Game tree algorithms and solution trees

Wim Pijls, Arie de Bruin

Department of Computer Science, Erasmus University,
P.O.Box 1738, 3000 DR Rotterdam, The Netherlands
e-mail {*pijls,adebruin*}*@few.eur.nl*

Abstract

In this paper a theory of game tree algorithms is presented, entirely based upon the concept of a solution tree. Two types of solution trees are distinguished: max and min trees. Every game tree algorithm tries to prune as many nodes as possible from the game tree. A cut-off criterion in terms of solution trees will be formulated, which can be used to eliminate nodes from the search without affecting the result. Further, we show that any algorithm actually constructs a superposition of a max and a min solution tree. Finally, we will see how solution trees and the related cutoff criterion are applied in major game tree algorithms like alphabeta and MTD.
Keywords: Game tree search, Minimax search, Solution trees, Alpha-beta, SSS*, MTD.

1 Introduction

A game tree models the behavior of a two-player game. Each node n in such a tree represents a position in a game. An example of a game tree with game values is found in Figure 1. The players are called Max and Min. Max is moving from the square nodes, Min from the circle nodes. The game value $f(p)$ for a position p may be defined as the guaranteed pay-off for Max. This function obeys the minimax property. An algorithm computing the guaranteed pay-off in a node n is called a game tree algorithm. Over the years many algorithms have been designed. Every algorithm tries to eliminate as many nodes as possible from the game tree search. So every algorithm has its own cut-off criterion. We will design a cut-off criterion derived from a theory of game trees. In this theory the notion of a solution tree is the key notion, which turns out to be a powerful tool for establishing such a criterion. We show that, in the family of search algorithms obeying the cut-off criterion, alphabeta is the depth-first instance, whereas MT-SSS is the best-first instance. Besides, we show in an obvious way that every algorithm necessarily builds a critical tree.

This paper is organized as follows. In Section 2 we recall some facts on solution trees mentioned earlier by Stockman[15]. In Section 3 the notion of a search tree is recalled. This notion has been introduced by Ibaraki [4]. Next, minimax functions on a search tree are defined, and the role of solution trees in a search tree is discussed. Section 4 presents a general theory on game tree algorithms based

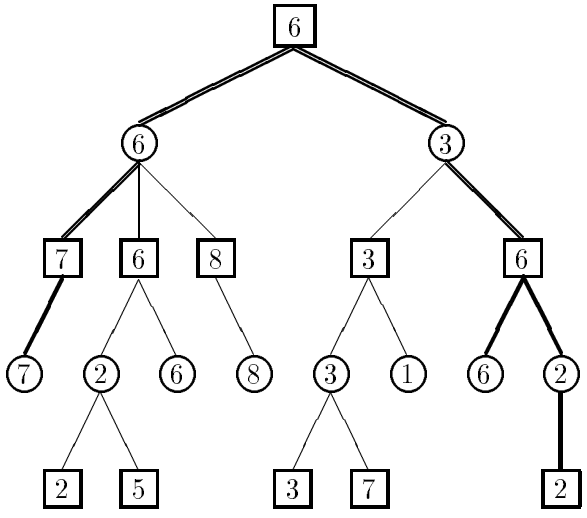

Figure 1: A game tree with f-values.

upon solution trees. A general cut-off criterion is the most important result. The Sections 5 and 6 link two well-known game tree algorithms to the cut-off criterion.

To conclude the opening section, some preliminaries are given. A game tree is denoted by G and its root is denoted by r throughout this paper. Given a statement related to a game tree, replacing the terms max/min by min/max yields the so-called *dual* statement.

2 Solution Trees

A *strategy* of Max in a tree G is defined as a subtree, including in each max node 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 such a subtree, Max is able to calculate the outcome for each series of choices that his opponent 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 have called a strategy for Max, will also be referred to as a *min solution tree*, or briefly a *min tree*. Dually a strategy for Min is defined, also called a *max solution tree* or a *max tree*. In Figure 1 the bold edges generate a max tree. A max tree is denoted by T^+ and a min tree by T^- in this paper.

Given a min solution tree, the most beneficial choice for Min in each min node is a move towards a terminal with minimal value. Consequently, in a given min tree (Max strategy) T^-, the profit for Max under optimal play of Min is equal to the minimum of all pay-off values in the terminals of T^-. Therefore we introduce the following function g for a max tree T^+ and a min tree T^-:

$$g(T^+) = \max\{f(p) \mid p \textit{ is a terminal in } T^+\} \quad (2.1)$$

$$g(T^-) = \min\{f(p) \mid p \textit{ is a terminal in } T^-\} \quad (2.2)$$

The intersection of a max tree T^+ and a min tree T^- consists of exactly one path. The g-definition implies that $g(T^-) \leq f(p_0) \leq g(T^+)$, where p_0 denotes the terminal at the end of the intersection path. It follows that $g(T^-) \leq g(T^+)$ for any two solution trees T^+ and T^- in a game tree.

