ON OPTIMAL ALGORITHMS FOR EVALUATING MONOTONIC BOOLEAN FUNCTIONS

by

Y. Breitbart and A. Reiter

Technical Report No. 31
January 1974
REFERENCES


1. Introduction

Boolean functions occur with great frequency in computing. There is hardly a program written without an "if clause" whose arguments must be evaluated. We shall refer to simple predicates of the form "A <relational operator> B" as Boolean variables. The usual practice in writing compilers is to generate code which skips over the evaluation of variables no longer relevant to the value of the expression as a whole (see e.g. [1], section 13.6) but not to attempt any optimization by trying to decide in what order to examine the variables.

Of special interest are the cases where evaluation of a variable is inherently time-consuming. Such is the case in implementing a query language over large databases, where a search is specified by a combination of qualifiers, each involving examining a data value located in secondary storage with an overhead of 50-100 milliseconds per access. Since the expression must be evaluated repeatedly, it is worthwhile expending some effort to analyze the structure of the expression in order to determine an optimal sequence in which to evaluate the variables. This paper is concerned with algorithms for obtaining the best evaluation sequences.

In general, the cost of evaluation differs among the variables, and in fact is not fixed (the "seek" time of a disk arm depends on its current position, that is, on the previous operation performed). Also, often there are a priori probabilities associated with the value of a variable. In the present work however we shall make the simplifying assumptions that the cost of evaluation is the same for all variables and that there are no a priori probabilities. Thus, the "cost" of a particular evaluation sequence is simply the number of variables examined until a conclusion is reached.

We shall also restrict ourselves to considering monotonic Boolean functions (functions in which negation does not occur) which we denote by MBF. Experience indicates that (in query languages at least) most of the functions which occur in practice either
are of this form or are readily so transformable by a change of variables. This assumption allows us to find good evaluation algorithms using simple and practical methods.

Given an MBF \( f(x_1, \ldots, x_n) \), an evaluation algorithm selects an \( x_i \), and substitutes its value into \( f \). If \( f \) is now identically constant, the algorithm terminates, else a new \( x_j \) is selected for the new function. Such algorithms are conveniently visually represented by binary decision trees.

**Example 1.** \( f = x_1x_2 \lor x_1x_3 \lor x_2x_4 \)

A decision tree for evaluating \( f \) (one of several different possible trees) is depicted in Fig. 1. The non-terminal nodes indicate the variable being examined, the branches indicate the value of the variable, and the termination nodes at which the value becomes known are depicted by *. The value of the function is necessarily the same as that of the last branch.

![Decision Tree](image)

In the sequel, when we speak of an evaluation algorithm for an MBF, we shall identify it with its decision tree.

We now introduce some precise definitions.
Let $f(x_1, \ldots, x_n)$ be an MBF, and $\phi$ an algorithm of the above type for evaluating $f$. For any point $\alpha = (\alpha_1, \ldots, \alpha_n)$ in the Boolean $n$-space $B^n$, let $N(f, \phi, \alpha)$ denote the cost of evaluating $f$ at $\alpha$ using $\phi$; in our case, $N$ is the number of nodes in the path of the decision tree $\phi$ corresponding to $\alpha$.

The expected cost of evaluating $f$ using $\phi$ over the entire $n$-space is defined by

$$E(f, \phi) = \sum_{\alpha \in B^n} \frac{N(f, \phi, \alpha)}{2^n}$$

Among all possible evaluation algorithms for $f$, some have lower expected cost than others. Let

$$E(f) = \min_{\phi} E(f, \phi).$$

We will say that an algorithm $\phi$ is optimal if $E(f, \phi) = E(f)$. Note that this optimality is global; for any given point, a better evaluation sequence may exist.

We give below (section 3) a practical algorithm which, for any given MBF $f(x_1, \ldots, x_n)$ constructs a decision tree. It is claimed (but not proved) that this decision tree is always optimal. Proofs of optimality are given for some special cases.

Analogous problems were investigated by Slagle [2] and Reinwald-Soland [3]. The latter give an algorithm of the branch-and-bound type to produce optimal decision trees for a wide class of decision tables. (We note in passing that every MBF can be represented as a decision table of this class.) As will be seen in the next section, their algorithm is not practical in real-life situations, requiring as it does too much computer time and storage.

