Minimizing the Average Query Complexity of Learning Monotone Boolean Functions

Vetle I. Torvik • Evangelos Triantaphyllou
Department of Psychiatry (MC 912) University of Illinois,
1601 West Taylor Street, Chicago, Illinois 60612-4321 USA
Department of Industrial and Manufacturing Systems Engineering,
Louisiana State University, Baton Rouge, Louisiana 70803-6409, USA
vtorvik@uic.edu • trianta@lsu.edu

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. For problems with up to 4 variables, the posets' optimality conditions are summarized in the form of an evaluative criterion. The evaluative criterion extends the computational feasibility to problems involving up to about 20 variables. A framework for unbiased average case comparison of monotone Boolean function inference algorithms is developed using unequal probability sampling. The unbiased empirical results show that an implementation of the subroutine considered as a standard in the literature performs almost twice as many queries as the evaluative criterion on the average. It should also be noted that the first algorithm ever designed for this problem performed consistently within two percentage points of the evaluative criterion. As such, it prevails, by far, as the most efficient of the many preexisting algorithms.

(*Artificial Intelligence; Programming: Integer: Algorithms: Branch and Bound; Networks-Graphs: Matchings; Statistics: Sampling*)

1. Introduction

Situations where a set of binary variables have a non-negative (i.e., monotone) relationship with a phenomenon are inherently frequent in applications. Suppose a computer tends to crash when it runs a particular word processor and web browser simultaneously. Then, the computer will probably crash if it, in addition, runs other software applications. Further, suppose this computer does not tend to crash when it runs a particular CD player and web browser simultaneously. Then, it will probably not crash when it only runs the web browser (or the CD player).

Recent literature contains a plethora phenomena that can be modeled by using monotone Boolean functions. Such diverse phenomena include, but are not limited to, social worker's decisions, lecturer evaluation and employee selection (Ben-David 1992), chemical carcinogenicity, tax auditing, and real estate valuation (Boros et al. 1994), breast cancer diagnosis and engineering reliability (Kovalerchuk et al. 1996), signal processing (Shmulevich 1997), rheumatology (Bloch and Silverman 1997), voting rules in the social sciences (Judson 1999), financial systems (Kovalerchuk and Vityaev 2000b), and record linkage in databases (Fellegi and Sunter 1969, Winkler 1995, and Judson 2001).

0899-1499/02/1402/0144$5.00
1526-5528 electronic ISSN

In practice, a great deal of time and effort is put into learning and understanding these monotone relationships. Software applications are tested, diseases are researched, database search engines are trained to be intelligent, 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 (*membership queries* submitted to an oracle). The *oracle* can be thought of as an entity that knows the underlying monotone Boolean function and provides a Boolean value in response to each query. In practice, it may take the shape as a human expert, or as the outcome of a task, such as running an experiment or searching a large database. 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 the oracle as a membership query. 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 in order 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 focuses on reducing the query complexity (Hansel 1966, Sokolov 1982, and Gainanov 1984). More recent work includes the computational complexity (Fredman and Khachiyan 1996, Boros et al. 1997, and Makino and Ibaraki 1997).

The problem of inferring monotone Boolean functions via membership queries is equivalent to many other computational problems in a variety of fields (Bioch and Ibaraki 1995, and Eiter and Gottlob 1995). For these applications, algorithms that are efficient in terms of query and computational complexity are used. In practice, queries often involve some sort of effort, such as consulting with experts or performing experiments. For such applications, queries far surpass computations in terms of cost. This paper is therefore focused on minimizing the query complexity as long as it is computationally feasible.

The paper is organized as follows: In Section 2 some background information on monotone Boolean functions is provided. An inference objective is proposed in Section 3 together with an approach for optimizing it. Section 4 describes an unequal probability sampling framework for generating monotone Boolean functions and comparing different inference algorithms. This framework is used in Section 5 to provide an extensive unbiased empirical comparison of various inference algorithms. In the end, Section 6 provides some concluding remarks.

2. 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) $v_i \le w_i$ for $i = 1, 2, \ldots, n$, 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 $v_i \ge w_i$ for $i = 1, 2, \ldots, n$. 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 $f(v) \le f(w) \forall (v, w)$: $v \preceq w$, and *non-increasing*, iff $f(v) \ge f(w) \forall (v, w)$: $v \preceq w$. This paper focuses on non-decreasing functions, which are referred to as monotone, as similar results hold for non-increasing functions.

The number of distinct monotone Boolean functions defined on $\{0,1\}^n$ is denoted by $\Psi(n)$. All known values of $\Psi(n)$ are given in Table 1. 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

Table 1 **History of Monotone Boolean Function Enumeration for $n = 1, 2, \ldots, 8$**

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

the best known asymptotic, is due to Korshunov (1981):

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

Figure 1 shows a *partially ordered set* (or *poset* for short). In general, posets can be formed by a set of vectors together with the precedence relation $\preceq$. A poset can be viewed as a directed graph 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 as a directed graph, the edges' directions and the redundant edges are often omitted without loss of information, as in Figure 1. The graph of a poset is acyclic and so all of the directions can be forced either up or down the page by ordering the vertices by layers. In Figure 1, the edges' directions all go upwards on the page. Relations that are transitively implied by other relations, are considered redundant. For example, the relation $(0000) \preceq (1100)$ is redundant because it is implied by the two relations $(0000) \preceq (1000)$ and $(1000) \preceq (1100)$.

An ordered set of related vertices $v^1 \preceq v^2 \preceq \cdots \preceq v^p$ is sometimes called a *chain*, while an *antichain* (or *layer*) consists of a set of unrelated vertices. When a set of vertices is partitioned into as few layers as possible, a *layer partition* is formed. For a particular layer partition, the layers can be ordered as $L^1, L^2, \ldots, L^r$ so that a vertex $v^i \in L^i$ cannot succeed another vertex $v^j \in L^j$ 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).

Two posets, P^1 and P^2, are said to be *isomorphic* if there exists a one-to-one mapping of the vertices in P^1 to the vertices in P^2, where the precedence relations are preserved. That is, if $v^1 \to v^2$ and $w^1 \to w^2$, 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. A poset P is called *connected* if all pairs of vertices v and w in P, are connected. That is, either v and w are directly related to each other, or there exists a sequence of vertices $v^1, v^2, \ldots, v^r$ in P for which all the pairs $(v, v^1), (v^1, v^2), \ldots, (v^r, w)$ are related.

A vector v^* is called a *lower unit* of a Boolean function f if $f(v^*) = 1$ and $f(v) < f(v^*)\,\forall v$: $v \preceq v^*$ and $v \neq v^*$. Similarly, a vector v^* is called an *upper zero* if $f(v^*) = 0$ and $f(v) > f(v^*)\,\forall v$: $v \succeq v^*$ 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 a Boolean function f. It is well known (Engel 1997) that $m(f)$ achieves its maximum value for a function that has all its border vectors on two of the most populous layers of $\{0,1\}^n$. That is, the following equation holds:

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

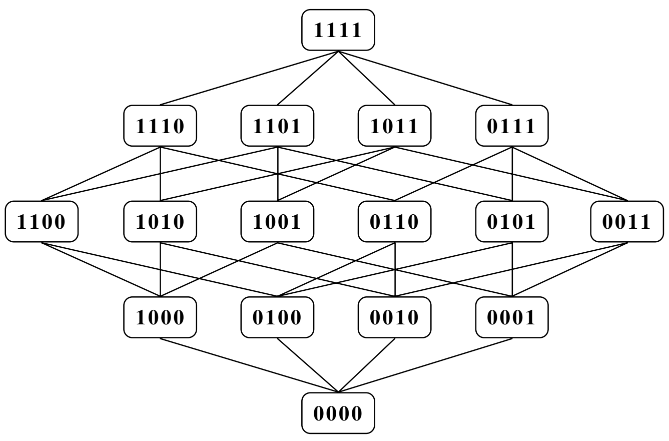

Figure 1 **The Poset Formed by $\{0,1\}^4$ and the $\preceq$ Relation**

where M_n denotes the set of all monotone Boolean functions defined on $\{0,1\}^n$. 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 $m(f)$ therefore works as a lower bound on the number of queries. The right hand side of equation (2) gives a flavor of the number of queries that may be required.

3. The Inference Objective and Its Solution

The purpose of this section is to develop an efficient algorithm for the guided inference problem. In Section 3.1, the efficiency of an inference algorithm is defined in the form of an optimization objective. An optimal solution to the proposed objective is developed in Sections 3.2 and 3.3. The resulting optimality conditions are summarized using an evaluative criterion in Section 3.4. The inference process using this criterion is demonstrated on a real life example in Section 3.6. Some other evaluative criteria are discussed in Section 3.5.

3.1. The Proposed Inference Objective

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 main existing algorithms (Hansel 1966, Sokolov 1982, and Gainanov 1984) focus on the upper bounds of their query complexities. Unfortunately, the worst-case scenario reflects the algorithm performance on a few select functions. It does not reflect what to expect when executing an algorithm on an arbitrary monotone Boolean function. For example, algorithms that implement the subroutine described in Gainanov (1984) indirectly suggest minimizing the upper bound on the number of queries per border vector. 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). This subroutine is also used in Valiant (1984), Makino and Ibaraki (1995), and Boros et al. (1997).

With no prior knowledge (other than monotonicity) about the inference application, each function is equally likely to be encountered. The functions 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:

$$Q(n)=\min_{A}\frac{\sum_{f\in M_n}\varphi(A,f)}{\Psi(n)}. \qquad (3)$$

Before the solution strategy is developed, a generalized version of problem (3) is described and solved in Section 3.2 via recursion. In Section 3.3, this recursive solution methodology is then used as a model for solving problem (3) but with an improved efficiency due to overlapping subproblems.

3.2. 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 $\{k_1,k_2,\ldots,k_n\}$ 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 $\langle(11100\ldots0),(10100\ldots0)\rangle$ is $5a$ $(=2a+3a)$, which is greater than the cost $2a+3b$ of the queries

$\langle(10100\ldots0),(11100\ldots0)\rangle$, even though the conclusion is potentially the same, i.e., that $f(10100\ldots0)=0$ (article(s) found) and $f(11100\ldots0)=1$ (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:

$$C(n)=\min_A \frac{\sum_{f\in M_n} C(A,f)}{\Psi(n)}. \tag{4}$$

Objective (4) involves arbitrary costs and is equivalent to the intuitive notion of minimizing the average inference cost. As such, it is a generalized version of 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. A path starts at the root, with the 4 unclassified vectors (00), (01), (10), and (11), and ends at one of the leaves with the empty set $\{\}$. Here, 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 $\langle(11),(10),(01),(00)\rangle$ 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, 1 and 0, respectively. The resulting tree contains 76 leaves, 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. Let c_i denote the cost of the i-th leaf from the top, for $i=1,2,\ldots,76$. 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 leaves. 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 leaves 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 leaves, will be used in the place of c_1,c_2,c_3 in the objective. Therefore, vector (01) should be selected if $c_1+c_2+c_3\le c_4+c_5+c_6$, otherwise (00) should be selected. If the sum of the costs are equal, either one of 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 $V=\{v^1,v^2,\ldots,v^p\}$ corresponding to the prior vector selections $W=\langle w^1,w^2,\ldots,w^r\rangle$ and their classifications $F=\langle f^1,f^2,\ldots,f^r\rangle$, define the cost function as follows:

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

where

$c(W,F)$ is the fixed cost associated with inferring the monotone function f defined by $f(w^1)=f^1$, $f(w^2)=f^2,\ldots,f(w^r)=f^r$ 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 $i=1,2,\ldots,p$ and $z=0,1$, and

