.R. Hoare, *An axiomatic basis for computer programming,* Communications of the ACM, 12 (1969), pp 576-580.

[10] Toshihide Ibaraki, *Generalization of alpha-beta and SSS* search procedures*, Artificial Intelligence 29 (1986), pp. 73-117.

[11] V. Kumar and L.N. Kanal, *A General Branch and Bound Formulation for Understanding and Synthesizing And/Or Tree Search Procedures*, Artificial Intelligence 21 (1983), pp. 179-198.

[12] V.Kumar and L.N. Kanal, *Parallel Branch and Bound Formulations for AND/OR Tree Search*, IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-6 (1984) no. 6, pp. 768-778.

[13] Donald E. Knuth and Ronald W. Moore, *An analysis of alpha-beta pruning*, Artificial Intelligence 6 (1975), no. 4, pp. 293-326.

[14] Richard E. Korf, *Best-first minimax search: First results*, Proceedings of the AAAI'93 Fall Symposium, American Association for Artificial Intelligence, AAAI Press, October 1993, pp. 39–47.

[15] T. A. Marsland, A. Reineveld, J. Schaeffer, *Low Overhead Alternatives to SSS*,* Artificial Intelligence 31 (1987) pp. 185-199.

[16] Judea Pearl, *Asymptotical properties of minimax trees and game searching procedures*, Artificial Intelligence **14** (1980), no. 2, 113–138.

[17] Judea Pearl, *Heuristics – intelligent search strategies for computer problem solving*, Addison-Wesley Publishing Co., Reading, MA, 1984.

[18] J. Pearl and R.E. Karp, *Search Techniques*, Ann. Rev. Comput. Sci. 2, 1987, pp. 451-467.

[19] Wim Pijls and Arie de Bruin, *Another view on the SSS* algorithm*, Algorithms, International Symposium SIGAL '90, Tokyo, Japan, August 16–18, 1990 Proceedings (T. Asano et. al. eds.), LNCS, vol. 450, Springer-Verlag, August 1990, pp. 211–220.

[20] Wim Pijls and Arie de Bruin, *Searching informed game trees*, Tech. Report EUR-CS-92-02, Erasmus University Rotterdam, Rotterdam, NL, October 1992, Extended abstract in Proceedings CSN 92, pp. 246–256, and Algorithms and Computation, ISAAC 92 (T. Ibaraki et al. eds), pp. 332–341, LNCS 650.

[21] Wim Pijls and Arie de Bruin, *SSS*†, Advances in Computer Chess 8 (H.J. van den Herik et al. eds.), University of Limburg, Maastricht, The Netherlands, 1997 (to appear).

[22] Wim Pijls, Arie de Bruin, Aske Plaat, *A theory of game trees, based on solution trees,* Tech. Report EUR-CS-96-xx, Erasmus University Rotterdam, Rotterdam, NL, November 1996.

[23] Aske Plaat, Jonathan Schaeffer, Wim Pijls and Arie de Bruin, *Best-first fixed depth game tree search in practice.* In: Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95), vol. 1 pp 273-279, 1995.

[24] Aske Plaat, Jonathan Schaeffer, Wim Pijls and Arie de Bruin, *A Minimax Algorithm Better than SSS*,* In: Artificial Intelligence, to appear.

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

[26] Alexander Reinefeld and Peter Ridinger, *Time-efficient state space search,* Artificial Intelligence 71 (1994), pp. 397-408.

[27] Igor Roizen and Judea Pearl, *A minimax algorithm better than alpha-beta? Yes and No*, Artificial Intelligence 21 (1983), pp. 199–230.

[28] G. Stockman, *A minimax algorithm better than alpha-beta?*, Artificial Intelligence 12 (1979), no. 2, pp. 179-196.

[29] J.C. Weill, *The NegaC* Search,* ICCA Journal 15 (1992), no. 1, pp. 3–7.

