Minimizing the Average Query Complexity of Learning Monotone Boolean Functions

Vetle I. Torvik and Evangelos Triantaphyllou / *Department of Industrial and Manufacturing Systems Engineering, Louisiana State University, Baton Rouge, LA 70803-6409, USA; Email: vtorvik@unix1.sncc.lsu.edu, trianta@lsu.edu; Web: http://cda4.imse.lsu.edu*

Last revision: March 10, 2000

Subject classifications: Computational learning.
Other key words: Monotone boolean functions, Membership queries, Average case complexity, Recursion, Partially ordered sets (posets), Isomorphism, Greedy algorithm, Unequal probability sampling, Unbiased estimators.

This paper addresses the problem of completely reconstructing deterministic monotone boolean functions via membership queries. The minimum average query complexity is guaranteed via recursion, where partially ordered sets (posets) make up the overlapping subproblems. The optimality conditions for all the 4,688 posets generated for the problem with up to 5 variables are summarized in the form of an evaluative criterion. The evaluative criterion extends the computational feasibility up to problems involving about 20 variables. A framework for unbiased average case comparison of monotone boolean function inference algorithms is developed using unequal probability sampling. Empirical results show that an implementation of the subroutine considered as a standard in the literature performs more than twice as many queries than the evaluative criterion on the average for 8 variables, and the gap increases with the number of variables. It should also be noted that the first algorithm ever designed for this problem performed consistently within 2% of the evaluative criterion, and consequently prevails, by far, as the most efficient of the many preexisting algorithms.

Situations where a set of binary variables have a non-negative (i.e., monotone) relationship with a phenomenon, are inherently frequent. For example, if a personal computer tends to crash when it runs a particular word processor and web browser simultaneously, it will probably crash if, in addition, it boots up its cd player. In practice, a great deal of effort is put into learning and understanding these monotone relationships. Software is tested, diseases are researched, and so on.

Due to the underlying monotonicity, asking questions can be much more efficient than observing cases as they present themselves.

Monotone boolean functions lay the ground for a simple question-asking strategy, which forms the basis of this paper. More specifically, monotone boolean functions are reconstructed by successive and systematic function evaluations. Initially, the entire set of 2^n boolean vectors, denoted by $\{0,1\}^n$, is considered unclassified. A vector is then selected from the set of unclassified vectors and is submitted to an oracle as a *membership query* (or *function evaluation*). After the vector's function value is provided by the oracle, the set of unclassified vectors is reduced. These queries are then repeated until all of the vectors are classified.

Given the classification of any vector in $\{0,1\}^n$, other vectors may be concurrently classified if the underlying function is assumed to be monotone. Therefore, only a subset of the 2^n vectors need to be evaluated to completely reconstruct the underlying function. Thus, a key problem is to select "promising" vectors so as to reduce the total number of queries (or *query complexity*). Earlier work on monotone boolean function inference (such as Hansel (1966), Sokolov (1982), and Gainanov (1984)) focuses on reducing the query complexity, while more recent work (like Fredman and Khachiyan (1996), Boros et al. (1997), and Makino and Ibaraki (1997)) includes the computational complexity.

The problem of inferring monotone boolean functions via membership queries is equivalent to many other computational problems in a variety of fields (see, for instance, Bioch and Ibaraki (1995), and Eiter and Gottlob (1995)). As a result, a solution to the inference problem, which is efficient in terms of query and computational complexity, can be used to solve other problems efficiently in terms of computational complexity. In practice however, queries often involve some sort of effort, such as consulting with experts or performing experiments, and therefore far surpass computations in terms of cost. Therefore, this paper is focused on minimizing the query complexity as long as it is computationally feasible.

Some background on monotone boolean functions is provided in section 1, before the proposed objective is defined in section 2. An inference approach that is guaranteed to be optimal, is also obtained in section 2, after which the optimality conditions are summarized using an evaluative criterion. Section 3 describes an unequal probability sampling framework for generating monotone

boolean functions and comparing different inference algorithms on the proposed inference objective. This framework is then used in section 4 to provide an extensive unbiased empirical comparison of existing inference algorithms and the evaluative criterion. In the end, section 5 provides some concluding remarks.

1. Some Key Properties of Monotone Boolean Functions

Let $\{0,1\}^n$ denote the set of boolean vectors defined on n boolean variables. A boolean function defined on $\{0,1\}^n$ is simply a mapping to $\{0,1\}$. A vector $v \in \{0,1\}^n$ is said to *precede* another vector $w \in \{0,1\}^n$, denoted by $v \preceq w$, if and only if (iff) $v_i \leq w_i$ for $i = 1, 2, ..., n$, where v_i (w_i) denotes the i-th element of vector v (w). Similarly, a vector $v \in \{0,1\}^n$ is said to *succeed* another vector $w \in \{0,1\}^n$, iff $v_i \geq w_i$ for $i = 1, 2, ..., n$. If a vector v either precedes or succeeds w, they are said to be *related*. A monotone boolean function f is called *non-decreasing*, iff $f(v) \leq f(w) \; \forall \; (v, w)$: $v \preceq w$, and *non-increasing*, iff $f(v) \geq f(w) \; \forall \; (v, w)$: $v \preceq w$. This paper focuses on non-decreasing functions, which are referred to as monotone, as similar results hold for non-increasing functions.

Table I. History of monotone boolean function enumeration for $n = 1,2, ..., 8$.

$Ø(1) = 3$, $Ø(2) = 6$, $Ø(3) = 20$	
$Ø(4) = 168$	by Dedekind (1897)
$Ø(5) = 7{,}581$	by Church (1940)
$Ø(6) = 7{,}828{,}354$	by Ward (1946)
$Ø(7) = 2{,}414{,}682{,}040{,}998$	by Church (1965)
$Ø(8) = 56{,}130{,}437{,}228{,}687{,}557{,}907{,}788$	by Wiedeman (1991)

The number of distinct monotone boolean functions defined on $\{0,1\}^n$ is denoted by $Ø(n)$. All known values of $Ø(n)$ are given in Table I. Wiedeman (1991) employed a Cray-2 processor for 200 hours to compute the value for n equal to 8. This gives a flavor of the complexity of computing the exact number of monotone boolean functions. For larger values of n the best known asymptotic, is due to Korshunov (1977):

$$Ø(n) \sim \begin{cases} 2^{\binom{n}{n/2}} e^{\left(\binom{n}{n/2-1}\left(\frac{1}{2^{n/2}}+\frac{n^2}{2^{n+5}}-\frac{n}{2^{n+4}}\right)\right)}, & \text{for even } n. \\ 2^{\binom{n}{n/2-1/2}+1} e^{\left(\binom{n}{n/2-3/2}\left(\frac{1}{2^{(n+3)/2}}-\frac{n^2}{2^{n+6}}-\frac{n}{2^{n+3}}\right)+\binom{n}{n/2-1/2}\left(\frac{1}{2^{(n+1)/2}}+\frac{n^2}{2^{n+4}}\right)\right)}, & \text{for odd } n. \end{cases} \quad (1)$$

An ordered set of related vectors $v^1 \preceq v^2 \preceq ... \preceq v^p$ is sometimes called a *chain*, while an *antichain* (or *layer*) consists of a set of unrelated vectors. When a set of vectors is partitioned into as few layers as possible, a *layer partition* is formed. For a particular layer partition, the layers can be ordered as $L^1, L^2, ..., L^r$ so that a vector $v^i \in L^i$ cannot succeed another vector $v^j \in L^j$ if $i < j$. The layer partition for the set $\{0,1\}^n$ is unique, while its chain partition is not unique. In fact, the way one partitions $\{0,1\}^n$ into chains can be used effectively in the inference of monotone boolean functions, as in the symmetric chain partition used by Hansel(1966) and Sokolov(1982).

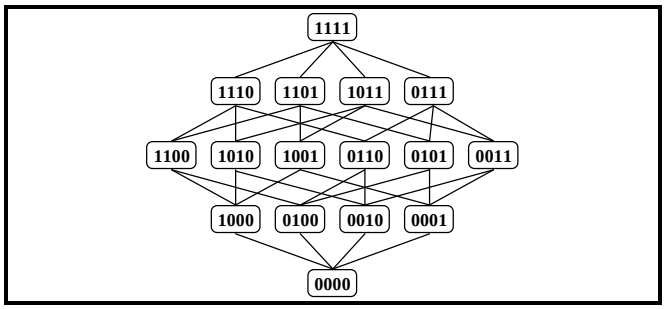

Figure 1. The poset formed by $\{0,1\}^4$ and the $\preceq$ relation.

Figure 1 shows a sample *partially ordered set* (or *poset* for short). In general, posets can be formed by a set of vectors together with the precedence relation $\preceq$. Posets can be viewed as directed graphs where each vertex corresponds to a vector and a directed edge from vertex v to vertex w, represents the precedence relation $v \preceq w$. When drawing a poset, the edge directions are often omitted since the graph is acyclic and so all of the directions can be forced either up or down the page by ordering the vertices by layers. For the purpose of reducing storage and simplifying the visualization of posets, redundant relations, i.e., ones that are transitively implied by other relations, are also omitted, as in Figure 1.

Two posets, P^1 and P^2, are said to be *isomorphic* if there exists a one-to-one mapping of the vectors in P^1 to the vectors in P^2, where the precedence relations are preserved. That is, if $v^1 \rightarrow v^2$ and $w^1 \rightarrow w^2$, where $v^1, w^1 \in P^1$ and $v^2, w^2 \in P^2$, then $v^1 \preceq w^1$ iff $v^2 \preceq w^2$. The *dual* of a poset P is the poset P^d resulting from reversing all of the precedence relations in P. All asymmetric posets have a dual poset that is not isomorphic to itself. A poset P is called *connected* if for any pair of vectors v and w in P, either v and w are directly related to each other, or there exists a sequence of vectors v^1, v^1, ..., v^r in P for which all the pairs (v, v^1), (v^1, v^2), ...,(v^r, w) are related.

A vector v^* is called a *lower unit* of a function f, if $f(v) < f(v^*)\ \forall\ v$: $v \preceq v^*$ and $v \neq v^*$. Similarly, a vector v^* is called an *upper zero* if $f(v) > f(v^*)\ \forall\ v$: $v \succeq v^*$ and $v \neq v^*$. Lower units and upper zeros are also referred to as *border* vectors. For any monotone boolean function f, the set of lower units and the set of upper zeros are unique, and either one of these two sets uniquely identifies f. Let $m(f)$ be the number of border vectors associated with f. It is well known (e.g., Engel (1997)) that $m(f)$ achieves its maximum value for the function(s) that has all its border vectors on two of the most populous layers of $\{0,1\}^n$. That is, the following equation holds:

$$\max_{f \in M_n} m(f) = \binom{n}{\lfloor n/2 \rfloor} + \binom{n}{\lfloor n/2 \rfloor + 1}, \tag{2}$$

where M_n denotes the set of all monotone boolean functions defined on $\{0,1\}^n$. Since the borders of any monotone boolean function f are the only vectors that require evaluations in order to completely reconstruct the function, the value of $m(f)$ works as a lower bound on the number of queries. As a result, the right hand side of equation (2) gives a flavor of the number of queries that may be required.

2. Inference Objective and Solution

An inference algorithm that performs fewer queries than another algorithm when reconstructing a particular monotone boolean function is considered more efficient on that particular function. However, it has not been clear how to compare algorithms on the entire class of monotone boolean functions defined on $\{0,1\}^n$. The existing algorithms (e.g., Hansel (1966), Sokolov (1982), Gainanov (1984)) focus on their upper bounds. Unfortunately, the worst case scenario reflects the algorithm performance on a few select functions and does not reflect what to expect when executing the algorithm on an arbitrary monotone boolean function. The algorithms that implement the subroutine described in Gainanov (1984), and also used in Valiant (1984), Makino and Ibaraki (1995), and Boros et al. (1997), for example, indirectly suggest minimizing the upper bound on the number of evaluations per border vector. As a result, these algorithms greatly favor the simplest functions (which may only have a single border vector), over the complex functions (with up to $\binom{n}{\lfloor n/2 \rfloor} + \binom{n}{\lfloor n/2 \rfloor + 1}$ border vectors).

2.1 The Proposed Inference Objective

With no prior knowledge (other than monotonicity) about the inference application, each function is equally likely to be encountered and should therefore carry the same weight in the objective. Let $\varphi(A, f)$ denote the fixed number of queries performed by algorithm A, in reconstructing the monotone boolean function f. The objective in this paper is to minimize the average number of queries over the entire class of monotone boolean functions defined on $\{0,1\}^n$. This can be expressed mathematically as follows:

$$Q^*(n) = \min_A \frac{\sum_{f \in M_n} \varphi(A, f)}{\varnothing(n)}. \tag{3}$$

Before the solution strategy is developed, a generalized version of problem (3) is described and solved in section 2.2 via recursion. In section 2.3, this recursive solution methodology is then used as a model for solving problem (3) but with an improved efficiency due to overlapping subproblems.