Suppose that the Max player confines himself to a certain tree T^-. Then Max achieves a pay-off of $g(T^-)$, if Min replies consistently towards a terminal with value equal to $g(T^-)$. If Min deviates from a path towards a terminal equal to $g(T^-)$, Max gets a higher pay-off. Hence, $g(T^-)$ is the guaranteed pay-off for Max playing in T^-. It follows that the highest attainable pay-off for Max is equal to the maximum of the values $g(T^-)$ in the set of all min trees T^-. Dually, the most beneficial pay-off from the viewpoint of Min is equal to the minimum of the values $g(T^+)$ in the set of all max trees T^+. Since the guaranteed pay-off is equal to the game value by definition, we come to the following equality holding in each node n of a game tree:

$$\begin{aligned} f(n) &= \max\{g(T^-) \mid T^- \textit{ a min tree rooted in } n\} \\ &= \min\{g(T^+) \mid T^+ \textit{ a max tree rooted in } n\} \end{aligned}$$

This equality can be proved formally by means of induction on the height of n. Since this equality is due to Stockman [15], it will be referred to as *Stockman's theorem* in this paper.

3 The Search Tree

So far, we were dealing with complete game trees. However, in every game tree algorithm the tree is built up step by step. At any time during execution a subtree of the game tree has been generated. Such a subtree is called a *search tree*. We assume that, as soon as at least one child of a node n is generated, all other children of n are also added to the search tree. If the children of a node n have been generated, n is called *expanded* or *closed*. If a non-terminal n has no children in a search tree (and hence n is a leaf in this search tree), then n is called *open*. A terminal n is called *closed* or *open* respectively, according to whether its pay-off value has been computed or not. The foregoing definitions of *open* and *closed* imply, that an open leaf in a search tree either is a non-terminal, whose children have not been generated yet, or is a terminal, whose game value has not been computed yet. Obviously, every closed leaf in a search tree is a terminal in the game tree.

Since $f(p)$ is not known yet in an open node p of a search tree S, the minimax function cannot be applied in S. To get an idea of the game values we assign two preliminary values to each open leaf. First, we assign $+\infty$ as a preliminary value. This gives rise to a function f^+ in a search tree S, defined as the minimax function in S assuming $f^+(p) = +\infty$ as game value in each open leaf p and $f^+(p) = f(p)$ in each closed leaf. Second, we assume $-\infty$ as game value in the open leaves. The related minimax function is called f^-. In every node n the inequality $f^-(n) \leq f(n) \leq f^+(n)$ holds, which can be shown by induction on the height of n. See Figure 2 for an instance of a search tree derived from

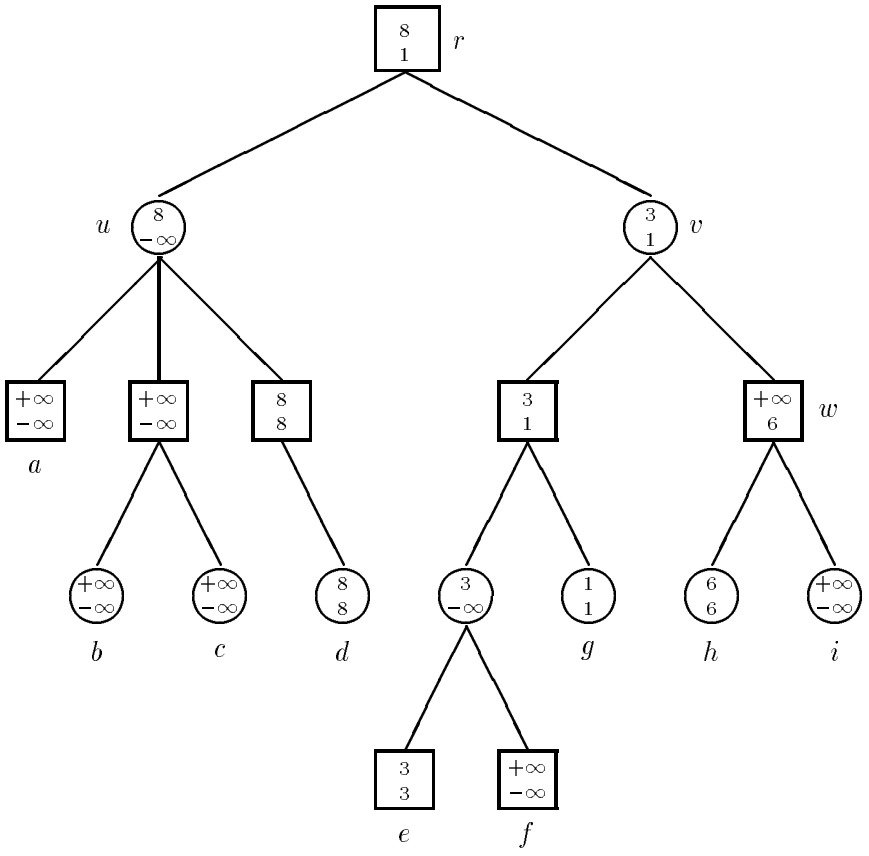

Figure 2: A search tree derived from Figure 1.

Figure 1. The nodes a, b, c, f and i are open leaves, whereas d, e, g and h are closed leaves (terminals that have their game values evaluated). In each node n of Figure 2 the top value denotes $f^+(n)$ and the bottom value denotes $f^-(n)$.

