Two Algorithms for Decision Tree Search

by
Runping Qi and David Poole

Technical Report 92-4
February, 1992

Department of Computer Science
The University of British Columbia
Vancouver, B. C. V6T 1Z2
Canada

email: qi@cs.ubc.ca, poole@cs.ubc.ca

©1992 Runping Qi and David Poole

Abstract

In this paper two algorithms for decision tree search are presented. The basic idea behind these algorithms is to make use of domain dependent information, in the form of an evaluation function as that used by AO*, along with a search mechanism similar to the alpha–beta technique for minimax trees. One of the advantages of our algorithms over AO* is that our algorithms need only linear space. The solution computed by the first algorithm can be either optimal or sub–optimal, depending on the *admissibility* of the evaluation function. The second algorithm is an approximate algorithm which can cast a tradeoff between computational efficiency and solution quality. Some results are presented on the correctness of the algorithms and on the quality of the solutions computed by the algorithms.

Description: decision tree searching

1 Introduction

Decision trees are a very commonly used representation tool for multi–stage decision problems. A decision tree representation of a sequential decision problem can be obtained directly from the problem [Raiffa, 1968] or derived from the influence diagram of the problem [Howard and Matheson, 1981]. A natural computational problem related to decision trees is to search for the optimal solution.

AO* [Pearl, 1984] is a well known heuristic algorithm that can be used for decision tree search. However, a major drawback of AO* is that it needs exponential space. In this paper, we develop two heuristic algorithms for decision tree search. The key idea behind the algorithms is to use domain dependent heuristic information and a branch–bound search technique, similar to the alpha–beta mechanism [Knuth and Moore, 1975] for minimax tree search, for decision tree search. The domain dependent heuristic information used by these algorithms is in the form of three functions: one *evaluation function* and two *ordering functions*. The evaluation function is similar to that used by AO*. One of the advantages of our algorithms is that they need only linear space.

The solution computed by the first algorithm can be either optimal or sub–optimal, depending on the *admissibility* of the evaluation function. The second algorithm is an approximate algorithm derived from A1 and can cast a tradeoff between computational efficiency and result quality. We also present some results on the correctness of the algorithms and the quality of the solutions computed by the algorithms. To our best knowledge, these algorithms are the first of their kind for decision tree search. In [Qi, 1992], these algorithms are applied to a decision problem arising from navigation in uncertain environments[Qi and Poole, 1991] [Qi and Poole, 1992].

It is interesting to note that, although there are many other techniques developed for minimax tree search (e.g., SSS* [Stochman, 1979], aspiration window [Shams *et*

al., 1991], conspiracy number [McAllester, 1988]), it seems that only the alpha–beta technique can be conveniently applied to decision tree search [1]. This is due to the difference between the ways node values are computed in decision trees and in minimax trees.

The remainder of the paper is organized as follows. The next two sections introduce some basic concepts about decision trees and decision tree search. In Sections 4 and 5, we present our algorithms for decision tree search, and give some theorems about these algorithms. The proofs of these theorems can be found in the Appendix. Finally, conclusions and the future work are discussed in Section 6.

2 Decision Trees

A *decision tree* [Raiffa, 1968] is a tree with two types of nodes: *decision nodes* and *nature nodes*. All successors of a node in a decision tree are of the same type. For the sake of convenience, we assume that the type of a node is always different from that of its parent (if it is not the root), and the root of a decision tree is always a decision node. Each decision node has a set of actions, each associated with an arc from this node to one of its children. Each action has a cost. Each nature node has a (discrete) probability distribution over its children. In other words, a probability is associated with each of the children of a nature node, and the probabilities of all the children of a nature node sum to unit. A subset of decision nodes is designated as terminals. Each terminal has a value associated with it.

A decision tree can possibly be interpreted as the representation of a process of sequential decision making . A decision node represents a situation where an agent has to select one of the actions to act. The root of a decision tree represents the initial situation. A nature node represents an uncertain situation which will eventually change to a situation represented by one of its successors with the probability associated with the successor. At a decision node, if an agent selects and takes an action, a cost associated with the action is incurred, and an uncertain situation is reached. From this uncertain situation, some new situation will be reached. This process repeats until a terminal is reached. A terminal node represents a situation where an assessment can be made. We assume that the value associated with a terminal is in the form of cost (instead of pay–off).

A *solution tree* T of a decision tree *DTREE* is a tree with the following characteristics:

1. The root of *DTREE* is the root of T;

[1] Although the treatment in algorithm A2 discussed in Section 5.2 in this paper is similar to the idea of aspiration window techniques, the cost for this treatment is that the result computed by A2 can be suboptimal.

2. If a nature node of *DTREE* is in T then all of its successors are in T;

3. If a nonterminal decision node of *DTREE* is in T then exactly one of its successors is in T.

3 Min–exp Evaluation of a Decision Tree

Let *DTREE* be a decision tree. A *min–exp evaluation* (in contrast to the minimax evaluation of a minimax tree [von Neumann and Morgenstern, 1947]) of *DTREE* is a real–valued function DT defined as follows:

1. If N is a terminal: $DT(DTREE, N) = value(N)$ where *value* is a real–valued function defined on terminals.

2. If N is a nature node and has l children, $N_1, ..., N_l$ in *DTREE*, let $p_1, ..., p_l$ be the probabilities associated with $N_1, ..., N_l$ respectively:

   $DT(DTREE, N) = \sum_{j=1}^{l} p_j * DT(DTREE, N_j)$.

3. If N is a decision node and has l children, $N_1, ..., N_l$ in *DTREE*, let $cost(N, j)$ denote the cost of the action associated with the arc from N to N_j for $j = 1, ..., l$: $DT(DTREE, N) = min_{j=1}^{l}\{cost(N, j) + DT(DTREE, N_j)\}$.

$DT(DTREE, N)$ is called the min–exp value of node N with respect to tree *DTREE*. The min–exp value of node N in tree *DTREE* can be interpreted as the minimal expected cost an agent is going to pay if it starts a sequential decision process from the situation represented by node N. Note that the above definition is applicable to a solution tree as well since a solution tree is a special decision tree. The problem considered in this paper is, *for a given decision tree DTREE, to find the optimal solution tree ST such that DT(ST, N) = DT(DTREE, N) where N is the root of ST.*

4 A Heuristic Search Algorithm

From the definition of DT, a recursive algorithm can be readily derived for computing the optimal solution of a decision tree. However, the algorithm needs to "visit" all of the nodes in a decision tree in order to compute the optimal solution.

In this section, we develop an algorithm for decision tree search. The algorithm uses a kind of domain dependent information and a pruning technique similar to the alpha–beta mechanism for minimax tree search [Knuth and Moore, 1975]. In order to develop the pruning technique, we contrast a decision tree to a minimax tree. A decision node in a decision tree can be regarded as a *min* node since we want to minimize the min–exp value of it. Consequently, a nature node is analogous to a *max* node. However,

a decision tree is different from a minimax tree in two major aspects. First, there is no cost or other information associated with the edges of a minimax tree, but in a decision tree, the information of this kind plays an important role in computing both the min–exp values of nodes and the optimal solution tree of the decision tree. Second and more importantly, the way to compute the minimax values in a minimax tree is different from the way to compute the min–exp values in a decision tree. In a minimax tree the minimax value of a *max* node is the maximum of the minimax values of its children, but in a decision tree the min–exp value of a *max* node is the expectation of the min–exp values of its children. These two differences make the original alpha–beta pruning rules not applicable to a decision tree.