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

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

PROOF: The correctness 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. □

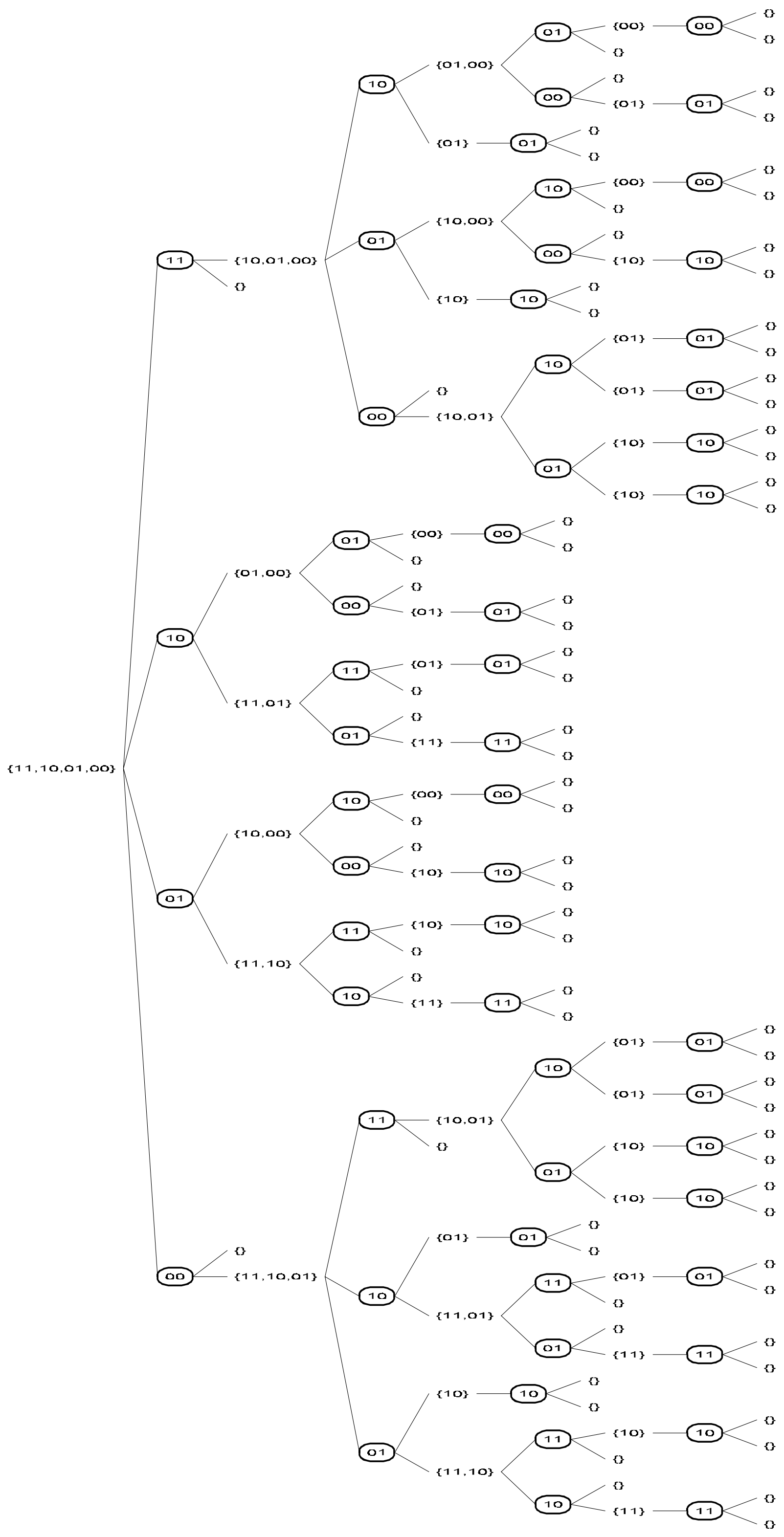

Figure 2 The Exhaustive Vector-Selection Tree for $\{0,1\}^2$

3.3. 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. 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 query 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 queries is then simply the sum of the individual costs, since it is independent of the order of the queries. As a result, the cost of evaluating a particular set of vectors is fixed, and can be written as a recursive function. For a node with the unclassified vectors $V = \{v^1, v^2, \dots, v^p\}$, define the cost function as follows:

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

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 individual query costs are fixed.*

Proof: Suppose the cost of queries are given by $c_q(v^i)$ for $i = 1, 2, \dots, p$. Then, the cost of evaluating a particular sequence of vectors does not depend on the order in which they were queried. In other words, the fixed cost function $c(\langle w^1, w^2, \dots, w^p\rangle, F)$ equals $c_q(w^1) + c_q(w^2) + \cdots + c_q(w^p)$. As a result, $C_2(V)$ equals $C_1(\langle\rangle, \langle\rangle, V)$. From Lemma 1 $C_1(\langle\rangle, \langle\rangle, \{0,1\}^n)$ 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. *If a set V of vectors consists of a set of mutually unrelated subsets $\{V^1, V^2, \dots, V^p\}$, the following equation holds:*

$$N(V) = N(V^1)N(V^2)\cdots N(V^p).$$

Furthermore, if the query costs are independent, then the following equation holds:

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

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 $w \in W = \{V^1, V^2, \dots, V^{i-1}, V^{i+1}, \dots, V^p\}$. 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 $N(V^1)N(V^2)\cdots N(V^p)$.

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 $W = \{V^1, V^2, \dots, V^{i-1}, V^{i+1}, \dots, V^p\}$, 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 $C_2(V^1) + C_2(V^2) + \cdots + C_2(V^p)$. □

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. Let the two possible vector subsets be denoted by V_z^i, for

$z = 0$ and 1, when vector v^i is selected from set V^i (for $i = 1, 2$).

LEMMA 4. *If the poset mapping* $(V^1, \preceq) \rightarrow (V^2, \preceq)$ *is isomorphic and maps vertex* v^1 *to vertex* v^2, *then the following equations hold*:

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

Furthermore, if the query costs are equal for all the vectors, then the following equations hold:

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

PROOF: This lemma can be proved by induction. As a basis of the induction consider the two non-isomorphic posets: a single vector v (from V^1 or V^2) and the empty set. The parameters $N(v) = 2$ and $C_2(v) = c$ are obviously fixed. 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, these values can be looked up.

COROLLARY 5. *For a poset* $P = (V, \preceq)$ *and its dual poset* $P^d = (V^d, \preceq)$, *the following equation holds*:

$$N(V) = N(V^d).$$

Furthermore, if the query costs are all equal, then the following equation holds:

$$C_2(V) = C_2(V^d).$$

PROOF: The proof is similar to that of Lemma 4 where the sets V and V^d replace the roles of the sets 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 query costs. In particular, only nodes corresponding to connected, non-dual, and non-isomorphic 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.

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 connected, non-dual, and non-isomorphic 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-dual, and non-isomorphic posets, starting from $\{0, 1\}^n$ and stopping when $\{\}$ is encountered. Here, an *edge vertex* of a poset, denotes a vertex that has either no related vertices preceding or no related vertices succeeding it. Lemma 6 verifies that the subroutine GENERATE-POSET-LIBRARY actually creates all possible connected, non-dual, and non-isomorphic posets encountered in the inference process.

In the second step, shown in lines 3 through 18 of MINIMIZE-AVE-Q, the N and Q parameters and the set of associated optimal vertices V 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 vertices) can be written as a 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

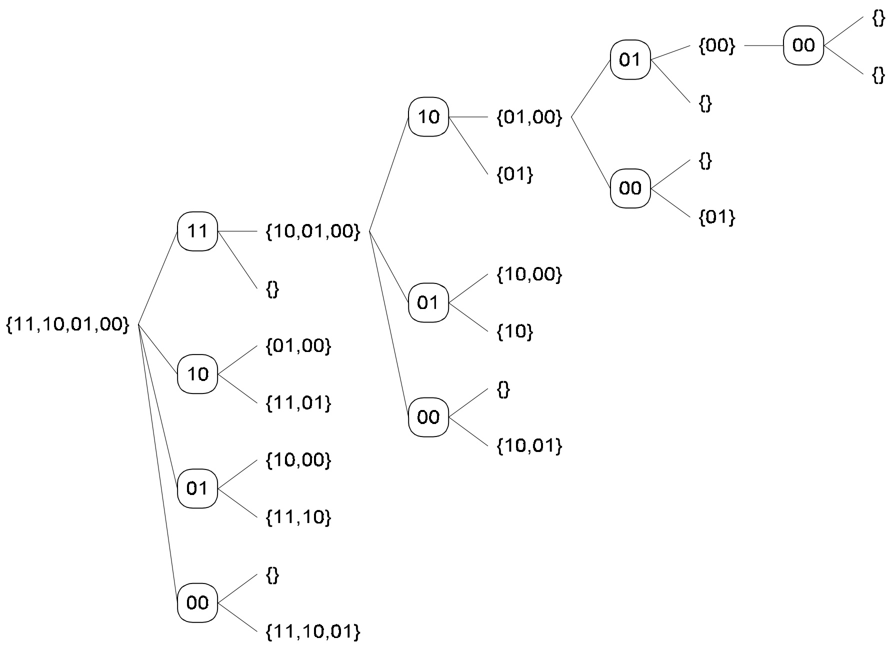

Figure 3 **The Vector-Selection Tree for $\{0,1\}^2$ Restricted to Connected, Non-Dual, and Non-Isomorphic Posets**

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, are 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 16 through 18 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 five subroutines in Figure 4 whose details have been left out for the purpose of brevity. The subroutine CONNECTED(P) should return TRUE if the poset P is connected and FALSE otherwise. The subroutine NONE-ISOMORPHIC(L, P) should return FALSE if library L contains a poset that is isomorphic to P, and TRUE otherwise. The subroutine REMOVE-CONE(P, v, f) should return the poset formed by removing the vertex 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) should return the largest partition of P into a set of posets so that they are mutually unrelated. The subroutine EDGE-VERTICES(P) should return the set containing all the edge vertices of poset P.

```
MINIMIZE-AVE-Q(n)
1   P ← ({0,1}^n,≼)
2   GENERATE-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         for each f ∈ {0,1}
9           P_f ← REMOVE-CONE(P, v, f)
10          N_f ← 1
11          Q_f ← 0
12          for each P_unr ∈ MAX-UNRELATED-POSETS(P_f)
13            N_f ← N_f × N(P_unr)
14            Q_f ← Q_f + Q(P_unr)
15        q(v) ← (N_1 × Q_1 + N_0 × Q_0)/(N_1 + N_0) + 1
16      Q(P) ← min{q(v): v ∈ P}
17      V(P) ← {v: q(v) = min{q(v): v ∈ P}}
18      N(P) ← N_1 + N_0

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

GENERATE(P)
1  if |P| > 0,
2    for each v ∈ EDGE-VERTICES(P)
3      P ← P - {v}
4      if CONNECTED(P),
5        if NONE-ISOMORPHIC(L[|P|],P)
6          L[|P|] ← L[|P|] ∪ {P}
7          GENERATE(P)
```

Figure 4 **The Algorithm Used to Compute $Q(n)$**

Lemma 6. *By recursively removing the edge vertices and only storing connected, non-dual, and non-isomorphic 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, and 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. Here, $L_k(i)$ denotes the i-th poset generated of size k. For example, the root of the tree $L_8(1)$ denotes $\{0, 1\}^3$. To reduce the size of the tree, only one branch is shown for vectors resulting in the same pair of isomorphic posets. This way, at least one optimal vector is preserved. For example, when branching on $L_8(1)$, the vector (000) results in the same pair of posets as the vector (111). As a result, they carry the same (N_0, N_1, Q_0, Q_1) values, and (000) is optimal iff (111) is. By branching on only one of them, at least one optimal vector is preserved. 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. In general, only $\lceil n/2 \rceil$ out of the 2^n branches out of the initial poset $\{0, 1\}^n$ are needed.

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 simplify visualization of the search tree. Also, the analysis in Section 3.4 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 connected, non-dual, and non-isomorphic posets that are observable in the inference problem defined on the set $\{0, 1\}^n$. It also finds all 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. For others, the optimal choices seem to be obscure. Section 3.4 addresses the issue of summarizing the optimality conditions for the posets in Figure 6 and for some posets observed for $n > 4$.

3.4. Summarizing the Poset Optimality Conditions via an Evaluative Criterion

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

When solving problem (3) for n greater than 6, the optimal approach presented in Section 3.3 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.

