Guided Inference of Stochastic Monotone Boolean Functions

Vetle I. Torvik[1] and Evangelos Triantaphyllou[2,] *

Dept. of Industrial and Manufacturing Systems Engineering

3128 CEBA Building, Louisiana State University

Baton Rouge, LA 70803-6409

1 Email: vtorvik@lsu.edu, Webpage: http://cda4.imse.lsu.edu/torvik/index.html

2 Email: trianta@lsu.edu, Webpage: http://imse.lsu.edu/vangelis

* Corresponding Author

Last Revision: October 29, 2001

Abstract

This paper addresses the problem of efficiently reconstructing monotone Boolean functions via membership queries to a stochastic oracle. An incremental maximum flow algorithm is developed to find a maximum likelihood monotone Boolean function after each query answer. This algorithm reduces the computational complexity to $O(V^2)$ per query, from the general maximum flow algorithms complexity of $O(V^3)$, where V is the query domain. The main goal of reducing the number of queries needed to make the maximum likelihood ratio converge to 1 is achieved by using certain evaluative criteria to select queries. Guiding the inference process by selecting queries from the inferred border vectors takes $O(V)$ time per query and reduces the average query complexity dramatically. The motivation for guided inference in the presence of errors is further strengthened by the fact that the number of queries also tends to increase faster than exponentially with the oracle's error rate.

Keywords: monotone Boolean functions, membership queries, maximum likelihood, incremental maximum flow algorithm, query selection criterion.

1. Introduction

Monotonicity is a natural property in a wide variety of applications. For example, the better grades a student has, the more likely he or she is to be accepted into a particular college, with all other factors constant. If a personal computer tends to crash when it runs a particular web browser and word processor simultaneously, then it will probably also crash when it, in addition, boots up an image viewer, and so on. In applications, the monotonicity property is usually easy to identify due to its intuitive nature. This is perhaps its most important feature when human interaction is involved, since people tend to make very good use of knowledge they understand.

This paper focuses on situations where the monotone function and its n variables are Boolean (i.e., take on two values, say 0 and 1). This does not necessarily limit the application domain as Kovalerchuk *et al.* (1995) establishes that any function can be represented by a sequence of monotonically non-increasing and non-decreasing Boolean functions. Recent literature contains a plethora of fields where monotone Boolean functions appropriately capture the phenomenon under study. Such diverse fields include, but are not limited to, social workers 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)), social sciences (Judson (1999)), finance (Kovalerchuk and Vityaev (2000)), and record linkage in databases (Judson (2001)).

In practice, a great deal of effort is put into the process of inferring a function underlying a phenomenon. Software is tested to establish its reliability, medical experiments are performed to establish diagnostic criteria for diseases, etc. This inference process generally involves gathering and analyzing data. Gathering the data often involves some sort of labor that far outweighs the computations used to analyze the data, in terms of cost. The main objective in this paper is, therefore, to minimize the cost associated with gathering the data, as long as it is computationally feasible.

Monotone Boolean functions lay the ground for a simple question-asking strategy where it may be easy to pinpoint questions whose answers make incomplete knowledge more general or stochastic knowledge more accurate. Due to the underlying monotonicity, a learning algorithm that actively chooses its observations (i.e., is guided) may significantly increase the learning rate, as a passive learner might not receive the relevant pieces of information. Therefore, it is highly desirable not only to be able to pose questions (or queries), but to pose "smart" questions.

The main problem addressed in this paper is how to identify these "smart" questions in order to completely and efficiently infer a monotone Boolean function defined on at most n variables. The monotone Boolean function can be thought of as a phenomenon, such as breast cancer or a computer crash, together with a set of n Boolean predictors or variables. It is assumed that the learning algorithm has access to an oracle that provides a 0-1 function value in response to each query. The oracle can be thought of as an entity that knows the underlying monotone Boolean function. In practice it may take the shape as a human expert or as the outcome of a tasks such as performing experiments. It is further assumed that the oracle misclassifies each vector (associated with a query) with a fixed probability $q \in (0,½)$. That is, for a given monotone Boolean function f, the oracle returns 1 for vector v with probability $p(v) = f(v)(1 - q) + (1 - f(v))q$, and 0 with probability $1- p(v)$. It is assumed that the oracle is not misleading the inference process and is better at classifying the vectors than completely random guessing, hence the oracle's misclassification probability is assumed to be less than ½.