Nevertheless, we still can design a similar pruning mechanism for decision trees if some *admissible evaluation functions* are available. An evaluation function for a decision tree is a function which estimates the min–exp values of the nodes of the decision tree. An evaluation function f for decision tree $DTREE$ is admissible if, for every node N in $DTREE$, $f(N) \leq DT(DTREE, N)$. As an example, the function that returns the Euclidean distance between the current and goal positions is an admissible evaluation function for the decision tree derived from a U–graph based navigation when the weights in the U–graph represent distances [Qi and Poole, 1992]. In such a decision tree, a decision node represents a navigation task of going from the current position to the goal position in a U–graph, and the min–exp value of a decision node is the minimal expected cost (distance) of the navigation task represented by the node.

4.1 The pruning mechanism

The pruning mechanism works by maintaining an upper limit on the min–exp value of every node in a decision tree. The basic idea of the pruning mechanism is: if it is known that the min–exp value of a node in a decision tree reaches its upper limit, the node, together with the subtree rooted at it, must not be in the optimal solution, thus, can be pruned. As a convention, we let N denote a non–terminal node, $N_1, ..., N_l$ denote all of the children of N, and $v, v_1, ..., v_l$ denote the min–exp values of node $N, N_1, ..., N_l$, respectively, in a given decision tree $DTREE$. Furthermore, let $c_1, ..., c_l$ denote the costs of the edges from N to $N_1, ..., N_l$, respectively, if N is a decision node, and let $p_1, ..., p_l$ denote the probabilities of $N_1, ..., N_l$, respectively, if N is a nature node.

Let f be an admissible evaluation function for $DTREE$. For any node N and an upper limit bp in $DTREE$, the pruning mechanism tries to answer the following two questions: (1) Is $DT(DTREE, N) < bp$? (2) What is $DT(DTREE, N)$ if it is less than bp? In order to illustrate the pruning mechanism, we need to consider the following cases.

- $bp \leq f(N)$. In this case, due to the admissibility of f, it can be concluded immediately that $bp \leq DT(DTREE, N)$. Thus, the subtree rooted at N need

not be searched.

- $bp > f(N)$ and N is a decision tree. In this case, the questions can be answered by searching the subtrees rooted at $N_1, ..., N_l$. Let $r_0 = bp$ and $r_i = min\{r_{i-1}, c_i + v_i\}$ for any $i, 1 \leq i \leq l$. If $r_{i-1} \leq c_i + v_i$, then $c_i + v_i$ is either no less than the upper limit for N, or no less than $c_j + v_j$ for some $j, 1 \leq j < i$. In either case, it is fruitless to search through the subtree rooted at N_i. Therefore, we can set $r_{i-1} - c_i$ as the upper limit for node N_i.

- $bp > f(N)$ and N is a nature node. In this case, a series of approximations of v, the min–exp value of N, can be obtained as the children of N are searched. Let $partial_i = \sum_{j=1}^{i} v_j * p_j + \sum_{j=i+1}^{l} f(N_j) * p_j$. $partial_i$ can be considered as the approximation of v when the values of nodes $N_1, ..., N_i$ have been obtained. It is obvious that $partial_i = partial_{i-1} + p_i * v_i - p_i * f(N_i)$ and $partial_l = \sum_{j=1}^{l} v_j * p_j = v$. Since f is admissible, $partial_{i-1} \leq partial_i$ for any $1 \leq i \leq l$. Thus, if, for some $i, 1 \leq i \leq l$, $partial_i \geq bp$, then, $v \geq bp$ for sure. Since $v_i \geq (bp - partial_{i-1})/p_i + f(N_i)$ implies $partial_i \geq bp$, we can use $(bp - partial_{i-1})/p_i + f(N_i)$ as the upper limit of node N_i.

4.2 A decision tree search algorithm

A decision tree search algorithm using the pruning mechanism discussed in the previous subsection, called A1, is shown in Fig. 1. In this algorithm, `MAXINT` is a large positive number, representing ∞; `cost(N, i)` and `prob(N, i)` correspond to c_i and p_i respectively. `f` corresponds to an admissible evaluation function, and `order-d` and `order-n` correspond to two *ordering* functions which can order the children of decision nodes and those of nature nodes respectively. These three functions are the abstraction of the domain dependent information of which A1 can make use.

The algorithm consists of two mutually recursive functions: `dnode1(N, bp)` and `nnode1`(N, bp). As we mentioned in the previous subsection, the pruning mechanism maintains an upper limit on the min–exp value of every node. In the algorithm, parameter `bp` is the upper limit for node N; variable `nbp` is the upper limit for the child to be searched next. In `dnode1`, variable `result` represents the intermediate back–up value of node N. As the children of node N are being searched, variable `result` is updated, and the upper limit (`nbp`) for the next child to be searched is computed. If the upper limit for a child is no more than the value given by the evaluation function, then the child need not be searched, thus the subtree rooted at the child is cut off. In `nnode1`, variable `partial` represents the series of approximations of the min–exp value of node N. As the children of node N are being searched, variable `partial` is updated and the upper limit for the next child to be searched is computed. It is important to note here that `partial` will never decrease as more children of a nature node are searched, due to the admissibility of the evaluation function. Therefore, as

```
dnode1(N, bp)
   if N is a terminal then
      if value(N) >= bp then return MAXINT; else return value(N);
   if f(N) >= bp then return MAXINT;
   result = bp;        j = # of children of N;
   let N1, N2, ..., Nj = order-d(N);
   for (i = 1 to j) do
       nbp = result -  cost(N, i);
       if nbp > f(Ni) then
          result = min {result; cost(N, i) + nnode1(Ni,nbp)};
   if result >= bp then return MAXINT; else return result;

nnode1(N, bp)
   j = # of children of N;
   let N1, N2, ..., Nj = order-n(N);
   partial = f(N1)* prob(N, 1) + ... + f(Nj) * prob(N, j);
   i = 0;
   while (partial < bp) and (i < j) do
     i = i + 1;
     nbp = (bp - partial)/prob(i) + f(N_i);
     partial=partial+prob(N, i)*(dnode1(Ni, nbp)-f(N1));
   if partial >= bp then return MAXINT; else return partial;
```

Figure 1: The pseudo codes of algorithm A1

soon as `partial` catches up with `bp`, it is surely known that the min–exp value of the nature node is equal to or over the upper limit. Thus no more children need to be searched and pruning happens. Since A1 is a depth first search algorithm, the size of the space it needs is linear in the depth of the tree, provided that the solution tree need not be constructed explicitly.

As an illustration of this algorithm, let us consider an example. For the purpose of convenience, we can think the algorithm, for a given decision tree, orders the tree first (using its ordering functions) and then search the ordered tree in the left–to–right order[2] (in contrast to integrating searching and ordering together). Suppose a decision tree, after being ordered by the ordering functions used by A1, is shown in Fig. 2 where we assume that all the terminals have value 10 and all the children of any nature node have the same probability (0.5). Suppose that the heuristic evaluation function f used by A1 returns zero for *every* node in the tree. Clearly, this heuristic function is admissible. The search algorithm starts from the root with ∞ as the upper limit. After node 8 is searched, the intermediate result for node 4 is 28. Thus, when node 9 is being explored, its upper limit is 3. After node 18 is explored, the approximation of the min–exp value of node 9 is 5 which exceeds its upper limit, thus node 19 is cut off. Another pruning happens right after node 10 is explored. The intermediate result of node 5 is 25. The upper limit for node 11 is negative, therefore, node 11, together with all the nodes below it is cut off. The last pruned node for this problem is node 27. Therefore, five nodes are cut off.