2.2 Minimizing the Average Inference Cost

In some applications the cost of inferring a particular monotone function depends on the vectors evaluated, as well as the order in which they were evaluated. As an example, consider searching for articles (among a set of a articles) containing as many of a specific set of keywords $\{k_1, k_2, ..., k_n\}$ as possible. For the sake of simplicity, assume that searching for p keywords in a single article incurs a cost of p units and that this cost is additive for several article searches. Suppose you find b articles containing keywords k_1 and k_3 (i.e., query (10100...0)). Then searching for articles containing keywords k_1, k_2 and k_3, (i.e., query (11100....0)) among the b articles found from query (10100...0) is easier than searching all of the a articles over again. Thus, the total cost for evaluating vector (10100...0) followed by (11100...0) is $2a + 3b$. If, on the other hand, the query order is reversed, and the query (11100....0) does not result in any articles, then all the a articles have to be searched again for query (10100...0). Now the total cost for evaluating vectors <(11100...0), (10100...0)> is $5a$ (= $2a + 3a$), which is greater than the cost $2a + 3b$ of the queries <(10100...0), (11100...0)>, even though the conclusion is potentially the same, i.e., that f(10100...0) = 0 (article(s) found) and f(11100...0) = 1 (no articles found).

To describe this general cost situation, let C(A, f) be the cost incurred by algorithm A in inferring the function $f \in M_n$. An objective that gives equal weight to each monotone boolean function can be written as:

$$C^*(n) = \min_A \frac{\sum_{f \in M_n} C(A, f)}{\emptyset(n)}. \quad (4)$$

Objective (4) is equivalent to the intuitive notion of minimizing the average inference cost. Furthermore, objective (4), which involves arbitrary costs, generalizes objective (3), which considers the queries themselves as the cost unit.

The tree in Figure 2 illustrates an approach for solving problem (4) for $n = 2$. This approach is especially useful when the query costs depend on the vectors evaluated, as well as the order in which they were evaluated. The tree shows all possible ordered sequences of vector selections. Starting at the root, with the 4 unclassified vectors (00), (01), (10) and (11), and ending at one of the leafs, with the empty set {}, a path denotes a specific ordered sequence of vector selections performed for a particular function. The topmost path, for example, corresponds to the sequence of queries <(11), (10), (01), (00)>, performed in inferring the uniform ONE function. On each path there are two types of nodes; one that corresponds to a set of unclassified vectors, and one that corresponds to a vector choice. For a node corresponding to a set of unclassified vectors, the out branches represent the possible vector choices. For a node (corresponding to a vector choice), the upward and downward branches represent the two possible function values, 0 and 1, respectively. The resulting tree contains 76 leafs, for the simple problem consisting of only 4 vectors.

A specific instance of problem (4) for $n = 2$, associates a cost with each leaf in this tree, say c_1, c_2, ..., c_{76}, where c_i corresponds to the cost of the i-th leaf from the top. To find the optimal choice for each node corresponding to a set of unclassified vectors, the minimum costs can be accumulated by backtracking from the leafs. As an example, consider the set {01,00} obtained after querying vectors (11) and (10) on the topmost path of the tree in Figure 2. If vector (01) is queried next, the costs c_1, c_2, c_3 associated with the three topmost leafs will be used in objective (4). If instead the vector (00) is queried next, the costs c_4, c_5, c_6 associated with the next three leafs, will be used in the place of c_1, c_2, c_3 in the objective. Therefore, vector (01) should be selected if $c_1 + c_2 + c_3 \leq c_4 + c_5 + c_6$, otherwise (00) should be selected. If the sum of the costs are equal, either one of

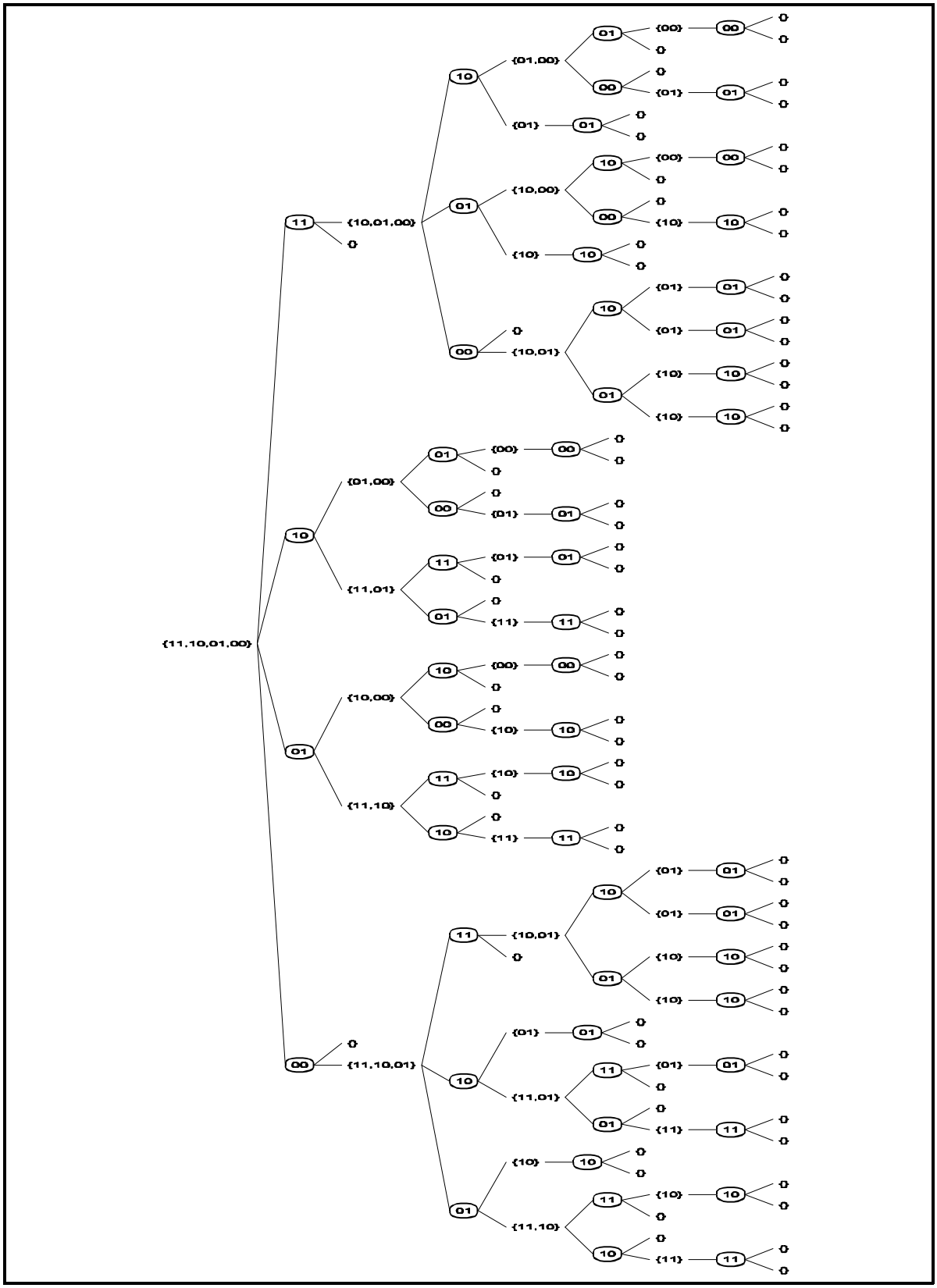

Figure 2. The exhaustive vector selection tree for $\{0,1\}^2$.

the vectors can be selected in order to minimize the total cost. Once the minimum cost is found for this node, it becomes a part of establishing the minimum cost for nodes preceding it on the path. This property is known as an *optimal substructure*, which can be used to compute the minimum cost for the initial set of vectors, in a recursive fashion, as Lemma 1 establishes.

For a node with the unclassified vectors $V = \{v^1, v^2, ..., v^p\}$ corresponding to the prior vector selections $W = <w^1, w^2, ..., w^r>$ and their classifications $F = <f^1, f^2, ..., f^r>$, we define the cost function as follows:

$$C_1(W,F,V) = \begin{cases} c(W,F), & \text{if } |V|=0, \\ \min\limits_{i=1,2,...,p} \left\{ \dfrac{N(V_0^i)C_1(<W,v^i>,<F,0>,V_0^i) + N(V_1^i)C_1(<W,v^i>,<F,1>,V_1^i)}{N(V_0^i) + N(V_1^i)} \right\}, & \text{otherwise.} \end{cases}$$

where

$c(W, F)$ is the fixed cost associated with inferring the monotone function f defined by $f(w^1) = f^1$, $f(w^2) = f^2$, ..., $f(w^r) = f^r$ via the sequence of queries W,

V_z^i is the set of unclassified vectors of V remaining after $f(v^i) = z$ has been established for $i = 1, 2, ..., p$ and $z = 0, 1$, and

$N(V)$ is the number of monotone boolean functions defined on V.

Lemma 1. *The recursive function $C_1(< >,< >,\{0,1\}^n)$ yields the minimum average inference cost $C^*(n)$.*

Proof:

The correctness of Lemma 1 follows by the construction of the function $C_1(W, F, V)$. The individual costs that make up the cost function $C^*(n)$ are recursively accumulated by selecting the set of costs that achieve the smallest average. □

2.3 Minimizing the Average Number of Queries

To guarantee the minimum average inference cost, the entire vector selection tree has to be traversed in order to accumulate the costs via backtracking. However, objective (3) possesses two properties that can be used to reduce this computational burden. The first apparent property is that there is a fixed query cost $c_q(v)$ associated with the evaluation of each vector $v \in V$. That is, the cost of

evaluating a particular vector does not affect the cost of evaluating any of the other vectors. The total cost associated with a sequence of vector evaluations is then simply the sum of the individual costs since it is independent of the order of the vector evaluations. As a result, the cost of evaluating a particular set of vectors is fixed, and can be written as a recursive function as follows in Lemma 2:

For a node with the unclassified vectors $V = \{v^1, v^2, ..., v^p\}$, we define the cost function as follows:

$$C_2(V) = \begin{cases} 0, & \text{if } |V|=0, \\ \min\limits_{i=1,2,\ ...,\ p} \left\{ \dfrac{N(V_0^i)(C_2(V_0^i) + c_q(v^i)) + N(V_1^i)(C_2(V_1^i) + c_q(v^i))}{N(V_0^i) + N(V_1^i)} \right\}, & \text{otherwise.} \end{cases}$$

where $c_q(v^i)$ is the cost incurred from evaluating vector v^i.

Lemma 2. *The recursive function $C_2(\{0,1\}^n)$ yields the minimum average inference cost $C^*(n)$ when the cost of the queries are independent and are given by the vector query costs $c_q(v)$.*

Proof:

Suppose the cost of queries are given by $c_q(v^i)$ for $i = 1, 2, ..., p$. That is, the cost of evaluating a particular sequence of vectors does not depend on the order in which they were queried. That is, the fixed cost function $c(<w^1, w^2, ...,w^p>, F)$ equals $c_q(w^1) + c_q(w^2) + ...+ c_q(w^p)$. As a result, $C_1(V)$ equals $C_2(<>, <>, V)$. From Lemma 1 $C_2(<>, <>, \{0,1\}^n)$ equals $C^*(n)$. □

Lemma 2 provides a way to compute the minimum average query cost in a more efficient manner than Lemma 1. In particular, branches out of nodes corresponding to unclassified vector subsets are required only for unique vector subsets. As an example, consider the node corresponding to the set {10,01} which appears twice in the tree in Figure 2. Suppose its associated parameters (i.e., the minimum average query cost $C_2(\{10,01\})$ and the number of monotone boolean functions $N(\{10,01\})$) are stored once they are computed at one of the two nodes. Then the other node can be bound, avoiding repetitive branching. A further bounding improvement based on the property of independent query costs is given next in Lemma 3.

