Probability-Based Game Tree Pruning

Liwu Li and *T. A. Marsland*

Computing Science Department
University of Alberta
Edmonton
Canada T6G 2H1

ABSTRACT

In game-tree search, a point value is customarily used to measure position evaluation. If the uncertainty about the value is to be reflected in the evaluation, which is described with probabilistic distribution or probabilities, the search process must back up distributions from leaf nodes to the root. It is shown that even though the merit value of a node is described by probabilities, α-β bounded windows can still be used to cut off some subtrees from search when a space-efficient depth-first traversal is applied to the game tree. Several variations of probability-based α-β game tree pruning are presented. We also show that probability-based α-β pruning can be viewed as a generalization of the standard α-β game tree search algorithm and that it inherits some good properties from its point-value version.

To appear, Journal of Algorithms, Fall 1989.

1. Introduction

In computer game playing, α-β pruning is a commonly used technique for speeding up search processes. Knuth and Moore credit the idea of pruning some nodes from the search process to McCarthy and his group at MIT, and trace back this search reduction method to 1958[1]. However, the first formal treatment of this topic appears to be that of Brudno in 1963[2]. In the α-β pruning algorithm, two bounds—α and β—are assigned to an interior node in the search tree that has been visited by a depth-first search process, and all the expected merit values of the node are between these two bounds. As the search of the game tree proceeds, the value of the best leaf node found so far changes, and the bounds α and β are also modified according to the value of the newly found best leaf node. As soon as it is known that the possible merit value of this node is out of this range, the search of the subtree rooted at this node is terminated.

Since it is rarely possible to search a game tree until truly terminal positions are reached, sufficiently deep positions have to be considered to be terminal. The part of a game tree that is to be explored by a search algorithm is usually called a *search tree*. The strengths, or merit values, of these "horizon" nodes or leaf nodes are evaluated to gain the best available information. There are three methods of describing the evaluation results. A simple but inflexible method uses point-values, such as material-balance, to measure the strength of a position. In a second method the upper and lower bounds are used to describe the possible merit values [3]. Finally, Palay suggests using a distribution function to describe the possible locations of the merit value of a node in the game tree[4]. In contrast, the point-value representation of game position merit totally ignores our uncertainty about the strength, it forces us to make a conclusion on the merit value of a position even when it is unclear. Another representation—range—also suffers from some similar problems. For example, two positions cannot be distinguished if their estimated ranges are the same. Probability distribution is a good alternative for describing the possible locations of the merit value of a position.

In computer game playing, move choice, or *decision making*, is based on game tree search. Best-first search strategies [3][4] combine decision making and game tree search naturally; depth-first game tree search leaves the problem of decision making open. Therefore, the existing probability-based move choice criterion, like the function *dominance-level* employed by algorithm PB* [4], can be applied to the probabilistic distributions backed up by depth-first search; if the decision cannot be made, some heuristics, like the iterative deepening [5] or some other selective deepening, can be used to guide the depth-first search to gain more information. As a matter of fact, because of its lower space requirement and lighter operating overhead, depth-first game tree search is almost universally used in computer game playing programs. The study of probability-based depth-first search could lead to some heuristics for selective deepening and so reduce the errors and costs in game tree search. In summary, the study of probability-based game tree search is essentially independent of the study of decision making, the former focuses on gaining more information with less cost, the latter, on analyzing the gained information. The depth-first search for game trees where the leaf nodes are evaluated with probability distributions, offers an interesting study of its own.

Since α-β pruning is such a popular and powerful technique, it is tempting to apply it in probability-based game tree search. All the prior discussions about α-β pruning have been restricted to point-value game tree search. The problem of how to apply the α-β pruning technique to probability-based game tree search will be addressed here. First, a computational model is presented, and a probability-based α-β pruning scheme is proposed for that model. The result shows that even though the merit value of a node is described by probabilities rather than a point-value, the α-β bounded windows can still be used to cut off the search of some nodes. It can be proven that probability-based α-β pruning is optimum in the sense that for some ordering of the successor nodes in a search tree, it will search the least number of leaf nodes to get the probabilities that describe the root position of the search tree. We show that the probability-based α-β algorithm can be viewed as a generalization of the α-β pruning employed by point-value search algorithms[1]. Several variations or applications of the probability-based algorithm are also presented. One of them applies α-β pruning to range-based game tree search. The heuristic information available at interior nodes of a search tree can also be used to improve the α-β bounds. We also show how α-β pruning can be incorporated into probability-based best-first search. Finally, the efficiency of this probability-based game tree pruning technique is tested with a C-language program which generates random trees.