The case when the oracle is deterministic (i.e., its misclassification probability q is equal to 0), has been studied extensively in the literature. A comparative study of existing algorithms for this problem can be found in Torvik and Triantaphyllou (2001a), and an extension to a pair of nested monotone Boolean functions can be found in Kovalerchuk *et al.* (1996) and Torvik and Triantaphyllou (2001b).

The stochastic inference problem involves estimating the misclassification parameter q as well as reconstructing the underlying function f. In this paper, these two tasks are based on a maximum likelihood framework. The monotone Boolean function that is the most likely to match the underlying function, given the observed queries, is referred to as the *inferred function* and is denoted by f^*. Associated with function f^* is the estimated misclassification probability which is denoted by q^*.

The inference process consists of two steps that are repeated successively. First a vector is submitted to the oracle as a query. After a vector's function value is provided by the oracle, both q^* and f^* may have to be updated. These two steps are repeated until the likelihood of the inferred function f^* matching the underlying function f is high relative to the likelihood of any of the other monotone Boolean functions matching f. That is, the underlying function is considered completely reconstructed when the maximum likelihood ratio for the inferred function $\lambda(f^*)$ reaches a value that is close to 1.

The paper is organized as follows. Before the details of the inference problem and the related methodology are described in Section 3, some background information on monotone Boolean functions and related inference problems is provided in Section 2. The conclusion given in Section 5 is based on the experimental results presented in Section 4.

2. Background Information

This section builds the framework used for the core methodology used in Sections 3 and 4. Section 2.1 provides some formal notation and definitions, and also some useful properties related to monotone Boolean functions. The focus of this paper is the fixed misclassification probability model of the oracle. This model, in addition to some more general stochastic models, is presented in Section 2.2. Some of the related model objectives and optimization procedures are also briefly described. Some of the literature pertaining to guided inference in the presence of errors is reviewed in Section 2.3.

2.1 Some Properties of Monotone Boolean Functions

Let $\{0,1\}^n$ denote the set of Boolean vectors defined on n Boolean variables. A deterministic 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, denoted by $v \preceq w$, if and only if (iff) $v_i \leq w_i$ for $i = 1, 2, ..., n$. Here v_i (w_i) denotes the i-th element of vector v (w, respectively). Similarly, a vector v is said to *succeed* another vector w, iff $v_i \geq w_i$ for $i = 1, 2, ..., n$. When v precedes (or succeeds) w, and the two vectors are distinct (i.e., $v \neq w$), then the vector v is said to *strictly precede* (or *strictly succeed*, respectively) w, denoted by $v \prec w$ (or $v \succ w$, respectively). A monotone Boolean function f is called *non-decreasing* if $f(v) \leq f(w)\ \forall\ (v, w)$: $v \preceq w$, and *non-increasing* if $f(v) \geq f(w)\ \forall\ (v, w)$: $v \preceq w$. This paper focuses on non-decreasing functions, which are referred to as just *monotone*, as analogous results hold for non-increasing functions.

Figure 1 shows a sample *partially ordered set* (or *poset* for short). In general, posets can be formed by a set of vectors together with the precedence relation $\preceq$. Posets can be viewed as directed graphs where each vertex corresponds to a vector and a directed edge from vertex v to vertex w, represents the precedence relation $v \preceq w$. When drawing a poset, the edge directions are often omitted since the graph is acyclic and so all the directions can be forced upwards on a page by ordering the vertices by layers. The poset in Figure 1 is shown with the directions because they are useful in the maximum flow algorithms described in Section 3. For the purpose of reducing storage and simplifying the visualization of posets, redundant relations, i.e., ones that are transitively implied by other relations, are also omitted, as in Figure 1.