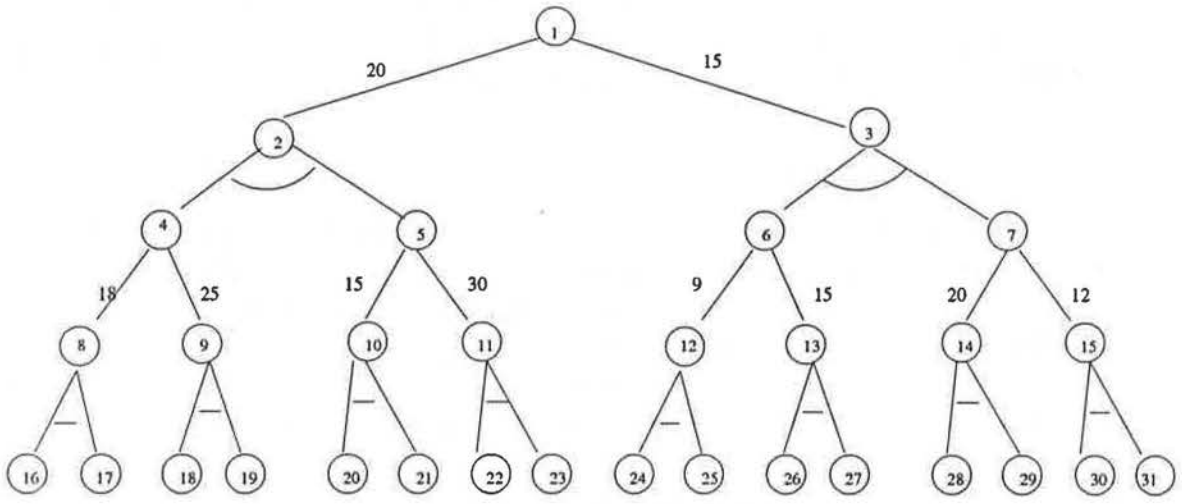

Figure 2: An illustration of the search algorithm

[2]This convention is used throughout this paper.

Let DT_1 be the function corresponding to Algorithm A1 and be defined as:

$$DT_1(N, bp) = \begin{cases} nnode1(N, bp) & \text{if } N \text{ is a nature node;} \\ dnode1(N, bp) & \text{otherwise.} \end{cases}$$

The following theorem establishes the correctness of algorithm A1.

Theorem 1 If the evaluation function used by A1 is admissible for a given decision tree $DTREE$, then DT and DT_1 satisfy:

$$DT_1(N, bp) = \begin{cases} DT(DTREE, N) & \text{if } DT(DTREE, N) < bp \\ \texttt{MAXINT} & \text{otherwise} \end{cases}$$

for any node N in the decision tree, and a number bp.

Corollary 1 If the evaluation function used by A1 is admissible for a given decision tree $DTREE$, then DT and DT_1 satisfy: $DT(DTREE, N) = DT_1(N, \infty)$ for any node N in the decision tree.

Let f_1 and f_2 be two evaluation functions. f_1 is said *more informed* than f_2 for a decision tree if $f_1(N) \geq f_2(N)$ for every nodes N in the decision tree. Suppose that both evaluation functions f_1 and f_2 are admissible and f_1 is more informed than f_2, it is clear that the performance of A1 with f_1 will be no worse than that of A1 with f_2 for the same decision tree.

As an illustration on the effect of the evaluation function, let us assume that for the same decision tree, we now have a more informed heuristic function f' defined as follows: $f'(N_i) = 16$ for $i = 2, ..., 7$ and $f'(N_i) = 7$ for $i = 8, ..., 31$. When applying A1 with f' to the decision tree ordered as shown in Fig. 2, nine nodes (nodes in the subtree rooted at nodes 9, 11, and 13) will be cut off.

4.3 Tree ordering

Note that the correctness of algorithm A1 is independent of the ordering functions. However, like minimax tree search, the order in which the children of nodes in a decision tree are searched has a great effect on the efficiency of the algorithm. Generally speaking, we want to search first the branch of a decision node that can result in the final (minimal) min–exp value of the decision node in hope that as many other branches as possible can be pruned; and we want to search first the child of a nature node which can contribute most to the min–exp value of the nature node in hope that the partial accumulation can reach bp as early as possible.

As an illustration on the effect of ordering, let us consider the decision tree shown in Fig. 3. This is the same decision tree as the one in the previous example except that the orderings of the children of some nodes are different. It can be verified that when algorithm A1 with heuristic function f is applied to this tree, nine[3] nodes (nodes 27,

[3]in contrast with five in the previous example.

29, 19, and nodes in the subtrees rooted at nodes 10 and 11) will be cut off; and when A1 with f' is applied to this tree, twenty one[4] nodes (nodes in the subtrees rooted at nodes 13, 14, and 2) will be cut off.

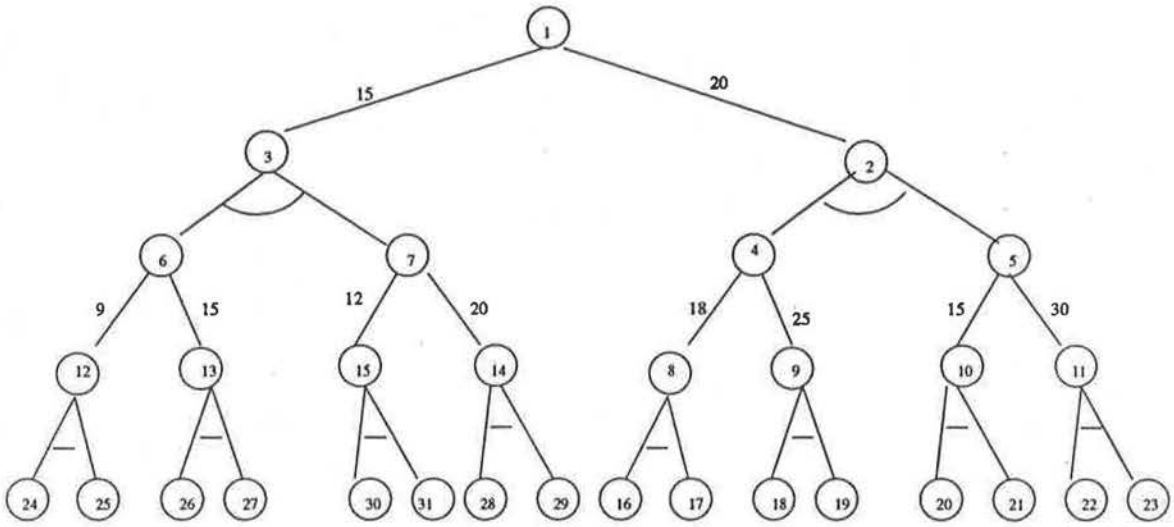

Figure 3: An illustration on the effect of children ordering

If we regard that, for a given tree, the functionality of an ordering function is to order the nodes in the tree according to the estimations of their min–exp values, and that the functionality of an admissible evaluation function is to estimate admissibly the min–exp values of the nodes in the tree, it seems fair to say that it is easier to find good ordering functions than to find a good admissible evaluation function for a decision tree. As a special case, we can use an admissible evaluation function to define the ordering functions. In particular, we can define $order_d$ in such a way that for a decision node N, $order_d(N) = N_1, ..., N_l$ satisfies:

$$f(N_i) + c_i \leq f(N_j) + c_j$$

for any $i, j, \quad 1 \leq i \leq j \leq l$. Similarly, for a nature node N, $order_n(N) = N_1, ..., N_l$ satisfies:

$$f(N_i) * p_i \geq f(N_j) * p_j$$

for any $i, j, \quad 1 \leq i \leq j \leq l$. With this definitions of the ordering functions, child N_j of a decision node N will be cut off if there exists a child N_i of N, $i < j$, such that:

$$DT(DTREE, N_i) + c_i \leq f(N_j) + c_j.$$

[4]in contrast with nine in the previous example.