Slagle [2] gives an algorithm for classes of Boolean functions different from ours, but also without a general proof of optimality.
2. Branch-and-bound Methods


Given a limited entry decision table with \( n \) conditions, \( 2^n \) rules, rule probabilities and time needed to test each condition, Reinwald and Soland obtain a lower bound for the processing time required by a full or a partially developed decision table. In the first-level search for the optimal tree (equivalent to the decision table) a condition (out of the \( n \) conditions) is picked which has the smallest lower bound for the processing time of trees starting with any one of these \( n \) conditions at its root. In the second level of the search, this tree is developed by picking one of the remaining \( (n - 1) \) conditions as the next condition to be tested. As any one of the \( (n - 1) \) conditions may be tested in \( Y \) or \( N \) branches resulting from testing the first condition, the number of possible trees equals \( (n - 1)^2 \) and a lower bound is computed for each of these \( (n - 1)^2 \) possibilities. If none of the lower bounds obtained now is smaller than the \( n \) lower bounds obtained during the first level search, the procedure backtracks to a condition which has a smaller bound and develops the \( (n - 1)^2 \) lower bounds for the new condition picked. The procedure is repeated until one of the \( (n - 1)^2 \) bounds in the second level is smaller than all the bounds in the first level. This tree is further developed by picking one of the remaining \( (n - 2) \) conditions as the next condition to be tested. As each of the two branches of the tree at the second level could have two branches the number of lower bounds to be computed at the second level is \( (n - 2)^4 \). In general, at the \( x \)-th level in the development of the tree at least \( [n - (x - 1)]^{2x-1} \) lower bounds would have to be computed. The procedure ends when the processing time calculated for a fully developed tree is less than or equal to all the lower bounds calculated in the immediately preceding level.

The amount of computation needed to obtain the optimal tree could be enormous.
For instance with \( n = 6 \) the number of bounds to be calculated at the fourth level is 6,561 and at the fifth level 65,636. During the search procedure, repeated backtracking to the previous level in the tree may arise which would further increase the search time for the optimal tree.

Actually, for tables corresponding to MBF, a somewhat simpler branch-and-bound algorithm can be formulated [5]. Nevertheless, the estimates above are still approximately valid. Thus, the usefulness of this technique for functions with more than six variables is limited.

3. Our Algorithm

Let \( f(x_1, \ldots, x_n) \in \text{MBF} \), and let \( K_1 \lor K_2 \lor \ldots \lor K_m \) be the representation of \( f \) in minimal disjunctive normal form \((*)\). Further, let \( f^*(x_1, \ldots, x_n) \) denote the dual function of \( f \) (obtained from \( f \) by interchanging conjunctions and disjunctions; \( f^* = \overline{f}(x_1, \ldots, x_n) \)) and let \( L_1 \lor L_2 \lor \ldots \lor L_p = \text{MDNF}(f^*) \). For each conjunction \( K \), let \( r(K) \) (rank of conjunction \( K \)) denote the number of variables in \( K \). Finally, let \( r(f) = \min_i (r(K_i)) \) denote the rank of the minimal conjunctions of \( f \).

We are interested in variables which appear in the conjunctions of minimal rank both in \( f \) and in \( f^* \). The set \( M \) of such variables is never empty for any \( f \in \text{MBF} \) by Lemma 1. The optimal algorithm consists of picking a variable in \( M \), obtaining its value, substituting it into \( f \) to obtain a new function \( g \) of \( (n-1) \) variables, and repeating the procedure until the function reduces to a constant. Since the procedure never backtrails, it is much faster than the branch-and-bound method.

Lemma 1. For \( f \in \text{MBF} \), every conjunction in \( \text{MDNF}(f) \) has a variable in common with

\( (*)\) As is well-known, this is unique for any MBF. In the sequel, we shall refer to such a representation as \( \text{MDNF}(f) \).
every conjunction in MDNF(f*).

Proof. Let \( x_{i_1} \ldots x_{i_k} \) be any conjunction in MDNF(f). Let \( \alpha \in \mathbb{B}^n \) be any point with 1's in coordinates \( x_{i_1} \ldots x_{i_k} \). Then \( f(\alpha) = 1 \). Suppose that
\[
 x_{j_1} \ldots x_{j_q}
\]
is some conjunction in MDNF(f*). Let \( \beta \in \mathbb{B}^n \) be a point with 0's in positions \( x_{j_1} \ldots x_{j_q} \) and 1's elsewhere. Hence \( f(\beta) = 1 \), or \( f(\beta) = 0 \). If none of the indices \( i,j \) intersect, \( \beta \) has 1's in positions \( x_{i_1} \ldots x_{i_k} \), contradicting the above. Q.E.D.