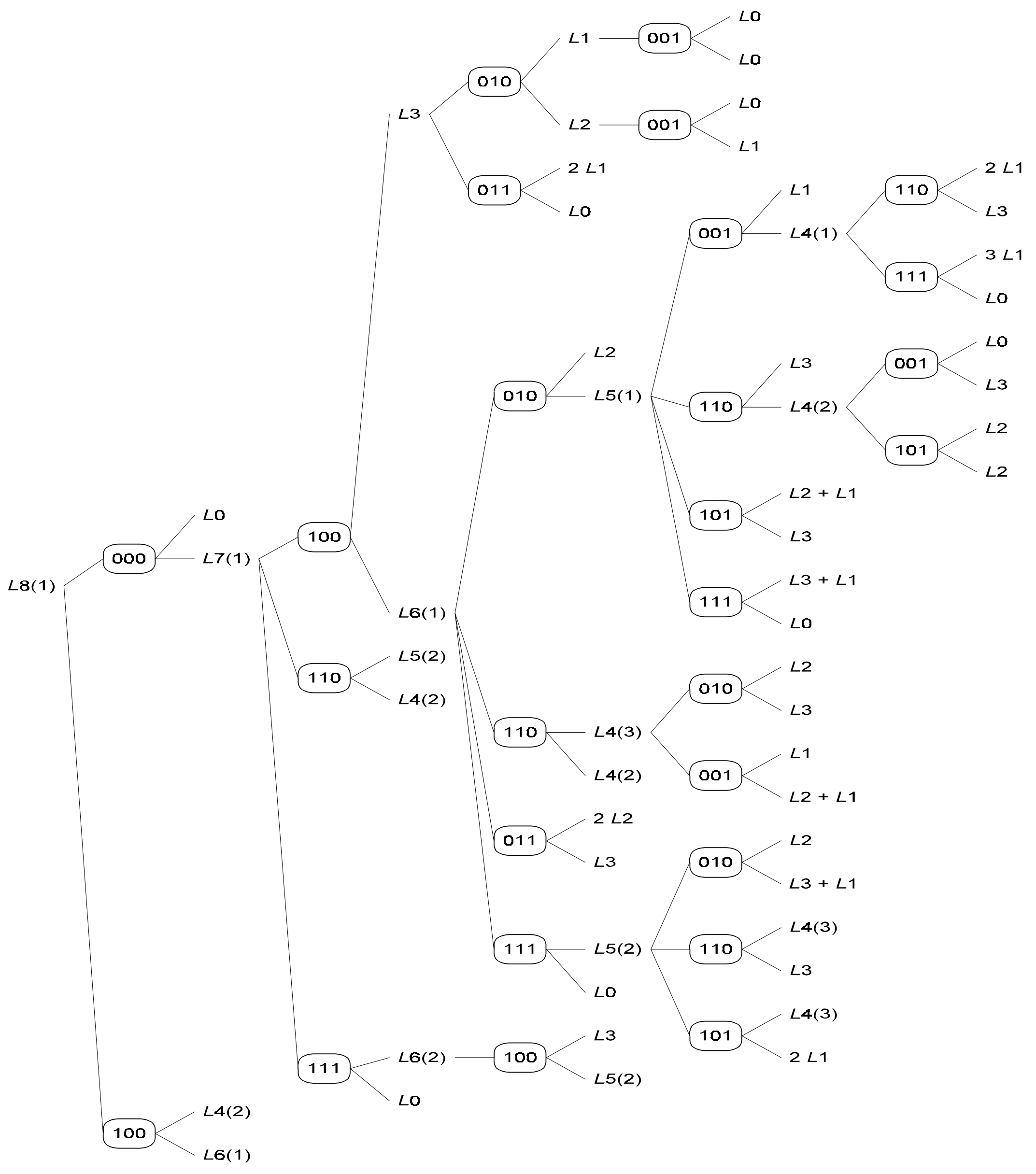

Figure 5 Part of the Tree Traversed by MINIMIZE-AVE-Q(3)

In solving problem (3) optimally for n equal to 4, many different and complex posets were encountered. The optimal vertices of these posets seemed to display two general properties. First, the optimal vertices tend to be in the *vertical middle*. More specifically, all posets in Figure 6 have at least one optimal vertex in the most populous layer. This observation alone is not sufficient to pinpoint an optimal vector. The second property observed is that the optimal vertices also tend to be *horizontal end points*. In particular, vertices related to only one other vertex in the posets in Figure 6 are always optimal. For posets that do not contain such a vertex, vertices with few related vertices tend to be optimal more often than others within a layer. To study the idea of a vertical middle and a 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.

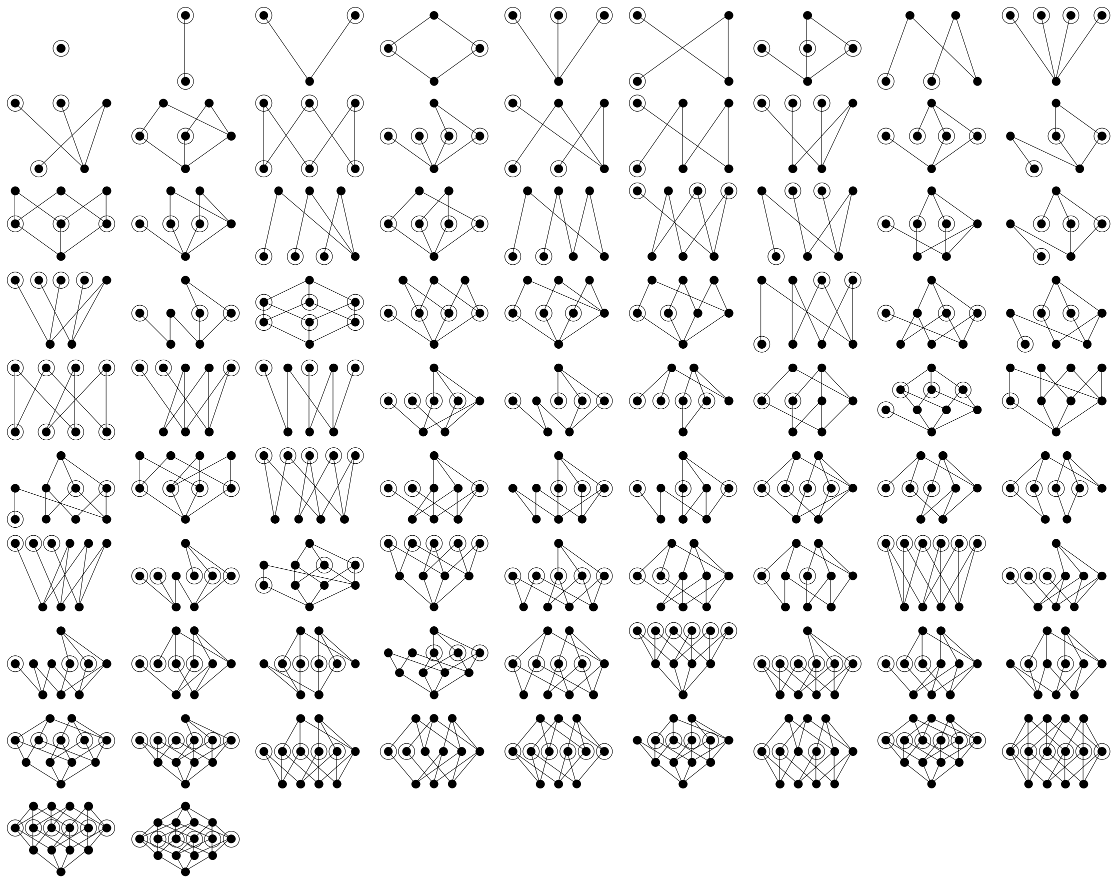

Figure 6 All the Connected, Non-Dual, and Non-Isomorphic Posets Generated by MINIMIZE-AVE-Q(4) With the Optimal Vertices Circled

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).

A chain is an arbitrarily tall connected poset with minimum width. For a chain with h vertices the number of monotone Boolean functions equals $h+1$. The minimum average number of queries is given by:

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

where $Q^c(0)=0$. These values are obtained by invariably selecting a middle vertex (i.e., the $\lceil h/2\rceil$-th vertex).

A sawtooth is an arbitrarily wide connected poset in which each vertex has two related vertices, except the end vertices which are related to only one other vertex each. Consider a sawtooth poset with w ver-

Table 2 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	3	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	13	84			4,688

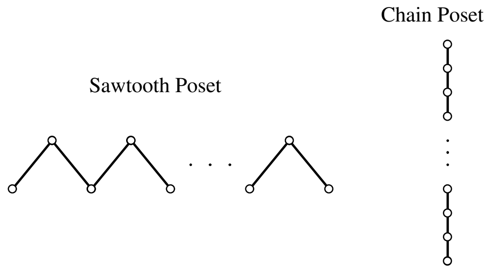

Figure 7 Illustration of the Sawtooth and Chain Posets

tices. The number of possible monotone Boolean functions $N(w)$ equals $N(w-1)+N(w-2)$. The minimum average number of queries is given by:

$$\begin{aligned} Q^s(w) &= \frac{Q^s(w-1)N(w-1)+Q^s(w-2)N(w-2)}{N(w-1)+N(w-2)}+1 \\ &\approx 0.7236068w+0.2291796, \end{aligned}$$

where $Q^s(1)=1, Q^s(0)=0, N(0)=1,$ and $N(1)=2$. These values are obtained by invariably selecting one of the end vertices. The surprising result about the sawtooth poset is that consistently selecting a vertex adjacent to an end vertex *maximizes* the average number of queries.

Figure 8 shows the average number of queries divided by w for the sawtooth poset with w vertices. Here, the vertices are invariably selected in three different ways: an end vertex (the lower curve), a middle vertex (the oscillating curve), and a vertex adjacent to an end vertex (the upper curve). For large w, selecting the end vertex results in approximately 72.36% queries on average, as the approximation for $Q^s(w)$ given above shows.

Now consider creating an evaluative criterion based on the ideas of vertical middle and horizontal end points. Suppose a subset of unclassified vectors, $V=\{v^1, v^2, \ldots, v^p\}$ is given. Let $K_1(v^i)$ and $K_0(v^i)$ be the number of vectors that are concurrently classified when $f(v^i)$ equals 1 and 0, respectively. Invariably selecting one of the vectors that have the minimum $|K_1-K_0|$ value guarantees the minimum average number of queries 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 $\min|K_1-K_0|$. These 14 posets are positioned in row and column numbers $(1,1)$, $(1,2)$, $(1,3)$, $(1,4)$, $(1,6)$, $(1,8)$, $(2,2)$, $(2,6)$, $(2,9)$, $(3,5)$, $(5,7)$, $(6,2)$, $(9,1)$, and $(10,2)$ starting from the upper left corner of 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

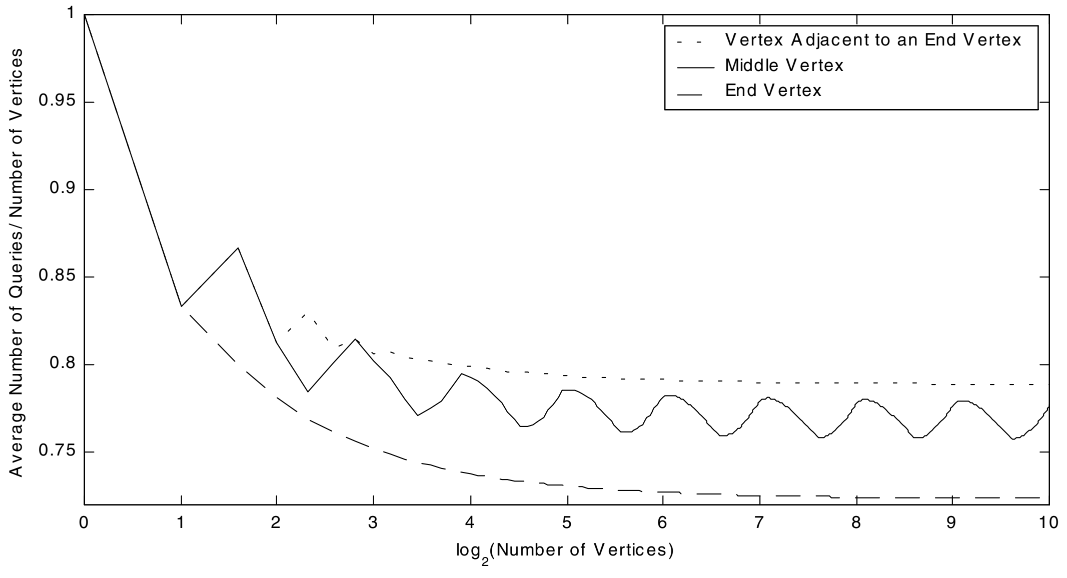

Figure 8 Average Query Complexity on the Sawtooth Poset

to 5. However, it is probably close to optimal since the larger posets eventually decompose into smaller posets. The next section addresses this issue further and discusses some other evaluative criteria.