In a search tree with minimax function f^+ Stockman's theorem can be applied. Likewise, this theorem can be applied to the f^--function. To rule out the annoying nodes with infinite values, we introduce a new definition. For a max and a min tree in a search tree this new g-definition will be given below (similar to the c-function in [6]). This definition is a generalization of definitions (2.1) and (2.2), which only hold for solution trees in a complete game tree, i.e., a tree with solely closed nodes.

$$g(T^+) = \max\{f(p) \mid p \textit{ is a } \text{closed } \textit{terminal in } T^+\} \tag{3.1}$$

$$g(T^-) = \min\{f(p) \mid p \textit{ is a } \text{closed } \textit{terminal in } T^-\} \tag{3.2}$$

Applying Stockman's theorem to f^+ and f^- respectively leads to the equalities below. Although Stockman's theorem deals with the old g-definition, these equalities are also valid for the new g-definition. By a *closed solution tree* we mean a solution tree in a search tree with solely closed leaves.

$$f^+(n) = \min\{g(T^+) \mid T^+ \textit{ is a } \text{closed } \textit{max tree with root } n\} \tag{3.3}$$

$$= \max\{g(T^-) \mid T^- \textit{ is a min tree with root } n\} \tag{3.4}$$

$$f^-(n) = \max\{g(T^-) \mid T^- \textit{ is a } \text{closed } \textit{min tree with root } n\} \tag{3.5}$$

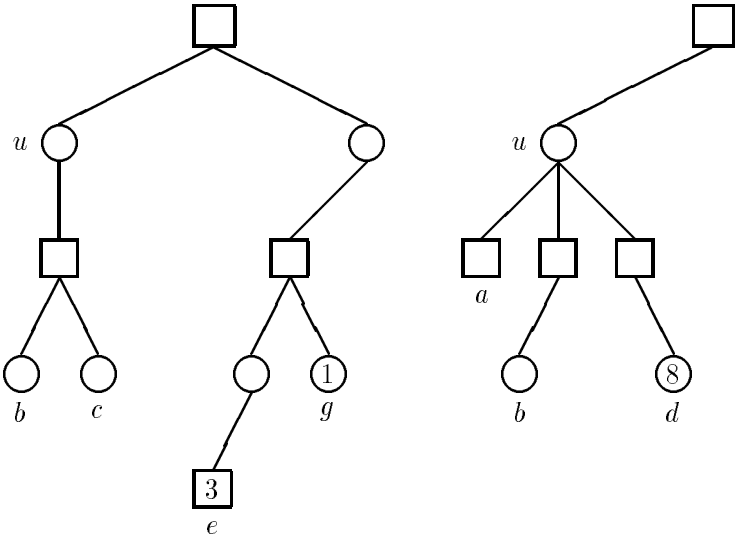

Figure 3: A max and a min tree, derived from Figure 2.

$$= \min\{g(T^+) \mid T^+ \text{ is a max tree with root } n\} \tag{3.6}$$

Here we assume that the minimum/maximum of the empty set is $+\infty/-\infty$.

We will comment on the formulas for f^+. (The formulas for f^- are dual.) For a min tree T^- with $+\infty$ as the game value in the open nodes and for a closed max tree T^+, the old and the new g-definition yield the same value. For a non-closed max tree T^+, the g-value in old sense equals $+\infty$. Consequently, the above equalities for $f^+(n)$ should be regarded as an application of Stockman's theorem, where non-closed max trees (with infinite g-value) are left out of consideration in the right-hand side of (3.3).

4 A General Theory

In this section a general theory on game tree algorithms is developed. A key role is played by the notions *alive* and *dead*.

4.1 Alive Nodes

In this subsection the definition and the significance of the notion *alive* is discussed. The definition of an *alive* node is as follows. *A node n in a search tree S is called alive if n is on the intersection path of a max tree T^+ and a min tree T^- (either rooted in r) with $g(T^+) < g(T^-)$.*

Given an alive node n in a search tree S, we can construct a game tree $G_n \supset S$, whose game value can only be obtained if one particular open descendant[1] of n is expanded. The construction of G_n proceeds as follows. Denote the actual values $g(T^+)$ and $g(T^-)$ by g_1 and g_2 respectively. The leaf p_0 at the end of the intersecting path must be open, since, if it was not, we would have

[1] In this paper each node n is assumed to be its own descendant.

$g(T^-) \leq f(p_0) \leq g(T^+)$. Choose a value f_0 with $g_1 \leq f_0 \leq g_2$. Define $f(p_0) = f_0$ and $f(p) \leq g_1$ for any open node $p \neq p_0$ in T^+ and $f(q) \geq g_2$ for any open node $q \neq p_0$ in T^-. To complete G_n, the other open nodes in S (if any) are closed arbitrarily. After being extended, both T^+ and T^- have g-values equal to f_0. Stockman's theorem entails, that the game value of G_n equals f_0. As long as p_0 is not closed in G_n, T^+ and T^- satisfy $g(T^+) = g_1 < g_2 = g(T^-)$ and every value in the range $[g_1, g_2]$ is still achievable as game value for r.

The above construction is illustrated using the Figures 2 and 3. Figure 3 shows solution trees T^+ and T^- with $g(T^+) = 3$ and $g(T^-) = 8$. Node u is on the intersecting path and is therefore alive. The game tree G_u is constructed by defining $f(b) = f_0$ with $f_0 \in [3, 8]$, $f(c) \leq 3$ and $f(a) \geq 8$. The nodes f and i in Figure 2 may be closed arbitrarily.

