A BRANCH-AND-BOUND ALGORITHM FOR
OPTIMAL EVALUATION OF MONOTONIC
BOOLEAN FUNCTIONS

by
Yu. Breitbart and A. Reiter

Technical Report No. 35
April 1974
ABSTRACT

For monotonic Boolean functions, a branch-and-bound algorithm is given for constructing an optimal decision tree (sequential evaluation procedure). The tree is optimal in minimizing the average number of variables which need to be examined.
1. **Introduction**

Let $f(x_1, \ldots, x_n)$ be a monotonic Boolean function (MBF). A sequential evaluation algorithm $\Phi(f)$ specifies which $x_i$ to examine first. Depending on whether $x_i=0$ or $x_i=1$, the evaluation terminates or another $x_i$ is specified for examination, etc. Such algorithms are conveniently represented graphically as binary decision trees. Thus, for the function $f = x_1 \lor x_2x_3$ we might have the decision tree of Figure 1.

![Decision Tree Diagram]

**Figure 1** A decision tree for $f = x_1 \lor x_2x_3$.

Nodes with variables will be called **interior** nodes, while nodes labelled * will be called terminal nodes.
For a given point \( \alpha = (\alpha_1, \ldots, \alpha_n) \) in Boolean \( n \)-space, a decision tree \( \Phi \) evaluates \( f(\alpha) \) in \( n = N(f, \Phi, \alpha) \) steps\(^*\). We define the expected cost of \( \Phi \) by
\[
E(f, \Phi) = \sum_{\alpha \in B^n} \frac{N(f, \Phi, \alpha)}{2^n}
\]

Among all possible decision trees for \( f \), some have lower expected cost than others. Denote the minimum by
\[
E(f) = \min_{\Phi} E(f, \Phi).
\]

We say that a decision tree \( \Phi \) is optimal iff \( E(f, \Phi) = E(f) \).

This paper is concerned with algorithms for constructing an optimal decision tree. In [1], the authors described a fast practical algorithm which however does not guarantee an optimal solution. The techniques employed there are combinatorial, and are based on the minimal disjunctive normal form representations of \( f \) and its dual function \( f^* \). The present work employs a branch-and-bound partial exhaustive search method such as used in solving the "traveling salesman" problem. Although the solution is optimal, the algorithm may require an enormous of time and space if the number of variables is large.

In section 2 we associate with each node of a decision tree a "node cost", and prove (Theorem 1) that the sum of these costs gives the expected cost of the

\(^*\) A step is the examination of the value of one variable. For motivation, assume that each \( x_i \) is located in secondary storage, requiring a time-consuming disk access.
tree in the sense of (1.1). We use these costs in formulating the algorithm in Section 3. This algorithm is actually a refinement of the algorithm proposed in [2] for converting decision tables into programs.

2. Mathematical Background

Let \( f(x_1, \ldots , x_n) \) be a monotonic Boolean function of \( n \) variables. We denote by \( f^0_i, f^1_i \) the functions derived from \( f \) by substituting \( x_i=0, x_i=1 \) respectively. Obviously if \( f \) is monotonic, so are \( f^0_i \) and \( f^1_i \).

For each coordinate \( x_i \) of \( B^n \), let \( z_i, t_i \), denote the cardinalities of the sets of points \( \{\alpha\} \) where \( f(\alpha)=1 \) and \( i\)-th coordinate of \( \alpha \) is 0 or 1 respectively. Define

\[
(2.1) \quad h_i = h_i(f,n) = t_i - z_i
\]

For this definition, \( f \) need not depend essentially(*) on all of its variables. The values of the \( h_i \)'s therefore depend not only on the function \( f \) but also on the dimensionality of the Boolean space over which \( f \) is defined.

