January 31, 2001

Nathan Segerlind

April 22, 2002

The boolean decision tree model of computation is introduced in both its deterministic and randomized forms. A specific function is demonstrated in which randomization provides a polynomial speedup over determinism.

1 Decision Trees

Definition 1.1 *A* decision tree *is a rooted binary tree with every internal node labelled with a variable, and every leaf is labelled with a* 0 *or a* 1*. The arcs leaving an internal node are labelled with* 0 *and* 1*.*

The branches of the tree can be identified with partial assignments. $Br(T)$ *denotes the set of partial assignments arising from branches of* T*. For each* $\pi \in Br(T)$, l_π *denotes the label on the leaf underneath the path* π*.*

We say that T *computes* f *if* $\forall \pi \in Br(T)$, $f \restriction_\pi = l_\pi$*.*

Notice that we could change the definition of decision trees to compute non-boolean functions by assigning the leaf labels from an arbitrary set.

The uniform analogue of a binary decision tree is a sublinear time random access machine.

Lemma 1 *If* f *is computed by a* $t(n)$ *deterministic random access machine, then for every input length* n, f_n *is computed by a decision tree of height* $\leq t(n)$*.*

Proof:

Let M a be a random access machine which computes $f(x)$ in time $t(|x|)$. Let n be given. We build a decision tree T of height $\leq t(n)$ computing f_b as follows:

Let $b_1, \ldots b_k$ be a sequence of bits with $k \leq t(n)$ so that it is possible for the $b_1, \ldots b_k$ to be the first k bits scanned by a computation of M. Let $C_{\vec{b}}$ be the configuration of M immediately after scanning the kth bit when the first k bits scanned are $\vec{b}$. Let $S(\vec{b}) = \{v_1, \ldots v_k\}$ denote the multi-set of variables scanned by M (within the first k scans) when the first k values scanned are $b_1, \ldots b_k$. Let $A(\vec{b})$ be the partial assignment to $S(\vec{b})$ induced by $\vec{b}$.

For each $\vec{b}$ of length l, which $\vec{b}$ is a consistent family values for the first l scans of M, we build a decision tree for $f \restriction_{A(\vec{b})}$ of height $\leq t(n) - l$.

Base case: $l = t(n)$. In this case, because M runs in time $t(n)$, $f \restriction_{S(\vec{b})}$ is constant, and the decision tree is simply a single node labelled with this value.

Let $l < t(n)$ be given, and assume that for all m, $l < m \leq t(n)$, for each $\vec{b}$ of length m, there is a decision tree of height $\leq t(n) - m$ for $f \restriction_{A(\vec{b})}$.

Let $\vec{b}$ be a sequence of bits of lenght l. If $f \restriction_{A(\vec{b})}$ is a constant, then T is simply a single node labelled with that value. Otherwise, the computation of M starting from configuration $C_{\vec{b}}$ must query another variable and halt within $\leq t(n) - l$ steps. Therefore, there are trees T_0, T_1, computing $f \restriction_{A(\vec{b},0)}$, $f \restriction_{A(\vec{b},1)}$, each of height $\leq t(n) - l - 1$.

Let x_j be the first variable read in the computation of M starting in configuration $C_{\vec{b}}$ (such a variable must exists because $f \restriction_{A(\vec{b})}$ is non-constant). Label the root of T with x_j. Underneath the branch labelled 0, place T_0 and underneath the branch labelled 1, place T_1.

■

Definition 1.2 *A* randomized decision tree *is rooted binary tree in which every internal node is labelled either with a variable or RANDOM, and every leaf is labelled with a value. The arcs leaving an internal node are labelled* 0 *and* 1.

The branches of the tree can be identified with a pair consisting of a partial assignment to the variables, π, and a sequence of random choices, $\vec{r}$. $Br(T)$ denotes the set of assignment, random string pairs arising from branches of T. For each $(\pi, \vec{r}) \in Br(T)$, $l_{\pi,\vec{r}}$ denotes the label on the leaf underneath the path $(\pi, \vec{r})$.