Let $\Delta = f(N_j) + c_j - (f(N_i) + c_i)$, the above inequality is equivalent to:

$$DT(DTREE, N_i) - f(N_i) \leq \Delta.$$

The left hand in the above inequality is the difference between the min–exp value of node N_i and its lower bound given by function f, and the right hand is the difference which determines the search order between N_i and N_j. Obviously, the more informed the evaluation function is, the better the chance for the brother nodes being pruned. Similarly, children $N_{j+1}, ...N_l$ of a nature node N will all be cut off if

$$\sum_{k=1}^{j} DT(DTREE, N_k) * p_k + \sum_{k=j+1}^{l} f(N_k) * p_k \geq \texttt{bp}.$$

It is easy to verify that with the ordering functions defined as above, if $f(N) = DT(DTREE, N)$, all the non–optimal successors of any decision node will be cut off, then the subtree searched by A1 is exactly the same as the optimal solution tree.

For a uniform tree of depth $2d + 1$ where each nonterminal decision node has b_1 children and each nature node has b_2 children, then the total number of the terminal nodes in the decision tree is $(b_1 b_2)^d$ while the number of the terminal nodes in the optimal solution tree is only b_2^d. Therefore, the ratio of the performance in the best case to that in the worst case is b_1^d. We call this ratio is the *ideal cutoff ratio*. It is clear that the bigger the b_1, the larger the ideal cutoff ratio, thus, the more potential A1 has. An intuitive interpretation on this is that the bigger b_1, the more important the domain knowledge.

5 Relaxing the Optimality Requirement

For a decision tree rooted at N, if it is acceptable to find a suboptimal solution then we can improve on the performance of decision tree search algorithms in two ways. The first one is that we can use inadmissible evaluation functions. The second one is that we can adjust the initial back–up values dynamically during search.

5.1 Using inadmissible evaluation functions

In the previous section, it is required that the evaluation function f must be admissible. For A* algorithm, Harris [Harris, 1974] has argued that the condition of admissibility is too restrictive. His arguments are applicable to decision tree search as well. Although it may be impractical to find a good evaluation function that never violates the admissibility condition, it is often easier to find a function that estimates the min–exp values well, but occasionally overestimates them. The following two theorems establish the relationship between inadmissible evaluation functions and the min–exp value of the resulting solution.

Theorem 2 Suppose A1 uses evaluation function f for decision tree $DTREE$. If f satisfies: $f(N) \leq DT(DTREE, N) + \delta$, for every node N in $DTREE$, then for every node N in the decision tree

$$DT(DTREE, N) + \delta \geq bp \text{ if } DT_1(N, bp) \geq bp \quad \text{and}$$

$$DT(DTREE, N) + \delta \geq DT_1(N, bp) \text{ if } DT_1(N, bp) < bp.$$

Theorem 3 Suppose A1 uses evaluation function f for decision tree $DTREE$. If the costs of all the edges in the decision tree are non–negative, and f satisfies:

$$f(N) \leq (1 + \delta) * DT(DTREE, N), \text{ for every node } N \text{ in } DTREE,$$

then for every node N in the decision tree

$$DT(DTREE, N) * (1 + \delta) \geq bp \text{ if } DT_1(N, bp) \geq bp \quad \text{and}$$

$$DT(DTREE, N) * (1 + \delta) \geq DT_1(N, bp) \text{ if } DT_1(N, bp) < bp.$$

5.2 A bounded quality search algorithm

Another algorithm, A2, is given in Fig. 4. A2 can be used to compute suboptimal solutions, with bounded quality, of decision trees.

Algorithm A2 is the same as A1 except for that the variable `nb` in function `dnode2` is intentionally decreased by a factor of $(1 - \epsilon)$ in the situations where variable `result` is less than variable `bp`[5]. When variable `nbp` is decreased, the next child to be explored is more likely to be cut off. Consequently, the computation speed increases. This algorithm can make an earlier resolution on the choices of a decision node which are all close to optimal. Therefore, A2 can be much more efficient than A1 for a decision tree where the min–exp values of all solution trees of the decision tree vary little but the optimal solution tree of the decision tree is toward the right of the tree.

For a given decision tree $DTREE$, and a constant $\epsilon \in [0, 1)$, A2 will compute a suboptimal solution with min–exp value no more than $DT(DTREE, N)/(1 - \epsilon)$ for every node N in the solution tree. More accurately, let DT_2 be the function corresponding to Algorithm A2 and be defined as:

$$DT_2(N, bp) = \begin{cases} nnode2(N, bp) & \text{if } N \text{ is a nature node;} \\ dnode2(N, bp) & \text{otherwise.} \end{cases}$$

The next two theorems establish the relationship between A2 and A1.

[5] In this case, at least one solution tree with min–exp value less than the initial back–up value, bp, is found for the sub–decision–tree rooted at node N. This treatment is similar to that of aspiration window [Shams *et al.*, 1991].

```
dnode2(N, bp)
   if N is a terminal then
      if value(N) >= bp then return MAXINT; else return value(N);
   if f(N) >= bp return MAXINT;
   result = bp;          j = # of children of N;
   let N1, N2, ..., Nj = order-d(N);
   for (i = 1 to j) do
       if result = bp then nbp = result -  cost(N, i);
          else nbp = (1 - epsilon) * (result -  cost(N, i));
       if nbp>f(Ni) then result=min{result;cost(N,i)+nnode2(Ni,nbp)};
   if result >= bp then return MAXINT; else return result;

nnode2(N, bp)      same as nnode1(N, bp) of A1
```

Figure 4: The pseudo codes of algorithm A2

Theorem 4 If the evaluation function f used by algorithms A1 and A2 is admissible for a decision tree $DTREE$, then for any node N in $DTREE$ and number bp, $DT_2(N, bp) < bp$ iff $DT_1(N, bp) < bp$.

The intuitive meaning of this theorem is that for any node N in a decision tree, if we apply algorithms A1 and A2 with the same upper limit bp to the subtree rooted at N, then either both algorithms report that the min–exp value of node N is less than bp, or both report that the min–exp value is no less than bp. This fact is crucial for the validness the next theorem.

Theorem 5 If the evaluation function f used by algorithms A2 is admissible for $DTREE$, then for any node N in $DTREE$, $DT_1(N, bp) \geq (1 - \epsilon) * DT_2(N, bp)$.

6 Discussion and Conclusions

In this paper we presented two algorithms for decision tree search. The basic idea behind these algorithms is to make use of domain dependent information and a search mechanism similar to alpha–beta technique for minimax trees. The domain dependent information used by our algorithms is in the form of three functions: one evaluation function and two ordering functions. The advantage of our algorithms over AO* is that they need only linear space while using similar domain dependent information.

The research presented in this paper was originally motivated by the problems of navigation with uncertainty [Qi and Poole, 1991, Qi and Poole, 1992]. In [Qi and Poole, 1991], we showed how the problem of devising navigation plans in uncertain

environments can be formulated as a problem of decision tree evaluation. For that particular domain, ordering functions and admissible evaluation function can be defined in a straightforward way.

Future work can be carried in several directions. First, we hope that a theoretic analysis, like that for game tree searching, can be carried out on the average performance of our algorithms. Second, more experimental studies on the performance of algorithms can be carried out by applying them to practical domains.

Acknowledgement The first author is supported by University Graduate Fellowship of UBC. The research reported in this paper is partially supported by NSERC grant under operation number OGPOO44121. The authors wish to thank A. Mackworth, M. Queyranne, K. Kanazawa, G. Lin and Y. Zhang for their valuable comments on the content and/or the presentation of this paper.

References

[Harris, 1974] L. R. Harris. The heuristic search under conditions of error. *Artificial Intelligence*, 5(3), 1974.