Lemma 3. *Suppose that a set V of vectors consists of a set of mutually unrelated subsets {V^1, V*

^{2}, ..., V^{p}}. If the query costs are independent, then the following equations hold:

$$N(V) = N(V^1)N(V^2) \ldots N(V^p)$$

and

$$C_2(V) = C_2(V^1) + C_2(V^2) + \ldots + C_2(V^p).$$

Proof:

For any monotone boolean function f defined on V, fixing the value of $f(v^i)$ for $v^i \in V^i$ does not restrict the value of $f(w)$ for $w \in W = \{V^1, V^2, \ldots, V^{i-1}, V^{i+1}, \ldots, V^p\}$. That is, for each distinct monotone boolean function defined on V^i, there are $N(W)$ distinct monotone boolean functions defined on V. Therefore, the number of monotone boolean functions defined on V is $N(W)N(V^j)$. By further reducing W in a similar manner for the other unrelated subsets, this expression can be reduced to $N(V^1)N(V^2) \ldots N(V^p)$.

Suppose that optimal vectors are selected from subset V^i until they are all classified, incurring a minimum cost of $C_2(V^i)$. Since vectors from V^i cannot concurrently classify any of the vectors belonging to $W = \{V^1, V^2, \ldots, V^{i-1}, V^{i+1}, \ldots, V^p\}$, all of the vectors in W are still unclassified. Therefore, the total average cost for V is accumulated by $C_2(V^i) + C_2(W)$. The average cost of the set of vectors W can be further reduced to $C_2(V^1) + C_2(V^2) + \ldots + C_2(V^p)$. □

As a result of Lemma 3, the computations can be distributed in a parallel fashion to unrelated vector subsets since the parameters of any vector set can be computed independently from its connected subsets.

A second apparent property of problem (3) is that the vector query costs are equal for all vectors. That is, $c(v) = c \; \forall \; v \in V$. This fact leads to Lemma 4, which provides even more general bounding rules.

Lemma 4. *Suppose an isomorphic poset mapping, $(V^1, \preceq) \rightarrow (V^2, \preceq)$, is given where v^1 is mapped to v^2. If v^i is selected from set V^i (for $i = 1, 2$), let the two possible remaining vector subsets be denoted by V_z^i, for $z = 0$ and 1. If the query costs are equal and independent for all vectors, then the following equations hold:*

$$N(V_0^1) = N(V_0^2), N(V_1^1) = N(V_1^2),$$

and

$$C_2(V_0^1) = C_2(V_0^2) \text{ and } C_2(V_1^1) = C_2(V_1^2).$$

Proof:

This lemma can be proved by induction. As a basis of the induction consider the two isomorphic posets: a single vector and the empty set. The parameters $N(v) = 2$ and $C_2(v) = c \; \forall \; v \in V^1$ (or V^2) are obviously fixed, since c is a constant. For the purpose of completeness also define $N(\{\}) = 1$ and $C_2(\{\}) = 0$. Now suppose two sets of vectors V^1 and V^2 are given for which the isomorphic mapping $(V^1, \preceq) \rightarrow (V^2, \preceq)$ maps v^1 to v^2. If v^1 is selected from V^1, then it results in a pair of posets that are isomorphic to the pair of posets that result from selecting v^2 from V^2. Thus, if the equations hold true for the resulting posets, then they also hold true for the original posets $(V^1, \preceq)$ and $(V^2, \preceq)$. This fact shows the recursive step, and completes the proof. □

Lemma 4 implies that a recursive look up procedure can focus on non-isomorphic posets instead of vector subsets, as implied by Lemma 2. That is, the vector subsets used for Lemma 2 are now formed into posets using the precedence relations. Since the mapping of vector subsets to non-isomorphic posets is many-to-one, the storage requirement of the algorithm is reduced. Once the N and C_2 values are computed for a particular poset, they can be stored. Later, when an isomorphic poset is encountered at another node in the tree, then these values can be looked up.

Corollary 5. *Suppose the poset $P = (V, \preceq)$ and its dual poset $P^d = (V^d, \preceq)$ are given. If the query costs are equal and independent, then the following equations hold:*

$$N(V) = N(V^d) \text{ and } C_2(V) = C_2(V^d).$$

Proof:

The proof is the same as that of Lemma 4 where V and V^d replace the roles of vectors V^1 and V^2, respectively. □

Lemmas 2 through 4 and Corollary 5 form the criteria for a bounding procedure under the assumptions of independent and equal queries. In particular, only nodes corresponding to non-isomorphic, non-dual, connected posets, are branched upon. Figure 3 shows the resulting reduced search tree when these bounding criteria are applied to the tree shown in Figure 2. As an example of bounding, consider the connected poset $(\{10,01, 00\}, \preceq)$. This poset is isomorphic to the dual of

($\{11, 10, 01\}, \preceq$) and therefore only one of them has to be branched upon, as seen in Figure 3. A size comparison of the trees in Figures 2 and 3 indicates the potential effectiveness of these bounding criteria.

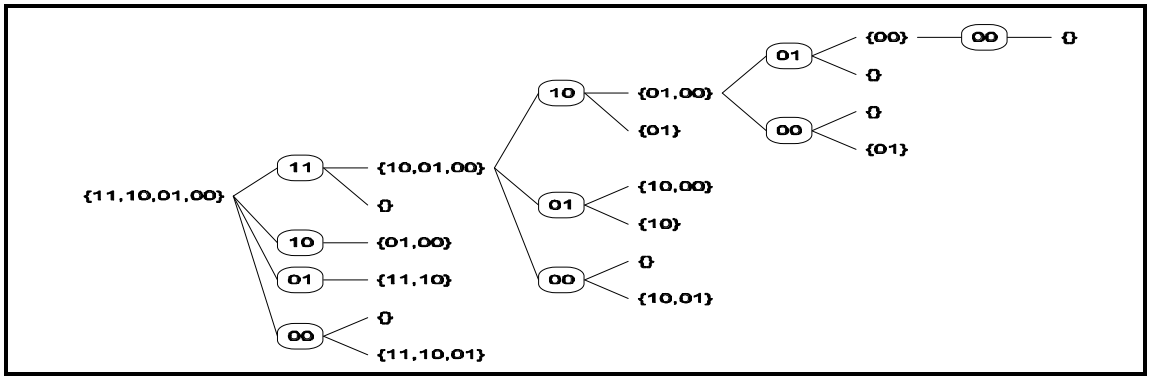

Figure 3. The vector selection tree for $\{0,1\}^2$ restricted to connected, non-dual, non-isomorphic posets.

To simplify the implementation, the algorithm was divided into two steps as shown in Figure 4. Since the cost unit is queries, $c(v) = 1 \; \forall \; v \in V$, the inference cost function C_2 is here denoted by Q. In the first step, all the non-isomorphic, non-dual, connected posets that can be encountered during the inference process are generated and stored in library L. This is realized in lines 1 and 2 of MINIMIZE-AVE-Q in Figure 4. In particular, the edge vertices are recursively removed from the connected, non-isomorphic, non-dual posets, starting from $\{0,1\}^n$ and stopping when {} is encountered. Here, an *edge vertex* of a poset, denotes a vertex that has no related vertices preceding (succeeding) it. Lemma 6 verifies that the subroutine CREATE-POSET-LIBRARY actually creates all possible non-isomorphic, non-dual, connected posets encountered in the inference process.

In the second step, shown in lines 3 through 22 of MINIMIZE-AVE-Q, the N and Q parameters and the set of associated optimal vectors W are computed for all of the posets stored in library L. Since the parameters of a poset of size k (i.e., one with k vectors) can be written as a

```
MINIMIZE-AVE-Q(n)
1  P ← ({0,1}^n, ≤)
2  CREATE-POSET-LIBRARY(P)
3  Q({}) ← 0,
4  N({}) ← 1
5  for i ← 1 to 2^n
6      for each P ∈ L[i]
7          for each v ∈ P
8              Q* ← 2^n,
9              W* ← NIL
9              for each f ∈ {0,1}
10                 P_f ← REMOVE-CONE(P, v, f)
11                 N_f ← 1,
12                 Q_f ← 0
13                 for each P_unr ∈ MAX-UNRELATED-POSETS(P_f)
14                     N_f ← N_f × N(P_unr)
15                     Q_f ← Q_f + Q(P_unr)
16             Q_v ← (N_1 × Q_1 + N_0 × Q_0)/(N_1 + N_0) + 1
17             if Q_v ≤ Q*
18                Q* ← Q_v
19                V* ← V* ∪ {v}
20         Q(P) ← Q*
21         W(P) ← W*
22         N(P) ← N_1 + N_0

CREATE-POSET-LIBRARY(P)
1  for i ← 0 to |P|-1
2      L[i] ← NIL;
3  L[|P|] ← P
4  GENERATE(P)

GENERATE(P)
1  if |P| > 0,
2     P ← P - {e} ; e = edge element
3     if CONNECTED(P),
4        if NONE-ISOMORPHIC(L[|P|],P)
5           L[|P|] ← L[|P|] ∪ {P}
```

Figure 4. An algorithm for minimizing the average number of queries .

combination of the parameters of posets of sizes strictly less than k, the posets are processed in increasing size, starting with the empty poset which has the parameters $N = 1$ and $Q = 0$.

Each time $Q(P)$, $N(P)$ or $V(P)$ is called in the algorithm MINIMIZE-AVE-Q, a search is performed for the poset in library L that is isomorphic to P or its dual. For the implementation of this algorithm, the size k (the simplest isomorphism invariant) and an index number i, is used to uniquely identify posets in L. Therefore, a specific poset can be denoted by $L_k(i)$, and the updates of $Q(P)$, $N(P)$ or $V(P)$ in lines 20 through 22 of the algorithm MINIMIZE-AVE-Q can be performed in one sweep by accessing $Q_k(i)$, $N_k(i)$ and $V_k(i)$.

There are four subroutines in Figure 4 whose details have been left out for the purpose of brevity. The subroutine CONNECTED(P) should output TRUE if the poset P is connected and FALSE otherwise. The subroutine NONE-ISOMORPHIC(L, P) should output FALSE if library L contains a poset that is isomorphic to P, and TRUE otherwise. The subroutine REMOVE-CONE(P,

v, f) returns the poset formed by removing the vector v and the vertices that precede v if $f = 0$ (the vertices that succeed v if $f = 1$) from P. The subroutine MAX-UNRELATED-POSETS(P) returns the largest partition of P into a set of posets so that they are mutually unrelated.

Lemma 6. *By recursively removing the edge vertices and only storing non-isomorphic, non-dual, connected posets, all the posets observable in the inference process are generated.*

Proof:

Suppose a non-edge vertex v is selected from a poset $P = (V, \preceq)$, and this query potentially reduces the vertices in V to V_0 (if $f(v) = 0$), and V_1 (if $f(v) = 1$). Then, there exists a pair of edge vertices v_0, $v_1 \in V$, for which $v_0 \preceq v \preceq v_1$. That is, $V - v_1$ contains V_1 and $V - v_0$ contains V_0. Therefore, none of the subsets obtained from fixing the function value for non-edge vertices are removed from consideration. □

Theorem 7. *The algorithm MINIMIZE-AVE-Q(n) computes the minimum average number of queries $Q^*(n)$.*

Proof:

The correctness of *MINIMIZE-AVE-Q(n)* follows from Lemmas 2, 3, 4, 6 and Corollary 5 as follows. Lemma 6 proves that all the needed connected, non-dual, non-isomorphic posets are generated in lines 1 and 2 of MINIMIZE-AVE-Q. Lemmas 2, 3, 4 and Corollary 5 prove that these posets are sufficient to compute the minimum average number of queries. MINIMIZE-AVE-Q is consistent with the recursive function of Lemma 2, while taking advantage of Lemmas 3 and 4, and Corollary 5 in computing the minimum average number of queries. □

Figure 5 shows a part of the tree that is traversed by the algorithm MINIMIZE-AVE-Q(n) when it is executed for n equal to 3. The root of the tree $L_8(1)$ denotes $\{0,1\}^3$, and in general $L_k(i)$ denotes the i-th poset generated of size k. To reduce the size of the tree, only one branch is shown for vectors with the same pair of isomorphic posets, without the loss of an optimal solution. For example, when branching on $L_8(1)$, the vector (000) results in the same pair of connected, non-dual, non-isomorphic posets as (111). As a result, they carry the same (N_0, N_1, Q_0, Q_1) values, and (000) is optimal if and only if (111) is, and by branching on only one of them, the optimal vector is not