By the monotonicity of \( f \), we deduce that

\[
(2.2) \quad h_i \geq 0 \text{ for all } i,
\]

and also that

\[
(2.3) \quad "h_i=0" \equiv "f \text{ does not essentially depend on } x_i".
\]

The number \( h_i \) can be thought of as measuring the degree of dependence of \( f \) on \( x_i \). Thus (2.3); also, \( h_i = 2^{n-1} \) iff \( f \) is totally dependent on \( x_i \); i.e.

\[
(*) f \text{ is said to depend essentially on } x_i \text{ iff } f^0_i \neq f^1_i.
\]
f(x_1, \ldots, x_n) = x_1. For all other functions, a higher value of \( h_1 \) means that the value of \( f \) is well correlated with the value of \( x_1 \). For a given \( f \), the computation of \( h_1 \) is straightforward; its solution is left to the reader.

Now denote

\[
H \equiv H(f,n) = 2 \sum_{i=1}^{n} h_i
\]

**Example 1**

Let \( f(x_1, x_2, x_3) = x_1 \lor x_2 \land x_3 \).

We have

\[
\begin{align*}
t_1 &= 4 & z_1 &= 1 & h_1 &= 3 \\
t_2 &= 3 & z_2 &= 2 & h_2 &= 1 \\
t_3 &= 3 & z_3 &= 2 & h_3 &= 1
\end{align*}
\]

\[ H = 10. \]

Also define

\[
(2.5) \quad \bar{h}_1 = 2^{n-1} - h_1
\]

Now let \( \Phi \) be an algorithm for evaluating \( f \in \text{MBF} \) over \( B^n \). Then \( \Phi \) is a tree whose non-terminal nodes each contain some variable \( x_i \), and have descending branches labelled 0 or 1. Let \( a_1, \ldots, a_m \) be an enumeration of the non-terminal nodes in some order. We associate with each node \( a_j \) a number \( c_j \) as follows. If \( a_j \) is the root node containing \( x_1 \), then \( c_j = \bar{h}_1 \). Otherwise, \( c_j \) is the \((k+1)th\)
node in a path leading directly from the root. Denote by $g$ the function of $n-k$ variables obtained from $f$ by substituting into $f$ the value (on the appropriate branch) of each of the $k$ variables which appear above $a_j$. Then

\begin{equation}
(2.6) \quad c_j = \bar{h}_j(g, n-k).
\end{equation}

We shall call $c_j$ "the cost of evaluating node $a_j". The motivation for this will become apparent later.

**Example 2**

For $f$ of example 1, let $\Phi$ be the tree of Figure 1. Then $c_1 = 2^2 - h_2 = 3$; $c_2 = 2^{2-1} - 2 = 0$; $c_3 = 2^{2-1} - 1 = 1$; $c_4 = 0$.

Instead of working with the expected cost function of (1.1) it is more convenient to define

\begin{equation}
(2.7) \quad L(f, \Phi) = 2^n \mathbb{E}(f, \Phi) = \sum_{\alpha \in B^n} N(f, \Phi, \alpha)
\end{equation}

The following theorem gives a convenient means for obtaining the expected cost of an evaluation algorithm.

**Theorem 1** Let $f \in MBF(n)$, $n \geq 1$; and let $\Phi$ be an evaluation tree whose non-terminal nodes are $(a_1, \ldots, a_m)$. Then

\begin{equation}
(2.8) \quad (f, \Phi) = H+2 \sum_{j=1}^{m} c_j
\end{equation}

**Proof:** Induction on $n$, the number of variables in $f$. $n=1$: Then $f$ is either constant (as a function of one variable), or is $x_1$: in either case it is easily verified that our statement is true.
**Induction step:** Assume the theorem true for all functions of (n-1) variables and their evaluation algorithms. Now let $f \in \text{MBF}(n)$ and let $\Phi$ be given. The $\Phi$ starts with some $x_i$ at node $a_k$ (see Figure 2), and induces evaluation algorithms $\Phi_0, \Phi_1$ on the functions of n-1 variables $f^0_i, f^1_i$ respectively.

We have (by definition 2.7)

\begin{equation}
L(f, \Phi) = 2^n + L(f^0_i, \Phi_0) + L(f^1_i, \Phi_1),
\end{equation}

and hence by our induction hypothesis

\begin{equation}
L(f, \Phi) = 2^n + H(f^0_i, n-1) + H(f^1_i, n-1) + 2 \sum_{j \neq k} c_j.
\end{equation}

since the non-terminal nodes of $\Phi_0$ and $\Phi_1$ are those of $\Phi$ except for the root node $a_k$.