4.2 Definition of the h-functions

In this subsection, we give an alternative definition for the notion *alive*. This definition implies a practical method to establish whether a node is alive, using the so-called h-functions. These h-functions in a search tree S are defined as:

$$h^-(n) = \min\{g(T^+) \mid T^+ \textit{ a max tree in } S \textit{ through } r \textit{ and } n\} \tag{4.1}$$

$$h^+(n) = \max\{g(T^-) \mid T^- \textit{ a min tree in } S \textit{ through } r \textit{ and } n\} \tag{4.2}$$

It is easily seen that a node n is *alive* iff $h^-(n) < h^+(n)$.
As a result of (3.4) and (3.6) respectively, the definition of the h-functions reduces in the root to $h^+(r) = f^+(r)$ and $h^-(r) = f^-(r)$. Since every solution tree considered in the above definition goes through r, r has a maximal h^+-value and a minimal h^--value in any given search tree S.
Extending the equality $f^+(r) = h^+(r)$, we will give formulas for the h-functions in any other node. Those formulas are of highly practical significance. To this end we need a new notion. Denote by $AMAX(n)/AMIN(n)$ the set of max/min nodes, that are proper ancestors of n. See Figure 4 for illustration. The set $AMIN\text{-}C(n)$ is defined as the set of children of the nodes in $AMIN(n)$, as far as these children are outside the path from r to n. The dual notion is $AMAX\text{-}C(n)$. The following interesting formulas hold for the h-values in a node n of a search tree S.

$$h^-(n) = \max\{f^-(m) \mid m \in \{n\} \cup AMAX(n)\} \tag{4.3}$$

$$h^+(n) = \min\{f^+(m) \mid m \in \{n\} \cup AMIN(n)\} \tag{4.4}$$

We only prove (4.3). As a result of (3.6), every node $m \in \{n\} \cup AMAX(n)$ is the root of a max tree T_m with $g(T_m) = f^-(m)$. In the superposition of all those trees T_m, we choose arbitrarily a max tree T^+ through r and n. By the definition of h^-, we have $h^-(n) \leq g(T^+)$. Every terminal value in the superposition under consideration is bounded above by $\max\{f^-(m) \mid m \in \{n\} \cup AMAX(n)\}$. Hence, $g(T^+)$ and also $h^-(n)$ are bounded above by that value as well. By definition, $h^-(n)$ is equal to the g-value of a max tree T_0 through r and m. Every node $m \in \{n\} \cup AMAX(n)\}$ is the root of a subtree T' of T_0. It follows that $h^-(n) = g(T_0) \geq g(T') \geq f^-(m)$, where the latest inequality is due to (3.6). It

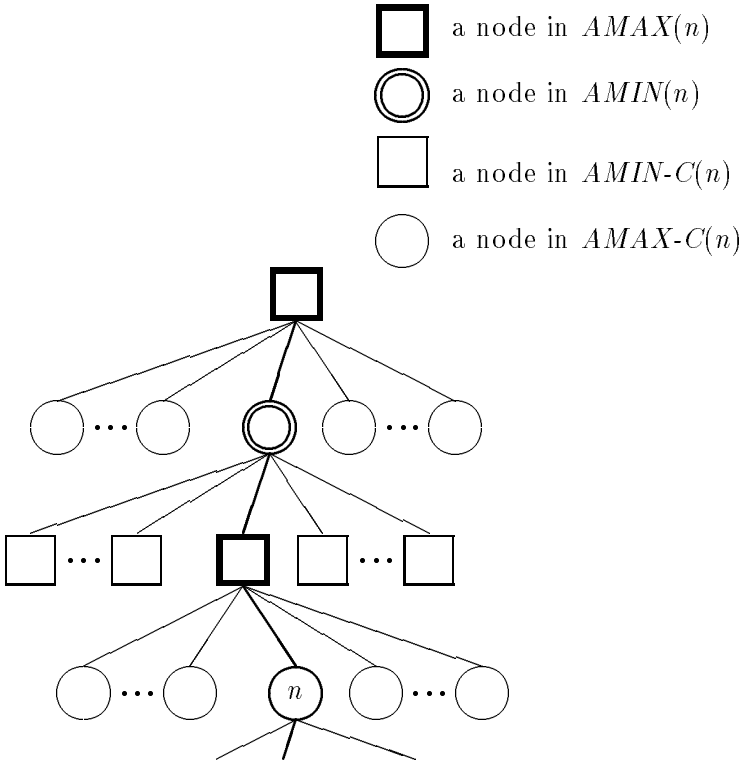

Figure 4: Definition of node sets.

follows that $h^-(n) \geq \max\{f^-(m) \mid m \in \{n\} \cup AMAX(n)\}$. The combination of some of the inequalities provides the desired result.

A similar reasoning can be given, when a max tree T_m with $g(T_m) = f^-(m)$ is considered in each node $m \in \{n\} \cup AMAX\text{-}C(n)\}$. This results into (4.5). The second formula is the dual counterpart.

$$h^-(n) = \max\{f^-(m) \mid m \in \{n\} \cup AMAX\text{-}C(n)\} \quad (4.5)$$

$$h^+(n) = \min\{f^+(m) \mid m \in \{n\} \cup AMIN\text{-}C(n)\} \quad (4.6)$$

4.3 Dead Nodes

A node that is not alive is called dead. It is easily shown that every ancestor of an alive node is alive as well. As a result, a descendant of a dead node is dead. In terms of the h-functions, we may state that n is dead iff $h^-(n) \geq h^+(n)$. The h-functions will be utilized in this subsection to derive some properties of dead nodes.

Consider a given max tree T^+ including an open dead node p. By the definition of h^- we have $g(T^+) \geq h^-(p)$. There is a node $m \in AMIN(p)$ with $f^+(m) = h^+(p)$ due to (4.4), and m is the root of a closed max tree T' with $g(T') = f^+(m)$ due to (3.3). Hence, $h^+(p)$ is associated not only with a min tree through r and p (by definition), but also with a max tree rooted in a node $m \in AMIN(p)$. We perform the

following transformation to the given tree T^+. Remove the subtree below m from T^+ and append T' to T^+ in m. Since $g(T^+) \geq h^-(p) \geq h^+(p) = f^+(m) = g(T')$, the g-value does not increase by this transformation. Since T' is a closed solution tree, the transformed tree has solely closed leaves below m. In a similar way, any other open dead node can be eliminated from T^+. The resulting tree does not include any open dead node and its g-value does not exceed the original value $g(T^+)$.