In the following, like Knuth and Moore[1], we assume that all successor positions of a node in a game tree have independent values, or equivalently that the searching algorithms have no knowledge about dependencies between these positions. As observed by

Palay[4], the effects of this assumption tend to be limited.

2. A Model for Probability-Based Game Tree Search

Although the model proposed by Palay [4] applies to our search scheme as well, for the sake of easy description and straight computation, our model describes the merit value of a position as a discrete random variable. The domain of merit values is a finite integer interval $[-i, i]$, for $i > 0$. The merit value of a position S is described by a set of probabilities, or a probability function P_S, such that $P_S(v)$ is the probability that the merit value is equal to v, for any $v \in [-i, i]$. A list of integer-real number pairs,

$$<v_1, P_S(v_1)>, <v_2, P_S(v_2)>, ..., <v_k, P_S(v_k)>, \tag{1}$$

is used to represent the probability function. The data structure (1) is called a VP-list if $P_S(v_j) > 0$, for each $j = 1,...,k$. In the following, the terms probability function and VP-list are used synonymously.

The negamax game tree search description method will be exploited, so that one player's winning will be another's loss, and *vice versa*. In probability-based game tree search, the operands are probability functions, and this reversed viewing method can be reflected by the function P_NEG. Let P_S be the VP-list describing the possible locations of the merit value D_S of position S. The probability $P\_NEG(P_S)(v)$, for any $v \in [-i, i]$, is defined as the probability that the random variable $-D_S$ is equal to v:

$$P\_NEG(P_S)(v) = Probability(-D_S = v).$$

Therefore, if a player is making use of a probability function P to represent the merit value of a position, the opponent will make use of $P\_NEG(P)$ to describe that same position. By the definition of P_NEG, it is easy to deduce the following formula:

$$P\_NEG(P)(v) = Probability(D_S = -v) = P(-v)$$

for any $v \in [-i, i]$. The above equations imply that if a merit value is represented from the point of view of one player by a VP-list

$$<v_1, p_1>, <v_2, p_2>, \ldots, <v_k, p_k>,$$

then it will be described by the opponent as

$$<-v_k, p_k>, <-v_{k-1}, p_{k-1}>, \ldots, <-v_1, p_1>.$$

In other words, to get the probability function for the opponent, just change the sign of the domain value component of each pair in the VP-list.

Searching a game tree means backing up the probabilities of the leaf nodes to the root position. The back-up from its successor positions $S_1, S_2, \ldots S_w$ to a node S can be described by function P_MAX, the operands of which are also probability functions. For the probability functions $P_{S_1}, P_{S_2}, \ldots P_{S_w}$, P_MAX can be defined in the following way:

When $w = 1$,

$$P_S = P\_MAX(P_{S_1}) = P_{S_1};$$

when $w = 2$, for any $v \in [-i, i]$,

$$P_S(v) = P\_MAX(P_{S_1}, P_{S_2})(v)$$

$$= P_{S_1}(v)P_{S_2}(v) + P_{S_1}(v)\sum_{t=-i}^{v-1} P_{S_2}(t) + P_{S_2}(v)\sum_{t=-i}^{v-1} P_{S_1}(t);$$

when $w \geq 3$,

$$P_S = P\_MAX(P_{S_1}, P_{S_2}, \ldots, P_{S_w})$$

$$= P\_MAX(P\_MAX(P_{S_1}, P_{S_2}), \ldots, P_{S_w}).$$

Similar formulae can be used to define another function—P_MIN, which is used to describe the complement operation of P_MAX:

When $w = 1$,

$$P\_MIN(P_{S_1}) = P_{S_1};$$

when $w = 2$, for any $v \in [-i, i]$,

$$P\_MIN(P_{S1}, P_{S2})(v)$$

$$= P_{S1}(v)P_{S2}(v) + P_{S1}(v) \sum_{t=v+1}^{i} P_{S2}(t) + P_{S2}(v) \sum_{t=v+1}^{i} P_{S1}(t);$$

when $w \geq 3$,

$$P\_MIN(P_{S1}, P_{S2}, ..., P_{S_w}) = P\_MIN(P\_MIN(P_{S1}, P_{S2}), ..., P_{S_w}).$$

3. Probability-Based α-β Pruning

Two concepts will be used in the information-lossless pruning of game trees. The lower bound of a probability function P, denoted as $lower\_bound(P)$, is defined as the smallest domain value $v \in [-i, i]$ for which $P(v) \neq 0$; the upper bound, $upper\_bound(P)$, is the greatest $v \in [-i, i]$ for which $P(v) \neq 0$.