The number of queries along a path is the number of nodes which are labelled with a variable.

The number of random choices along a path is the number of nodes whic are labelled RANDOM.

The probability of a branch is 2^{-k} where k is the number of random choices along that branch.

Definition 1.3 *Let f_n be a function in n boolean variables. $BPD_\epsilon(f_n)$ is the minimum q so that there exists a randomized decision tree making $\leq q$ queries along each path, with the probability of correctness $\geq 1 - \epsilon$.*

Definition 1.4 *Let f_n be a function in n boolean variables. $ZDiff(f_n)$ is the minimum q so that there exists a randomized decision tree so that every branch yields the correct answer, and the expected number of queries made is $\leq q$.*

2 Read-Once Boolean Formulas

Definition 2.1 *A* boolean formula *is a binary tree, whose internal nodes are labelled with* $\vee, \wedge$*, and whose leaves are labelled with input variables or their negations.*

Definition 2.2 *A* read-once boolean formula *is a boolean formula so that the leaf labels are distinct, positive literals.*

Lemma 2 *If f is computed by a read-once boolean formula with n leaves, then $DT(f_n) = n$.*

Proof: Clearly, $\mathrm{DT}(f_n) \leq n$.

We will show that $\mathrm{DT}(f_n) \geq n$ by induction on n. Clearly the lower bound holds when $n = 1$.

Let $n > 1$ be given and assume that the lemma holds for all read-once boolean formulas of $m < n$ variables. Let f_n be a function computed by a read-once boolean formula F. Let T be a shortest decision tree computing f_n. Notice that every variable queried by T appears on a leaf of F.

Let x_i be the first variable queried by T. Let g be the label on the node immediately above the leaf labelled x_i in F, and let H be the sub-formula sibling of x_i. Suppose that g is an OR gate. Let G be F with the sub-formula under g removed, so that $F = G(x_i \vee H(\vec{x}), \vec{y})$, where $\vec{x}$ and $\vec{y}$ are disjoint sets of k and $n-k-1$ variables, respectively. Notice that $T \restriction_{x_i=0}$ computes $G(H(\vec{x}), \vec{y})$. This is a read-once boolean formula on $n-1$ variables, and by the induction hypothesis, it requires height at least $n-1$. Therefore, T has height at least n. The case when g is an AND gate is handled similarly by considering $T \restriction_{x_i=1}$.

■

3 Randomized Evaluation of Game Trees

Definition 3.1 *A boolean game tree is a balanced read-once boolean formula so that along each path from the root to a leaf, the $\wedge$ and $\vee$ gates alternate.*

The following randomized algorithm evaluates game trees.

randEval($T, \vec{x}$)
    if T is a leaf labelled with x_i,
        return the value of x_i
    let T_0 and T_1 be the subtrees of T directly underneath the root
    choose $b \in \{0, 1\}$ at random
    let $t =$ randEval($T_b, \vec{x}$)

if the root of T is labelled with an OR and $t = 1$,
    return 1
if the root of T is labelled with an AND and $t = 0$,
    return 0
let $t' = \text{randEval}(T_{1-b}, \vec{x})$
    return t'

Theorem 3 *randEval evaluates a read-once boolean formula with n leaves in time* $O(n^{0.793})$.

Proof:

G_k is the expected time to evaluate a depth k formula when either *(i)* the root is labelled $\vee$ and the formula evaluates to 1, or, *(ii)*, the root is labelled $\wedge$ and the formula evaluates to 0.

B_k is the expected time to evaluate a depth k formula when either *(i)* the root is labelled $\vee$ and the formula evaluates to 0, or, *(ii)*, the root is labelled $\wedge$ and the formula evaluates to 1.