To illustrate the above transformation, see Figure 2. It is easily seen using (4.3) and (4.4) that $h^-(i) = 6$ and $h^+(i) = 3$, meaning that i is dead. Any max tree T^+ through i has g-value ≥ 6. The subtree below v in such a tree T^+ may be replaced by the max tree rooted in v and ending up in the terminals e and g.

Given an alive node n, the above transformation can be applied to a max tree associated with $h^-(n)$, i.e., a max tree T^+ such that $g(T^+) = h^-(n)$. This results into a new tree avoiding open dead nodes and not exceeding $h^-(n)$ by its g-value. Since $f^+(m') \geq h^+(n) > h^-(n) = g(T^+)$ for every node $m' \in \{n\} \cup AMIN(n)$, replacing a subtree of T^+ rooted in m' with a closed subtree would raise the g-value. We conclude that no node from $\{n\} \cup AMIN(n)$ is involved in the above transformation of T^+ eliminating open dead nodes. It follows that, given an alive node n, a solution tree T^+ through n associated with $h^-(n)$ can be found avoiding any open dead node. As long as the algorithm does not expand an open node of this tree T^+, the value $h^-(n)$ is unaffected. Therefore, while expanding a dead node, the h^--value of any alive node in a search tree is not affected. For reasons of duality, any alive h^+-value isn't affected either.

4.4 Main Theory

We now come to our theory consisting of four observations.

a) We have shown in subsection 4.1 that, if n is alive, a game tree G_n can be constructed, in which $f(r)$ is unknown as long as one particular open descendant p_0 of n is not expanded. The conclusion is that any alive node n cannot be discarded.

b) As a result of a), an algorithm must continue as long as the search tree contains any alive nodes. Therefore, the algorithm may only stop when all nodes in the search tree are dead. We might say therefore, that any game tree algorithm actually aims at killing the entire search tree.

c) All nodes in a search tree S are dead iff $g(T^+) \geq g(T^-)$ for any two solution trees T^+ and T^- in S. As a result of (3.4) and (3.6), this condition is equivalent to the equality $f^-(r) = f^+(r)$, which is the stop criterion of every game tree algorithm therefore. When the condition $f^-(r) = f^+(r)$ is achieved, both a closed max tree and a closed min tree with g-value equal to $f(r)$ are present in the search tree. The superposition of these two trees is called a *critical tree*. This notion has been introduced in [7], with a totally different definition however. Since the algorithm must continue until $f^-(r) = f^+(r)$ holds, we conclude that every game tree algorithm

needs to build a critical tree.
The intersection of the max and the min tree in a critical tree is a path with constant f-value, as can easily be shown using Stockman's theorem.

d) Expanding descendants of a dead node does not affect the h-values of any alive node, as we have shown in subsection 4.3. Consequently, an alive node can only be killed by expanding an alive node. For a game tree algorithm to achieve its goal, every node needs to be killed. Therefore, expanding a dead node is useless. Since every dead node has solely dead descendants, a dead node along with the subtree underneath may be neglected during the search.

Notice that the notes a) and d) constitute a general cut-off criterion for game tree algorithms: alive nodes must be respected, dead nodes may be neglected. Note c) describes the situation on termination of a game tree algorithm.

5 Alpha-beta revisited

An extensive treatment of the alphabeta procedure can be found in [7]. The same paper also includes a historical survey of the rise of this procedure. In this section, we will show, how the alphabeta algorithm complies with our theory on solution trees. The main result is the characterization of alphabeta, presented in Theorem 5.2. Figure 5 shows the code of the *alphabeta* procedure. We present a postcondition of alphabeta, which extends the postconditions in [7] and [3], in that it relates the new functions f^+ and f^- to the return value of an alphabeta call. The accompanying precondition is: $\alpha < \beta$.

Theorem 5.1 *The following postcondition holds for an alphabeta call with return value v.*

$$\begin{aligned} \textit{low failure:} \quad & v \leq \alpha \quad && \Rightarrow v = f^+(n), && (5.1)\\ \textit{success:} \quad & \alpha < v < \beta \quad && \Rightarrow v = f(n) && (5.2)\\ \textit{high failure:} \quad & v \geq \beta \quad && \Rightarrow v = f^-(n). && (5.3) \end{aligned}$$

In case of a low or high failure, n has exactly one optimal closed max or min tree with g-value equal to $f^+(n)$ or $f^-(n)$ respectively.