Proposition 1. Given two probability functions P_1 and P_2,

(1) if $lower\_bound(P_1) \geq upper\_bound(P_2)$, then

$$P\_MAX(P_1, P_2) = P_1;$$

(2) if

$$lower\_bound(P_1) < upper\_bound(P_2)$$

and

$$lower\_bound(P_2) < upper\_bound(P_1),$$

then

$$P\_MAX(P_1, P_2) \neq P_1$$

and

$$P\_MAX(P_1, P_2) \neq P_2.$$

Proof.

By the definitions of *lower_bound*, *upper_bound* and *P_MAX*. □

Suppose the current state of a node S is represented by P_S and that of one successor node S_j is represented by P_{S_j}. The current state of S will be described by

$$P\_MAX\,(P_S, P\_NEG\,(P_{S_j})).$$

The property of operation *P_MAX* presented in Proposition 1 suggests that only if $lower\_bound\,(P_S) \geq -lower\_bound\,(P_{S_j})$ can the search of the remaining successors of S_j be cut off. Based on this observation, a probability-based α-β algorithm can be designed.

The standard α-β pruning technique, which uses α-β bounded windows to limit the backed-up point-values[1], will be generalized for probability-based game tree search. The lower bounds of the probability functions that describe the current states of the ancestor nodes will be used in setting the cut-off bounds α and β before searching a node; these bounds are also improved dynamically when searching the subtree rooted at the node.

Algorithm 1. Probability-Based α-β Pruning Algorithm.

The recursive function P_AB(S: position; α, β: integer) will return a VP-list, the upper bound of which is less than or equal to β and the lower bound greater than or equal to α. In this function, two local variables (TP0 and TP1, respectively) maintain the current state of position S and that of the successor being explored. The special VP-list <v, 1> with a single integer-real pair represents a probability function that takes value v without doubt (i.e., with probability 1).

```
function P_AB(S: position; α, β: integer): VP-list;
var
  TP0, TP1: VP-list;
  j: integer;
```

```
begin
1.  if $S$ is a leaf node then
        return $P\_MIN(P\_MAX(P_S, <\alpha, 1>), <\beta, 1>)$;
    comment: Probability function $P_S$ is obtained by evaluating $S$
             from the corresponding player's viewpoint.
2.  determine the successor positions, $S_1, ..., S_w$, of $S$, where $w > 0$;
3.  TP0 := $<\alpha, 1>$;
4.  for $j$ := 1 to $w$ do
      begin
5.      TP1 := $P\_NEG$(P_AB($S_j$, $-\beta$, $-\alpha$));
6.      TP0 := $P\_MAX$(TP0, TP1);
7.      $\alpha$ := $lower\_bound$(TP0)
8.      if $\alpha = \beta$
          then return TP0;
      end;
9.  return TP0;
end.
```

The properties of function P_AB proposed in the following lemma will be used to prove the above algorithm.

Lemma 1. If P_S is the probability function that describes the merit value of a position S in a negamax game tree search, and integers $\alpha < \beta$, then

P_AB(S, α, β) = $<\alpha, 1>$, if $upper\_bound(P_S) \le \alpha$;

P_AB(S, α, β) = $P\_MIN(P\_MAX(P_S, <\alpha, 1>), <\beta, 1>)$,

if $upper\_bound(P_S) > \alpha$ and $lower\_bound(P_S) < \beta$;

P_AB(S, α, β) = $<\beta, 1>$, if $lower\_bound(P_S) \ge \beta$.

Proof.

These three loop invariant statements can be summarized by the following generalization,

$$\text{P\_AB}(S, \alpha, \beta) = P\_MIN(P\_MAX(P_S, <\alpha, 1>), <\beta, 1>).$$

The separation of this statement is just for convenience in proving Theorem 1 later.

Lemma 1 will be proved by induction on the height of position S in the search tree.