Glossary

G	a game tree
T^+, T^-	a max/min solution tree
S	the search tree
S^+, S^-	the search tree with $+\infty/-\infty$ as game value in the
For each above tree denotation A:	
$A(n)$	the subtree of A, rooted in n
$\mathcal{M}(n), \mathcal{N}(n)$	the set of max/min solution trees in S through the root and n
$f(n)$	the minimax value of n
$f(G)$	the minimax value of the root of G
$g(T)$	(T a closed solution tree) the minimax value of the root of T, restricted to T, or:
$g(T^+)$	(T^+ a closed solution tree) $\max\{f(p) \mid p$ a terminal in $T^+\}$
$g(T^-)$	(T^- a closed solution tree) $\min\{f(p) \mid p$ a terminal in $T^-\}$
$c(T^+)$	the minimax value in S^- of the root of T^+ or: $\max\{f(p) \mid p$ a closed terminal in $T^+\}$,
$c(T^-)$	the minimax value in S^+ of the root of T^- or : $\min\{f(p) \mid p$ a closed terminal in $T^-\}$.
$f^+(n), f^-(n)$	the minimax value of n in S^+/S^-
$h^+(n), h^-(n)$	the minimum/maximum of value $c(T)$, $T \in \mathcal{M}/\mathcal{N}$

A Equivalence of SSS* and MT-SSS

In this appendix, we give the proof of the equivalence of SSS* and MT-SSS. To that end, we first present an extended postcondition of the classical alphabeta algorithm. Each proof in this appendix applies induction on the height of the calling tree, generated by nested recursive calls.

Definition A.1 *A search tree S with root n is a γ-milestone if $f^+(n) \leq \gamma$ and S contains a max solution tree T^+ such that, in every min node of T^+, the children left to T^+ have an f^--value $> \gamma$ and the children right to T^+ are open.*

The following intuitive interpretation holds for a milestone: a γ-milestone is the minimum search tree necessary to achieve an upperbound $\leq \gamma$, when the game tree is searched from left to right. Given a value γ, the key solution tree T^+ in any γ-milestone is determined uniquely, because it is the leftmost solution tree with g-value $\leq \gamma$.
If a search tree S with root n is a γ-milestone, $S$is also a γ'-milestone for any γ' in the interval $[f^+(n), \gamma]$.

Lemma A.1 *When a call alphabeta(n, α, β) terminates with a low failure, then S is an α-milestone.*

Proof
We give a proof by induction on the height of the calling tree. In the basic step, we consider the case that n is a terminal. Here, the postcondition holds almost trivially.
The induction step is divided into two cases: n is a max node and n is a min node respectively.
If a fail low happens in a max node n, then every child c of n has been parameter in a subcall and every subcall has ended with a fail low. By the induction hypothesis, every child c is the root of an α-milestone. It follows immediately from the definition of max milestone that n is also the root of such a milestone.
If a fail low happens in a min node n, then one subcall, say to a child c_0, has ended with a fail low. The induction hypothesis, referring to a), says that c_0 is the root of a max milestone with pivot α. Every call to an older brother c of c_0 has ended with $v' = f^-(c) > \alpha$. Every younger brother of c_0 is still open. Again, we conclude from the definition of a milestone, that n is the root of an α-milestone. $\Box$.

The foregoing proof shows, that after a high failure, the key solution tree as defined in Section 6 and the key solution tree in the related milestone are identical. Now, we show that AB-SSS is equivalent to SSS*. To that end, the code of the alphabeta procedure in small window search is tailored to AB-SSS. This code can be found in Figure 13. Moreover this code is enhanced with calls of a procedure *List-op*, operating on a List. The call *List-op*(i, n) means that the operations of Case i in Figure 7 have to be executed. We assume, that LIST is initialized to $\langle r, \mathit{live}, \infty\rangle$, like in SSS*.