A description of the algorithm, including the tie-breaking procedure, is now given.

Step 1. Obtain MDNF representations of \( f \) and \( f^* \).

Then \( f = K_1 \lor \ldots \lor K_m \), \( f^* = L_1 \lor \ldots \lor L_p \).

Let \( b_1 < b_2 \ldots < b_{s_1} \) and \( c_1 < c_2 \ldots < c_{s_2} \) denote the ranks of the respective conjunctions. Also let \( B_1 \) and \( C_1 \) represent the respective sets of conjunctions of ranks \( b_1 \) and \( c_1 \).

Step 2. Obtain the variables \( x_{i_1} \ldots x_{i_t} \) which appear both in \( B_1 \) and \( C_1 \).

For each such \( x_i \), let \( w_1(x_i) \) denote the number of occurrences of \( x_i \) in the conjunctions of \( B_1 \) and \( C_1 \). Pick the \( x_i \) which occurs the most times. If there is only one such, proceed to step 3. Otherwise, for all \( x_i \)'s with most occurrences in the sets \( B_1 \) and \( C_1 \), examine \( w_2(x_{i_1}) \) (occurrences in \( B_2 \) and \( C_2 \)). Continue until all ties are broken. If ties remain unresolved when all ranks are exhausted, pick any variable among those which remain.

Step 3. Having chosen \( x_i \), obtain its value and substitute to obtain a new function. Repeat step 1 until a constant is reached.
4. Examples

Example 2. \( f = x_1 x_3 x_4 \lor x_1 x_5 x_6 \lor x_1 x_3 x_5 \lor x_2 x_4 \lor x_2 x_3 x_5 \lor x_2 x_3 x_6 \). 

This example shows that it is important to examine \( f^* \) as well as \( f \). We have 

\[ f^* = x_1 x_2 \lor x_3 x_4 \lor x_5 x_6 \]

Looking at \( f \) only, it is not obvious whether we should start with \( x_2 \) or \( x_4 \). \( f^* \) tells us that \( x_2 \) is preferable. The costs for the best algorithms starting with \( x_2 \) and \( x_4 \) respectively are 186/64 and 192/64. The optimal tree for evaluating \( f \) is given in Fig. 2.

![Fig. 2.](image-url)
Example 3. \( f = x_1(x_3 x_4 v x_5 x_6 v x_7 x_8 x_9) v x_2(x_3 x_4 v x_5 x_6). \)

This example shows that the algorithm must be iterated as long as necessary to break ties. \( x_1 \) and \( x_2 \) appear an equal number of times in \( B_1 \cup C_1 \), but \( x_1 \) appears more times in \( B_2 \cup C_2 \). The respective costs for starting with \( x_1 \) and \( x_2 \) are 2028/512 and 2172/512.

Example 4. \( f = x_1(x_2 v x_3 ... v x_n) v x_2 x_3 v x_4 x_5 ... v x_{n-1} x_n. \)

This example shows that our algorithm by no means gives the only optimal solution. \( x_1 \) appears arbitrarily more often in conjunctions of optimal rank than the other variables, yet the cost of starting with any variable is equal!

5. Optimality proofs for some simple cases

We are unfortunately unable to prove the general optimality of our algorithms. However, we are able to provide proofs for some cases where the rank of the minimal conjunction is 1 or 2.

For a given \( f \in \text{MBF} \), let \( \Phi \) be any algorithm which starts by evaluating \( x_1 \). Denote by \( f(0), f(1) \) the functions obtained by setting \( x_1 = 0, x_1 = 1 \) respectively. It is easy to see that

\[
E(f, \Phi) = 1 + \frac{E(f(0), \Phi) + E(f(1), \Phi)}{2}
\]

5.1

Theorem 1. Let \( f \) be a function with minimal rank of 1; i.e. \( f = x_1 v ... v x_K v g(y_1, ..., y_n) \). Then every optimal algorithm begins with one of \( x_1, ..., x_K \), and

\[
E(f) = 2 - 1/2^{k-1} + \frac{E(g)}{2^k}
\]

5.2

Proof. We might as well assume that the \( x \)'s and \( y \)'s are distinct. By symmetry in the \( x_1 \)'s, also assume that all algorithms evaluate the \( x_1 \)'s in sequence.
Let $\Phi$ be an algorithm which first evaluates $x_1, \ldots, x_k$ and then proceeds optimally on $g$. We have, by 5.1

$$E(f, \Phi) = 1 + \frac{1}{2} \ldots + 1/2^{k-1} + E(g)/2^k.$$ 

We must show that $\Phi$ is optimal and that every optimal algorithm begins with an $x$. We do so by induction on $n$.