3.5. Some Other Evaluative Criteria

It seems reasonable to attempt to classify as many vectors as possible for each query. The two criteria $\max(K_1(v)+K_0(v))$ and $\max(K_1(v)K_0(v))$ are consistent with this philosophy (see Judson 1999). They are extremely counterproductive to minimizing the average query complexity and should be avoided.

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. 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. The criterion $\max(K_1(v)K_0(v))$ ties the entire set of vectors. This shows how intuition can lead to poor and ambiguous choices.

There is also a logical explanation for why these two evaluative criteria are counterproductive. Vectors that are able to concurrently classify more vectors, are also more likely to be classified by others. Following this line of thought, the evaluative criterion $\min(K_1(v)+K_0(v))$ seems reasonable. This criterion is similar to $\min|K_1(v)-K_0(v)|$, but it does not satisfy the same optimality conditions for the inference problem when n is equal to 4.

Consider the poset in the 9th row and 1st column in Figure 6. Here, the five optimal vectors have the values $(K_1,K_0)=(2,4)$, or $(3,4)$, and the seven non-optimal vectors have the values $(1,12)$, $(4,2)$, $(5,2)$, $(1,7)$, or $(1,8)$. The criterion $\min|K_1(v)-K_0(v)|$ is equal to 1 for an optimal vector with the values $(3,4)$. It is therefore guaranteed to select an optimal vector for this poset. On the other hand, the criterion $\min(K_1(v)+K_0(v))$ is equal to 2 for optimal and non-optimal vectors with the values $(2,4)$ and $(4,2)$, respectively. Therefore, it may select one of the non-optimal vectors for this poset.

All the posets generated when n is equal to 3 are given in rows and columns $(1,1)$, $(1,2)$, $(1,3)$, $(1,4)$, $(1,5)$, $(1,6)$, $(1,7)$, $(1,8)$, $(2,2)$, $(2,3)$, $(3,1)$, and $(4,3)$, starting from the upper left corner of Figure 6. The criterion $\min|K_1(v)-K_0(v)|$ is optimal for these posets and their duals. Unfortunately, the two values K_1 and K_0 are not sufficient to construct an evaluative criterion that is optimal for all the posets generated when n is greater than 3, as Theorem 8 establishes.

THEOREM 8. *An evaluative criterion defined as a function of the two values K_1 and K_0 cannot be optimal for all the posets observable in the inference problem when n is greater than* 3.

PROOF: Let $z(K_1,K_0)$ denote a function of the parameters K_1 and K_0. Without loss of generality, suppose that an evaluative criterion is defined as $\min z(K_1,K_0)$. First, consider the poset in the 8th row and 6th column in Figure 6. In this poset, all the optimal vectors satisfy $(K_1,K_0)=(1,4)$, and four of the non-optimal vectors satisfy $(K_1,K_0)=(4,2)$. For the criterion $\min z(K_1,K_0)$ to select an optimal vector for this poset, the inequality $z(1,4)<z(4,2)$ has to hold. Similarly, the inequality $z(4,1)<z(2,4)$ is implied by the dual of this poset. Notice that this poset and its dual are both observable in the inference process when $n>3$.

Now, consider the poset in the 9th row and the 6th column in Figure 6. In this poset, all the optimal vectors satisfy $(K_1,K_0)=(2,4)$, and one of the non-optimal vectors satisfy $(K_1,K_0)=(1,4)$. For the criterion $\min z(K_1,K_0)$ to select an optimal vector for this poset, the inequality $z(2,4)<z(1,4)$ has to hold. Similarly, the inequality $z(4,2)<z(4,1)$ is implied by the dual of this poset. Notice that this poset and its dual are both observable in the inference process when $n>3$.

This leads to an impossibility as follows $z(1,4)<z(4,2)<z(4,1)<z(2,4)<z(1,4)$. In other words, no evaluative criterion defined as a function of the two parameters K_1 and K_0 can select optimal vectors for both of these two posets and their duals. □

Theorem 8 shows that the non-optimality of the criterion $\min|K_1-K_0|$ for some of the posets considered is not due to the criterion itself but rather due to lack of information. An optimal evaluative criterion for $n>4$, has to be based on more information than just the K_1 and K_0 values, but hopefully less than the entire poset. The objective $Q(n)$ assumes that the underlying function should be completely reconstructed. Suppose a limited number of queries are allotted. Then the objective should be changed and consequently the evaluative criterion should be modified. Figure 9 shows the average number of vectors remaining as a function of the number of queries performed for $n = 5$. Two evaluative criteria are used: one corresponding to a greedy algorithm, the other corresponding to the algorithm that sacrifices earlier to perform better in the end.

The greedy approach (the solid line in Figure 9) maximizes the instantaneous reduction in the average number of remaining vectors between the vector selections. The other approach (the dotted line in Figure 9) selects the vector which is the most frequent border vector among the remaining monotone Boolean functions. The latter approach draws its motivation from the fact that each border vector eventually has to be evaluated. Rather than relying on immediate gratification, it tends to sacrifice early in the inference process for the benefit of requiring fewer queries overall.

In Figure 9, the greedy approach corresponds to the curve that achieves the steepest instantaneous descent. After 6 queries, this approach loses its momentum. From there on, the other algorithm achieves a greater instantaneous descent. By the time both algorithms are finished, the greedy algorithm comes in second with an average of 13.9 queries for complete reconstruction. The algorithm based on maximizing the fraction border vectors, queries 13.6 vectors on the average.

In comparison to these two evaluative criteria, the criterion $\min|K_1 - K_0|$ queries 13.7 vectors on the average. However, the criterion $\min|K_1 - K_0|$ 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. Moreover, they both tie all the vectors in the chain poset, and the greedy approach ties all the vectors in the sawtooth poset.

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 1). Since there are about 10^7 monotone Boolean

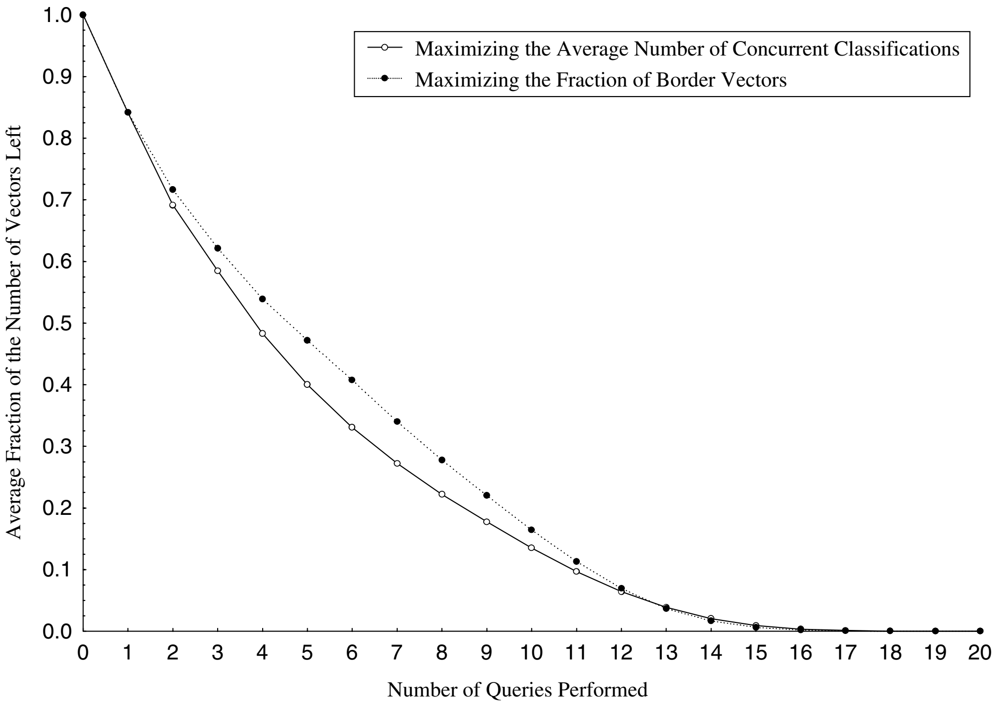

Figure 9 Classification Efficiency of Two Evaluative Criteria Over All Functions in M_5

functions for n equal to 6, and about 10^{12} for n equal to 7, these two criteria seem feasible for at most 6 variables with current storage capacities. In contrast, the criterion $\min|K_1 - K_0|$ may be computed for problems with up to about 20 variables, which involves about 1 million vectors.

3.6. An Illustration of the Criterion $\min|K_1 - K_0|$: Breast Cancer Diagnosis

To demonstrate how the inference process works using the evaluative criterion $\min|K_1 - K_0|$ consider one of the monotone Boolean functions inferred by interviewing a radiologist in Kovalerchuk et al. (1996). The inferred function that describes their "cancer subproblem" is defined as follows: $f(x) = x_1x_2 \vee x_3 \vee x_1x_5 \vee x_2x_5 \vee x_4x_5$, where $f(x) = 1$ if a tumor with the features described by x is highly suspicious for malignancy and 0 otherwise. Here x_i describes the i-th diagnostic feature which is 1 if it is "pro cancer" and 0 if it is "contra cancer". The individual features are defined as follows:

$x_1 =$ *Amount and volume of calcifications,*
$x_2 =$ *Shape and density of calcifications,*
$x_3 =$ *Ductal orientation,*
$x_4 =$ *Comparison with previous exam, and*
$x_5 =$ *Associated findings.*

It should be noted that x_1 and x_2 are also monotone Boolean functions that have to be inferred prior to the inference of f. The details of that decomposition are left out for the purpose of simplifying this illustration. The interested reader is referred to Kovalerchuk et al. (1996 and 2000a) for the details.

Figure 10 shows the sequence of queries and their answers for the cancer application, when the queries are selected based on the criterion $\min|K_1 - K_0|$. Below each query, the remaining unclassified vectors are

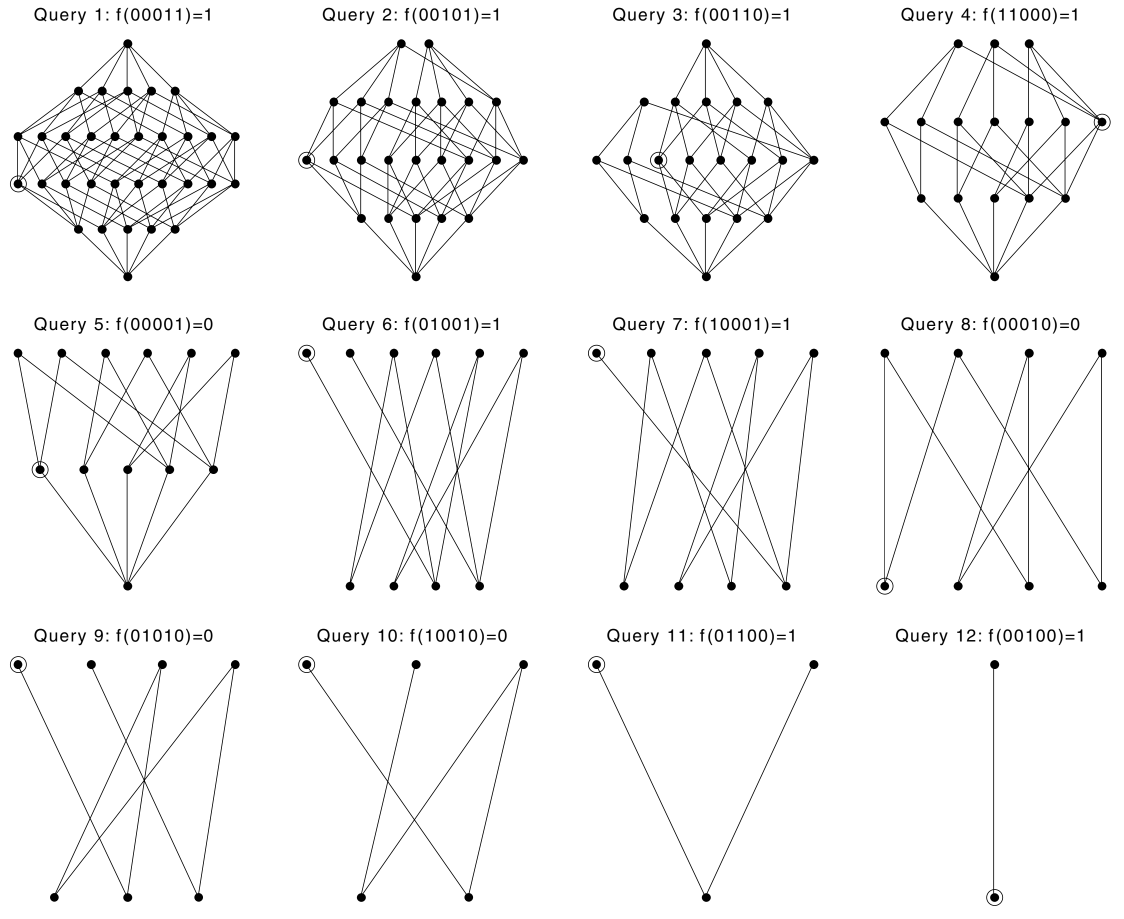

Figure 10 The Breast Cancer Inference Process Using the Criterion $\min|K_0$-$K_1|$

shown in the form of a poset with the selected vector circled. Initially, all of the vectors $\{0,1\}^5$ are unclassified. The vectors on the two middle layers all possess the same minimum value $|K_1 - K_0| = 4$. One of these vectors (00011) is arbitrarily selected for the first query. The radiologist is asked whether a tumor with "pro cancer" features in *comparison with previous examination* ($x_4 = 1$) and in *associated findings* ($x_5 = 1$), is considered highly suspicious for malignancy. After the radiologist answers yes ($f(00011) = 1$), the set of unclassified vectors is reduced and forms the poset shown below the 2nd query.