Theorem A.1 *During execution of MT-SSS, the following conditions apply to the calls List-op(i, n) and to the call MT(n, γ).*

```
function MT($n, \gamma$);
if terminal ($n$) then
        if open(n) then
            $v$ :=eval($n$);
            List-op($4, n$);
        else $v := n.f^+$ or $n.f^-$;
else if max($n$) then
        if open($n$) then
            generate the children of $n$;
            List-op($6, n$);
        $v := -\infty$;
        $c$ :=first($n$) ;
        while $v < \gamma$ and $c < \perp$ do
            if $c.f^+ \geq \gamma$ then
                $v'$ :=MT($c, \gamma$);
                if $v' \geq \gamma$ then List-op($1, c$);
            else $v' := c.f^+$ ;
            $v :=$ max($v, v'$);
            $c :=$ next($c$);
else if min($n$) then
        if open($n$) then
            generate the children of $n$;
            List-op($5, n$);
        $v := +\infty$;
        $c$ :=first($n$) ;
        while $v \geq \gamma$ and $c < \perp$ do
            if $c.f^- < \gamma$ then
                $v'$ :=MT($c, \gamma$);
                if $v' \geq \gamma$ then
                    if $c <$ last(n) then List-op($2, c$) else List-op($3, c$);
            else $v' := c.f^-$;
            $v :=$ min($v, v'$)
            $c :=$ next($c$);

weak storage($v$);
return $v$;
```

Figure 13: The MT-procedure in MT-SSS

a) precondition of List-op(i, n):
LIST includes a triple $\langle n, s, \gamma\rangle$, being the leftmost triple with maximal merit; the restrictions in Case i of Figure 7 are satisfied for this triple;

b) precondition of $MT(n, \gamma)$:
if n is open, then $\langle n, open, \gamma\rangle$ is in LIST and n in the leftmost node in LIST with maximal merit;
if n is closed, $S(n)$ is a γ-milestone and $\gamma = f^+(n)$; every leaf x of the key solution tree T^+ is represented in LIST by a triple $\langle x,$ closed, $f(x)\rangle$; one of these leaves of T^+ is the leftmost node in LIST with maximal merit.

c) postcondition of $MT(n, \gamma)$:
if $v < \gamma$, then $S(n)$ is a γ-milestone and $v = f^+(n)$; every leaf x of the key solution tree T^+ is represented in LIST by a triple $\langle x,$ closed, $f(x)\rangle$;
if $v \geq \gamma$, then $\langle n,$ closed, $\gamma\rangle$ is in LIST

In case b) and c), no other descendants of n are included in LISTand every node of the key solution tree T^+ and every left child of T^+ has truthful storage values.

Proof
We will prove that the specification consisting of the precondition in b) and the postcondition of c) is correct for the procedure *MT*. A correct specification means that the postcondition is satisfied, provided that the precondition is satisfied. The proof is by induction on the height of the calling tree. This proof is divided into two parts. First, we show that if the precondition holds for a call, it also holds for all subcalls. Second, we prove that the postcondition is met. In both parts, the assumption is already made that the specification is correct for each subcall.
As a side effect, we prove that the precondition of *List-op* holds, whenever it is called.
To complete the proof of the theorem, we must show that the calls $MT(r, \gamma)$ in the main program of MT-SSS satisfy the precondition in b). This is shown as follows, using the correctness of the specification of *MT*. In the first iteration, n is open and $\langle r, open, \infty\rangle$ is in LIST. Each *MT* generates a milestone S with value v and each next *MT* call starts with this milestone S, replacing v by γ. Of course, the relation between LIST and S accomplished in an iteration, holds at the start of the next iteration.

Now, we give the two parts, which make up the correctness of $MT(n, \gamma)$.

Precondition of MT
We distinguish between n being open and n being closed.
Assume the precondition holds for a call to an open node n. First consider the case that n is a max node. By assumption, $\langle n, open, \gamma\rangle$ is in LIST, with n being the leftmost node in LIST with maximal merit. The precondition for *List-op*$(6, n)$ holds. This operation replaces the triple including n by a series of triples, each including a child of n. When c is parameter, the subcalls (if any) to older brothers b has ended with $v' < \gamma$. By the induction hypothesis, after each such call, the descendant terminals of b in LIST have a merit equal to their f-value. Since

$g(T') = v' < \gamma$, also each of these merits is $< \gamma$. It follows that, when c is parameter, $\langle c, open, \gamma\rangle$ is in LIST, being the leftmost triple with maximal merit. Hence the precondition holds for c.
Second, consider n being a min node. By assumption, $\langle n, open, \gamma\rangle$ is in LIST and this triple is the leftmost triple with maximal merit. The precondition to *List-op*$(5, n)$ holds. The operation *List-op*$(5, n)$ causes the precondition to be met for the oldest child c of n. As long as each subcall ends with $v' \geq \gamma$, the while loop is continued. Before each such subcall, a triple $\langle c, s, \gamma\rangle$ is in LIST, with $s =$*open*, being the leftmost triple with optimal merit. After the subcall, s has changed to *closed*. The precondition for the call *List-op*$(2, c)$ holds and the related operation replaces this triple by $\langle next(c),\ open,\ \gamma\rangle$. We conclude that the precondition of MT holds for each next subcall.