$n = 2$. Then $g(y_1, y_2) = y_1 y_2$. Let $\Theta$ be an algorithm which starts with $y_1$ (say). Then, by 5.1

$$E(f, \Theta) = 1 + E(x_1 v \ldots v x_k v y_1 v y_2) = 1 + \frac{E(y_1 y_2)}{2^k} = 1 + \frac{2 - 1/2^{k-1} + 1/2^k}{2^k} = 3 - 3/2^{k+1}$$

while

$$E(f, \Phi) = 2 - 1/2^{k-1} + \frac{E(y_1 y_2)}{2^k} = 2 - 1/2^{k-1} + 1/2^k(1 + 1) = 2 - 1/2^{k+1}.$$ 

Hence $\Phi$ is optimal, while $\Theta$ is not.

Now assume that for all functions $g(y_1, \ldots, y_n)$, $n \leq n-1$, the theorem is true. Let $g = g(y_1, \ldots, y_n)$ and let $\Theta$ be an algorithm which starts with $y_1$ (say) and proceeds optimally. We have

$$E(f, \Theta) = 1 + \frac{E(x_1 v \ldots v x_k v g(y_1 = 0)) + E(x_1 v \ldots v x_k v g(y_1 = 1))}{2},$$

with neither $g(y_1 = 0)$ nor $g(y_1 = 1)$ being identically 1. By induction, we proceed optimally by looking at $x_1$ in each function on the right.

Hence

$$E(f, \Theta) = 2 + \frac{E(x_2 v \ldots v x_k v g(y_1 = 0)) + E(x_2 v \ldots v g(y_1 = 1))}{4}. $$

But if $\Phi$ starts with $x_1$ and proceeds optimally, we have

$$E(f, \Phi) = 1 + \frac{E(x_2 v \ldots v x_k v g)}{2} \leq 1 + \frac{E(x_2 v \ldots v g(y_1 = 0)) + E(x_2 v \ldots v g(y_1 = 1))}{4}.$$

Hence $\Phi$ is better than $\Theta$. Q.E.D.
We now turn our attention to functions with minimal rank $\geq 2$, and prove the following lemma:

**Lemma 2.** Let $f(y_1, \ldots, y_n) \in MBF$, be of minimum rank $\geq 2$. Then $3/4E(f) - E(x_1x_2 \lor f) \leq -1$.

**Proof:** Induction on $n$. For $n=2$, the result follows by straightforward computation using 5.1 and 5.2. Assume lemma true for all $f \in MBF$ of minimum rank $\geq 2$ with no more than $n-1$ variables. Let $f$ now have $n$ variables, and consider an optimal evaluation algorithm $\Phi$ for $x_1x_2 \lor f(y_1, \ldots, y_n)$. If $\Phi$ starts with $x_1(x_2)$, we have by straightforward computation $E(x_1x_2 \lor f) = 3/2 + 3/4E(f)$, thus proving the lemma. Assume now that $\Phi$ starts with $y_1$. We have by 5.1

$$E(x_1x_2 \lor f) = 1 + \frac{1}{4}(E(x_1x_2 \lor f(y_1=0)) + E(x_1x_2 \lor f(y_1=1))).$$

Note that $f(y_1=0)$ may be identically zero, but otherwise must be of minimum rank $\geq 2$. $f(y_1=1)$ may however be of minimal rank $= 1$ so that we have to be careful in applying our induction hypothesis. By the induction hypothesis,

$$E(x_1x_2 \lor f(y_1=0)) \geq 1 + 3/4E(f(y_1=0)).$$

Write $f(y_1=1) = y_2 \lor \ldots \lor y_{k+1} \lor g(y_{k+2}, \ldots, y_n)$, where $g$ is of minimum rank $\geq 2$.