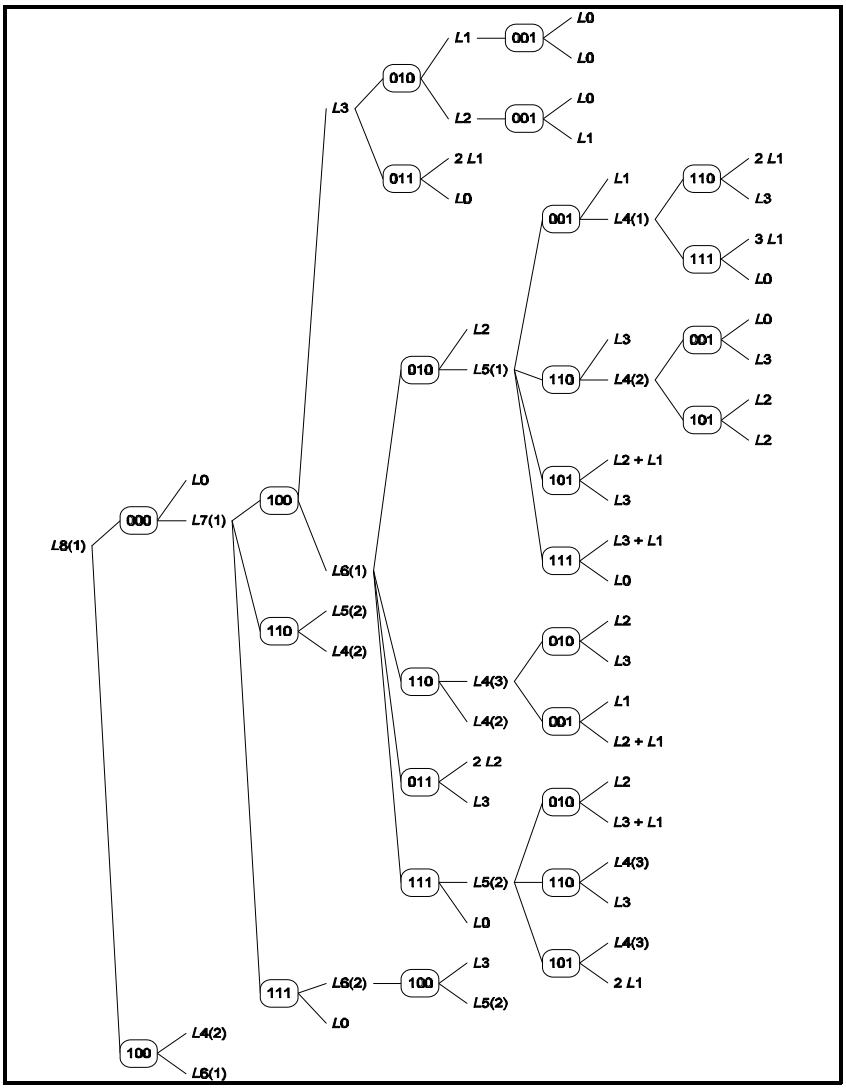

Figure 5. Part of the tree traversed by MINIMIZE-AVE-Q(3).

removed from consideration. The same relation holds between the vectors (100), (010), (001), (110), (101) and (011). Therefore, only one vector from each group, say (000) and (100), needs to be branched on, as seen in Figure 5.

At a first glance, it could seem as though this bounding criterion could further improve upon the algorithm MINIMIZE-AVE-Q. However, determining whether two vectors result in the same pair of isomorphic posets, is computationally equivalent to finding their parameters, which is exactly what MINIMIZE-AVE-Q does in the first place. Therefore, this bounding criterion is only used to

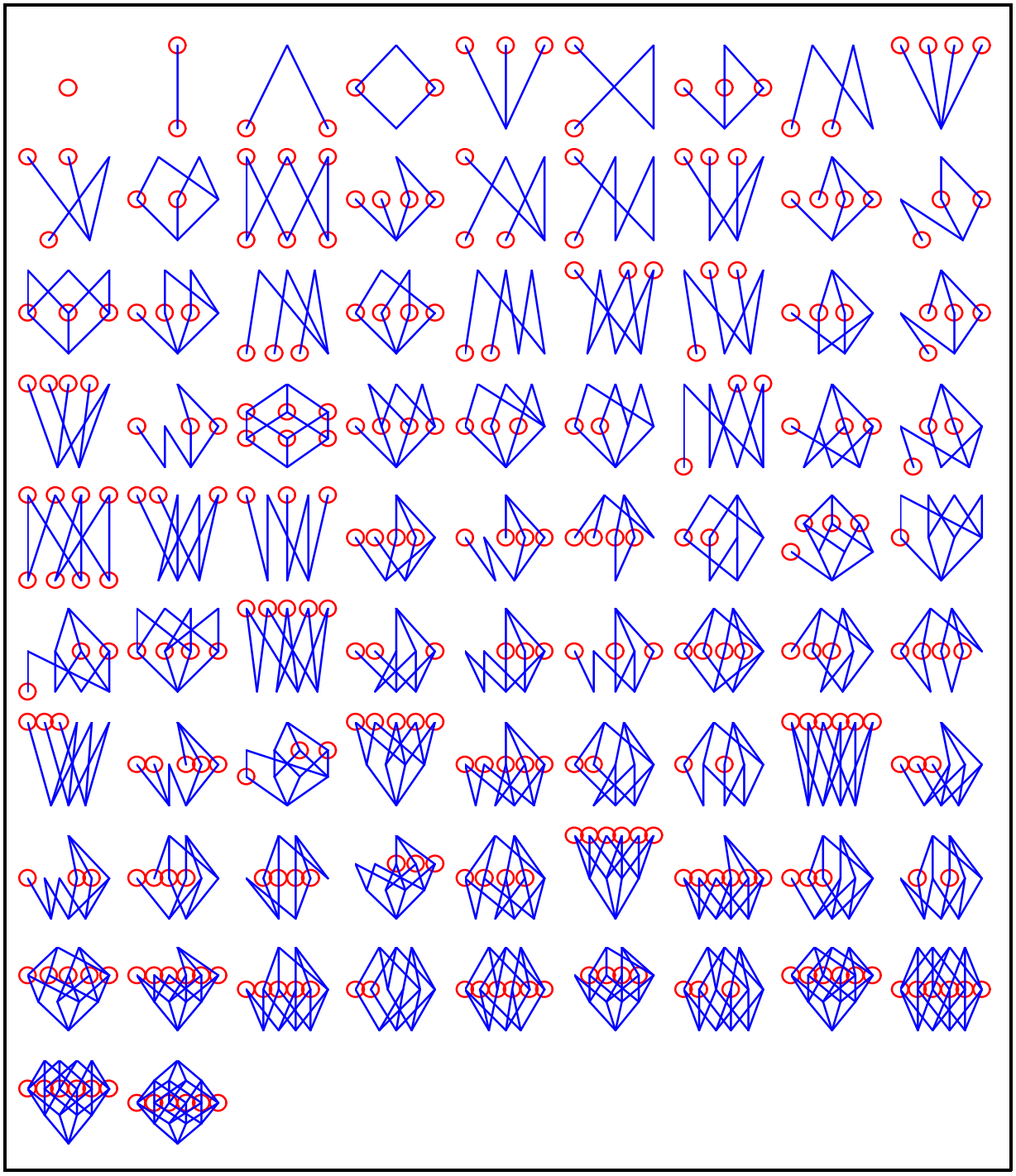

Figure 6. All connected, non-dual, non-isomorphic posets generated for $n = 4$, with optimal vertices circled.

simplify visualization of the search tree. Also, the analysis in section 2.4 requires knowing all the optimal solutions. For that purpose, finding a single optimal solution is not sufficient.

The algorithm MINIMIZE-AVE-Q(n) generates all possible non-isomorphic, non-dual,

connected subposets of $\{0,1\}^n$ and finds the corresponding optimal vertices. Figure 6 shows these posets and their corresponding optimal vertices for *n* equal to 4. For some of the posets, the optimal choices seem to be intuitive. However, for others the optimal choices seem to be obscure. Section 2.4 addresses the issue of summarizing the optimality conditions for the posets in Figure 6 and for some posets observed when *n* is greater than 4.

2.4 Summarizing Poset Optimality Conditions via an Evaluative Criterion

The total number of connected, non-dual, non-isomorphic posets generated by MINIMIZE-AVE-Q(*n*) for *n* equal to 1, 2, ..., 5 is given in Table II. There are at least half as many posets as the number of monotone boolean functions for *n* equal to 1, 2, ..., 5 (given in Table I). The number of posets for *n* equal to 6 is therefore probably of a magnitude greater than 10^6, making it close to intractable to store all of them.

Table II. The number of posets generated by MINIMIZE-AVE-Q(*n*).

Poset Size	*n* = 1	*n* = 2	*n* = 3	*n* = 4	*n* = 5		Poset Size	*n* = 5
0	1	1	1	1	1		17	476
1	1	1	1	1	1		18	373
2	1	1	1	1	1		19	262
3		1	1	1	1		20	187
4		1	2	3	3		21	120
5			2	4	4		22	82
6			2	8	8		23	48
7			1	11	18		24	33
8			1	14	40		25	18
9				13	71		26	12
10				10	130		27	6
11				6	217		28	5
12				5	338		29	2
13				2	462		30	2
14				2	577		31	1
15				1	609		32	1
16				1	576			
Total	3	5	12	84				4,688

When solving problem (3) for *n* greater than 6, the optimal approach presented in section 2.3

is currently computationally infeasible. However, the optimality conditions for the posets generated for n up to and including 5 can be summarized in a simpler form than storing the posets in their entirety. The goal of this section is to establish a simple summary of the poset optimality conditions in the form of a vector evaluative criterion. Since an inference algorithm that tackles larger posets will eventually decompose into smaller posets for which optimal vectors are known, this evaluative criterion will hopefully be close to the optimal for larger problems.

In solving problem (3) optimally for n equal to 5, many different and complex posets were encountered. The optimal vectors of these posets seemed to display two general properties. First, the optimal vectors seem to be in the *vertical middle*. More specifically, all posets in Figure 6 have at least one optimal vector in the most populous layer. However, this observation alone is not enough to pinpoint an optimal vector. The second property observed is that the optimal vectors seem also to be *horizontal end points*. In particular, vectors related to only one other vector in the posets in Figure 6 are always optimal. For posets that do not contain such a vector, vectors with few related vectors tend to be optimal more often than others within a layer. To study the idea of vertical middle and horizontal end point, consider two specific posets, namely a *chain* and a *sawtooth*, as shown in Figure 7. Examples of these posets can also be seen in Figure 6. The first row from the top has two chain posets (1st and 2nd columns from left) and five sawtooth posets (1st, 2nd, 3rd, 6th and 8th column from the left).

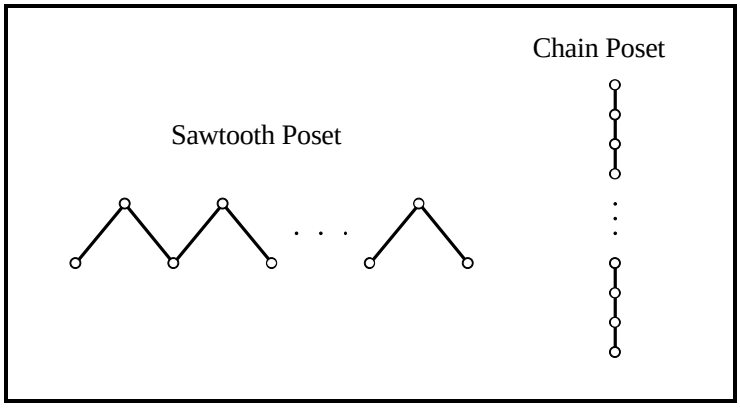

Figure 7. Illustration of the sawtooth and chain posets.

A chain is an arbitrarily tall poset with minimum width. For a chain of height h, the number of monotone boolean functions $N(h)$ equals $h+1$, and the minimum average number of questions, given by

$$Q(h) = \frac{Q(\lfloor h/2 \rfloor)(\lfloor h/2 \rfloor+1) + Q(\lfloor h/2-1/2 \rfloor)(\lfloor h/2-1/2 \rfloor+1)}{h+1}+1 \begin{cases} \geq \log_2(h+1) \\ \leq \lfloor \log_2(h) \rfloor + 1, \end{cases}$$

is obtained by repeatedly selecting a middle vector. Here, $Q(1) = 1$ and $Q(0) = 0$.

A sawtooth is an arbitrarily wide poset with minimum height. Consider a sawtooth poset with width w. The number of possible monotone boolean functions $N(w)$ equals $N(w\text{-}1) + N(w\text{-}2)$ and the minimum average number of questions, given by

$$Q(w) = \frac{Q(w-1)(N(w-1) + Q(w-2)N(w-2)}{N(w-1) + N(w-2)}+1 \approx .7236068w + .2291796,$$

is obtained by invariably selecting one of the edge vectors. Here, $Q(1) = 1$, $Q(0) = 0$, $N(0) = 1$ and $N(1) = 2$. The surprising result about the sawtooth poset is that consistently selecting the vertex next to the edge vectors maximizes the average number of questions. Figure 8 shows the average number of questions divided by w for the sawtooth poset of width w, when consistently selecting the edge vertex (the lower curve), the middle vector (the oscillating curve), and the vertex next to the edge vertex (the upper curve).