[Howard and Matheson, 1981] R. A. Howard and J. E. Matheson. Influence diagrams. In R. A. Howard and J. E. Matheson, editors, *The Principles and Applications of Decision Analysis, VolumII.* Strategic Decision Group, Mento Park, CA., 1981.

[Knuth and Moore, 1975] D. E. Knuth and R. W. Moore. An analysis of alpha beta pruning. *Artificial Intelligence*, 6(4), 1975.

[McAllester, 1988] D. A. McAllester. Conspiracy numbers for minmax search. *Artificial Intelligence*, 1988.

[Pearl, 1984] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley Publishing Company, 1984.

[Qi and Poole, 1991] Runping Qi and David Poole. High level path planning with uncertainty. In *Proc. of the Seventh Conference on Uncertainty in AI*, 1991.

[Qi and Poole, 1992] Runping Qi and David Poole. A framework for high level path planning with uncertainty. Technical report, Department of Computer Science, University of British Columbia, 1992.

[Qi, 1992] Runping Qi. A study of high level path planning with uncertainty. Technical Report Ph. D. thesis, forthcoming, Department of Computer Science, University of British Columbia, 1992.

[Raiffa, 1968] Howard Raiffa. *Decision Analysis.* Addison–Wesley Publishing Company, 1968.

[Shams *et al.*, 1991] R. Shams, H. Kaindl, and H. Horacek. Using aspiration windows for minimax algorithms. In *IJCAI–91*, 1991.

[Stochman, 1979] G. C. Stochman. a minmax algorithm better than alpha–beta? *Artificial Intelligence*, 12(2), 1979.

[von Neumann and Morgenstern, 1947] J. von Neumann and O. Morgenstern. *Theory and Games and Economic Behavior.* Princeton University Press, 1947.

A The Proofs of Theorems

Theorem 1 For a given decision tree $DTREE$ and an admissible evaluation function f, DT and DT_1 satisfy the following relation:

$$DT_1(N, bp) = \begin{cases} DT(DTREE, N) & \text{if } DT(DTREE, N) < bp \\ \texttt{MAXINT} & \text{otherwise.} \end{cases}$$

for any node N in $DTREE$ and a number bp.

Proof We first observe two facts. The first one is that function DT is equivalent to DT_0 defined as follows:

Case 1 N is a terminal:

$$DT_0(N) = value(N). \tag{1}$$

Case 2 N is a nature node:

$$DT_0(N) = t^0. \tag{2}$$

where $t^0 = t_l^0$ and $t_i^0, 1 \leq i \leq l$ is recursively defined as follows:

$$\begin{array}{l} t_0^0 = \sum_{j=1}^{l} f(N_j) * p_j; \\ t_i^0 = t_{i-1}^0 + p_i * (DT_0(N_i) - f(N_j)). \end{array} \tag{3}$$

If f is admissible, then $t_i^0 \leq t_{i+1}^0$ for $i = 1, ..., l-1$.

Case 3 N is a decision node:

$$DT_0(N) = t^0 \tag{4}$$

where $t^0 = t_l^0$ and $t_i^0, 1 \leq i \leq l$ is recursively defined as follows:

$$\begin{array}{l} t_0^0 = \infty; \\ t_i^0 = min\{t_{i-1}^0, c_i + DT_0(N_i)\}. \end{array} \tag{5}$$

In the above mathematical characterization, c_i denotes the cost of the edge from a decision node to its i-th child, and p_i denotes the probability associated with the i-th child of a natural node. They correspond to `cost(N, i)` and `prob(N, i)` respectively in algorithms A1 and A2. This convention will be used in the rest of the paper. Thus, it suffices to prove

$$DT_1(N, bp) = \begin{cases} DT_0(N) & \text{if } DT_0(N) < bp \\ \texttt{MAXINT} & \text{otherwise.} \end{cases} \tag{6}$$

for every node N in $DTREE$.

The second fact is that the definition of DT_1 can be further refined as follows:

Case 1 N is a terminal;

$$DT_1(N, bp) = \begin{cases} value(N) & \text{if } value(N) < bp \\ \texttt{MAXINT} & \text{otherwise.} \end{cases} \tag{7}$$

Case 2 N is a nature node:

$$DT_1(N, bp) = \begin{cases} t^1 & \text{if } t^1 < bp \\ \texttt{MAXINT} & \text{otherwise.} \end{cases} \tag{8}$$

where $t^1 = t_l^1$ and $t_i^1, 1 \leq i \leq l$ is recursively defined as follows:

$$\begin{array}{l} t_0^1 = \sum_{j=1}^{l} f(N_j) * q_j; \\ nbp_i^1 = (bp - (t_{i-1}^1 - f(N_i) * p_i))/p_i; \\ t_i^1 = \begin{cases} t_{i-1}^1 & \text{if } t_{i-1}^1 \geq bp. \\ t_{i-1}^1 + p_i * (DT_1(N_i, nbp_i^1) - f(N_i)) & \text{otherwise.} \end{cases} \end{array} \tag{9}$$

Case 3 N is a decision node:

$$DT_1(N, bp) = \begin{cases} t^1 & \text{if } t^1 < bp \\ \texttt{MAXINT} & \text{otherwise.} \end{cases} \tag{10}$$

where $t^1 = t_l^1$ and $t_i^1, 1 \leq i \leq l$ is recursively defined as follows:

$$\begin{array}{l} t_0^1 = bp; \\ t_i^1 = \begin{cases} t_{i-1}^1 & \text{if } t_{i-1}^1 - c_i \leq f(N_i) \\ min\{t_{i-1}^1, c_i + DT_1(N_i, t_{i-1}^1 - c_i)\} & \text{otherwise} \end{cases} \end{array} \tag{11}$$

Based on these two facts, relation (6) can be proved by induction on the tree.

1. N is a terminal. The relation is obvious.

2. N is a decision node. Suppose the relation holds for all the children of N. Our first observation here is that, under the induction hypothesis, equation (11) is equivalent to the following simpler one:

$$\begin{array}{l} t_0^1 = bp; \\ t_i^1 = min\{t_{i-1}^1, c_i + DT_1(N_i, t_{i-1}^1 - c_i)\} \end{array} \tag{12}$$

Our second observation here is that the conclusion of the theorem holds too for this case, according to equations (10) and (12), if we can prove the following claim:

> $t_i^1 < bp$ iff $t_i^0 < bp$; and if $t_i^0 < bp$, then $t_i^0 = t_i^1$, otherwise, $t_i^1 = bp$, where t_i^1 and t_i^0 are defined by equations (12), (5) respectively.

We prove the above claim by induction on i.

$i = 1$. $t_1^1 = min\{bp, c_1 + DT_1(N_1, bp - c_1)\}$ and $t_1^0 = c_1 + DT_0(N_1)$.

If

$$t_1^0 = c_1 + DT_0(N_1) \geq bp$$

then,

$$DT_0(N_1) \geq bp - c_1.$$

By the outer induction assumption, we have:

$$DT_1(N_1, bp - c_1) = \texttt{MAXINT} > bp.$$

Therefore

$$t_1^1 = bp.$$

If

$$t_1^0 = c_1 + DT_0(N_1) < bp$$

then,

$$DT_0(N_1) < bp - c_1.$$

By the outer induction assumption, we have:

$$DT_0(N_1) = DT_1(N_1, bp - c_1);$$

$$c_1 + DT_1(N_1, bp - c_1) = c_1 + DT_0(N_1) < bp.$$

Therefore, $t_1^1 = c_1 + DT_0(N_1) = t_1^0$. The induction base holds.