In this poset, several vectors possess the minimum $|K_1 - K_0|$ value of 0. The vector 00101 is one of them. The radiologist is asked whether a tumor with "pro cancer" features in *ductal orientation* ($x_3 = 1$) and in *associated findings* ($x_5 = 1$), is considered highly suspicious for malignancy. After the radiologist answers yes ($f(00101) = 1$), the set of unclassified vectors is further reduced and forms the poset shown below the 3rd query.

This process continues for an additional 10 queries shown in Figure 10. After these 12 queries, the value of the diagnosis function f is known for all the vectors v in $\{0,1\}^5$. That is, the function is completely reconstructed after 12 queries out of a possible maximum of 32 queries. It should also be noted that this function has the set of 8 border vectors: {11000, 00100, 10001, 01001, 00011, 10010, 01010, 00001}. Therefore, the minimum possible number of queries is 8.

The previous sections provided the theoretical motivation for using the evaluative criterion $\min|K_1 - K_0|$ and demonstrated how this criterion is used in practice. It was shown that this criterion minimizes the average query complexity for $n = 1, 2, 3, 4$ and resulted in 12 queries in a breast cancer diagnosis problem. In the next section, a framework for further analyzing the efficiency of this approach is developed.

4. A Sampling Framework for Comparing the Inference Algorithms

Often, a random sample of test cases is generated to estimate parameters such as the average 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. When some test cases are more likely to end up in the sample, 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., its *inclusion probability*) is known, then an unbiased estimator can be obtained. The estimators that are presented next in Section 4.1 can be found in most textbooks (e.g., Thompson 1992) on sampling. The algorithm used to randomly generate monotone Boolean functions which allows for computing these estimators is presented in Section 4.2.

4.1. An Unbiased Estimator for the Average Case Complexity

Consider the finite universe of fixed quantities $\{\varphi_1, \varphi_2, \ldots, \varphi_{\Psi(n)}\} = \{\varphi(A, f) : f \in M_n\}$ 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:

$$\hat{\tau} = \sum_{i=1}^{\nu} \frac{\varphi_i}{p_j},$$

where $\{\varphi_1, \varphi_2, \ldots, \varphi_\nu\}$ are the quantities corresponding to the set of ν 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 $\hat{\tau}$ 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:

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

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

$$\hat{\mu} = \frac{\hat{\tau}}{\Psi(n)}, \tag{5}$$

and a corresponding unbiased estimate of its variance is:

$$\widehat{Var}(\hat{\mu}) = \frac{\hat{Var}(\hat{\tau})}{\Psi(n)^2}.$$

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 discussed further in the next section.

4.2. Randomly Generating 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 (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 into the trap of generating functions from a distribution that deviates significantly from the uniform distribution. As an illustrative example, consider the algorithm GENERATE-MBF as outlined in Figure 11. 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 1/2 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:

```
GENERATE-MBF(n)
U ← $\{0,1\}^n$
while U ≠ {},
  v ← SELECT-AT-RANDOM(U, 1/|U|)
  f(v) ← SELECT-AT-RANDOM({0,1},1/2)
  if f(v) = 0
    W ← {w: w ∈ U, w ≼ v}
    f(w) ← 0, ∀ w ∈ W
    U ← U - W
  else
    W ← {w: w ∈ U, w ≽ v}
    f(w) ← 0, ∀ w ∈ W
    U ← U - W
return f
```

Figure 11 The Algorithm Used to Generate Monotone Boolean Functions Randomly Without Inclusion Probabilities

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

Korshunov's asymptotic to $\Psi(n)$ (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.

Instead of the exact inclusion probability of the ONE function, a lower bound 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. In situations where nothing is known a priori about the number of queries, it is also desirable for the inclusion probabilities to be close to uniform. This 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. 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 12 illustrates how the pair of monotone Boolean functions are created from a non-monotone Boolean 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 lower left of the figure. The lower units of f uniquely define the second target monotone function f_G shown on the lower right.

The algorithm GENERATE-MBF-P(n, p) shown in Figure 13 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.

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 as 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 <u>or</u> the greatest function,

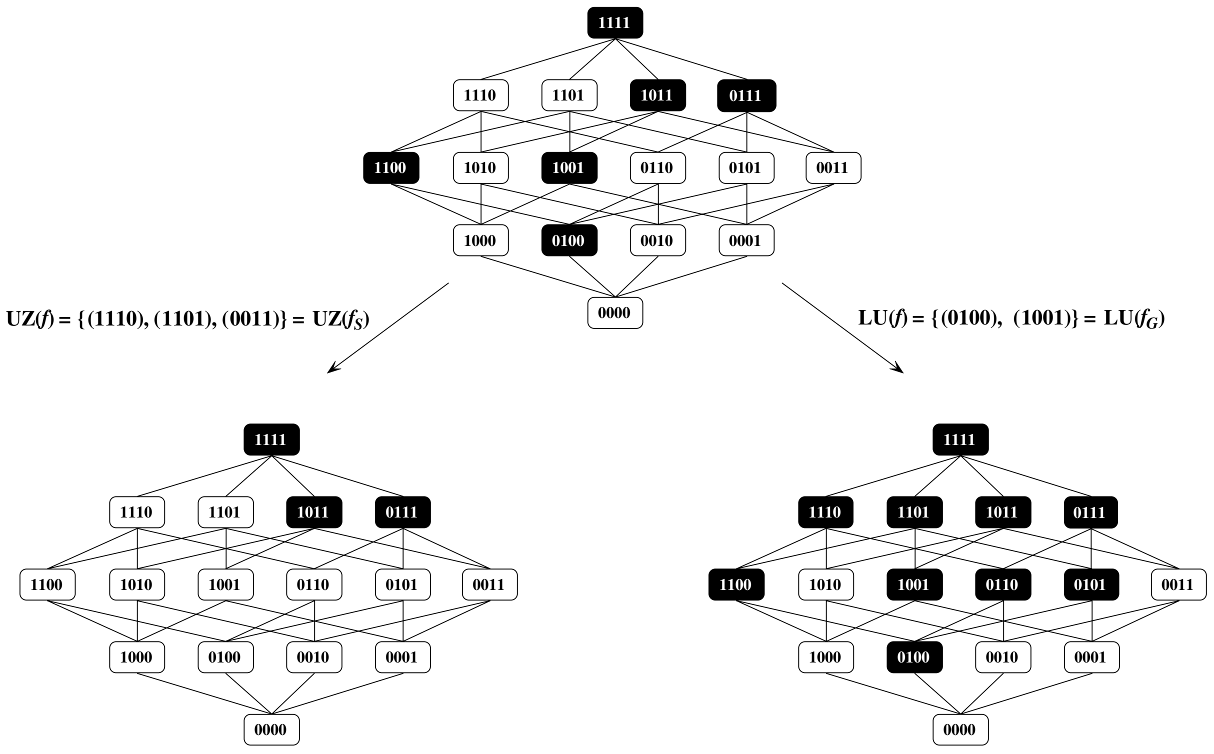
1111
1110
1101
1011
0111
1100
1010
1001
0110
0101
0011
1000
0100
0010
0001
0000
UZ(f) = {(1110), (1101), (0011)} = UZ(f_S)
LU(f) = {(0100), (1001)} = LU(f_G)

Figure 12 Two Monotone Boolean Functions Created from a Non-Monotone Boolean Function Defined on $\{0,1\}^4$

```
GENERATE-MBF-P(n,p)
1  G ← {}, S ← {}
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 ∈ G}
6      if W = {}
7        G ← G ∪ {v}
8      else
7        G ← G - W
8    else
9      W ← {w: v ≽ w, w ∈ S}
10     if W = {}
11       S ← S ∪ {v}
12     else
13       S ← S - W
14 return G, S
```

Figure 13 The Algorithm Used to Generate Monotone Boolean Functions Randomly With Inclusion Probabilities

given by:

$$\Pr\{F_S = f \vee F_G = f\} = \Pr\{F_S = f\} + \Pr\{F_G = f\} - \Pr\{F_S = f \wedge F_G = f\}, \quad (7)$$

where

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

and similarly,

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

and finally

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

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

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

The reasoning behind this particular definition of $p(v)$ is provided immediately after this example. Suppose the general function shown in the top of Figure 12 was generated as a result. Consider computing the inclusion probabilities for the function f_G shown on the bottom right of Figure 12. 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:

$$\begin{aligned}q_1 &= p(0100)p(1001)(1-p(1010))(1-p(0011))\\ &\quad\times(1-p(1000))(1-p(0010))(1-p(0001))\\ &\quad\times(1-p(0000)) + (1-p(1010))(1-p(0011))p(0100)\\ &\quad\times p(1100)p(1001)p(0110)p(0101)p(1110)p(1101)\\ &\quad\times p(1011)p(0111)p(1111) - p(1111)p(1110)p(1101)\\ &\quad\times p(1011)p(0111)p(1100)p(1001)p(0110)p(0101)\\ &\quad\times p(0100)(1-p(1010))(1-p(0011))(1-p(1000))\\ &\quad\times(1-p(0010))(1-p(0001))(1-p(0000))\\ &= \left(\frac{5}{42}\right)\left(\frac{1}{2}\right)^3\left(\frac{37}{42}\right)^3\left(\frac{167}{168}\right) + \left(\frac{1}{2}\right)^6\left(\frac{5}{42}\right)\left(\frac{37}{42}\right)^4\\ &\quad\times\left(\frac{167}{168}\right) - \left(\frac{167}{168}\right)^2\left(\frac{37}{42}\right)^7\left(\frac{1}{2}\right)^6\left(\frac{5}{42}\right)\\ &\approx 0.01047\end{aligned}$$

If, for example, GENERATE-MBF-P was executed 20 times, the inclusion probability is equal to $q_{20} \approx 1 -$

$(1-0.01047)^{20} \approx 0.1898$ 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 $\Psi_k(n)$ 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 $\Psi_k(n)/\Psi(n)$. 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 $\Psi_k(n)$.

$$\Psi_k(n) = \begin{cases} 1, & k=0 \\ \Psi(n-1), & k=1 \\ \Psi(n)/2, & k=n/2,\ n \text{ is even} \\ \Psi(n)-\Psi(n-1), & k=n-1, \\ \Psi(n)-1 & k=n \end{cases} \qquad (8)$$

Equation (8) indicates that computing $\Psi_k(n)$ is just as hard as computing $\Psi(n)$. We computed some of the exact values for $\Psi_k(n)$, which are given in Table 3. The solid curves in Figure 14 correspond to $\Psi_k(n)/\Psi(n)$ for $n=1,2,\ldots,6$. 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:

$$\frac{\Psi_k(n)}{\Psi(n)} = \frac{e^{\alpha(n,k)(k-n/2)}}{1+e^{\alpha(n,k)(k-n/2)}}. \qquad (9)$$