Now consider creating an evaluative criterion based on the ideas of vertical middle and horizontal end points. Suppose a subset of unclassified vectors, $V = \{v^1, v^2, ..., v^p\}$ is given. Let $K_1(v^i)$ and $K_0(v^i)$ be the number of vectors that are concurrently classified when $f(v^i) = 1$ and $f(v^i) = 0$, respectively. Invariably selecting one of the vectors that have the minimum $|K_1 - K_0|$ value guarantees the minimum average number of questions for arbitrary sized sawtooth and chain posets, as well as for the inference problems with n strictly less than 5. This can be verified by considering the subset of posets encountered by the criterion $min|K_1 - K_0|$ in Figure 6. Unfortunately, this criterion is not optimal for all the posets generated for n equal to 4. It is only optimal for the posets encountered when using the criterion $min|K_1 - K_0|$. Another drawback is that it is not optimal for the inference problem when n is equal to 5.

It seems reasonable to attempt to classify as many vectors as possible for each question, and the two criteria max $(K_1(v) + K_0(v))$ and max $K_1(v)K_0(v)$ are consistent with this philosophy (see

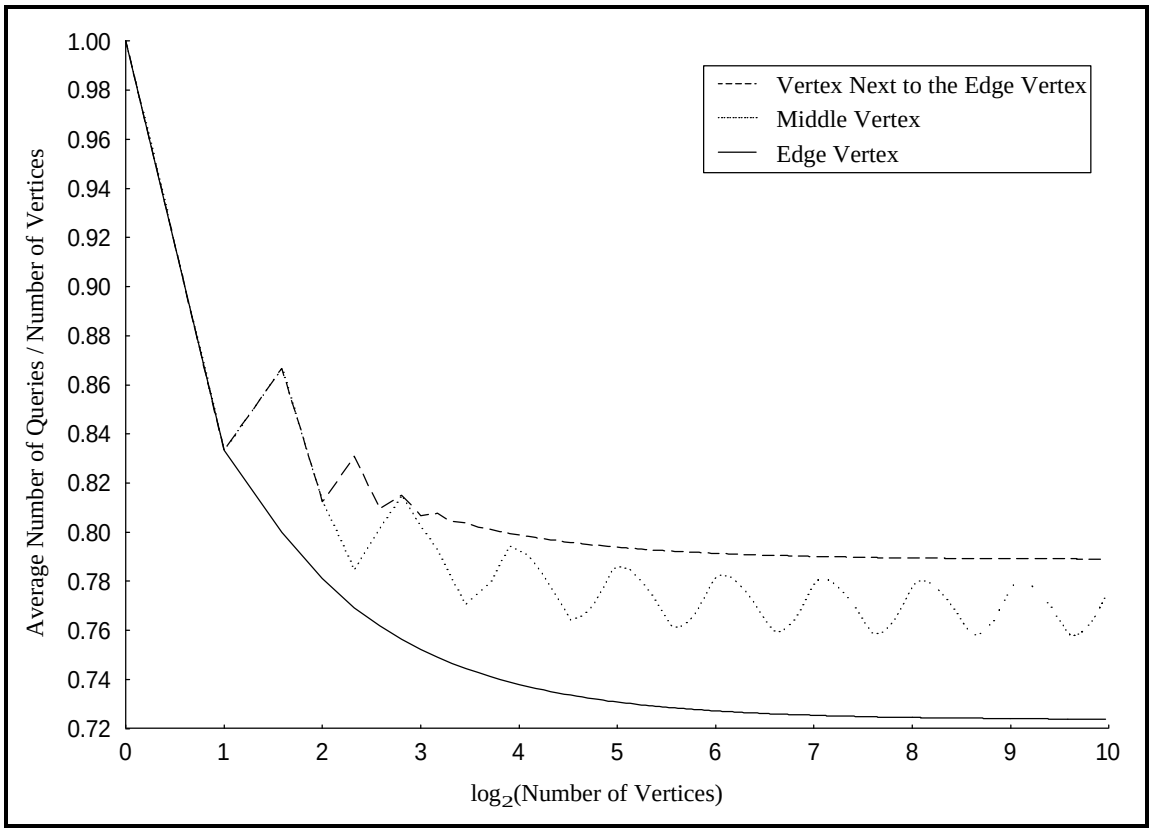

Figure 8. Query complexity on the sawtooth poset.

Judson (1999)). However, they are extremely counterproductive and should be avoided. One of the reasons for this is that vectors that are able to concurrently classify more vectors, are also more likely to be classified by others. As an example consider the set of vectors $\{0,1\}^4$. All the vectors on the middle layer are optimal, as can be observed in Figure 6. However, the criterion max $(K_1(v) + K_0(v))$ selects either the (0000) or the (1111) vector, which happens to maximize the average number of queries, while $K_1(v)K_0(v)$ ties the entire set of vectors. This shows how intuition can lead to poor and ambiguous choices.

Unfortunately, the two values K_1 and K_0 alone are not sufficient to construct a criterion that is optimal for all the posets encountered for the inference problem when n is equal to 5. To find a better selection criterion the vector properties for a particular poset need to be summarized using more than just their K_1 and K_0 values, but hopefully less than the entire poset.

Throughout this paper, it was assumed that the function should be completely reconstructed. If, however, a limited number of queries are allotted, then the previous objective changes and

consequently the selection criterion should be modified. In this situation, the average number of vectors remaining can be considered as a function of the number of queries performed. Figure 9 shows two such functions for n equal to 5; one corresponding to a greedy algorithm, the other corresponding to an algorithm that sacrifices earlier to perform better in the end.

The greedy approach to this problem (the solid line in Figure 9) maximizes the instantaneous reduction in the average number of remaining vectors between the vector selections. The other algorithm (the dotted line in Figure 9) selects the vector which is the most frequent border vector among the remaining monotone boolean functions, and draws its motivation from the fact that each border vector eventually has to be evaluated. Rather than relying on immediate gratification, this selection criterion tends to sacrifice early in the evaluation process for the benefit of requiring fewer evaluations overall. In Figure 9 the greedy approach corresponds to the graph that achieves the steepest instantaneous descent. After 6 questions, however, this approach loses its momentum and from there on, the other algorithm achieves a greater instantaneous descent. By the time they are both finished, the greedy algorithm comes in second.

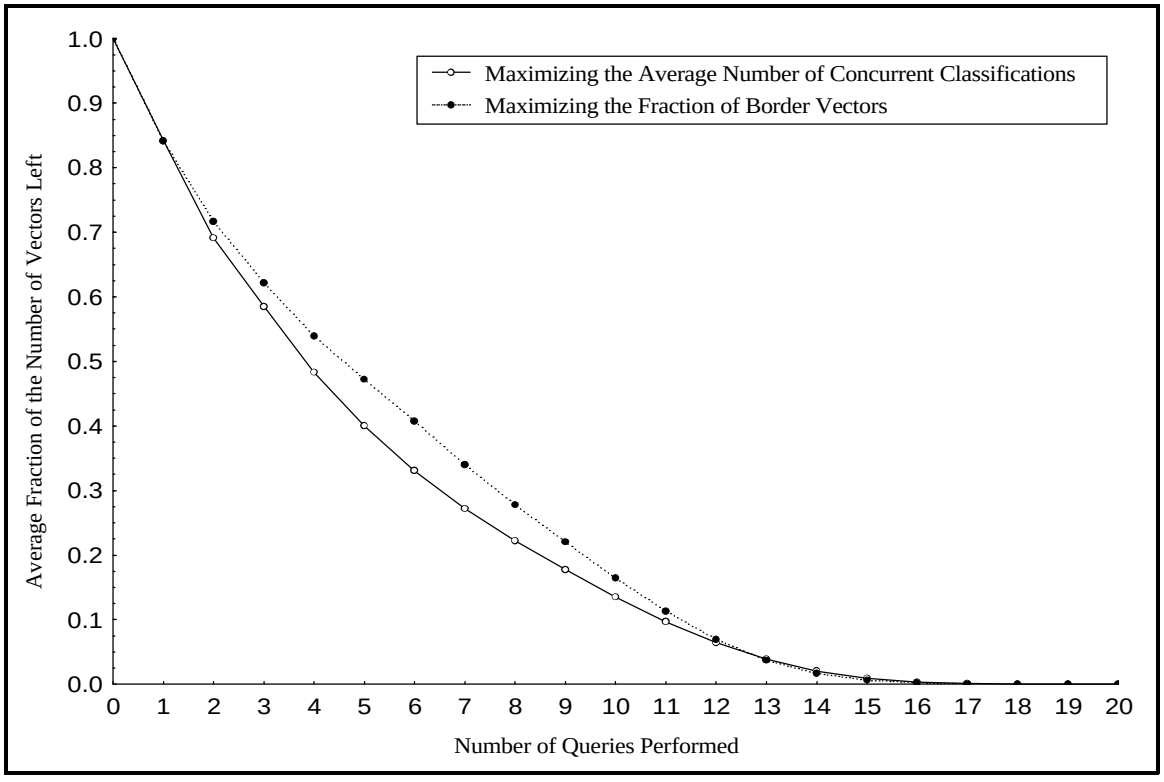

Figure 9. Classification efficiency of two evaluative criteria over all functions in M_5.

The algorithm based on maximizing the fraction border vectors, queries on average 13.6 vectors for complete reconstruction, while the algorithm based on maximizing the average number of concurrent classifications queries 13.9 vectors on the average. Furthermore, the evaluative criterion $min|K_1\text{-}K_0|$ queries 13.7 vectors on the average. However, the criterion $min|K_1\text{-}K_0|$ only requires the subset of vectors in order to be computed. The two other criteria are computationally burdened by the fact that they need to generate and store all the monotone boolean functions. The problem of generating and storing all monotone boolean functions defined on at most n variables is much harder than just enumerating them, which has only been done for n up to and including 8 (see Table I). Since there are about 10^7 monotone boolean functions for n equal to 6, and about 10^{12} for n equal to 7, these two heuristics seem feasible for at most 6 variables with current storage capacities.

3. A Framework for an Unbiased Empirical Comparison of Algorithms

3.1 An Unbiased Estimator for the Average Case Complexity

Often a random sample of test cases is generated to estimate parameters such as the expected value of the algorithm complexity over the entire population of test cases. For an estimator to be *unbiased*, its expected value has to be equal to the actual parameter value it is estimating. If some test cases are more likely to end up in the sample, then the average complexity of the sample puts too much weight on the more likely test cases, and too little weight on the less likely test cases. If the probability of each test case being included in the sample, i.e., their *inclusion probabilities*, are known, then an unbiased estimator can be obtained. The estimators that are presented next can be found in most textbooks (e.g., Thompson (1992)) involving unequal probability sampling.

Consider the finite universe of fixed quantities $\{\varphi_1, \varphi_2, ..., \varphi_{Ø(n)}\} = \{\varphi(A, f): f \in M_n\}$ associated with a particular monotone boolean function inference algorithm A. Let p_i denote the probability of including the i-th element in the sample. Horvitz and Thompson (1952) introduced an unbiased estimator for the total number of queries in the universe as:

$$\hat{\delta} = \sum_{i=1}^{\acute{\imath}} \frac{\varphi_i}{p_i},$$

where $\{\varphi_1, \varphi_2, ..., \varphi_{\acute{\imath}}\}$ are the quantities corresponding to the set of $\acute{\imath}$ unique monotone boolean functions in the sample. Notice that if the sample was taken with replacement, some monotone

boolean functions might be selected more than once, while their corresponding quantity is used but once in the estimate.

An unbiased estimator for the variance of $\hat{\delta}$ requires the joint pairwise inclusion probabilities. Let p_{ij} be the probability that elements *i* and *j* are included together in the sample. Then, an unbiased estimator is given by:

$$\hat{Var}(\hat{\delta}) = \sum_{i=1}^{\nu} \frac{1 - p_i}{p_i^2}\varphi_i^2 + \sum_{i=1}^{\nu}\sum_{j\neq i} \left(\frac{p_{ij} - p_i p_j}{p_i p_j}\right)\frac{\varphi_i\varphi_j}{p_{ij}}$$

When $\Phi(n)$ is known, an unbiased estimate of the universe mean is:

$$\hat{\mu} = \frac{\hat{\delta}}{\Phi(n)}, \tag{5}$$

and a corresponding unbiased estimate of its variance is:

$$\hat{Var}(\hat{\mu}) = \frac{\hat{Var}(\hat{\delta})}{\Phi(n)^2} .$$

At a first glance, the only benefit of using the inclusion probabilities, are unbiased estimators. If the sampling technique allows for adjusting the inclusion probabilities, however, it may be possible to adjust them so as to reduce the variance of the estimators. This issue is further discussed in section 3.2.