If the height of S is 0, statement 1 will return a probability function

$$P\_MIN(P\_MAX(P_S, <\alpha, 1>), <\beta, 1>).$$

Therefore, the loop invariant is true for a leaf node S.

Suppose that the lemma is true for all nodes of height less than h, and S is a position whose height in the search tree is h. In the **for** statement of function P_AB, statement 5 returns a probability function

$$\text{TP1} = P\_NEG(P\_MIN(P\_MAX(P_{S_j}, <-\beta, 1>), <-\alpha, 1>))$$

$$= P\_MIN(P\_MAX((P\_NEG(P_{S_j})), <\alpha, 1>), <\beta, 1>)).$$

If $lower\_bound$(TP1) is equal to β for some j, by the definition of the backed-up merit value, we must have $lower\_bound(P_S) \geq \beta$. Therefore,

$$\text{P\_AB}(S, \alpha, \beta) = P\_MIN(P\_MAX(P_S, <\alpha, 1>), <\beta, 1>)$$

$$= <\beta, 1>.$$

This justifies the **return** of statement 8. Otherwise, we can prove

$$P\_MIN(P\_MAX(P_S, <\alpha, 1>), <\beta, 1>)$$

$$= P\_MIN(P\_MAX(P\_MAX(P\_NEG(P_{S_1}), ..., P\_NEG(P_{S_w})), <\alpha, 1>), <\beta, 1>)$$

$$= P\_MAX(P\_NEG(P\_MIN(P\_MAX(P_{S_1}, <-\beta, 1>), <-\alpha_1, 1>)), ...,$$

$$P\_NEG(P\_MIN(P\_MAX(P_{S_w}, <-\beta, 1>), <-\alpha_w, 1>))),$$

for any integers $\alpha_1, ..., \alpha_w$, where $\alpha = \alpha_1 \leq \alpha_2 \leq ... \leq \alpha_w = \max(lower\_bound(P_S), \alpha)$. Hence the lemma is proved. $\square$

By the above lemma, given a position S as the root of a search tree, probability function P_S can be calculated by calling

$$\text{P\_AB}(S, -i, +i).$$

Theorem 1. For any position S, function call P_AB(S, $-i$, $+i$) will return a probability function that describes the merit value of S. □

Note that in function P_AB we can replace statement 1 by

```
if $S$ is a leaf node then
    return $P_S$,
```

provided we change statement 8 into

```
if $\alpha \geq \beta$
  then return TP0.
```

But statement 1

```
return $P\_MIN(P\_MAX((P_p), <\alpha, 1>), <\beta, 1>)$
```

made the proof of Lemma 1 easier, and it usually returns a VP-list with tighter bounds.

4. A Generalization of Standard α-β Pruning

Probability-based α-β pruning algorithm, in fact, is a generalization of the standard α-β algorithm analyzed by Knuth and Moore[1]. If the evaluation result for each leaf node S is a point-value, i.e.

$$P_S(v) = 1$$

for some domain value v, then the function P_AB has a form that is essentially the same as the algorithm F2 encoded by Knuth and Moore[1]. Therefore, our probability-based α-β algorithm will have similar properties to the point-valued version. In particular, since for any search tree there always exists an ordering of nodes for which the α-β algorithm examines the fewest leaf nodes, an equivalent property must exist for the probability-based version.

Theorem 2. Probability-based pruning is optimum in the sense that for any search tree and any algorithm, denoted as algorithm X, that computes the probability function for the root position, there is a way of ordering successor nodes so that every leaf node examined by probability-based α-β pruning method under this reordering is also examined by the algorithm X.

Before proving the theorem, let us introduce a symbol ψ into the functions P_NEG, P_MAX and P_MIN. The symbol ψ stands for an unknown real value between 0 and 1 that satisfies

$$\psi \times r = \psi,\ \psi \times \psi = \psi,$$

$$\psi + t = \psi,\ \psi + \psi = \psi,$$

where $\times$ and + are the arithmetic multiplication and addition, r is a non-zero real number, and t is any real number. Then the "probability function" P_S for a node S in the given search tree will be defined as follows:

If S is a leaf node left unexamined by algorithm X, for each value $v \in [-i, i]$

$$P_S(v) = \psi;$$

if S is an examined leaf node, then P_S is determined by the evaluation function of algorithm X;

if S is a non-leaf node, then

$$P_S = P\_MAX(P\_NEG(P_{S_1}), ..., P\_NEG(P_{S_w})).$$

If there is only one value $v \in [-i, i]$ for which $P_S(v) = \psi$, then let

$$P_S(v) = 1 - \sum_{j \neq v} P_S(j).$$

By induction on the height of a node S in the given search tree, we can prove that if function $P_S(v) = \psi$ for some $v \in [-i, i]$, then different probability functions can be assigned to the unexamined leaf nodes to get different backed-up probability functions for the node S. The appearance of symbol ψ in a probability function P_S means that algorithm X cannot solve the game tree rooted at node S, because by assigning different probabilities to unexamined leaf nodes, P_S will be changed.