Proof
We only prove the extended part of the theorem, viz. the results in case of a low or high failure. These results are proved by induction on the height of the calling tree. This means that, we show that the theorem holds for a call, under the assumption that the theorem holds for any recursive subcall to a child c. For reasons of duality, only the case that n is a max node, is studied.
Suppose that the call ends with a low failure. Then every child has been parameter in a subcall and every subcall has ended with a low failure. Then $v = \max\{v'_c \mid c \textit{ a child of } n\}$, where v'_c denotes the result of the subcall with parameter c. Since each v'_c corresponds to a unique max tree (assumed by induction), v corresponds to a unique max tree too.
Suppose the call ends with a high failure. Then the last subcall, say to c_0, ended

```
function alphabeta(n, α, β);
if terminal (n) then  v :=f(n);
else if max(n) then
        v := −∞;
        α′ := α;
        c :=first(n);
        while v < β and c ≠ ⊥ do
              v′ :=alphabeta(c, α′, β);
              v := max(v, v′);
              α′ := max(α′, v′);
              c := next(c);
else if min(n) then
        v := +∞;
        β′ := β;
        c :=first(n);
        while α < v and c ≠ ⊥ do
              v′ :=alphabeta(c, α, β′);
              v := min(v, v′)
              β′ := min(β′, v′);
              c := next(c);
return v;
```

Figure 5: The alpha-beta procedure

with a high failure with return value v'_{c_0}. The return value of the main call is $v = v'_{c_0}$. The unique optimal min tree for c_0 (assumed by induction) is also the unique optimal min tree for n, since the elder children provided a smaller return value. □

The exact value of a game tree is computed by a call *alphabeta*$(r, -\infty, +\infty)$. When this call is executed and a node n is parameter in a subcall, n is the left-most open alive node in the actual search tree, as we will show in the following theorem. So, a characterization of the alphabeta algorithm as depth-first instance is obtained.

Theorem 5.2 *Suppose a call alphabeta*$(r, -\infty, +\infty)$ *is performed. Then at every nested call alphabeta*(n, α, β)*, the relation* $h^-(n) = \alpha < \beta = h^+(n)$ *holds and every node to the left of* n *is dead.*

Proof
This is proved by induction on the depth of n. The theorem holds trivially at depth 0. We will show, that the theorem holds at depth $d+1$, provided that it holds at depth d ($d \geq 0$). Assume n is max node at depth d with $h^-(n) = \alpha$ and $h^+(n) = \beta$. The case that n is a min node is dual.
Notice that the oldest child c of n has $h^-(c) = \alpha = \alpha'$ and $h^+(c) = \beta$, when the subcall *alphabeta*(c, α', β) starts. Now, we discuss some properties of the subcalls *alphabeta*(c, α', β) that are followed by a next one. So the subcall to the youngest child of n and the subcalls ending with $v' \geq \beta$ are left out of consideration for

our goal. After a subcall ending with $\alpha' < v'$, the h^--value for n and all its children change into v', and hence α' is updated. After any subcall ending with $v' \leq \alpha'$, the h-values for n and its children are unaffected and hence α' remains unchanged. We conclude that the h-values for any call at depth $d+1$ are in accordance with the theorem.
At any subcall with parameter c, the elder brothers (if any) of c satisfy $f^+(c) \leq \alpha' = h^-(c)$ and are dead therefore. By the induction hypothesis, any other node to the left of c is already dead, when n is expanded. □

The set *AMIN-C*(n) for a given n can be split up into two sets, called *AMIN-C-Left*(n) and *AMIN-C-Right*(n), containing the nodes to the left or right respectively of n. Likewise, *AMAX-C*(n) can be split up. Let S_0 denote the search tree when a nested call *alphabeta*(n, α, β) is executed (and when Theorem 5.2 applies). As an easy extension to Theorem 5.2, we can show, that any $x \in$ *AMIN-C-Right*(n) is open in S_0. This enables us to determine a static value for $\beta = h^+(n)$ in S_0, as we will show. Suppose that the game tree is completed under the nodes in *AMIN-C-Left*(n). Expanding descendants of nodes in *AMIN-C-Left*(n) does not affect $h^+(n)$, since those nodes are dead. Hence, $h^+(n)$ remains equal to β. In the enhanced tree, each descendant x of a node in *AMIN-C-Left*(n) has $f^+(x) = f(x)$. Applying (4.6) in the enhanced tree results into $\beta = h^+(n) = \min\{f(x) \mid x \in$ *AMIN-C-Left*$(n)\}$. A dual formula holds for α. Here, we re-cover the formulas presented in [1].

6 The MTD-algorithm

In this section, we discuss the MTD-algorithm. First we present the algorithm. Next, we give a characterization for one particular instance of MTD, viz. MT-SSS.

6.1 The Description of the MTD algorithm

MTD stands for *Memory Test Driver*. The algorithm has its roots in the *Test* routine, introduced in [9]. This routine 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 γ, equal to the greater parameter $\beta = \alpha + 1$. The assumption is made that any game value is integer, so no game values between $\gamma - 1$ and γ are assumed. Consequently, the *success* ending in the postcondition cannot apply. So, the return value v of the *Test* procedure establishes either a lower bound $v = f^-(n)$ or an upper bound $v = f^+(n)$.
We use an extended *Test* procedure. When several *Test* calls are executed successively (each with a different null window), the search tree can be retained in memory and bounds like $f^+(n)$ and $f^-(n)$ can be stored at each node n. The *Test* procedure, which exploits previous bounds and stores new bounds, is named MT (Memory Test). The code of MT is presented in Figure 6. The code of the MTD algorithm including a number parameter f is shown in Figure 7. In most actual applications of game tree search, a so-called transposition table is maintained, containing all positions visited earlier. This table can also serve to register bounds to the nodes of the search tree. The bounds are stored into the

```
function MT(n, γ);
if terminal (n) then
        if open(n) then v :=eval(n)
        else v := n.f+ or n.f−;
else
        if open(n) then generate the children of n;
        if max(n) then
                v := −∞;
                c :=first(n);
                while v < γ and c ≠ ⊥ do
                        if c.f+ ≥ γ then v' :=MT(c, γ) else v' := c.f+ ;
                        v := max(v, v');
                        c := next(c);
        if min(n) then
                v := +∞;
                c :=first(n);
                while v ≥ γ and c ≠ ⊥ do
                        if c.f− < γ then v' :=MT(c, γ) else v' := c.f− ;
                        v := min(v, v');
                        c := next(c);
if v < γ then n.f+ := v else n.f− := v;
return v;
```

Figure 6: The code of the function MT

fields $n.f^+$ and $n.f^-$ of a record associated to each node n of the transposition table In case of a high failure $n.f^-$ is set; in case of a low failure a value $n.f^+$ is set. Hence, we assume that only one bound per node is stored. As soon as $n.f^+$ is set, the alternate variable $n.f^-$ is undefined, even though it may have been given a value in the past. A variable that is undefined, is assumed to have an infinite value.