Now, we treat the case that n is a closed node. Assume the precondition holds for an inner node n. Since γ is the maximum of the game values of the leaves of T^+, the maximal merit in LIST is equal to γ. The properties described in b) are obvious for each closed child c, except the property that c is the leftmost node LIST with maximal merit. We will explicitly show this property for every child c, whenever it is parameter in a subcall. Further, we prove that the precondition holds, when an open child c is addressed.
A child c of a max node n, which is parameter in a subcall, is in the key solution tree and has $f^+(c) = \gamma$. When c is parameter, every older brother b has $f^+(b) < \gamma$. Therefore c is ancestor of the leftmost node in LIST with maximal merit.
If n is a min node, the child of n in T^+ is the only closed child that undergoes a subcall (the older children x satisfy $f^-(x) > \gamma$). It follows that this child is also the ancestor of the leftmost node in LIST with maximal merit. When, later on, an open child c of n is a parameter in a subcall, then every preceding subcall to an older brother has ended with $v' \geq \gamma$. Similarly to the the situation with n open (see above), the precondition holds for every subcall to an open child.

Postcondition of MT
First, consider n being a terminal. Since the precondition of MT holds, also the precondition to List-op$(4, n)$ is met. On exit, n is in LIST with status *closed*, and either *merit*$= f(n) = v < \gamma$ or *merit* $= \gamma \leq f(n) = v$. In both cases, the postcondition holds.
Second, consider n being an inner max node. If every subcall ends with $v' < \gamma$, then $v < \gamma$ on exit. The properties mentioned in c) hold for S, because they hold for each subtree $S(c)$. If at least one subcall to a child c ends with $v' = \gamma$, then after this call $\langle c, closed, \gamma\rangle$ is in LIST. This triple is the leftmost triple with maximal merit, due to the fact that the calls to older brothers of c have ended with $v' < \gamma$, The operation *List-op*$(1, c)$ causes the postcondition to be met for n.
Third, consider n being a min node. As soon as a subcall $MT(c_1, \gamma)$ has return value $v' < \gamma$, the *while* loop stops. Every brother at the left side of c_1 has $f^-(c_1) \geq \gamma$ and every brother at the right side is still open. By the induction hypothesis, $S(c_1)$ is a max milestone and consequently $S(n)$ is. They have the same key solution tree, whose leaves are in LIST. If all subcalls end with $v' \geq \gamma$,

then, on termination of the while loop, $\langle \mathit{last}(n), \mathit{closed}, \gamma\rangle$ is in LIST. Since *last*(n) has no older brothers in LIST, this node is the leftmost node in LIST with maximal merit. The precondition to *List-op*$(3, \mathit{last}(n))$ is satisfied and after this operation, the postcondition of MT holds for n. $\square$

During MT-SSS, we descend top-down in the search tree and we execute recursively a call $NT(n, \gamma)$, where γ is constant and $f^+(n) = \gamma$, as shown above. Descending top-down in this way, we may encounter a dead node n. This is the case, if n has $\gamma = f^-(n)$ (beside $\gamma = f^+(n)$). (The first dead node, that is encountered, if any, is a child of a min node, because, if it was a max node's child, this max node is also dead). Since n is also the root of a γ-milestone, we conclude that n is the starting point of a path P to a terminal z with $f(z) = \gamma$, such that every node in P that is a child of a min node, is the youngest child of that min node. Unless n is the max father of a terminal or n itself is a terminal, this is a rare situation. Therefore, apart from visits to closed terminals, MT-SSS with weak storage hardly differs from the version with heavy storage.