Since $\alpha(n,k)$ is a function of n and k, no information is lost or gained by the transformation. The problem of approximating $\Psi_k(n)/\Psi(n)$ is merely transformed into the equivalent problem of approximating the function $\alpha(n,k)$. However, an approximate relationship between the known $\alpha(n,k)$ values is more transparent in Table 4 than between the known $\Psi_k(n)$ values given in Table 3. Solving (9) for $\alpha(n,k)$ gives,

$$\alpha(n,k) = \frac{\ln(\Psi(n)-\Psi_k(n)) - \ln(\Psi_k(n))}{n/2-k},$$
$$\text{for } n>0,\ n\neq 2k, \text{ and } k=0,1,\ldots,n.$$

Furthermore, any value for $\alpha(n,k)$ in equation (8) will give the correct value of 1/2 for $\Psi_k(n)/\Psi(n)$ when $n=2k$. From equation (8), it is known that $\Psi_0(n)=1$, and therefore the following equation holds:

$$\alpha(n,0) = \frac{\ln(\Psi(n)-1)}{n/2}, \text{ for } n>0. \qquad (10)$$

Equation (8) also provides the relationship $\Psi_1(n) = \Psi(n-1)$ which further implies the following equation:

$$\alpha(n,1) = \frac{\ln(\Psi(n)-\Psi(n-1)) - \ln(\Psi(n-1))}{n/2-1},$$
$$\text{for } n>2.$$

To compute the values for $\alpha(n,0)$ and $\alpha(n,1)$ only $\Psi(n)$ is needed. For n up to and including 8, the exact values of $\Psi(n)$ are known and were given in Table 1. For n greater than 8 Korshunov's approximation given in equation (1) can be used. However, to compute $\alpha(n,k)$ for $k>1$, a generalization is needed. Based on the observed values in Table 4, it seems reasonable to use

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

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

Figure 14 shows the approximated curves, for $n=1,2,\ldots,6$, resulting from using (10) and generalizing with $\alpha(n,k)\approx\alpha(n-1,k-1)$ and $\alpha(n,k)=\alpha(n-$

Table 3 **Some Exact Values of $\Psi_k(n)$**

$\Psi_k(n)$	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$
$k=0$	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

Table 4 **Some Exact Values of $\alpha(n,k)$**

$\alpha(n,k)$	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$	$n=7$	$n=8$
$k=0$	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		

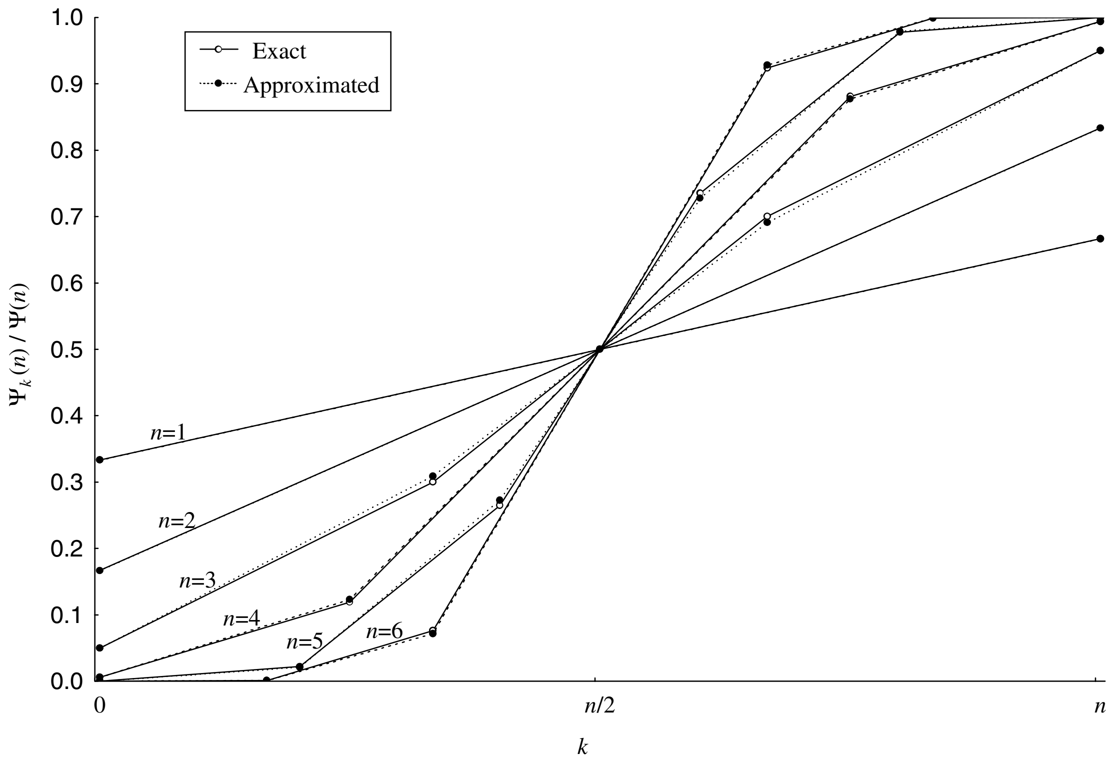

Figure 14 Comparing the Exact Values to the Approximations of $\Psi_k(n)/\Psi(n)$

$k, k)$ for $k = 1, 2, \ldots, \lceil n/2 \rceil - 1$. The approximations based on $\alpha(n, k) \approx \alpha(n-1, k-1)$ are probably worse for larger k. Approximations of $\Psi_k(n)/\Psi(n)$ that are off by a few percentage points will not have a major effect on the experiments described in Section 5.3. Fortunately, these approximations are only needed for n up to 11 (i.e., for k up to 5). 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, $\Psi(11) > 10^{144}$ using Korshunov's asymptotic given in equation (1).

5. Empirical Comparisons of the Inference Algorithms

The purpose of this section is to highlight the differences and similarities of various approaches to the inference problem. Section 5.1 provides an overview of the main three existing algorithms and some of their properties. Section 5.2 demonstrates details of these three algorithms on the breast cancer diagnosis problem described in Section 3.5. The query complexities of the three existing approaches and the evaluative criterion $\min|K_1 - K_0|$ are then compared in Section 5.3.

5.1. Some Preexisting Algorithms

Only someone that knows the function ahead of time, i.e., an all knowing *Teacher*, can achieve the minimum number of queries for every single monotone Boolean function. That is, the following equation holds: $\varphi(\mathit{Teacher}, f) = m(f)$, $\forall f \in M_n$. 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 queries. That is, the following inequality always holds: $m(f) \leq \varphi(A, f)$, $\forall f \in M_n$.

It turns out that it is possible to achieve fewer or the same number of queries 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. The key property of the Hansel chains is that once the function values are known for the chains of length k, the function values are unknown for at most two vectors in each of the next length $k+2$. Proof of this property can be found in both Hansel (1966) and Sokolov (1982).

When n is odd, the shortest chains contain two vectors each, and there are a total of $\binom{n}{\lfloor n/2\rfloor}$ chains. Therefore, the maximum number of queries used by an algorithm searching the Hansel chains in increasing size is $2\binom{n}{\lfloor n/2\rfloor}$. When n is even, there are $\binom{n}{n/2}-\binom{n}{n/2+1}$ chains of length one, and $\binom{n}{n/2+1}$ chains of length greater than one. Therefore, the maximum number of queries used by an algorithm searching the Hansel chains in increasing size is $\binom{n}{n/2}-\binom{n}{n/2+1}+2\binom{n}{n/2+1}=\binom{n}{n/2}+\binom{n}{n/2+1}$. That is, the following inequality holds:

$$\begin{aligned}\varphi(Hansel, f) &\le \max_{f\in M_n} m(f)\\ &= \binom{n}{\lfloor n/2\rfloor}+\binom{n}{\lfloor n/2\rfloor+1},\\ &\qquad \forall f\in M_n. \qquad (11)\end{aligned}$$

The algorithm described in Sokolov (1982) is also based on the Hansel chains. It 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 $\lfloor \log_2(n+1)\rfloor+1$ queries, while Hansel's algorithm needs at least $\binom{n}{\lfloor n/2\rfloor}$ queries. 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 $\{v: |v|=n/2 \text{ or } n/2+1\}$ and $m(f)=\binom{n}{n/2}+\binom{n}{n/2+1}$. 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 $|w^2|=\lceil 3n/4\rceil > n/2$ and $n/2+1$. Therefore, the following inequality holds:

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

In an attempt to provide a unified efficiency testing platform, Gainanov (1984) proposed to evaluate algorithms based on the number of queries 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 $n+1$ queries. 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:

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

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.

5.2. Illustration of the FIND-BORDER, Hansel's, and Sokolov's Algorithms on the Breast Cancer Diagnosis Application

For the purpose of contrasting the different algorithms, the monotone Boolean function for the breast cancer diagnosis application described in Section 3.5 will be used. Please recall that the function was defined as follows: $f(x)=x_1x_2\vee x_3\vee x_1x_5\vee x_2x_5\vee x_4x_5$. The inference processes for the subroutine FIND-BORDER, Hansel's algorithm, and Sokolov's algorithm are given in Figures 15, 16, and 17, respectively. These figures are similar to Figure 10 shown in Section 3.6 for the criterion $\min|K_1-K_0|$.

The FIND-BORDER algorithm may select vectors that are already classified. The posets in Figure 15

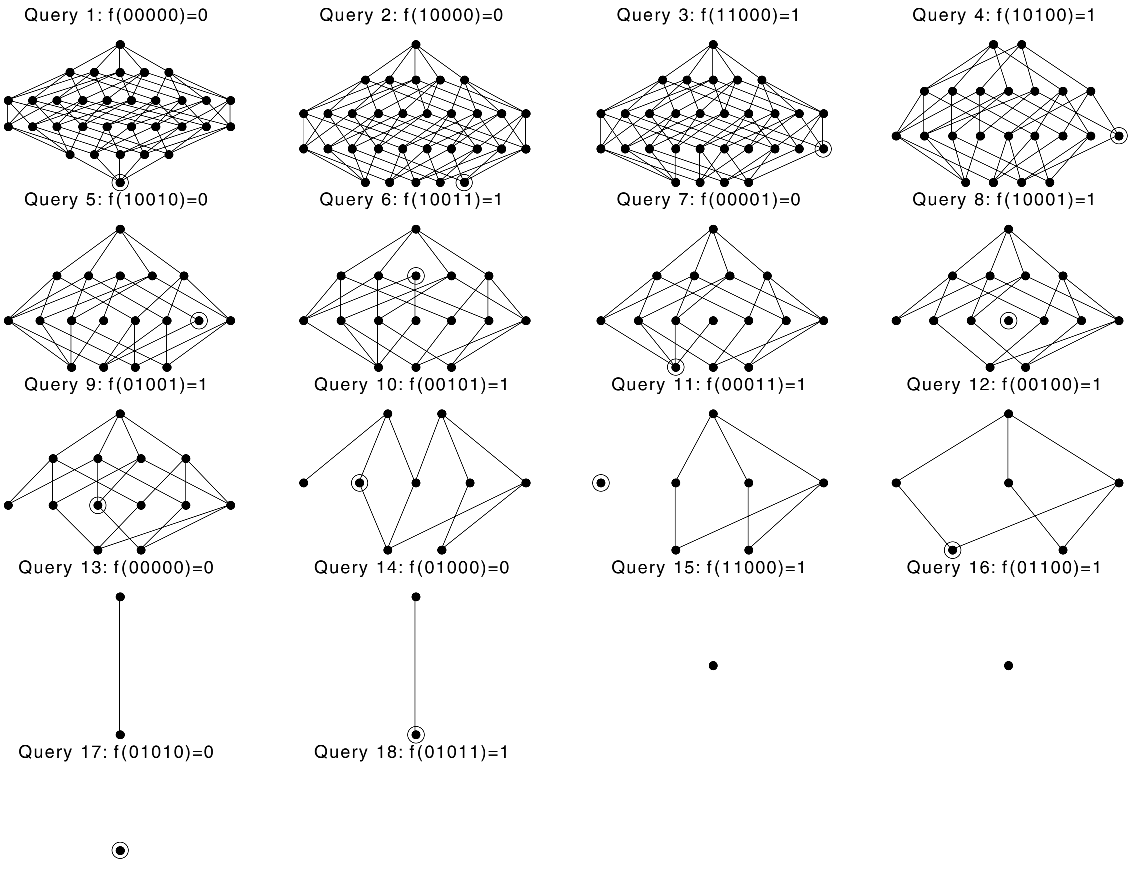

Figure 15 The Breast Cancer Inference Process Using the FIND-BORDER Algorithm

only contain the unclassified vertices, and therefore some of the posets do not have a circle (queries 13, 15, 16 and 18). In a practical situation these queries can be answered by the function inferred thus far and is therefore not posed to the radiologist. The total number of actual queries used by this algorithm is therefore, $18 - 4 = 14$. The first vector fed to FIND-BORDER is 00000, and the first sequence of queries are (00000, 10000, 11000, 10100, 10010, 10011). The second vector fed to FIND-BORDER is 00001, and the second sequence of queries are (00001, 10001, 01001, 00101, 00011). Similarly, the third and fourth sequence of queries are (00100, 00000), and (01000, 11000, 01100, 01010, 01011).