By the definition of $H$ it is easy to verify that

\begin{equation}
H(f, n) = H(f^0_i, n-1) + H(f^1_i, n-1) + 2 (t_i - z_i),
\end{equation}

where $t_i$ and $z_i$ are computed for $f$. Since $x_i$ is the root node, we have

\begin{equation}
c_k = h_i = 2^{n-1} - (t_i - z_i).
\end{equation}

Substituting (2.11) and (2.12) into (2.10), we obtain
Theorem 1 can be interpreted as follows. Given an evaluation tree $\Phi$, the cost of using $\Phi$ is the sum of the fixed "overhead" $H$ plus the costs of evaluating each one of its nodes.

3. An algorithm for building an optimal decision tree

Let $\Psi$ be a given partially constructed evaluation tree (PET) for $f$ with root $x_i$.

Let $L=L(f,\Psi)$ be the (partial) sum of (2.7) formed over the nodes of $\Psi$.

Then by Theorem 1 $L$ is a lower bound for the cost of any evaluation tree which is an extension of $\Psi$. We make use of this fact in the following branch-and-bound algorithm for constructing an optimal decision tree.

To start, construct for each $x_i$ a PET which consists only of the root $x_i$, and compute the partial cost. For the tree with the lowest cost, construct the two branches corresponding to $x_i=0$ and $x_i=1$ respectively, and to each branch attach in turn each of the $x_i$'s on which the resulting function depends essentially. Compute the partial costs for each possibility. Of all the resulting PETs, pick the one with lowest cost. This may be one with $x_i$ at the root, or a different one. Continue to develop the tree with the lowest cost.
one level at a time by adding 0- and 1- branches at each of its non-terminal nodes. Eventually a constant value is reached in each path of the lowest-cost PET. This tree can be seen to be optimal.

The flow chart for the algorithm is given in Figure 3. If the algorithm always develops the same tree, i.e. always adds nodes to the PET most-recently worked on, we say that the algorithm is one-pass; otherwise it is said to back-track. When there is no backtracking the algorithm is reasonably fast (the computation of costs is relatively simple); unfortunately, for most functions we will do a fair amount of backtracking.

The data structure which may be used for the implementation is shown in Figure 4. Each node in a PET consists of the variable $x_i$, the associated node-cost $c_j$, and three continuation pointers: one each for the $x_i=0$ and $x_i=1$ branches, and one to the PET which has the same starting path but continues with a different $x_k$ at this point.

The following recursive procedure traverses the PET structure, computing the minimum cost and building a tree of pointers to the corresponding PET. This tree, which we call an E-tree, is a temporary structure, repeatedly built and freed. Each node consists of three entries: pointers LSON and RSON for the corresponding 0- and 1-branches (if any), and a pointer PETNODE to the corresponding PET node. The procedure calls itself to find optimal subtrees for its left and right sons, and then connects the two by adding a root node. When finished, the E-tree defines the PET with minimum cost.
Procedure mincost (a,c,p);

comment computes minimum cost and corresponding E-subtree for PET node.

Input: a is a pointer to the PET node.

Output: c is the computer cost,
        p is a pointer to the E-subtree root.

temporaries:

        min is the current best value for LSON or RSON, and
        g points to the corresponding E-subtree.
        g, r are temporary pointer variables.
        v holds temporarily a computed son cost.

procedures:

allocate (t) allocates one E-tree node from tree storage,
returns pointer t.

free (t) deallocates an E-subtree, whose root is pointed to by t
(if t=0, does nothing);

1. Initialization. Allocate (p). Set PETNODE(p)=a. Set cost(p) = ̃h(a)

2. Compute minimum cost and subtree for left sons. If LSON(a) = 0 or -1,
nothing to do: set LSON(p) = 0 and go to step 3. Otherwise:

2.1 "Initialize". Set min=∞, g=0, q=LSON(a).