3.2 Generating Random Samples of Monotone Boolean Functions

The problem of generating all monotone boolean functions is much harder than just enumerating them, and so an exhaustive analysis becomes intractable when *n* is greater than 6. As a remedy, a sample of functions can be generated. However, a sample that is not drawn randomly is subject to conditional results. Also, if one generates monotone boolean functions that contain a certain number of border vectors (e.g., Makino et al. (1999)), for example, then one simply cannot make claims towards the general class of monotone boolean functions.

It is practically impossible to generate monotone functions from a uniform distribution. In fact, it is easy to fall in the trap of generating functions from a distribution that deviates significantly from the uniform distribution. As an illustrative example, consider the algorithm GENERATE-MBF

```
GENERATE-MBF(n)
1  U = {0,1}^n
2  while U ≠ {}
3        v ← SELECT-AT-RANDOM(U, 1/|U|)
4        f(v) ← SELECT-AT-RANDOM({0,1},1/2)
5        if f(v) = 0
6           W ← {w: w ∈ U, w ≤ v}
7           f(w) ← 0, ∀ w ∈ W
8           U ← U - W
9        else
10          W ← {w: w ∈ U, w ≥ v}
11          f(w) ← 0, ∀ w ∈ W
12          U ← U - W
13 output f
```

Figure 10. Randomly generating monotone boolean functions without inclusion probabilities.

as outlined in Figure 10. It starts with all the vectors $\{0,1\}^n$ as unclassified. Then it randomly selects a vector from the unclassified vectors so that each is selected with equal probability. In the last step, it classifies the selected vector as 0 or 1 with (equal) probability of ½ and classifies all of the vectors, that are covered according to the monotonicity property, to their appropriate values. This process is repeated until all of the vectors are classified.

The algorithm GENERATE-MBF will indeed randomly generate monotone boolean functions, but the functions do not have equal probabilities of being included in the sample. In fact, these inclusion probabilities vary significantly. For example, the probability that the zero vector is classified as 1, which only corresponds to the uniform ONE function, is greater than $2^{-(n+1)}$, the probability of selecting the zero vector during the first iteration and classifying it as 1. That is, the inclusion probability for the ONE function is much greater than what it would be in the uniform sampling as the following inequality indicates:

$$\Pr\{f = \text{ONE}\} \geq 2^{-(n+1)} \ggg \text{Ø}(n)^{-1} \tag{6}$$

Korshunov's asymptotic to Ø(n) (given in equation (1)) provides an idea of the magnitude of the difference in inclusion probabilities for the ONE function.

The fact that certain functions have extremely high (relative to the uniform case) inclusion probabilities, comes at the expense of other functions having extremely low inclusion probabilities. This algorithm seems to generate monotone boolean functions with few border vectors more frequently and functions with many border vectors less frequently than in the uniform case. Suppose GENERATE-MBF is used and the number of queries for a particular inference algorithm significantly increases with the number of border vectors. Then an estimate of the average query complexity

which ignores the inclusion probabilities is seriously **biased** downwards.

A lower bound on, instead of the exact value of, the exact inclusion probability of the ONE function was given by inequality (6). This is due to the fact that inclusion probabilities are hard to compute for GENERATE-MBF, because there are so many different ways of generating the same function. Since the inclusion probabilities are essential to unbiased estimators, they need to be easy to compute. If nothing is known a priori about the number of queries, then it is also desirable for the inclusion probabilities to be close to the uniform case which will lower the variance of the estimator given in equation (5).

Finding the probability of a particular set of random classifications is easiest if the classifications are performed independently. However, if all of the vectors are classified independently, the resulting function f may not necessarily be monotone. Therefore, consider imposing monotonicity onto the function f, by creating the pair of unique "smallest" and "greatest" monotone functions, denoted respectively by f_S and f_G, that sandwich the original function by $f_S \leq f \leq f_G$.

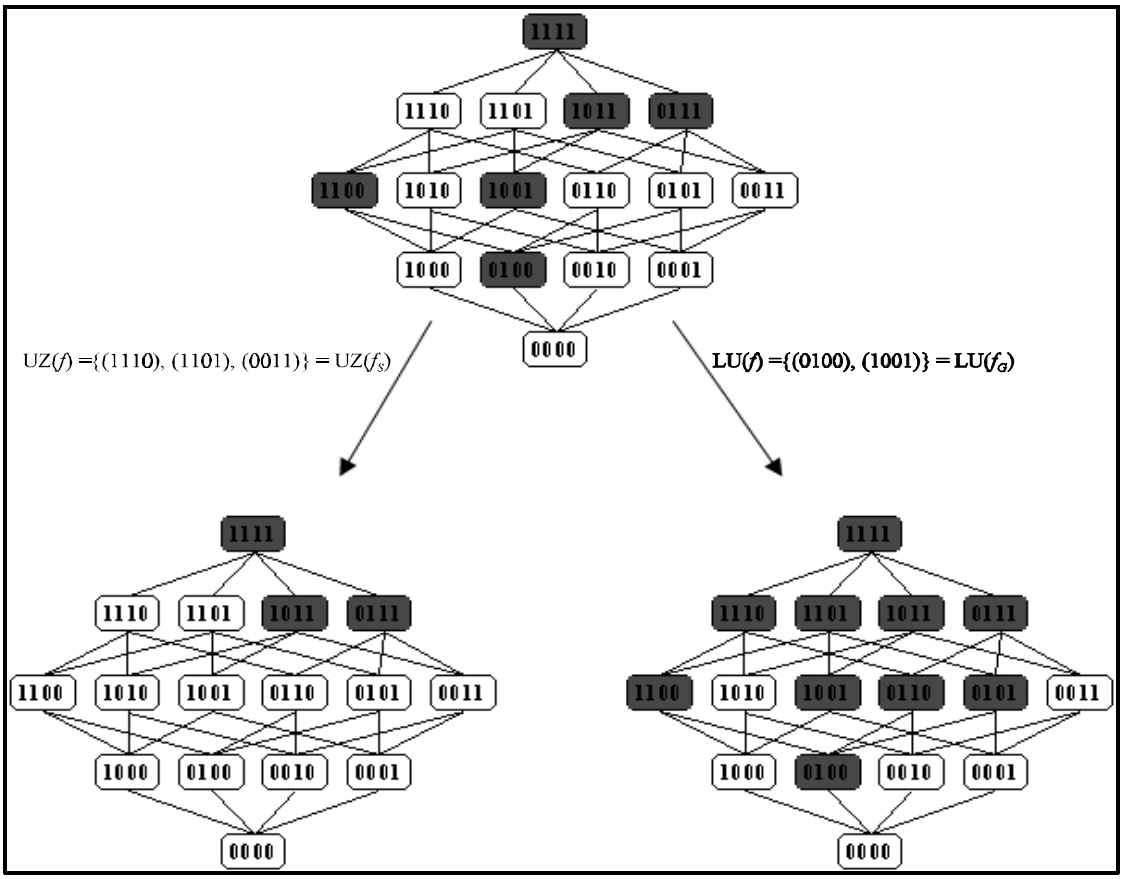

Figure 11. Two sandwich monotone functions created by a non-monotone function defined on $\{0,1\}^4$.

Figure 11 illustrates how the pair of monotone functions are created from a non-monotone function. The shaded vectors carry a function value of 1, while the non-shaded vectors carry a function value of 0. The non-monotone function f shown in the top of the figure has upper zeros UZ(f) = {(1110), (1101), (0011)} and lower units LU(f) = {(0100), (1001)}. The upper zeros of function f uniquely define the first target monotone function f_S shown in the bottom left of the figure, while the lower units of f uniquely define the second target monotone function f_G shown on the bottom right.

The algorithm GENERATE-MBF-P(n, p) shown in Figure 12 creates the general function f, by classifying $f(v)$ as 1 with probability $p(v)$, or zero with probability 1- $p(v)$ for all vectors in $\{0,1\}^n$. It then places all the upper zeros of f into the set S, and all the lower units into the set G. The first target monotone function f_S is then defined by the upper zeros in S, while the second target monotone function f_G is defined by the lower units in G.

```
GENERATE-MBF-P(n,p)
1  G = NIL, S = NIL
2  for each v ∈ {0,1}^n
3     f(v) ← SELECT-AT-RANDOM({0,1}, {1-p(v),p(v)})
4     if f(v) = 1
5        W = {w: v ≤ w, w in G}
6        if W = NIL
7           G ← G ∪ {v}
8        else
7           G ← G - W
8     else
9        W = {w: v ≥ w, w in S}
10        if W = NIL
11          S ← S ∪ {v}
12       else
13          S ← S - W
14 return G, S
```

Figure 12. Randomly generating monotone boolean functions with inclusion probabilities.

After the two sandwich functions have been generated, their inclusion probabilities, denoted by p_G and p_S, have to be computed. To that end, define the two random monotone boolean functions F_S and F_G by the output of a single execution of the random process GENERATE-MBF-P. Then, the inclusion probability for function f is the probability that it was generated as the smallest or the greatest function, given by:

$$\Pr\{F_S = f \vee F_G = f\} = \Pr\{F_S = f\} + \Pr\{F_G = f\} - \Pr\{F_S = f \wedge F_G = f\}, \tag{7}$$

where

$$\Pr\{F_S = f\} = \Pr\{F(v) = 1 \ \forall\, v \in \mathrm{LU}(f),\ F(v) = 0 \ \forall\, v \in \{w\colon f(w) = 0\}\}$$
$$= \prod_{v \in LU(f)} p(v) \prod_{v \in \{w\colon f(w)=0\}} (1-p(v))$$

and similarly,

$$\Pr\{F_G = f\} = \Pr\{F(v) = 0 \ \forall\, v \in \mathrm{UZ}(f),\ F(v) = 1 \ \forall\, v \in \{w\colon f(w) = 1\}\}$$
$$= \prod_{v \in UZ(f)} (1-p(v)) \prod_{v \in \{w\colon f(w)=1\}} p(v)$$

and finally

$$\Pr\{F_S = f \wedge F_G = f\} = \Pr\{F(v) = f(w) \ \forall\, v \in \{0,1\}^n\}$$
$$= \prod_{v \in \{w\colon f(w)=1\}} p(v) \prod_{v \in \{w\colon f(w)=0} (1-p(v))$$

Suppose the vector classification probability function $p(v)$, defined as follows, is used as input for the algorithm GENERATE-MBF-P.

$$p(v) = \begin{cases} 1/168, & v = (0000) \\ 20/168 = 5/42, & v \in \{(1000),(0100),(0010),(0001)\} \\ 84/168 = 1/2, & v \in \{(1100),(1010),(1001),(0110),(0101),(0011)\} \\ 148/168 = 37/42, & v \in \{(1110),(1101),(1011),(0111)\} \\ 167/168, & v = (1111) \end{cases}$$

This particular definition of $p(v)$ comes from $\emptyset_{|v|}(n) / \emptyset(n)$ which is described immediately after this example. Suppose the general function shown in the top of Figure 11 was generated as a result. Consider computing the inclusion probabilities for the function f_G shown on the bottom right of Figure 11. The lower units of f_G are {(0100), (1001)}, while its upper zeros are {(1010), (0011)}. Let q_k denote its inclusion probability when GENERATE-MBF-P is executed k times. Then the inclusion probability for this function is given as follows:

$$q_k = 1 - (1 - q_1)^k$$

Here the inclusion probability for a single execution can be computed using equation (7) as follows:

$$\begin{aligned} q_1 &= p(0100)p(1001)(1-p(1010))(1-p(0011))(1-p(1000))(1-p(0010))(1-p(0001))(1-p(0000)) \\ &\quad + (1-p(1010))\,(-p(0011))\,p(0100)\,p(1100)\,p(1001)\,p(0110)\,p(0101)\times \\ &\quad\; p(1110)\,p(1101)\,p(1011)p(0111)\,p(1111) \\ &\quad - \; p(1111)p(1110)p(1101)p(1011)p(0111)p(1100)p(1001)p(0110)p(0101)p(0100)\times \\ &\quad (1-p(1010))(1-p(0011))(1-p(1000))(1-p(0010))(1-p(0001))(1-p(0000) \end{aligned}$$

$$= \left(\frac{5}{42}\right)\left(\frac{1}{2}\right)^3\left(\frac{37}{42}\right)^3\left(\frac{167}{168}\right) + \left(\frac{1}{2}\right)^6\left(\frac{5}{42}\right)\left(\frac{37}{42}\right)^4\left(\frac{167}{168}\right) - \left(\frac{167}{168}\right)^2\left(\frac{37}{42}\right)^7\left(\frac{1}{2}\right)^6\left(\frac{5}{42}\right) \approx 0.01047$$

If, for example, GENERATE-MBF-P was executed 20 times, the inclusion probability is equal to $q_{20} \approx 1 - (1 - .01047)^{20} \approx 0.1898$ for function f_G. The inclusion probability for other monotone boolean functions can be computed in a similar fashion.