Hansel's algorithm searches the 10 Hansel chains in increasing size given by: (00011, 10011), (00101, 10101), (00110, 00111), (01001, 01101), (01010, 01011), (00001, 10001, 11001, 11101), (00010, 10010, 11010, 11011), (00100, 10100, 10110, 10111), (01000, 01100, 01110, 01111), (00000, 10000, 11000, 11100, 11110, 11111). As noted in Section 4.1, the number of queries per chain is guaranteed to be less than or equal to 2. Figure 16 shows the Hansel query process where the respective number of queries are here 1, 1, 1, 1, 1, 2, 2, 1, 0, 1, for a total of 11 queries.

Sokolov's algorithm searches the 10 Hansel chains in decreasing size: (00000, 10000, 11000, 11100, 11110, 11111), (01000, 01100, 01110, 01111), (00100, 10100, 10110, 10111), (00010, 10010, 11010, 11011), (00001, 10001, 11001, 11101), (01010, 01011), (01001, 01101), (00110, 00111), (00101, 10101), (00011, 10011). It performs a binary search within each of these chains. Figure 17 shows this process, where the respective number of queries per chain are here 3, 2, 2, 2, 2, 2, 1, 0, 0, 1, for a total of 15 queries.

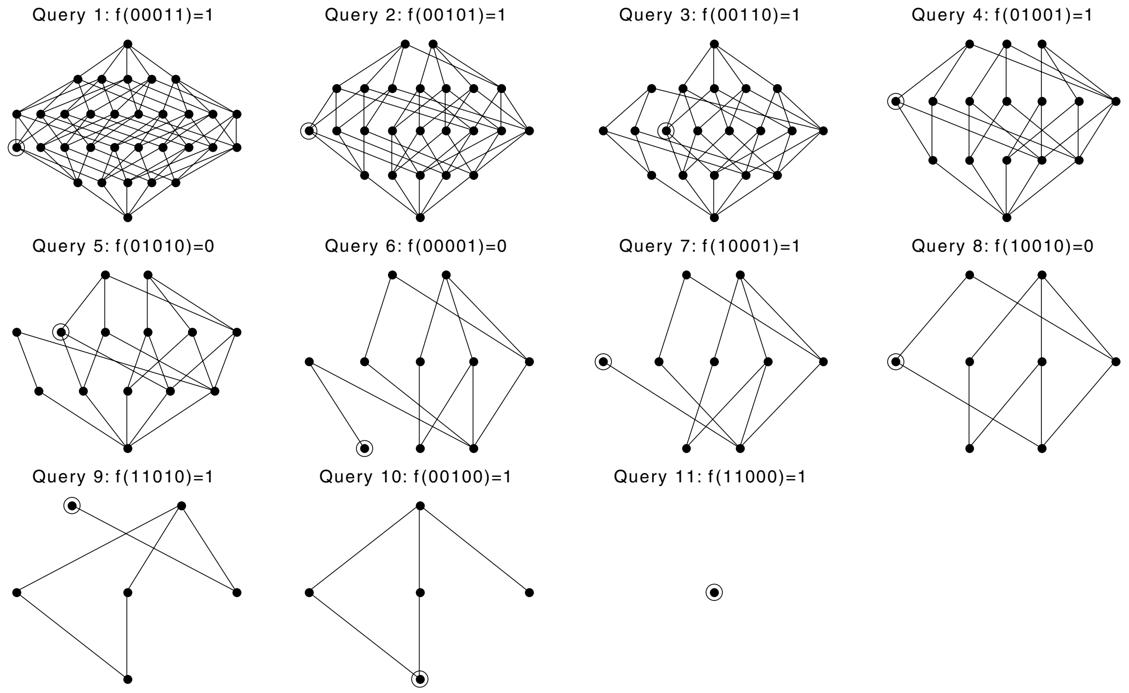

Figure 16 **The Breast Cancer Inference Process Using Hansel's Algorithm**

The number of queries performed by each of the four methods Hansel, $\min|K_1 - K_0|$, FIND-BORDER, and Sokolov for the breast cancer diagnosis application were 11, 12, 14, and 15, respectively. The eight border vectors for this function are {11000, 00100, 10001, 01001, 00011, 10010, 01010, 00001}. That is, the number of queries used by *Teacher* is 8, since it represents the minimum number of queries required to infer the function. The number of queries per border vector are $11/8 = 1.375$, $12/8 = 1.5$, $14/8 = 1.75$, and $15/8 \approx 1.875$ for the Hansel, $\min|K_1 - K_0|$, FIND-BORDER and Sokolov algorithms, respectively. This represents the algorithms' performance on only one of the 7,581 monotone Boolean functions defined on $\{0, 1\}^5$, and is far from conclusive. The next section provides the results from more extensive comparisons.

5.3. 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 $\min|K_1 - K_0|$ which was described in Section 3.4 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.

Figure 18 shows the number of queries for each of the inference algorithms distributed over all monotone Boolean functions in the set M_5. The average number of queries are labeled with small squares on the x-axes of the respective histograms. In this figure, the average number of queries for an algorithm A is given by:

$$\sum_{f \in M_5} \frac{\varphi(A, f)}{\Psi(5)}.$$

For the Teacher, the evaluative criterion, Hansel's algorithm, Sokolov's algorithm, and the algorithm based on the subroutine FIND-BORDER, the averages are 9.4, 13.7, 14.5, 16.0, and 18.3, respectively. Here, the evaluative criterion is probably close to the optimal.

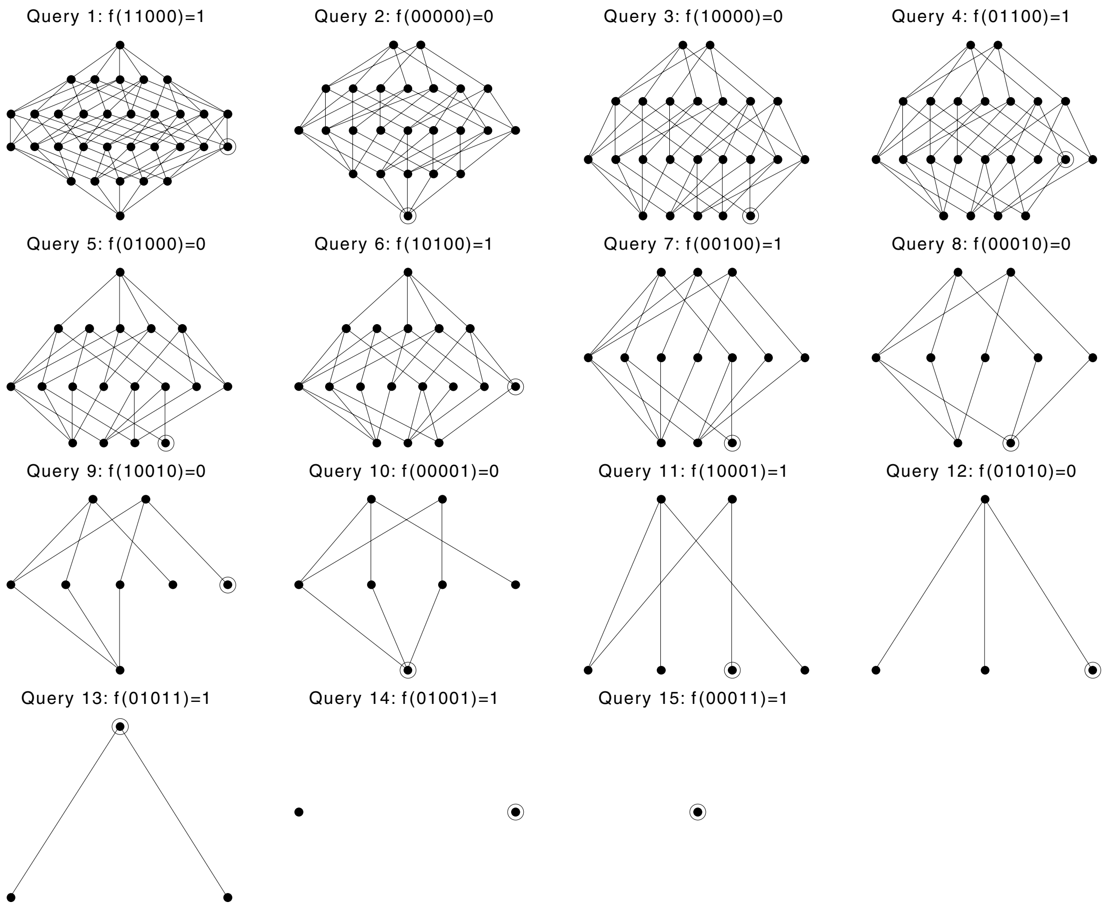

Figure 17 The Breast Cancer Inference Process Using Sokolov's Algorithm

Another nice characteristic of the evaluative criterion is that it is the most consistent of all the algorithms. It performs between 10 and 18 queries for 99.6% of the monotone Boolean functions. In contrast, the algorithm based on the subroutine FIND-BORDER is the least consistent with between 8 and 25 queries for 99.6% of the monotone Boolean functions.

The results in Figure 19 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 $\Psi_{|v|}(n)/\Psi(n)$ constructed in Section 4.2 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 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 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 queries is normalized by the maximum possible number of queries 2^n so that the magnitude of all the averages in Figure 19 are not overshadowed by the large values

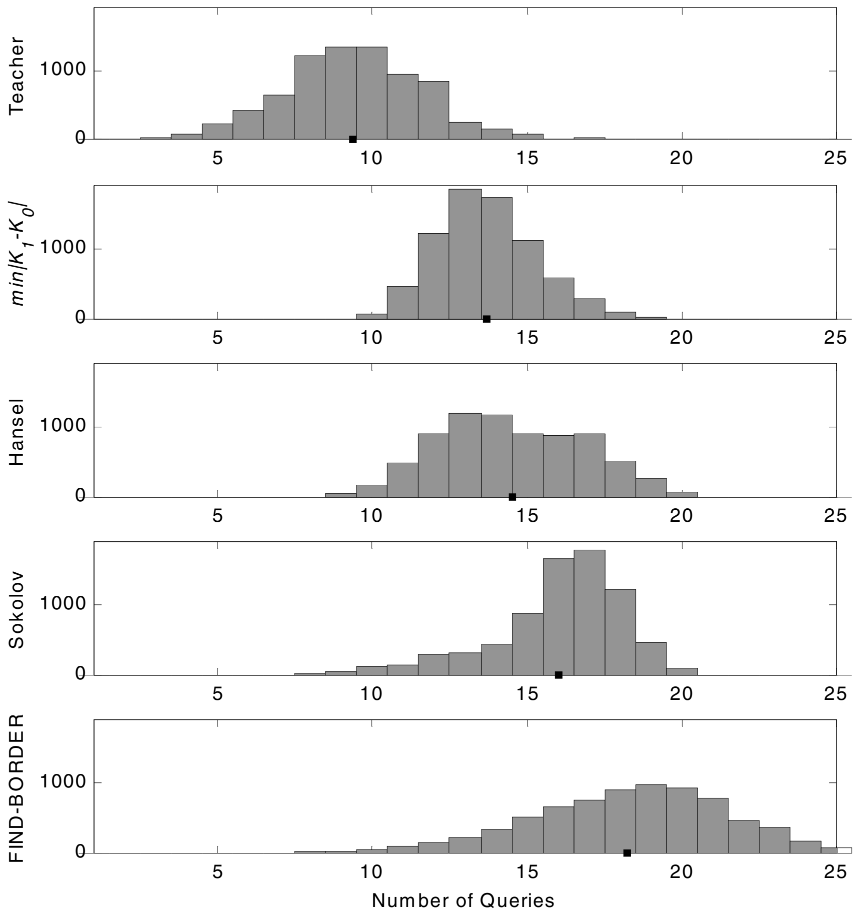

Figure 18 The Query Complexities on the Set $\{0,1\}^5$

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 19 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 monotonicity 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 queries for every single function. Therefore, it is expected that a few extra queries are required on the average. Since the heuristic based on the evaluative criterion $\min|K_1 - K_0|$ achieves the minimum average number of queries 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 19 paints a clear picture of how the pre-existing 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 differences between Hansel and Sokolov, and Sokolov and FIND-BORDER seem to increase with n, the corresponding difference in the average number of queries increases at rate greater than exponentially with n.