2.2 Call mincost (q,v,r)

2.3 "Compare to previous best." Is v < min? If not, free(r), go to 2.5.
   If yes,

2.4 "Set new best". free(g), set min=v, g=r.
2.5 "Test next brother". Set q=NEXTPET(q); if q=0 go to 2.6 otherwise repeat step 2.2.

2.6 Set cost(p)=cost(p) + min, LSON(p)=g. Go to step 3.

3. Repeat step 2 with RSON instead of LSON.

4. Return.

The algorithm can be modified to make use of some additional information. In particular, the following result from [1] is relevant:

Theorem 2. Let f∈MBF have the form \( x_0 \lor g(x_1, \ldots, x_n) \) or \( x_0 \land g(x_1, \ldots, x_n) \). Then an optimal evaluation algorithm exists which starts with \( x_0 \).

Another useful modification is to look for symmetries among the variables. This can prevent extensive backtracking.

Even with these modifications, the proposed algorithm is not practical for functions of more than six variables. Worst-case estimates show that the number of PETs which have to be constructed approaches \( [n-(x-1)]^{2x-1} \) at level x.
For each $x_i$, form the PET consisting only of root $x_i$

Find a PET with minimum cost ("mincost" procedure)

For this PET: At every non-terminal node add branches corresponding to 0 and 1. Form new PETs by extensions: put on each new branch a node for every $x_k$ on which the resulting function depends essentially.

Were any extensions made?

Yes

Done. The minimum-cost PET is the optimal (complete) evaluation tree.

No

Figure 3. Constructing an optimal decision tree.
VARIABLE $x_i$  | COST: node cost $c_i$
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LSON: pointer to 0-branch PET extension</td>
<td>RSON: pointer to 1-branch PET extensions</td>
</tr>
</tbody>
</table>

a) One node in the tree. For LSON and RSON pointers, 0 denotes constant value (no extension possible, i.e. terminal node) and -1 denotes "no extension yet".

b) Graphic interpretation of the pointers for $f=x_1 \lor x_2x_3$. Only some of the PETs are shown.

**Figure 4.** Data structure for the PETs.
4. An example

For most functions, our algorithm begins by developing the optimal tree, and (after jumping around) returns to it. The reader is encouraged to try the algorithm on some simple functions. It may seem that we can dispense with backtracking and simply continue to develop the PET with lowest initial cost. Unfortunately, it is possible to construct functions where the initial PETs indicate a lower partial cost for non-optimal trees than for optimal ones. The following example is due to J. Perl. Let

\[ f = \neg y \lor x_1 x_3 \lor x_2 x_5 \lor x_1 x_4 x_5 \lor x_1 x_4 x_6 \lor x_1 x_5 x_6 \lor x_2 x_4 x_5 \lor x_4 x_5 x_6 \]

We have \( h_y = 28 \), and \( \bar{h}_y = 36 \) can be seen to be minimal. Thus the PET we must develop starts with \( y \). (See Theorem 2 above). At the next step we extend the \( y=0 \) branch (for \( y=1 \) the node is terminal). The two best main possibilities are \( x_1 \) with \( \bar{h}_{x_1} = 20 \) and \( x_3 \) with \( \bar{h}_{x_3} = 18 \) (see Figure 5). We develop both branches of the \( x_3 \) PET; the optimal continuations are \( x_4 \) and \( x_1 \) on the 0- and 1-branches respectively (Figure 6). As the total cost for this PET is now higher than the previously obtained estimate for starting with \( x_1 \), that PET is now further developed. Both fully developed best PETs are shown in Figure 7. (The algorithm will of course try other possibilities also; we have ignored these for simplicity). Note that the wrong PET was chosen for initial development.

Figure 5. Two of the PETs for (4.1), at stage 2
a) Cost=72. Choices of $x_4$ and $x_1$ are optimal

Figure 6. The two PETs of Fig. 5, after one more iteration. Backtrack to b) will be made at the next step.
Figure 7. The optimal evaluation tree for (4.1), and another one. Note how algorithm will switch back and forth between the two trees during first five levels. Tree on the right will be abandoned after level 5 is reached.

a) Total cost = 92

b) Total cost = 96
REFERENCES