The alternation of the operations gives us a crucial recurrence relation: if an instance is bad, then *both* recursive sub-instances will be good. Therefore, evaluating a bad instance of depth k requires the same number of queries as two good instances, each of depth $\leq k-1$.

$$B_k \leq 2G_{k-1}$$

When we evaluate a good instance of depth k, with probability $\geq \frac{1}{2}$ we only evaluate one subtree, a depth $\leq k-1$ bad instance. In the other case, we evaluate a depth $\leq k-1$ bad instance and a depth $\leq k-1$ good instance.

$$G_k \leq \frac{1}{2}B_{k-1} + \frac{1}{2}\left(G_{k-1} + B_{k-1}\right)$$

We combine these two inequalities and we have:

$$G_k \leq \frac{1}{2}(2G_{k-2}) + \frac{1}{2}\left(G_{k-1} + 2G_{k-2}\right) = 2G_{k-2} + \frac{1}{2}G_{k-1}$$

By induction we have $G_k \leq 3^{\frac{k}{2}}$, and therefore $B_k \leq 2 \cdot 3^{\frac{k-1}{2}} = O\left(3^{\frac{k}{2}}\right)$.

In a game tree with n leaves, $k = \log_2 n$, so the number of queries needed is $O\left(3^{\frac{k}{2}}\right) = O\left(3^{\frac{\log_2 n}{2}}\right) = O\left(n^{\frac{\log_2 3}{2}}\right)$. ■

4 Real Valued Game Trees

The same technique can be used in the evaluation of real-valued game trees, a problem which occurs in artificial intelligence.

Definition 4.1 *A* real-valued game tree *is a balanced binary tree in which the internal nodes are labelled with* $\min_a$ *and* $\max_b$ *and the leaves are labelled with real numbers. Furthermore, all internal nodes of the same depth have the same label, and the labels alternate between levels.*

The problem is to determine value of the tree.

Suppose that the root is labelled with $\min_a$. If the value of the tree is a, then at least one subtree has the value a, and if were to evaluate this subtree before evaluating the other, we would save time, exactly as in the the previous example, by evaluating one bad case.

If the root is labelled with $\min a$, and value of the tree is some c, $a < c \leq b$, then we must evaluate both subtrees. However, by randomly choosing which subtree to evaluate, we can get a chance of savings. Suppose one subtree evaluates to e and the other to f, with $e < f$. If we evaluate the subtree evaluating to e first, and then evaluate the other subtree *with the modification that its upper bound is now* e, the second evaluation will be a good instance (for a tree with root $\max_e$ of depth $k - 1$).

Here is the algorithm.

randEvalReal($T, \vec{x}$)
    if T is a leaf labelled with x_i,
        return the value of x_i
    let T_0 and T_1 be the subtrees of T directly underneath the root
    choose $r \in \{0, 1\}$ at random
    let $t =$ randEvalReal($T_r, \vec{x}$)
    if the root of T is labelled with $\min_a$
        if $t \leq a$,
            return a
        otherwise,
            let T' be T_{1-r} with the root label changed from $\max_c$ to $\max_t$
                (but if T_{1-r} is a leaf don't do anything to it)
            let $t' =$ randEvalReal($T', \vec{x}$)
            return $\min\{t, t'\}$
    if the root of T is labelled with an $\max_b$
        if $t \geq b$,
            return b
        otherwise,
            let T' be T_{1-r} with the root label changed from $\min_c$ to $\min_t$
                (but if T_{1-r} is a leaf don't do anything to it)
            let $t' =$ randEvalReal($T', \vec{x}$)

return $\max\{t, t'\}$

The recurrences give us

$$G_k \le \frac{1}{2}B_{k-1} + \frac{1}{2}\left(B_{k-1} + B_{k-1}\right) = \frac{3}{2}B_{k-1}$$

$$B_k \le \frac{1}{2}(2B_{k-1}) + \frac{1}{2}\left(B_{k-1} + G_{k-1}\right) = \frac{3}{2}B_{k-1} + \frac{1}{2}G_{k-1}$$

Combining these gives us

$$B_k \le \frac{3}{2}B_{k-1} + \frac{3}{4}B_{k-2}$$

We can solve the characteristic polynomial of the recurrence:

$$x^2 - \frac{3}{2}x - \frac{3}{4}$$

and bound $B_k \le (1.9)^k = (1.9)^{\log_2 n} = 2^{\log_2 n \log_2 1.9} = n^{\log_2 1.9}$.