For each such generalized probability function P_S, two integer ranges, C_S and U_S, can be defined so that the lower bound of C_S is the minimum domain value v for which $P_S(v) \neq \psi$ and $P_S(v) > 0$, and that the upper bound of C_S is the maximum domain value v for which $P_S(v) \neq \psi$ and $P_S(v) > 0$. Similarly, the lower bound of U_S is the minimum domain value v for which $P_S(v) = \psi$, the upper bound of U_S is the maximum domain value v for which $P_S(v) = \psi$. Note that for a position S, one, but not both, of the ranges C_S and U_S may be empty.

Lemma 2. For a node S, if both the ranges C_S and U_S are not empty, then one of the following two situations must be true.

$$lower\_bound(C_S) > upper\_bound(U_S),$$

or

$$lower\_bound(U_S) > upper\_bound(C_S).$$

Proof.

By induction on the height of the position S in the given game tree, and the details are omitted here. $\square$

The above lemma characterizes the structure of the probability function P_S for any position S. In fact, we can prove that if the range U_S is nonempty, then for any $v \in U_S$, we have

$$P_S(v) = \psi.$$

For any such value v, $P_S(v)$ can assume at least two (in fact, infinitely many) different probabilities by independently varying the probability function of an unexamined leaf node.

According to the ranges U_S and C_S, we will say that the position S is *solved*, *semi-solved* or *live* if the range U_S is empty, not empty and not equal to [-i, i], or equal to [-i, i], respectively. Note that algorithm X computes the probability function for the merit value of a position only if the status of the corresponding node is solved.

Lemma 3. For any solved non-leaf node S in the given tree, one of its successors S_t must be a solved node such that the upper bound of C_{S_t} is less than or equal to the lower bound of U_{S_j}, for all $j \neq t$.

Proof.

Without loss of generality, suppose node S has only two successors, S_1 and S_2. First, let us assume both ranges U_{S_1} and U_{S_2} are non-empty. We will show that this assumption will lead to a conclusion that node S is not a solved position.

Let $U_{S_1} = [l_1, u_1]$ and $U_{S_2} = [l_2, u_2]$, and suppose $u_1 \leq u_2$. Then

$$P\_MAX(P\_NEG(P_{S_1}), P\_NEG(P_{S_2}))(v) = \psi$$

for any $v \in [-u_1, -l_1]$. This shows that S cannot be solved by algorithm X.

Now assume range U_{S_1} is empty and U_{S_2} is not empty. If the upper bound of C_{S_1} is greater than the lower bound of U_{S_2}, the probability function P_S will have

$$P_S(-c_1) = \psi$$

and

$$P_S(-c_1 + 1) = \psi,$$

where c_1 is the upper bound of the range C_{S_1}. Therefore, we must have $c_1 \leq l_2$, and this implies the conclusion presented in Lemma 3. $\square$

If the range U_S is understood as the *uncertain part* of the domain range [-i, i] for the position S, we will say that the α-β algorithm P_AB(S, α, β) *visits* the certain part of position S when the intersection of the two ranges $[\alpha, \beta]$ and U_S consists of at most one integer, which is either α or β.

Lemma 4. If algorithm P_AB visits an interior solved node S, then there is an ordering of the nodes under S such that P_AB will visit a successor S_j of S only if algorithm X visits it, and when P_AB visits S_j, P_AB will visit its certain part.

Proof.

Let P_AB visit S_t first, as determined by Lemma 3. Note that S_t is such a solved node that the upper bound of C_{S_t} is less than or equal to the lower bound of U_{S_j} for all $j \neq t$. Then the reset of α and β will make P_AB either visit the certain part of a successor, S_j ($j \neq t$), or return from S. □

Lemma 5. If algorithm P_AB visits the certain part of a non-leaf node S, there is an ordering of the nodes under S such that P_AB will visit a leaf node under S only if that leaf is visited by algorithm X.