Suppose the claim is true for $i = k$. For $i = k + 1$, we have:

$$t_{k+1}^1 = min\{t_k^1, c_{k+1} + DT_1(N_{k+1}, t_k^1 - c_{k+1})\};$$

$$t_{k+1}^0 = min\{t_k^0, c_{k+1} + DT_0(N_{k+1})\}.$$

Now, we have two cases:

(A) $t_{k+1}^0 \geq bp$. In this case, we have $t_k^0 \geq bp$ and $c_{k+1} + DT_0(N_{k+1}) \geq bp$. Thus we can obtain $t_k^1 = bp$ by the inner induction assumption. Furthermore, since

$$DT_0(N_{k+1}) \geq bp - c_{k+1},$$

then, by the outer induction assumption, we have:

$$DT_1(N_{k+1}, t_k^1 - c_{k+1}) = DT_1(N_{k+1}, bp - c_{k+1}) = \texttt{MAXINT}.$$

Therefore, $t_{k+1}^1 = bp$.

(B) $t_{k+1}^0 < bp$, In this case, we need to consider three subcases:

(a) $t_k^0 \geq bp$ and $c_{k+1} + DT_0(N_{k+1}) < bp$. In this subcase, we have:

$$DT_0(N_{k+1}) < bp - c_{k+1};$$

$$t_{k+1}^0 = c_{k+1} + DT_0(N_{k+1}) < bp.$$

By the inner induction assumption, we have: $t_k^1 = bp$. By the outer induction assumption, we obtain:

$$DT_1(N_{k+1}, bp - c_{k+1}) = DT_0(N_{k+1}).$$

Thus,

$$c_{k+1} + DT_1(N_{k+1}, t_k^1 - c_{k+1}) = DT_0(N_{k+1}) + c_{k+1} < bp.$$

Therefore, $t_{k+1}^1 = c_{k+1} + DT_1(N_{k+1}, t_k^1 - c_{k+1}) = t_{k+1}^0$.

(b) $t_k^0 < bp$ and $t_k^0 \leq c_{k+1} + DT_0(N_{k+1})$. In this subcase, we have:

$$t_{k+1}^0 = t_k^0 = t_k^1 \text{ (by the inner induction assumption);}$$

$$DT_0(N_{k+1}) \geq t_k^0 - c_{k+1} = t_k^1 - c_{k+1}.$$

Thus, by the outer induction assumption, we have:

$$DT_1(N_{k+1}, t_k^1 - c_{k+1}) = \texttt{MAXINT}.$$

therefore, $t_{k+1}^1 = t_k^1 = t_{k+1}^0$.

(c) $t_k^0 < bp$ and $t_k^0 > c_{k+1} + DT_0(N_{k+1})$ In this subcase, we have:

$$t_k^0 = t_k^1 \text{ (by the inner induction assumption);}$$

$$t_{k+1}^0 = c_{k+1} + DT_0(N_{k+1})$$

$$DT_0(N_{k+1}) < t_k^0 - c_{k+1} = t_k^1 - c_{k+1}.$$

Thus, by the outer induction assumption, we have:

$$DT_1(N_{k+1}, t_k^1 - c_{k+1}) + c_{k+1} = DT_0(N_{k+1}) + c_{k+1} < t_k^1.$$

Therefore, $t_{k+1}^1 = DT_1(N_{k+1}, t_k^1 - c_{k+1}) + c_{k+1} = t_{k+1}^0$.

In summary, the claim holds for $i = k + 1$. Consequently, the claim holds by induction.

3. N is a nature node. Suppose the conclusion of the theorem holds for all the children of N. Similarly, the conclusion follows from the the following claim:

   $t_i^1 < bp$ iff $t_i^0 < bp$; and if $t_i^0 < bp$, then $t_i^0 = t_i^1$, where t_i^1 and t_i^0 are defined by equations (9), (3) respectively.

This claim can be proved by a similar induction on t_i^0 and t_i^1. In the proof of this claim, we need to use the fact that $t_i^0 \leq t_{i+1}^0$ for $i = 0, ..., l-1$.

In summary, the theorem holds in general. □

Theorem 2 Suppose A1 uses evaluation function f for decision tree $DTREE$. If f satisfies:

$$f(N) \leq DT(DTREE, N) + \delta, \text{for all node } N \text{ in } DTREE,$$

then

$$DT(DTREE, N) + \delta \geq bp \text{ if } DT_1(N, bp) \geq bp;$$

and

$$DT(DTREE, N) + \delta \geq DT_1(N, bp) \text{ if } DT_1(N, bp) < bp.$$

Theorem 3 Suppose A1 uses evaluation function f for decision tree $DTREE$. If the costs of all the edges in the decision tree are non–negative, and f satisfies:

$$f(N) \leq (1 + \delta) * DT(DTREE, N), \text{for every node } N \text{ in } DTREE,$$

then for every node N in the decision tree

$$DT(DTREE, N) * (1 + \delta) \geq bp \text{ if } DT_1(N, bp) \geq bp;$$

and

$$DT(DTREE, N) * (1 + \delta) \geq DT_1(N, bp) \text{ if } DT_1(N, bp) < bp.$$

Since DT is equivalent to DT_0, all the occurrences of $DT(DTREE, N)$ in the above theorems can be replaced with $DT_0(N)$. Therefore, for Theorem 2, it suffices to prove that for every node N in $DTREE$

$$DT_0(N) + \delta \geq bp \text{ if } DT_1(N, bp) \geq bp;$$

and

$$DT_0(N)\delta \geq DT_1(N, bp) \text{ if } DT_1(N, bp) < bp$$

For Theorem 3, it suffices to prove that for every node N in $DTREE$

$$DT_0(N) * (1 + \delta) \geq bp \text{ if } DT_1(N, bp) \geq bp;$$

and

$$DT_0(N) * (1 + \delta) \geq DT_1(N, bp) \text{ if } DT_1(N, bp) < bp$$

Actually, the proofs of Theorems 2 and 3 are very similar. Here we just present the proof of Theorem 3.

Proof of Theorem 3

Case 1 N is a terminal. Trivial.

Case 2 N is a nature node.
Suppose the relations hold for all the children of node N. We need to consider the following two cases:

- $DT_1(N, bp) < bp$. According to equations (8) and (9), we have $t^1 = t_l^1 < bp$, thus, $DT_1(N, bp) = t_l^1$. Furthermore, by the induction assumption, we have: $DT_1(N_i, nbp_i^1) < nbp_i^1$, thus, $DT_1(N_i, nbp_i^1) \leq (1+\delta) * DT_0(N_i)$ for $i = 1, ..., l$. According to equations (3) and (2), we obtain:

$$DT_0(N) = t_l^0 = \sum_{i=1}^{l} p_i * DT_0(N_i)$$

According to equation (9), we obtain:

$$t_l^1 = \sum_{i=1}^{l} p_i * DT_1(N_i, nbp_i^1) \leq \sum_{i=1}^{l} p_i * DT_1(N_i) * (1+\delta) = t_l^0 * (1+\delta)$$

Thus, $DT_1(N, bp) \leq DT_0(N) * (1+\delta)$.