To generate the functions close to uniformly, a random 0 or 1 vector classification should occur according to the vectors respective fraction of monotone boolean functions that classify it as 0 or 1. Let $\emptyset_k(n)$ be the number of monotone boolean functions defined on $\{0,1\}^n$ that classifies a vector v, for which $|v| = k$, as 1. Therefore, a vector with hamming weight k should be classified 1

with probability $Ø_k(n) / Ø(n)$. This procedure does not result in complete uniformity, yet it is a step in its direction using independent classifications. Equation (8) gives the known generalizations of $Ø_k(n)$.

$$Ø_k(n) = \begin{cases} 1, & k=0 \\ Ø(n-1), & k=1 \\ Ø(n)/2 & k=n/2,\ n\ \textit{is even} \\ Ø(n)-Ø(n-1), & k=n-1 \\ Ø(n)-1, & k=n \end{cases} \tag{8}$$

Equation (8) indicates that computing $Ø_k(n)$ is just as hard as computing $Ø(n)$. We computed some of the exact values for $Ø_k(n)$, which are given in Table III.

Table III. Some exact values for $Ø_k(n)$.

$Ø_k(n)$	$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 5$	$n = 6$
$k = 0$	1	1	1	1	1	1
$k = 1$	2	3	6	20	168	7,581
$k = 2$		5	14	84	2,008	595,649
$k = 3$			19	148	5,573	3,914,177
$k = 4$				167	7,413	7,232,705
$k = 5$					7,580	7,820,773
$k = 6$						7,828,353

The solid curves in Figure 13 correspond to $Ø_k(n)/Ø(n)$ for $n = 1, 2, ..., 6$. These curves exhibit symmetric S-shapes that change rapidly as n increases. This is the motivation behind the transformation of the distribution function into a sigmoid function as follows,

$$\frac{Ø_k(n)}{Ø(n)} = \frac{e^{á(n,k)(k-n/2)}}{1 + e^{á(n,k)(k-n/2)}}. \tag{9}$$

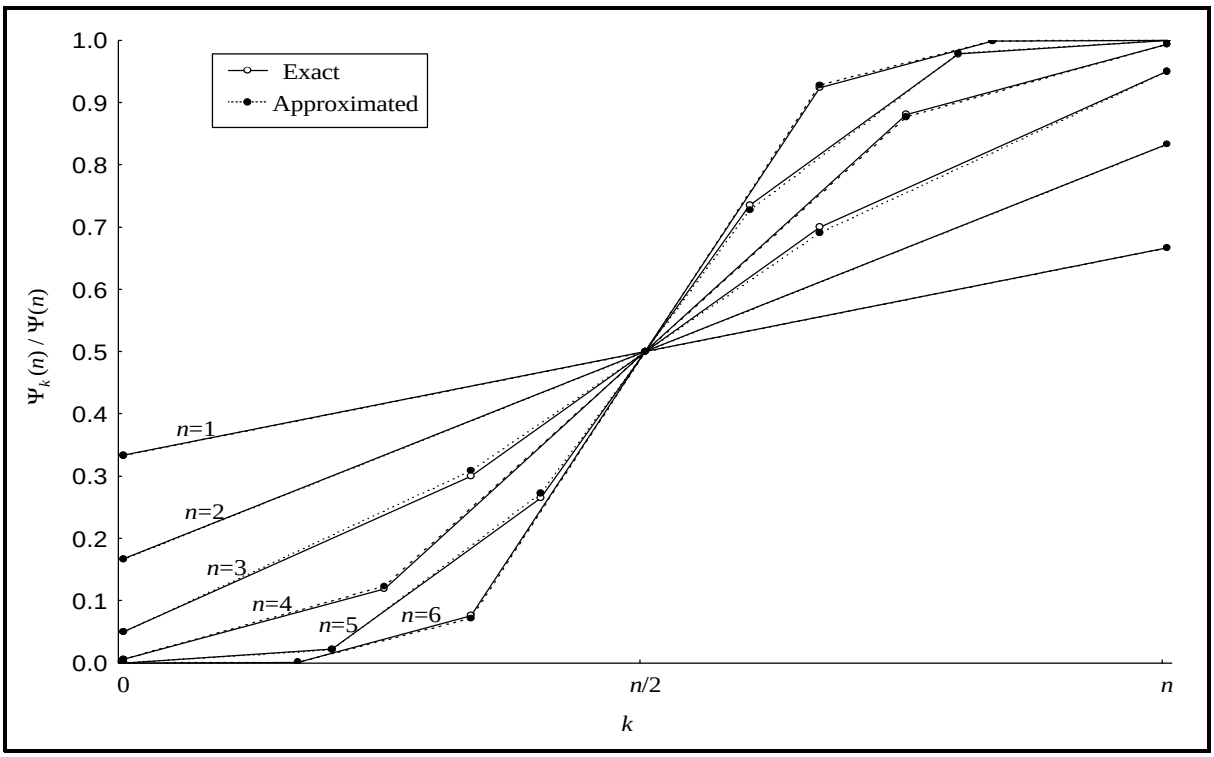

Figure 13. Comparing the exact values to the approximations of $Ø_k(n)/Ø(n)$.

Since á(n, k) is a function of n and k, no information is lost or gained by the transformation. The problem of approximating $Ø_k(n)/Ø(n)$ is merely transformed into the equivalent problem of approximating the function á(n, k). However, an approximate relationship between the known á(n, k) values is more transparent (see Table IV) than between the known $Ø_k(n)$'s in Table III.

Table IV. Some values of á(n,k).

á(n,k)	n = 1	n = 2	n = 3	n = 4	n = 5	n = 6	n = 7	n = 8
k = 0	1.386	1.609	1.963	2.560	3.573	5.291	8.147	13.10
k = 1			1.695	2.002	2.525	3.490	5.056	7.957
k = 2					2.042	2.497		

Solving (9) for á(n, k) gives,

$$\text{á}(n,\ k) \ = \ \frac{\ln(Ø(n) \ - \ Ø_k(n)) \ - \ \ln(Ø_k(n))}{n/2 - k}\ , \ \text{for } n > 0,\ n \neq 2k, \text{ and } k = 0,\ 1,\ ...,\ n.$$

Furthermore, any value for á(n, k) in equation (8) will give the correct value of ½ for $Ø_k(n)/Ø(n)$ when $n = 2k$. From equation (8), it is known that $Ø_0(n) = 1$, and therefore the following equation holds:

$$á(n,0) = \frac{\ln(Ø(n) - 1)}{n/2}, \text{ for } n > 0 \text{ and } n \neq 2k. \quad (10)$$

Equation (8) also provides the relationship $Ø_1(n) = Ø(n\text{-}1)$ which further implies the following equation:

$$á(n,1) = \frac{\ln(Ø(n) - Ø(n-1)) - \ln(Ø(n-1))}{n/2-1}, \text{ for } n>0 \text{ and } n\neq 2k.$$

To compute the values for á(n, 0) and á(n, 1) only Ø(n) is needed. For n up to and including 8, the exact values of Ø(n) are known and were given in Table I. For n greater than 8 Korshunov's approximation given in equation (1) can be used. However, to compute á(n, k) for $k > 1$, a generalization is needed. Based on the observed values in Table IV, it seems reasonable to use

$$á(n, k) \approx á(n\text{-}1, k\text{-}1) \text{ for } k = 1, 2, ..., \lceil n/2 \rceil\text{-}1,$$

and then use the fact that á(n, k) = á(n-k, k) to find the á(n, k) values for $k = \lceil n/2 \rceil+1, ..., n$.

Figure 13 shows the approximated curves, for n = 1, 2, ..., 6, resulting from using (10) and generalizing with á(n, k) = á(n-1, k-1) and á(n, k) = á(n-k, k). Approximations of $Ø_k(n)/Ø(n)$ that are off by a few percent points will not have a major effect on the experiments described in section 5. The approximations based on á(n, k) ≈ á(n-1, k-1) are probably worse for larger k, however, these approximations are only needed for n up to 11 (i.e., for k up to 5) in the experiments performed here. It should be noted that it may be necessary to work with the inclusion probabilities in a log base as they get extremely small. For example, $Ø(11) > 10^{144}$ using Korshunov's asymptotic given in equation (1).

4. Comparison of Inference Algorithms

4.1 Some Preexisting Algorithms

Only someone that knows the function ahead of time, i.e., an all knowing *Teacher*, can achieve the

minimum number of evaluations for every single monotone boolean function, thus φ (*Teacher*, f) = $m(f)$, $\forall\ f \in M_n$. For any algorithm A, that does not have prior knowledge about the underlying function f other than that it is monotone, $m(f)$ can be considered as a lower bound on the number of questions. That is, the following inequality always holds: $m(f) \le \varphi(A, f), \forall\ f \in M_n$ and A $\neq$ *Teacher*.

It turns out that it is possible to achieve fewer or the same number of function evaluations as the upper bound on $m(f)$ given in inequality (2), for all monotone boolean functions defined on $\{0,1\}^n$. One realization of this is based on partitioning the set $\{0,1\}^n$ into chains, with increasing length as described in Hansel (1966), and then searching the chains in that order. Proof of the following inequality can be found in both Hansel (1966) and Sokolov (1982):

$$\varphi(\textit{Hansel},\ f\ ) \le \max_{f \in M_n} m(\ f\ )\ ,\ \forall\ f \in M_n\ . \tag{11}$$

The algorithm described in Sokolov (1982) is also based on the Hansel chains, but considers the chains in the reverse order (i.e., in decreasing length) and performs a binary search of each chain. It turns out that Sokolov's algorithm is much more efficient for functions that have all their border vectors in the longer Hansel chains. As an example, consider the uniform ONE function, which has only one border vector and this border is located in the longest chain. For this function, Sokolov's algorithm performs at most $\log_2(n+1)$ evaluations, while Hansel's algorithm needs at least $\binom{n}{\lfloor n/2 \rfloor}$ evaluations. However, Sokolov's algorithm does not satisfy the upper bound (11), as the following example shows. Suppose that $n > 4$ and even, and the monotone boolean function to be inferred is defined by $f(v) = 1$ if $|v| > n/2$, and 0 otherwise. Then the set of border vectors is $\{v,$ $|v| = n/2$ or $n/2\text{-}1\}$ and $m(f) = \binom{n}{\lfloor n/2 \rfloor} + \binom{n}{\lfloor n/2 \rfloor + 1}$. In Sokolov's algorithm, the first vector w^1, submitted for evaluation is a border vector since $|w^1| = n/2$. The second vector w^2 is not a border vector because $|w^2| = \lceil 3n/4 \rceil \neq n/2$ and $n/2\text{-}1$. Therefore, the following inequality holds:

$$\varphi(\textit{Sokolov}, f\ ) > \binom{n}{\lfloor n/2 \rfloor} + \binom{n}{\lfloor n/2 \rfloor + 1}\ ,\ \text{for at least one } f \in M_n.$$

In an attempt to provide a unified efficiency testing platform, Gainanov (1984) proposed to evaluate algorithms based on the number of evaluations needed for each border vector. To that end, he presented an algorithm that searches for border vectors one by one. At the core of the algorithm

is a subroutine that takes as input any unclassified vector v, and finds a border vector by successively evaluating adjacent vectors. This subroutine, which will be referred to as FIND-BORDER(v), is also used in the algorithms of Boros et al. (1997), Makino and Ibaraki (1995), and Valiant (1984). Given any unclassified vector as input, this subroutine will find a border vector using at most n+1 evaluations. As a result, an algorithm A that utilizes any method to generate unclassified vectors and uses the subroutine FIND-BORDER to find the border vectors, satisfies the following upper bound:

$$\varphi(A, f) \leq m(f)(n+1), \quad \forall \ f \in M_n.$$

However, for the majority of monotone boolean functions, the expression $m(f)(n+1)$ is greater than or equal to 2^n, in which cases the bound is trivial.

4.2 Experimental Results

The different inference algorithms described in section 4.1, do not specify which vector to select when there are ties. In particular, the Sokolov and Hansel algorithms may have to choose between two vectors that make up the middle of a particular chain. Furthermore, an algorithm that uses the subroutine FIND-BORDER needs to be fed an unevaluated vector, of which there may be many. Even the evaluative criterion $min|K_1\text{-}K_0|$ which was described in section 2.5 may result in ties. For the purpose of comparing the algorithms on the same ground and without introducing another aspect of randomness, ties were broken by selecting the first vector in the list of tied vectors.