Proof.

This lemma is proved by induction on the height of S in the given search tree. In the following, it is assumed that S is visited with a window $[\alpha, \beta]$. By Lemma 4, we can suppose position S is semi-solved and, for simplicity, it has only two successors S_1 and S_2. If both of the ranges U_{S_1} and U_{S_2} are subsets of $[-upper\_bound(U_S), -lower\_bound(U_S)]$, when algorithm P_AB visits S_1 or S_2, P_AB will also visit the certain part, and the lemma is true by the induction hypothesis. If one of the ranges, for example, U_{S_1}, is not contained in $[-upper\_bound(U_S), -lower\_bound(U_S)]$, then, because

$$P\_NEG(P_S) = P\_MIN(P_{S_1}, P_{S_2}),$$

we must have

$$-lower\_bound(U_S) < upper\_bound(U_{S_1}).$$

This fact implies

$$-\beta \geq upper\_bound(U_{S_2})$$

and

$$P_{S_2}(v) = 0$$

for any $v \in [upper\_bound(U_{S_2})+1, i]$, or $-\alpha \leq \min(lower\_bound(U_{S_1}), lower\_bound(U_{S_2}))$. In the former case, if P_AB visits S_2 first, then P_AB will visit the certain part of node S_2 with window $[-\beta, -\alpha]$, and the function P_AB(S_2, $-\beta$, $-\alpha$) returns $<-\beta, 1>$. Therefore, P_AB will prune S_1. In the latter case, the visiting window $[-\beta, -\alpha]$ does not intersect with either of the two ranges, U_{S_1} or U_{S_2}. This proves the lemma. □

Lemma 5 implies that there is an ordering of successor nodes in the given search tree such that if P_AB visits the certain part of a node S that is visited by algorithm X, then P_AB will visit only those nodes under S that are also visited by algorithm X. We now turn to the proof of Theorem 2.

Proof of Theorem 2.

Because the root position S_0 of the given tree must be a solved node, P_AB(S_0, $-i$, i) visits the certain part of S_0. By Lemma 5, Theorem 2 follows. □

Since a search tree is finite, there must be an algorithm, say A, that visits the least number of leaf nodes among all the algorithms that solve the search tree. Theorem 2 implies that by ordering successors in the search tree, P_AB will visit only those leaf nodes that are visited by the algorithm A. Therefore, by reordering successor nodes in a search tree, the algorithm P_AB will visit the least number of leaf nodes among all the algorithms that solve the tree.

5. Applications

Some variations or applications of the probability-based α-β algorithm P_AB are now presented. It is expected that the algorithm P_AB will have as many applications as its point-value version, as studied by Knuth and Moore[1]. For example, P_AB can be used in the iterative deepening search version for point-value game tree search[5], and also to other well-studied α-β pruning based algorithms[6]. Only three examples of the applications of P_AB will be briefly described here.

5.1. Range-Based Game Tree Search

In range-based game tree search, the merit value D_S of a position S is described by a range $[l, u]$, where l and u are the lower and upper bounds of all possible merit values of S[3]. If S is a leaf node in the search tree, the bounds are returned by an evaluation function; otherwise, they will be backed-up from its successors $S_1, ..., S_w$ by

$$u = \max(-l_1, ..., -l_w) \text{ and } l = \max(-u_1, ..., -u_w),$$

where $S_1, ..., S_w$ are described by the ranges $[l_1, u_1], ..., [l_w, u_w]$, respectively.

The probability-based α-β algorithm can be directly used as a range-based one if we assume that the merit value D_S for any leaf node S is uniformly distributed among the values of the range $[l, u]$, and then ignore the probabilities of the values between the lower and upper bounds of the backed-up probability function for the root position. In this way, the range for the possible merit values of an initial position can be backed up by P_AB. Since the formulas for the operations of range-based game tree search are much simpler than those for probability operations, the following range functions *R_NEG*, *R_MAX* and *R_MIN* can be substituted for *P_NEG*, *P_MAX* and *P_MIN* in the function *P_AB*

$$R\_NEG([l, u]) = [-u, -l],$$

$$R\_MAX([l_1, u_1], ..., [l_w, u_w]) = [\max(l_1, ..., l_w), \max(u_1, ..., u_w)]$$