- $DT_1(N, bp) \geq bp$. According to equations (8) and (9), we know that $t^1 = t_l^1 \geq bp$. This implies that either $t_0^1 \geq bp$ or there exists $k, 1 \leq k \leq l$ such that $t_{k-1}^1 < bp$ and $t_k^1 \geq bp$. In the former case, we have:

$$DT_0(N) * (1+\delta) = \sum_{i=1}^{l} p_i * DT_0(N_i) * (1+\delta) \geq \sum_{i=1}^{l} p_i * f(N_i) = t_0^1 \geq bp$$

In the latter case, we can obtain:

$$DT_1(N_j, nbp_j^1) \geq nbp_j^1 \quad \text{for} \quad 0 \leq j < k$$

$$DT_1(N_k, nbp_k^1) \geq nbp_k^1$$

where $nbp_k^1 = (bp - (t_{k-1}^1 - f(N_k) * p_k))/p_k$. Consequently, we have:

$$DT_0(N_j) * (1+\delta) \geq DT_1(N_j, nbp_j^1) \text{ for } 0 \leq j < k$$

and

$$(1+\delta) * DT_0(N_k) \geq nbp_k^1$$

Therefore:

$$p_k * (1+\delta) * DT_2(N_k) \geq bp - (t_{k-1}^1 - f(N_k) * p_k))$$

$$t^1_{k-1} + p_k * (1+\delta) * DT_2(N_k) - f(N_k) * p_k > bp$$

According to equation (9), we have:

$$t^1_{k-1} = \sum_{j=1}^{k-1} DT_1(N_j, nbp^1_j) * p_j + \sum_{j=k}^{l} f(N_j) * p_j$$

Thus,

$$\sum_{j=1}^{k-1} DT_1(N_j, nbp^1_j) * p_j + p_k * (1+\delta) * DT_2(N_k) + \sum_{j=k+1}^{l} f(N_j) * p_j \geq bp$$

$$\sum_{j=1}^{k} (1+\delta) DT_0(N_j) * p_j + \sum_{j=k+1}^{l} (1+\delta) DT_0(N_j) * p_j \geq bp$$

Therefore,

$$(1+\delta) DT_0(N) = \sum_{j=1}^{l} (1+\delta) DT_0(N_j) * p_j \geq bp$$

Case 3 N is a decision node.

Suppose the relations hold for all the children of node N. The relations hold too for this node if we can prove

$$t^1_i \leq (1+\delta) * t^0_i \tag{13}$$

for $i = 0, ..., l$, where t^0_i and t^1_i are defined by equations (3) and (16) respectively. This inequality can be proved by induction on i.

Basis: $i = 0$, trivial.

Induction. Suppose the inequality is true for $i = k$. Consider the case when $i = k+1$.

- $t^0_k \leq c_{k+1} + DT_0(N_{k+1})$. In this case, we have $t^0_k = t^0_{k+1}$. On the other hand, it is easy to verify that $t^1_{k+1} \leq t^1_k$. By the inner induction assumption, we conclude $t^1_{k+1} \leq (1+\delta) * t^0_{k+1}$.
- $t^0_k > c_{k+1} + DT_0(N_{k+1}) = t^0_{k+1}$. In this case, if $t^1_k - c_{k+1} \leq f(N_{k+1})$, then,

$$t^1_{k+1} = t^1_k \leq c_{k+1} + f(N_{k+1}) \leq c_{k+1} + (1+\delta) * DT_0(N_{k+1})$$

Since $c_{k+1} \geq 0$ and $\delta \geq 0$, we have $t^1_{k+1} \leq (1+\delta) * t^0_{k+1}$.
If $t^1_k - c_{k+1} > f(N_{k+1})$, then,

$$t^1_{k+1} = min\{t^1_k, c_{k+1} + DT_1(N_{k+1}, t^1_k - c_{k+1})$$

When $t_k^1 - c_{k+1} \leq DT_1(N_{k+1}, t_k^1 - c_{k+1})$, by the outer induction assumption, we have:

$$(1+\delta) * DT_0(N_{k+1}) \geq t_k^1 - c_{k+1}$$

$$t_{k+1}^1 = t_k^1 \leq (1+\delta) * DT_0(N_{k+1}) + c_{k+1}$$

Since $c_{k+1} \geq 0$ and $\delta \geq 0$, we have $t_{k+1}^1 \leq (1+\delta) * t_{k+1}^0$. when $t_k^1 - c_{k+1} > DT_1(N_{k+1}, t_k^1 - c_{k+1})$, by the outer induction assumption, we have:

$$(1+\delta) * DT_0(N_{k+1}) \geq DT_1(N_{k+1}, t_k^1 - c_{k+1})$$

$$t_{k+1}^1 = c_{k+1} + DT_1(N_{k+1}, t_k^1 - c_{k+1}) \leq c_{k+1} + (1+\delta) * DT_0(N_{k+1})$$

Since $c_{k+1} \geq 0$ and $\delta \geq 0$, we have $t_{k+1}^1 \leq (1+\delta) * t_{k+1}^0$.

□

In order to prove Theorems 4 and 5, we first observe that DT_2 can be further refined as follows.

Case 1 N is a terminal.

$$DT_2(N, bp) = \begin{cases} value(N) & \text{if } value(N) < bp \\ \texttt{MAXINT} & \text{otherwise.} \end{cases} \tag{14}$$

Case 2 N is a nature node:

$$DT_2(N, bp) = \begin{cases} t^2 & \text{if } t^2 < bp \\ \texttt{MAXINT} & \text{otherwise.} \end{cases} \tag{15}$$

where $t^2 = t_l^2$ and $t_i^2, 1 \leq i \leq l$ is recursively defined as follows:

$$\begin{array}{l} t_0^2 = \sum_{j=1}^{l} f(N_j) * p_j; \\ nbp_i^2 = (bp - (t_{i-1}^2 - f(N_i) * p_i))/p_i; \\ t_i^2 = \begin{cases} t_{i-1}^2 & \text{if } t_{i-1}^2 \geq bp. \\ t_{i-1}^2 + p_i * DT_2(N_i, nbp_i^2) & \text{otherwise.} \end{cases} \end{array} \tag{16}$$

Case 3 N is a decision node:

$$DT_2(N, bp) = \begin{cases} t^2 & \text{if } t^2 < bp \\ \texttt{MAXINT} & \text{otherwise.} \end{cases} \tag{17}$$

where $t^2 = t_l^2$ and $t_i^2, 1 \leq i \leq l$ is recursively defined as follows:

$$\begin{array}{l} t_0^2 = bp; \\ nbp_i^2 = \begin{cases} bp - c_i & \text{if } t_{i-1}^2 = bp \\ (1-\epsilon) * (t_{i-1}^2 - c_i) & \text{otherwise;} \end{cases} \\ t_i^2 = \begin{cases} t_{i-1}^2 & \text{if } nbp_i^2 \leq f(N_i) \\ min\{t_{i-1}^2, c_i + DT_2(N_i, nbp_i^2)\} & \text{otherwise} \end{cases} \end{array} \tag{18}$$

The following proofs are based on the above characterization of DT_2.

Lemma 1 If the evaluation function f used by algorithm A2 is admissible for a decision tree $DTREE$, then for any node N in $DTREE$, and a number bp, $DT_0(N) \leq DT_2(N, bp)$.

Proof By induction.

It is trivial for the cases of N being a terminal and a nature node. Let us consider the case of N being a decision node. Suppose the conclusion of the lemma is true for all the children of N. We observe that the conclusion in the lemma is true in this case as well if we can prove the following claim:

> $t_i^2 = bp$ or $t_i^2 \geq t_i^0$ for $i = 1, ..., l$, where t_i^2 and t_i^0 are defined by equations (18) and (5) respectively.

This claim can be proved by a simple induction on i. □

Lemma 2 If the evaluation function f used by algorithms A1 and A2 is admissible for a decision tree $DTREE$, then for any node N in $DTREE$ and numbers bp_1 and bp_2, if $bp_1 \geq bp_2$, then $DT_1(N, bp_1) \leq DT_2(N, bp_2)$.

Proof By induction.

It is trivial for the cases of N being a terminal and a nature node. Let us consider the case of N being a decision node. Suppose the conclusion in the lemma is true for all the children of N.

Observe that the conclusion in the lemma is true in this case as well if we can prove the following claim:

> $t_i^1 \leq t_i^2 \leq bp_1$ for $i = 1..l$, where t_i^1 and t_i^2 are defined by equations (12) and (18) respectively.