The results in Figure 14 are based on an exhaustive analysis (i.e., all the monotone functions were generated) for n up to and including 5. For n greater than 5 random samples of functions were generated by the algorithm GENERATE-MBF-P(n, p) using the estimate of $Ø_{|v|}(n) / Ø(n)$ constructed in section 3.2 for $p(v)$ for all $v \in \{0,1\}^n$. The algorithm was executed 1,000 times for n equal to 6, 7 and 8, while the number of executions was reduced to 100 times for n equal to 9, 10 and 11 because of time limitations. That is, 2000 functions were generated for n equal to 6, 7 and 8, and 200 functions for n equal to 9, 10 and 11, since each execution results in a pair of functions. This is the

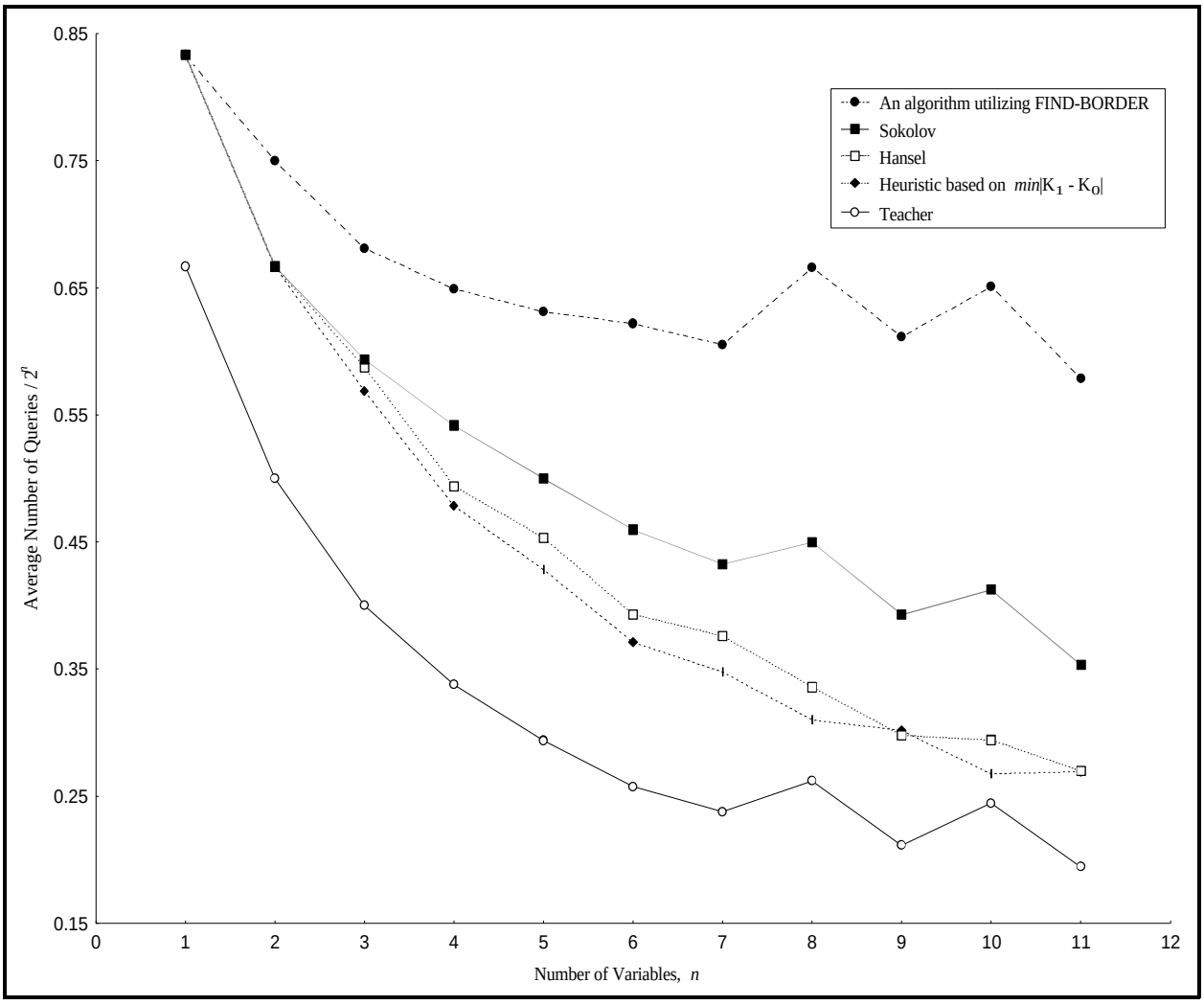

Figure 14. Query complexity for different inference algorithms.

maximum number of functions used in the estimate, because the functions are generated with replacement. However, since the likelihood of generating the same functions more than once is small (especially for larger values of n) the effective sample size was generally close to the these maxima.

The Horvitz-Thompson estimator given by equation (5) is used to compute the averages for n greater than 5. The average number of questions is normalized by the maximum possible number of questions 2^n so that the magnitude of all the averages in Figure 14 are not overshadowed by the large values obtained for n equal to 11. As a consequence, two algorithms that result in parallel curves in such a plot, have an exponential (in n) difference in the average number of queries. Also, the gap between the curves in Figure 14 and the horizontal line *Average Number of Queries* / 2^n = 1 (not shown in the figure) can be thought of as the benefit of the monotone assumption. This is due

to the fact that 2^n is the number of required queries when the underlying function is not necessarily monotone.

The curve titled "Teacher" represents the lower bound on the number of questions for every single function. Therefore, it is expected that a few extra questions are required on the average. Since the heuristic based on the evaluative criterion $min|K_1\text{-}K_0|$ achieves the minimum average number of questions for n up to 4, it can be thought of as a lower bound on the average, and its gap between "Teacher" quantifies the benefits of knowing the actual function beforehand.

Figure 14 paints a clear picture of how the preexisting inference algorithms fare against each other. Hansel's algorithm was the best performer by far, Sokolov's came in second and an algorithm using the subroutine FIND-BORDER (also used by Gainanov (1984), Valiant (1984), Makino and Ibaraki (1995), and Boros et al. (1997)) was a distant third. In fact, since the curve differences between Hansel and Sokolov, and Sokolov and the subroutine FIND-BORDER implementation, seem to increase with n, the corresponding difference in the average number of queries increases at rate greater than exponentially with n. Furthermore, the difference between the curves for Hansel and "Teacher" decreases as n increases. However, since the heuristic based on $min|K_1\text{-}K_0|$ seems to have a curve that is almost parallel to Hansel's curve, it is unlikely that Hansel's curve will approach the "Teacher" curve for large n.

Our evaluative criterion $min|K_1\text{-}K_0|$ seemed to perform about 2% better than Hansel's algorithm. This decrease is especially clear in Figure 14 for n up to and including 8. For larger values of n, the high variance of our estimates makes it hard to distinguish the two curves, but the overall decreasing trends remain intact. It might seem that a 2% decrease is insignificant, but writing it as $2^n \times 0.02$ shows its real magnitude.

6. Conclusions

The recent focus on the computational complexity has come at the expense of a drastic increase in the query complexity. In fact, the more recent the inference algorithm is, the worse it performs in terms of average query complexity. The subroutine, here referred to as FIND-BORDER, is the most commonly used in the recent literature (Gainanov (1984), Valiant (1984), Makino and Ibaraki (1995), and Boros et al. (1997)), and its performance was by far the worst. Therefore, the

framework for unbiased empirical comparison of inference algorithms developed in this paper seems to be long overdue.

Even though guaranteeing the minimum average number of queries is currently only computationally feasible for relatively few variables (i.e., up to 5 or 6), the recursive algorithm presented here revealed the non-intuitive nature of the optimal solutions. While only the monotone boolean functions defined on $\{0,1\}^n$ were studied here, these particular vector subsets are associated with many applications in their own right. Furthermore, the recursive algorithm executes just as well for monotone boolean functions defined on any reasonably sized set of vectors. The only requirement to guarantee the minimum average query cost is that the assumption of independent and equal query costs holds.

The optimal solutions paved the way for the evaluative criterion $min|K_1\text{-}K_0|$ that probably would not have been developed (due to its non-intuitive nature) without the consultation of the optimal solutions. The heuristic using this evaluative criterion extends the feasible problem sizes to up to about 20 variables (which involves about 1 million vectors). When the number of variables exceeds 20, computing the evaluative criterion might become intractable, while Hansel's algorithm will most likely still perform the best on the average. When creating the chain partition used in Hansel (1966) and Sokolov (1982) becomes intractable, finding border vectors one by one using the subroutine FIND-BORDER is the last but perhaps still computationally feasible resort.

Acknowledgments

The authors would like to thank Dr. Ilya Shmulevich at the Tampere University of Technology in Finland for suggesting the sandwich functions used in the algorithm GENERATE-MBF-P and Dr. Dean Judson at the US Census Bureau for stimulating discussions on different evaluative criteria. Also, the authors gratefully acknowledge the support from the Office of Naval Research (ONR) Grants N00014-95-1-0639 and N00014-97-1-0632.

References

Bioch, J.C., T. Ibaraki. 1995. Complexity of Identification and Dualization of Positive Boolean Functions. *Information and Computation* **123** 50-63.

Boros, E., P.L. Hammer, T. Ibaraki., K. Makino. 1997. Polynomial-Time Recognition of 2-Monotonic Positive Boolean Functions Given by an Oracle. *SIAM Journal on Computing* **26** 1 93-109.

Church, R. 1940. Numerical Analysis of Certain Free Distributive Structures. *Duke Mathematical Journal* **6** 732-734.

Church, R. 1965. Enumeration by Rank of the Free Distributive Lattice with 7 Generators. *Notices of the American Mathematical Society* **11** 724.

Dedekind, R. 1897. Ueber Zerlegungen von Zahlen durch ihre Grössten Gemeinsamen Teiler. *Festschrift Hoch. Brauhnschweig u. ges Werke II*, 103-148.

Eiter, T., Gottlob G. 1995. Identifying the Minimal Transversals of a Hypergraph and Related Problems. *SIAM Journal on Computing* **24** 6 1278-1304.

Engel, K. 1997. *Encyclopedia of Mathematics and its Applications 65: Sperner Theory*. Cambridge University Press, Cambridge, MA.

Fredman, M.L., L. Khachiyan.1996. On the Complexity of Dualization of Monotone Disjunctive Normal Forms. *Journal of Algorithms* **21** 618-628.

Gainanov, D.N. 1984. On One Criterion of the Optimality of an Algorithm for Evaluating Monotonic Boolean Functions. *U.S.S.R. Computational Mathematics and Mathematical Physics* **24** 4 176-181.

Hansel, G. 1966. Sur Le Nombre Des Foncions Booleenes Monotones De n Variables. *C. R. Acad. Sc. Paris* **262** 1088-1090.

Horvitz, D.G., Thompson D.J. 1952. A Generalization of Sampling without Replacement from a Finite Universe. *Journal of the American Statistical Association* **47**, 663-685.

Judson, D.H. 1999. On the Inference of Semi-Coherent Structures from Data. A Master's Thesis, University of Nevada, Reno, NV.

Korshunov, A.D. 1977. Solution of Dedekind's Problem on the Number of Monotonic Boolean Functions. *Soviet Mathematics Doklady* **18** 2 442-445.

Makino, K., T. Ibaraki. 1995. A Fast and Simple Algorithm for Identifying 2-Monotonic Positive Boolean Functions. *Proceedings of ISAACS'95, Algorithms and Computation*, Springer-Verlag, Berlin, Germany. 291-300.

Makino, K., T. Ibaraki. 1997. The Maximum Latency and Identification of Positive Boolean Functions. *SIAM Journal on Computing* **26** 5 1363-1383.

Makino, K., T. Suda, H. Ono, T. Ibaraki. 1999. Data Analysis by Positive Decision Trees. *IEICE Transactions on Information and Systems* **E82-D** 1 76-88.

Sokolov N.A. 1982. On the Optimal Evaluation of Monotonic Boolean Functions. *U.S.S.R. Computational Mathematics and Mathematical Physics* **22** 2 207-220.

Thompson, S. K. 1992. *Sampling*. John Wiley & Sons, Inc., New York, NY.

Valiant, L.G. 1984. A Theory of the Learnable. *Communications of the ACM* **27** 11 1134-1142.

Ward, M. 1946. Note on the Order of the Free Distributive Lattice. *Bulletin of the American Mathematical Society* **52** 135 423.

Wiedemann, D. 1991. A Computation of the Eight Dedekind Number. *Order* **8** 5-6.