$$R\_MIN([l_1, u_1], ..., [l_w, u_w]) = [\min(l_1, ..., l_w), \min(u_1, ..., u_w)]$$

to get the so called range-based α-β pruning algorithm R_AB. Note that this range-based algorithm makes use of depth-first traversing to search a tree, and so is different from B*[3].

```
function R_AB($S$: position; $\alpha$, $\beta$: integer): range;
var
  TR0, TR1: range;
  j: integer;
begin
1.  if $S$ is a leaf node then
        return $R\_MIN(R\_MAX(R_S, [\alpha, \alpha]), [\beta, \beta])$;
    comment: Range $R_S$ is obtained by evaluating $S$
              from the corresponding player's viewpoint.
2.  determine the successor positions, $S_1, ..., S_w$, of $S$, where $w > 0$;
3.  TR0 := $[\alpha, \alpha]$;
4.  for $j$ := 1 to $w$ do
     begin
5.    TR1 := $R\_NEG$(R_AB($S_j$, $-\beta$, $-\alpha$));
6.    TR0 := $R\_MAX$(TR0, TR1);
7.    $\alpha$ := lower_bound(TR0)
8.    if $\alpha = \beta$
        then return TR0;
     end;
9.  return TR0;
end.
```

5.2. Informed Game Tree Search

Ibaraki[7] recently proposed an "informed" game tree search model based on the availability of some heuristic information, embodied as upper bound, *U_bound*(S), and lower bound, *L_bound*(S), on the merit value of an interior node S. The utilization of this kind of information has been recognized as a key factor for designing good game-playing programs[7][6]. The heuristic information available at interior nodes can be used in probability-based α-β pruning by replacing the statement 3 of P_AB with the following series of statements:

```
3.1    $\alpha := \max(\alpha, L\_bound(S))$;
3.2    $\beta := \min(\beta, U\_bound(S))$;
3.3    **if** $\alpha = \beta$ **then return** $<\beta, 1>$;
3.4    TP0 := $<\alpha, 1>$;
```

Because the new window after the execution of statement 3.4 is usually a proper subset of the parameter window, this narrower window can be used to prune more nodes.

5.3. Probability-Based B*—PB*

The generalization of B* algorithm, PB*[4], makes use of a best-first search strategy. Since relatively reliable bounds for a leaf position can be generated with some such technique as the null-move[8], we can suppose that the expansion of a leaf node in the best-first search, and back-up of information from its successors will not increase the upper bound or decrease the lower bound of the possible merit values of the expanded node. This assumption is similar to the one employed by Ibaraki[7].

It can be proved that PB* works in the model proposed in Section 2 as well. As a matter of fact, the probability-based α-β pruning technique can be incorporated into PB* to cut off some nodes from the search. In PB*, the following three operations are repeated until a best-move is found: find a potential best node, find a path from the root position of search tree to a leaf, and expand the leaf node. Suppose a node S is called with a search window $[\alpha, \beta]$. When we choose one node S_j from the successors S_1, ..., S_w of a node S according to the ProveBest or DisproveRest strategy (cf. [4]), the probability function

$$P = P\_MIN(P\_MAX(P\_NEG(P\_MIN(P_{S_1}, ..., P_{S_w})), <\alpha, 1>), <\beta, 1>)$$

can be used to set a new α by

$$\alpha := lower\_bound(P).$$

If $\alpha = \beta$, then the node S_j should not be searched and a new strategy should be chosen; otherwise, the node S_j will be visited with search window [$-\beta$, $-\alpha$]. When a node S is expanded, each successor S_j is examined as above, and if $\alpha = \beta$, the successors will not be included in the search tree. In this way, both the search time and the memory required to store the search tree will be reduced.

6. Pruning Efficiency

Random trees can be used to assess the pruning efficiency of algorithm P_AB. Given integers d and w, a random tree can be generated so that each interior node has less than w successor nodes and the tree consists of at most d levels. The lower and upper bounds of the probability function for the merit value of each node is also generated randomly. An assumption is that the root position of each such random tree has a preselected probability function for its merit value. Therefore, using a variation on an earlier scheme [6], the probability function (or its bounds) for the root position is generated first, followed by the probability functions for the successor positions. To generate consistent probability functions for the successor nodes, if a node S has lower and upper bounds l and u respectively, one of the successors is randomly chosen and its upper bound is set to the minimum upper bound $-l$. The minimum lower bound $-u$ is similarly assigned to another successor. The lower bounds of other successor nodes will be determined by randomly choosing integers between $-u$ and i, and the upper bounds by choosing integers between $-l$ and i, where i is the domain bound. Since the lower bound and upper bound of a successor are chosen randomly and independently, the former may be greater than the latter; if this is so, the two bounds are exchanged.