Now, we will prove that $f(r) = v$ upon termination of the MTD-algorithm. The situation that the MT call in the initial step ends with a low failure ($f^+(r) = v < \gamma$) is discussed. The alternate case is dual. The first MT call in the main loop starts with $f^+(r) = \gamma$. A low failure ending of this call amounts to $f^+(r) = v < \gamma$. A high failure is equivalent to $f^-(r) = v = \gamma$. (Given the start condition $f^+(r) = \gamma$, an ending with $f^-(r) = v > \gamma$ cannot happen.) Therefore, a high failure causes the stop criterion to apply. When a low failure happens, the MT call in the subsequent iteration starts with $f^+(r) = \gamma$. For each iteration a similar reasoning holds. When the stop criterion holds after any iteration, the condition $f^+(r) = \gamma$ holding at the start of latest MT call, along with the stop criterion $f^-(r) = v = \gamma$ yields $f(r) = v$.

The instance with $f = \infty$ is called MT-SSS, due to its similarity with SSS*. This similarity has been clarified in [13]. Extensive tests with MT-SSS, MT-Dual and MTD(f) are described in [12, 13]. It turns out that, combining MTD(f) with

```
function MTD(r, f);
initial step:
γ := f;
v :=MT(r, γ);
main loop:
if v < f then
    repeat
        γ := v;
        v :=MT(r, γ);
    until v = γ;
if v ≥ f then
    repeat
        γ := v + 1;
        v :=MT(r, γ);
    until v = γ − 1;
return v;
```

Figure 7: The code of MTD(f).

iterative deepening and taking the value of each previous iteration as the next f-value results into a fast and efficient algorithm.

6.2 Characterization of MT-SSS

In this subsection, we will characterize MT-SSS. The main result is expressed by Theorem 6.1.
In Lemma 6.1, we use the notions left and right child respectively of a max tree. For any max tree T^+, a node x is called a left node for T^+, if x is an elder brother of a node c being the single child in T^+ of a min node. Analogously a right node of T^+ is defined.

Lemma 6.1 *For each nested call $MT(n, \gamma)$ during execution of MT-SSS, the following pre- and postcondition holds.*
Precondition*:*
If n is not open then $f^+(n) = \gamma$ and there is a unique optimal closed max tree T^+. Every $x \in T^+$ has $f^+(x) = x.f^+$ and every left node y of T^+ satisfies $y.f^- = f^-(y) > \gamma$.
Postcondition*:*
If a call $MT(n, \gamma)$ ends with a low failure ($v < \gamma$), the newly constructed search tree below n contains a unique optimal closed max tree T_1^+. Every $x \in T_1^+$ has $f^+(x) = x.f^+$ and every left node y of T_1^+ satisfies $y.f^- = f^-(y) \geq \gamma > v$.

Proof(outline)
The proof of the postcondition is by induction on the height of the calling tree, similarly to the proof of Theorem 5.1. The proof holds under the condition that the precondition holds.
In each (sub)call in the initial step, the node parameter n is open. The precondition for each call $MT(r, \gamma)$ in the main loop is a result of the postcondition of

the previous call. The proof of the precondition for a nested call during the main loop is by induction on the depth of n, similarly to the proof Theorem 5.2. □

Lemma 6.2 *For any call MT(n, γ) during execution of MT-SSS, each node x in AMAX-C-Left(n) has $f^+(x) < \gamma$ and each node x in AMIN-C-Left(n) has $f^-(x) \geq \gamma$.*

Proof(sketch)
The proof is by induction on the depth of n, similarly to the proof of Theorem 5.2. The proof utilizes the fact (along with its dual counterpart), that a child c of max node n is only parameter in a subcall, when every subcall to an elder child c' has ended with a low failure or has $c'.f^+ < \gamma$ in the transposition table. □

Theorem 6.1 *For any call MT(n, γ) during execution of MT-SSS, $h^+(n)$ is maximal in S and every node to the left of n is dead or has a lower h^+-value.*

Proof
In a call $MT(r, \gamma)$, the algorithm descends from the root to the search tree. In each closed max node n, a closed child c is chosen, such that $f^+(n) = f^+(c) = \gamma$, and in each open max node, any parameter c of a subcall has $f^+(n) = f^+(c) = \infty$. We conclude, that the transition from a closed node n to an open child c is made in a min node n. Now, it can be shown by induction, that an open min tree T^- with $g(T^-) = \gamma$ crosses every node n being parameter in a nested MT call. It follows that $h^+(n) \geq \gamma$. At the start of a call $MT(r, \gamma)$, each min tree in the search tree has g-value $\leq f^+(r) = \gamma$. The conclusion is that $h^+(n) = \gamma$ and no node has a higher h^+-value.