This claim can be proved by a simple induction on t_i^1 and t_i^2.

For $i = 1$, the claim is true by the above assumption and equations (12) and (18).

Suppose the claim is true for $i = k$. According to equations (12) and (18), it suffices to prove

$$DT_2(N_{k+1}, nbp_{k+1}^2) + c_{k+1} \geq t_{k+1}^1. \tag{19}$$

We need to consider two cases here. (A) $t_k^1 > DT_1(N_{k+1}, (t_k^1 - c_{k+1})) + c_{k+1}$. In this case, we have:

$$t_{k+1}^1 = DT_1(N_{k+1}, (t_k^1 - c_{k+1})) + c_{k+1};$$
$$DT_1(N_{k+1}, (t_k^1 - c_{k+1})) < t_k^1 - c_{k+1};$$

and (by Theorem 1)

$$DT_1(N_{k+1}, (t_k^1 - c_{k+1})) = DT_0(N_{k+1}).$$

Thus, by Lemma 1, we have: $DT_0(N_{k+1}) \leq DT_2(N_{k+1}, nbp_k^2)$. Therefore, inequality (19) is established.

(B) $t_k^1 \leq DT_1(N_{k+1}, t_k^1 - c_{k+1}) + c_{k+1}$. In this case, we have:

$$DT_1(N_{k+1}, t_k^1 - c_{k+1}) \geq t_k^1 - c_{k+1};$$

$$t_k^1 = t_{k+1}^1.$$

By Theorem 1, we have:

$$DT_0(N_{k+1}) \geq t_k^1 - c_{k+1}.$$

Thus, by Lemma 1, we have $DT_2(N_{k+1}, nbp_k^2) \geq t_k^1 - c_{k+1}$. Therefore, inequality (19) is established. □

Theorem 4 If the evaluation function f used by algorithms A1 and A2 is admissible for a decision tree $DTREE$, then for any node N in $DTREE$ and number bp, $DT_2(N, bp) < bp$ iff $DT_1(N, bp) < bp$.

Proof By induction.

It is trivial for the cases of N being a terminal and N being a nature node. Let us consider the case of N being a decision node. Assuming that the theorem holds for all the children of N. First, $DT_1(N, bp) < bp$ if $DT_2(N, bp) < bp$ can be derived immediately from Lemma 2. Now suppose that $DT_1(N, bp) < bp$. There must exist at least one child, say N_j, of N such that $DT_1(N_j, bp - c_j) < bp - c_j$. If $t_{j-1}^2 < bp$, then $t^2 < bp$. Thus $DT_2(N, bp) < bp$. Otherwise, $t_{j-1}^2 = bp$. According to equation (18), $t_j^2 = min\{bp, c_j + DT_2(N_j, bp - c_j)\}$. By the induction assumption, we have:

$$DT_2(N_j, bp - c_j) < bp - c_j.$$

Therefore, $t_j^2 = c_j + DT_2(N_j, bp - c_j) < bp$. Thus, $t^2 < bp$. In summary, $t^2 < bp$ can be derived from $DT_1(N, bp) < bp$. Therefore, we can conclude

$$DT_1(N, bp) < bp \text{ iff } DT_2(N, bp) < bp.$$

The theorem holds by induction. □

Lemma 3 For any node N in a decision tree and number bp, $DT_2(N, bp) < bp$ or $DT_2(N, bp) = \texttt{MAXINT}$.

Proof This result can be verified directly by examining the definition of function DT_2. □

Theorem 5 If the evaluation function f used by algorithms A2 is admissible for a decision tree $DTREE$, then for any node N in $DTREE$, $DT_1(N, bp) \geq (1 - \epsilon) *$

$DT_2(N, bp)$.

Proof of Theorem 5 By induction.

It is trivial for the cases of N being a terminal and N being a nature node. Let us consider the case of N being a decision node. Assuming that the conclusion of this theorem holds for all the children of N. The conclusion of this theorem will also hold for N if we can prove

$$t_i^1 \geq (1-\epsilon) * t_i^2 \tag{20}$$

for $i = 0, ..., l$, where t_i^1 and t_i^2 are defined by equations (12) and (18) respectively.

In proving inequality (20), we will use the fact

$$(1-\epsilon) * (t_k^1 - c_{k+1}) \leq nbp_{k+1}^2 \leq t_k^1 - c_{k+1} \tag{21}$$

which can be derived from equation (18) directly.

For $i = 0$, $t_0^1 = bp = t_0^2$. Inequality (20) is true.

Suppose inequality (20) is true for $i = k$. For $i = k+1$, we need to consider three subcases.

(A) $t_{k+1}^1 = t_k^1$. Since $t_{k+1}^2 \leq t_k^2$, inequality (20) follows from the inner induction assumption.

(B) $t_{k+1}^1 < t_k^1$ and $t_{k+1}^2 < t_k^2$.

In this case, we have:

$$t_{k+1}^1 = c_{k+1} + DT_1(N_{k+1}, t_k^1 - c_{k+1})$$

and

$$t_{k+1}^2 = c_{k+1} + DT_2(N_{k+1}, nbp_{k+1}^2)) < t_k^2 \leq bp.$$

Thus,

$$DT_2(N_{k+1}, nbp_{k+1}^2)) < nbp_{k+1}^2 \text{ (by lemma 3).}$$

By Lemma 1, Theorem 4 and Theorem 1, we have:

$$DT_1(N_{k+1}, nbp_{k+1}^2) < nbp_{k+1}^2.$$

By Theorem 1 and equation (21), we can deduce:

$$DT_1(N_{k+1}, t_k^1 - c_{k+1}) = DT_0(N_{k+1});$$

and

$$DT_1(N_{k+1}, nbp_{k+1}^2) = DT_0(N_{k+1}).$$

On the other hand, we can obtain from the outer induction assumption:

$$DT_1(N_{k+1}, nbp_{k+1}^2) \geq (1-\epsilon) * DT_2(N_{k+1}, nbp_{k+1}^2).$$

Thus,

$$DT_0(N_{k+1}) \geq (1-\epsilon) * DT_2(N_{k+1}, nbp^2_{k+1}).$$

Therefore,

$$t^1_{k+1} = c_{k+1} + DT_0(N_{k+1}) \geq c_{k+1} + (1-\epsilon) * DT_2(N_{k+1}, nbp^2_{k+1});$$

$$t^1_{k+1} \geq (1-\epsilon) * t^2_{k+1}.$$

(C) $t^1_{k+1} < t^1_k$ and $t^2_{k+1} = t^2_k$. In this case, we have:

$$t^1_{k+1} = c_{k+1} + DT_1(N_{k+1}, t^1_k - c_{k+1});$$

and

$$t^2_k \leq DT_2(N_{k+1}, nbp^2_{k+1}) + c_{k+1}.$$

Thus, we have:

$$DT_2(N_{k+1}, nbp^2_{k+1}) \geq (1-\epsilon) * (t^2_k - c_{k+1}).$$

According to Theorem 4, we obtain:

$$DT_1(N_{k+1}, nbp^2_{k+1}) \geq (1-\epsilon) * (t^2_k - c_{k+1}).$$

Therefore, $DT_0(N_{k+1}) \geq (1-\epsilon) * (t^2_k - C_{k+1})$. From now on, we can derive as follows:

$$(1-\epsilon) * c_{k+1} + DT_0(N_{k+1}) \geq (1-\epsilon) * t^2_k;$$

$$t^1_{k+1} = c_{k+1} + DT_0(N_{k+1}) \geq (1-\epsilon) * t^2_k.$$

In summary, inequality (20) is true in all the three subcases, hence, is true for $i = k+1$. Therefore, the conclusion of the theorem is true for the case of N being a decision node. Now, we conclude that the theorem is true in general by induction. □