For the efficiency experiment on the probability-based α-β pruning scheme, ten random trees were generated for each combination of d and w, where d = 5, ..., 8 and w = 4, ..., 7. Here, the domain value i was set to 6, i.e., the domain of the probability functions was the integer range [-6, 6]. With these settings, Table 1 presents the total number of leaf nodes for each set of ten trees. In the recursive function P_AB, probability functions are passed as parameters, but only their lower and upper bounds are used in the pruning. In this case, since only the bounds of the probability functions are used, P_AB reduces to the range-based pruning algorithm, R_AB. For our test data, the number of leaf nodes visited by P_AB (or R_AB) for each combination of d and w is shown in Table 2. Table 3 presents the efficiency of the pruning and also shows that the relative efficiency, as measured by the fraction of leaf nodes that are pruned by P_AB, increases as either the width or depth of search tree increases. Informally speaking, the larger the search tree, the greater the fraction of nodes that will be pruned by probability-based α-β.

Table 1. The Number of Leaf Nodes in Ten (d, w)-Random-Trees

Depth (d)	Width Limit (w)			
	4	5	6	7
5	323	919	1943	2022
6	623	2865	5621	9827
7	2149	9529	21820	30402
8	3537	26371	70289	220723

Table 2. The Number of Leaf Nodes Visited by P_AB

Depth (*d*)	Width Limit (*w*)			
	4	5	6	7
5	250	692	1100	1044
6	538	1481	3074	3827
7	1371	5110	10076	11528
8	2593	14268	29867	72564

Table 3. The Efficiency of P_AB for Different (*d*, *w*)

Depth (*d*)	Width Limit (*w*)			
	4	5	6	7
5	.22	.24	.43	.48
6	.13	.48	.45	.61
7	.36	.46	.53	.62
8	.26	.45	.57	.67

7. Concluding Remarks

The efficient search scheme—depth-first traversal—is introduced into probability-based game tree search. We show that the α and β bounded search windows can still be used here, and generalize the α-β pruning technique for probability-based game tree search. The so-called probability-based game tree pruning technique inherits some good properties from the point-value version. Several applications of this pruning method are illustrated, one of which is the "degeneration" of this probability-based algorithm into a range-based one. It is expected that this probability-based game tree pruning technique will have as many applications as its point-value version. Probability experiments are used to show that it can be exploited to effectively prune the search of some subtrees.

Acknowledgement

Partial funding of this research was provided through Grant A7902 from the Natural Sciences and Engineering Research Council of Canada.

References

1. D. E. Knuth and R. W. Moore, An Analysis of Alpha-Beta Pruning, *Artificial Intelligence 6*, (1975), pp. 293-326.
2. A. L. Brudno, Bounds and Valuations for Abridging the Search of Estimates, *Problems of Cybernetics 10*, (1963), pp. 225-241. Translation of Russian original appearing in Problemy Kibernetiki 10, pp. 141-150.
3. H. Berliner, The B* Tree Search Algorithm: A Best-First Proof Procedure, *Artificial Intelligence 12*, (1979), pp. 23-40.
4. A. J. Palay, *Searching with Probabilities*, Pitman Advanced Publishing Program, Boston, 1985. Also, Ph.D. Thesis, CMU, 1983.
5. D. J. Slate and L. R. Atkin, Chess 4.5-The Northwestern University Chess Program, in *Chess Skill in Man and Machine*, P. Frey (ed.), Springer-Verlag, 1977, pp. 82-118.
6. T. A. Marsland, A. Reinefeld and J. Schaeffer, Low Overhead Alternatives to SSS*, *Artificial Intelligence 31*, (1987), pp. 185-199.
7. T. Ibaraki, Generalization of Alpha-Beta and SSS* Search Procedures, *Artificial Intelligence 29*, (1986), pp. 73-117.
8. D. Beal, A Generalized Quiescence Search Algorithm, *Artificial Intelligence*, 1988, (to appear). Also, Experiments with the null move, *Advances in Computer Chess 5*, Elsevier.