Then

$$E(x_1x_2 \lor f(y_1=1)) = 2 - 1/2^{k-1} + 1/2^k E(x_1x_2 \lor g),$$

and

$$E(f(y_1=1)) = 2 - 1/2^{k-1} + 1/2^k E(g).$$

Applying the induction hypothesis to $x_1x_2 \lor g$ and combining, we get

$$E(x_1x_2 \lor f) \geq 1 + 1/2(1 + 3/4E(f(y_1=0)) + 2 - 1/2^{k-1} + 1/2^k (1 + 3/4E(g))) \geq 1 + 3/4(1 + 3/4E(f(y_1=0)) + 2 - 1/2^{k-1} + 1/2^k (1 + 3/4E(g))) \geq 1 + 3/4E(f).$$

Q.E.D.

This lemma enables us to prove

**Theorem 2.** Let $f = x_1x_2 \lor g(y_1, \ldots, y_n)$, $g$ of minimal rank $> 2$.

Then there are optimal algorithms for $f$ which start with $x_1(x_2)$. 
**Proof:** Induction on \( n \). For \( n=3 \) this is obtained by straightforward computation.

For the induction step, let \( \Phi \) be an algorithm which starts with \( x_1 \) proceeding optimally; assume there is an optimal algorithm which starts with \( y_1 \). Then

\[
E(f) = 1 + \frac{1}{2}[E(x_1x_2v g(\bar{y}_1=0)) + E(x_1x_2v g(\bar{y}_1=1))].
\]

By induction hypothesis,

\[
E(x_1x_2v g(\bar{y}_1=0)) = \frac{3}{2} + \frac{3}{4} E(g(\bar{y}_1=0)).
\]

By lemma 2,

\[
E(x_1x_2v g(\bar{y}_1=1)) \geq 1 + \frac{3}{4} E(g(\bar{y}_1=1)).
\]

Combining we get

\[
E(f) \geq \frac{9}{4} + \frac{3}{8}(E(g(\bar{y}_1=0)) + E(g(\bar{y}_1=1)).
\]

On the other hand,

\[
E(f,\Phi) = 1 + \frac{E(x_2v g) + E(g)}{2} = \frac{3}{2} + \frac{3}{4} E(g) \leq \frac{3}{2} + \frac{3}{4} \left[3/2 + \frac{3}{4} + \frac{3}{8} (E(g(\bar{y}_1=0)) + E(g(\bar{y}_1=1))\right];
\]

i.e. \( E(f,\Phi) \leq E(f) \), or \( \Phi \) is optimal \( \Box \).

**6. Heuristic arguments for optimality**

In lieu of a general proof, we advance two arguments for the optimality of our proposed algorithm.

The first one is based on information theory considerations, as follows.

The function \( f \) divides \( B^n \) into disjoint subspaces on which it takes the values 0 or 1. Each conjunction in the MDNF representation of \( f \) and \( f^* \) further subdivides \( B^n \) into subsets defined by all of the particular coordinates being 1. The maximum information is obtained by interrogating that \( x_i \) which has maximum "differentiation" with \( f = 1 \) for most points where \( x_i = 1 \) and \( f = 0 \) for most points where \( x_i = 0 \). Similar considerations hold for \( f^* \). Conjunctions of minimal
rank correspond to the largest subspaces (fewest fixed coordinates). Hence an \( x_i \) which occurs in a minimal-rank conjunction both in \( f \) and in \( f^* \) gives us the most information.

The second argument is based on the following theorem, whose proof is omitted.

**Theorem 3.** Let \( \Phi \) and \( \Theta \) be two decision trees for \( f \), each having \( t \) terminal nodes. Suppose that \( \Phi \) starts with a conjunction of minimal rank, but \( \Theta \) does not. Then

\[
E(f,\Phi) \leq E(f,\Theta).
\]

It is conjectured that optimal algorithms also have the minimal number of terminal nodes. This, taken with Theorem 3, suggests strongly that the algorithm of Section 3 is indeed minimal.

### 7. Conclusions

The proposed algorithm is easily programmed, and works rapidly even with many variables. Although lacking an optimality proof, the algorithm seems to give optimal results. The validity of the hypothesis is currently being tested experimentally by a computer program, which also compares the cost of constructing the decision tree versus the algorithm proposed in [5].

While it doesn't seem worthwhile to apply these techniques to evaluating simple Boolean expressions of the type encountered in programs, the application to query languages implementation does seem plausible.

**ACKNOWLEDGMENT.** The authors wish to acknowledge their debt to D. Gelperin for many ideas and counterexamples.