If a node to the left of n has an ancestor x in *AMAX-C-Left*(n), then $f^+(x) < \gamma$ and every min tree through r and any descendant of x has g-value $< \gamma$. If ancestor x is in *AMIN-C-Left*(n), then $f^-(x) > \gamma$ and every max tree through any descendant of x and r has g-value $\geq \gamma$. For those nodes x, $h^+(x) \leq f^+(r) \leq \gamma \leq h^-(x)$ and hence, any descendant of x is dead. □

Notice that a node n being expanded during MT-SSS is not guaranteed to be alive. By theorem 6.1, a closed parameter n in a nested call satisfies $n.f^+ = f^+(n) = \gamma$. However, the inequality $f^+(n) > f^-(n)$ needs not to hold. Instead, n may have identical boundary values. The most trivial occurrence of such a parameter is a terminal n that is revisited. It is also possible that n is an inner node or even $n = r$. This is illustrated by Figure 8. This tree may be (a part of) the search tree after a MT call during MT-SSS. In this tree, b is a closed terminal with $f(b) = 3$, $f^+(a) > 3$ and n' is still open. It is easily seen that $f^+(a) = f^+(n) = f^+(m) = f^+(r) > 3$ and $f^-(b) = f^-(n) = 3$. In the next iteration, we have a call $MT(r, \gamma)$ with $\gamma = f^+(r) > 3$. Supposing that the subcall to a ends with return value 3, we get $f^+(a) = f^+(n) = f^+(m) = f^+(r) = 3$. Since $f^+(n) = f^-(n) = 3$, everything below n is dead. Nevertheless, MT-SSS performs a new iteration, which expands open descendants of a and revisits b. Assume that n is the single child of m, i.e, n' is assumed to be removed from the game tree. When the above situation with $f^+(a) = f^+(n) = f^+(m) = f^+(r) = 3$

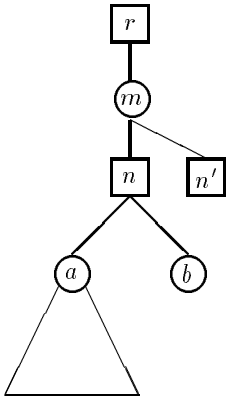

Figure 8: Side effects on boundaries.

is reached, then also $f^-(r) = 3$ and r is dead. Nevertheless, MT-SSS continues with a call MT$(r, 3)$. So, it turns out that even a dead root can be parameter in an MT call.

In order to make sure that only alive nodes are expanded, we must store other bounds at some nodes n in the transposition table. For the root, $f^+(r)$ and $f^-(r)$ must be stored and the stop criterion must be converted into $r.f^+ = r.f^-$. Inspecting closely the code in Figure 6, we see, that only the f^+-value is looked up in a child of a max node. A dual situation holds in a min node. Therefore, $f^-(n)$ must be stored in every max node n in the transposition table, and $f^+(n)$ must be stored in every min node. With these enhancements in the code of MT-SSS, the condition $f^+(n) \geq \gamma > f^-(n)$ will hold for every node n visited by an MT call. Consequently $h^+(n) = \gamma > h^-(n)$ holds and hence, only alive nodes are visited. This implies, among others, that a closed terminal will never be re-visited by an MT call.
Storing the new bounds requires, that $f^-(n)$ is also available after a low failure in a max node n, and dually, $f^+(n)$ is available after a high failure in a min node n. This can only be achieved, if the procedure *Test* returns two values, viz. $f^+(n)$ and $f^-(n)$, rather than one value v which equals either $f^+(n)$ or $f^-(n)$. These return values are computed using their minimax properties.
As a matter of fact, every enhancement should be reflected in the code of MT and MTD(f).

Theorem 6.1 showed that $h^+(n) = \gamma$ for any call $MT(n, \gamma)$ during MT-SSS. It can be shown by extending Theorem 6.1 and Lemma 6.2, that every node in *AMIN-C-Right*(n) is open. When n itself is open at an MT call, parameter γ has a value that can be determined statically. We can compute this value in exactly the same way, as we did for parameter β in the alphabeta algorithm. It follows that γ and β have identical values. Again, we have re-covered an old result, see [14].

7 Concluding remarks

We have developed a new theory built around the notion of solution tree. It was shown, how alphabeta and MT-SSS fit into this theory. Two fairly important algorithms are not discussed here, viz. negascout and proof number search. The role of solution trees in those algorithms was described in [2]. Due to Stockman's theorem, the notion of strategy or solution tree has pushed aside the minimax function. To our feeling, the former notion is closer to the human intuition than the latter is.

References

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

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

[3] Finkel, R.A., Fishburn, J.P. Parallelism in alpha-beta search. *Artificial Intelligence*, 19:89-106, ISSN 0004-3702, 1982.

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

[5] Toshihide Ibaraki, *Search Algorithms for Minimax Game Trees,* Conference paper at *Twenty years NP-completeness*, Sicily, June 1991.

[6] 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.

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

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

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

[10] W. Pijls and A. 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.

[11] W. Pijls and A. de Bruin, *A theory of game trees, based on solution trees*, in: F. Plasil et al. (eds.), Proceedings Sofsem '97, Theory and Practice of Informatics, LNCS, vol. 1338, Springer-Verlag, 1997, pp. 539-546.

[12] 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.

[13] Aske Plaat, Jonathan Schaeffer, Wim Pijls and Arie de Bruin, *A Minimax Algorithm Better than SSS**, In: Artificial Intelligence 84 (1996), pp. 299-337.

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

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