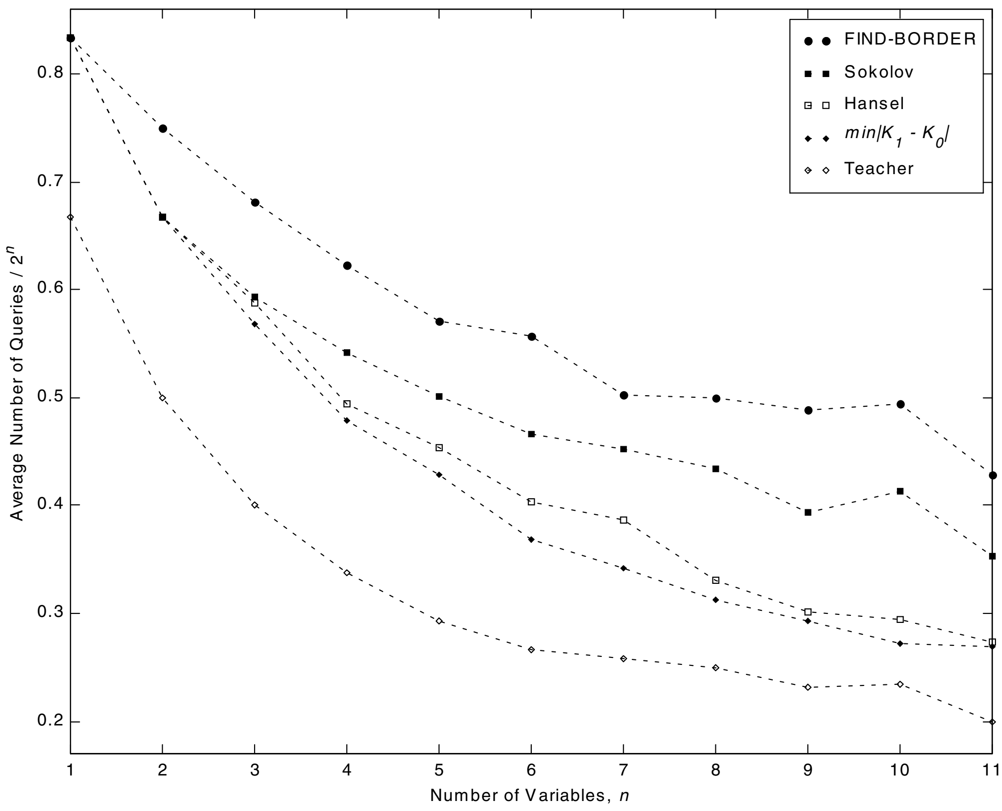

Figure 19 The Average Query Complexities on the Set $\{0, 1\}^n$

The difference between the curves for Hansel and "Teacher" decreases as n increases. However, the evaluative criterion $\min|K_1 - K_0|$ seemed to perform about 2% better than Hansel's algorithm. This difference is especially clear in Figure 19 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.

Figure 20 shows the number queries each of the inference algorithms uses per border vector on the average. That is, for each inference algorithm A, the quantity given by:

$$\sum_{f \in M_n} \frac{\varphi(A, f)}{\varphi(Teacher, f)},$$

is computed for $n = 1, 2, \ldots, 11$. Again, the average values for $n = 1, 2, \ldots, 5$ are exact values, and the average values for $n = 6, 7, \ldots, 11$ are estimated. The same ranking as for the average query complexity holds for $n = 1, 4, 5, \ldots, 11$. An algorithm using the subroutine FIND-BORDER seems to perform about 2 queries per border vector for larger values of $n(> 4)$, while the criterion $\min|K_1 - K_0|$ seems to level off close to 1.1 queries per border vector for $n = 9, 10$ and 11. Even in situations where one knows that the number of border vectors tends to be small, the criterion $\min|K_1 - K_0|$ still is the best alternative, unless $n = 2$ or 3 when Sokolov's algorithm is the best alternative.

6. Concluding Remarks

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 (and average query complexity per border vector for $n > 3$). The subroutine, here

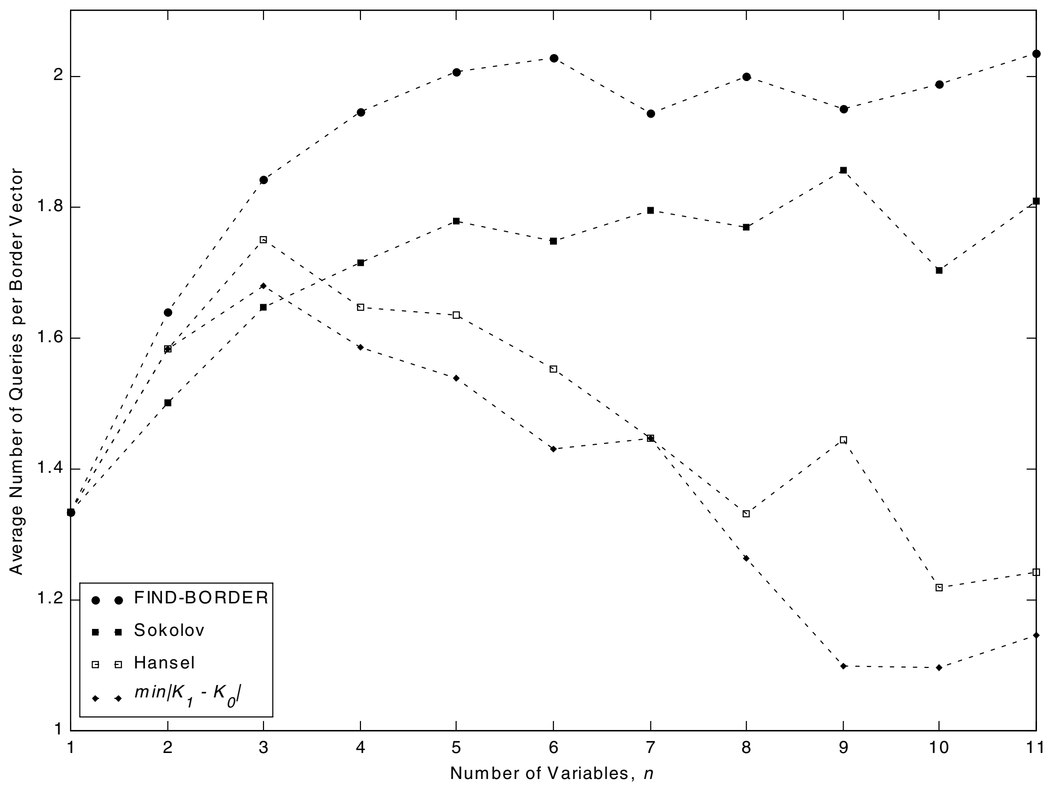

Figure 20 The Average Query Complexities per Border Vector on the Set $\{0,1\}^n$

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 equal query costs holds.

The optimal solutions paved the way for the evaluative criterion $\min|K_1 - K_0|$ that probably would not have been developed (due to its non-intuitive nature) without the consultation of the optimal solutions. This near optimal 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.

In some applications, where additional properties may be known about the underlying monotone Boolean function, it may be beneficial to use a modified objective. For example, the application may put a limit on the number of lower units, shifting the focus of the optimal vertices from the vertical center to the vertical edge of the poset. It would also be interesting to see how the optimal dialogue with the oracle changes as the equal query cost assumption is modified, as in the keyword search described in Section 3.2.

The problem of minimizing the average query complexity is extended to that of inferring a monotone Boolean function from a stochastic oracle in Torvik and Triantaphyllou (2001a), and to that of inferring a pair of nested monotone Boolean functions from a pair of deterministic oracles in Torvik and Triantaphyllou (2001b). For a unified and more detailed version of these articles the interested reader is referred to Torvik (2002).

Acknowledgments

The authors would like to thank Dr. Ilya Shmulevich at the Tampere University of Technology in Finland, Dr. Dean Judson at the US Census Bureau, and the anonymous reviewers whose thoughtful comments enhanced the quality of this research. The authors also gratefully acknowledge the support from Office of Naval Research (ONR) Grants N00014-95-1-0639 and N00014-97-1-0632.

References

Ben-David, A. 1992. Automatic generation of symbolic multiattribute ordinal knowledge-based DSSs: methodology and applications. *Decision Sciences* **23** 1357–1372.

Bioch, J. C., T. Ibaraki. 1995. Complexity of identification and dualization of positive Boolean functions. *Information and Computation* **123** 50–63.

Bloch, D. A., B. W. Silverman. 1997. Monotone discriminant functions and their applications in rheumatology. *Journal of the American Statistical Association* **92** 144–153.

Boros, E., P. L. Hammer, J. N. Hooker. 1994. Predicting cause-effect relationships from incomplete discrete observations. *SIAM Journal on Discrete Mathematics* **7** 531–543.

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** 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. Über zerlegungen von zahlen durch ihre grössten gemeinsamen teiler. *Festschrift Hoch. Brauhnschweig u. ges Werke II* 103–148 (in German).

Eiter, T., G. Gottlob. 1995. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing* **24** 1278–1304.

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

Fellegi, I. P., A. B. Sunter. 1969. A theory for record linkage. *Journal of the American Statistical Association* **64** 1183–1210.

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** 176–181.

Hansel, G. 1966. Sur le nombre des foncions Booleenes monotones de n variables. *C. R. Acad. Sci. Paris* **262** 1088–1090 (in French).

Horvitz, D. G., D. J. Thompson. 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.* Master's thesis, Dept. of Mathematics, University of Nevada, Reno, NV.

Judson, D. H. 2001. A partial order approach to record linkage. *Federal Committee on Statistical Methodology Conference, Arlington, VA, November 14–16.*

Korshunov, A. D. 1981. On the number of monotone Boolean functions. *Problemy Kibernetiki* **38** 5–108 (in Russian).

Kovalerchuk, B., E. Triantaphyllou, A. S. Deshpande. 1996. Interactive learning of monotone Boolean functions. *Information Sciences* **94** 87–118.

Kovalerchuk, B., E. Triantaphyllou, J. F. Ruiz, V. I Torvik, E. Vitayev. 2000a. The reliability issue of computer-aided breast cancer diagnosis. *Computers and Biomedical Research* **33** 296–313.

Kovalerchuk, B., E. Vityaev. 2000b. *Data Mining in Finance*. Kluwer Academic Publishers, Boston, MA.

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** 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** 76–88.

Shmulevich, I. 1997. *Properties and Applications of Monotone Boolean Functions and Stack Filters.* Ph.D. dissertation, Dept. of Electrical Engineering, Purdue University, West Lafayette, IN.

Sokolov, N. A. 1982. On the optimal evaluation of monotonic Boolean functions. *U.S.S.R. Computational Mathematics and Mathematical Physics* **22** 207–220.

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

Torvik, V. I., E. Triantaphyllou. 2001a. Guided inference of stochastic monotone Boolean functions. Working paper, Dept. of Psychiatry, University of Illinois at Chicago.

Torvik, V. I., E. Triantaphyllou. 2001b. Guided inference of nested monotone Boolean functions. Working paper, Dept. of Psychiatry, University of Illinois at Chicago.

Torvik, V. I. 2002. *Data Mining and Knowledge Discovery: A Guided Approach Based on Monotone Boolean Functions.* Ph.D. dissertation, Dept. of Industrial Engineering, Louisiana State University, Baton Rouge, LA.

Valiant, L. G. 1984. A theory of the learnable. *Communications of the ACM* **27** 1134–1142.

Ward, M. 1946. Note on the order of the free distributive lattice. *Bulletin of the American Mathematical Society* **52** 423.

Wiedemann, D. 1991. A computation of the eight dedekind number. *Order* **8** 5–6.

Winkler, W. 1995. Matching and record linkage. B. G. Cox, D. A. Binder, B. N. Chinnappa, eds. *Business Survey Methods*. John Wiley & Sons, New York.

Accepted by John N. Hooker, Jr; received March 2000; revised June 2001, September 2001; accepted November 2001.