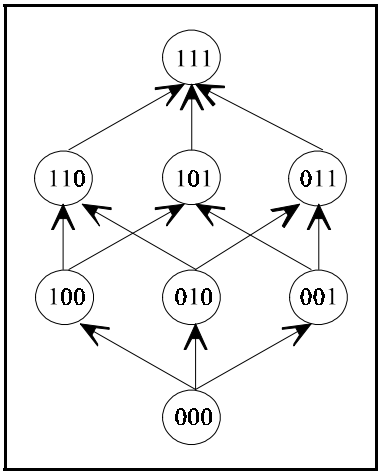

Figure 1. The poset formed by $\{0,1\}^3$ and the $\preceq$ relation.

A vector v^* is called an *upper zero* of a function f, if $f(v^*) = 0$ and $f(v) > f(v^*)\ \forall\ v$: $v^* \prec v$. Similarly, a vector v^* is called a *lower unit* of a function f, if $f(v^*) = 1$ and $f(v) < f(v^*)\ \forall\ v$: $v \prec v^*$. Lower units and upper zeros are also referred to as *border vectors*. For any monotone Boolean function f, the set of lower units $LU(f)$, and the set of upper zeros, $UZ(f)$ are unique and either one of these two sets uniquely identifies f. Boolean functions are often written in Disjunctive Normal Form (DNF) or Conjunctive Normal Form (CNF). For any monotone Boolean function f there is a one-to-one relationship between its lower units $LU(f)$ and its terms in DNF as follows:

$$f(v_1, v_2, \ldots, v_n) = \bigvee_{w \in LU(f)} \left(\bigwedge_{i:\, w_i=1} v_i \right).$$

Similarly, there is a one-to-one relationship between the upper zeros $UZ(f)$ of a monotone Boolean function f, and its clauses in CNF as follows:

$$f(v_1, v_2, \ldots, v_n) = \bigwedge_{w \in UZ(f)} \left(\bigvee_{i:\, w_i=0} v_i \right).$$

For example, the monotone Boolean function defined by its lower units {110, 101} can be written in DNF as $v_1v_2 \vee v_1v_3$. The corresponding upper zeros are {011, 100} and CNF is $v_1(v_2 \vee v_3)$. Since the lower units and upper zeros are unique to a monotone Boolean function, so are its representations in DNF and CNF. The set of monotone

functions defined on $\{0,1\}^n$ is denoted by M_n. For example, the set of monotone Boolean functions defined on at most 2 variables written in DNF is given by $M_2 = \{F, v_1v_2, v_1, v_2, v_1 \vee v_2, T\}$. Here, the uniform functions F and T are defined as $f(v) = 0 \; \forall \; v \in \{0,1\}^n$, and $f(v) = 1 \; \forall \; v \in \{0,1\}^n$, respectively.

The number of monotone Boolean functions defined on $\{0,1\}^n$ is denoted by $\Psi(n)$. All of the known values for $\Psi(n)$ are given in Table 1. Wiedeman (1991) employed a Cray-2 super computer for 200 hours in order to compute the value for n equal to 8. This gives a flavor of the complexity of computing the exact number of monotone Boolean functions. For larger values of n the best known asymptotic is due to Korshunov (1981):

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

Table 1. History of Monotone Boolean Function Enumeration for n = 1,2, ..., 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)

Many monotone Boolean functions are of similar form. For example, by permuting the variables in the function $v_1 \vee v_2v_3$, the two functions $v_2 \vee v_1v_3$ and $v_3 \vee v_1v_2$ are obtained. Also, by switching the roles of the operations $\vee$ and $\wedge$ in the function $v_1v_2v_3$, the function $v_1 \vee v_2 \vee v_3$ is obtained. To formally describe this similarity, define the following two posets: $P_z(f) = (\{v \in \{0,1\}^n: f(v) = z\}, \preceq)$, for z = 0 and 1. 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 $v^1 \rightarrow v^2$ and $w^1 \rightarrow w^2$, then $v^1 \preceq w^1$ iff $v^2 \preceq w^2$, $\forall \; v^1, w^1 \in P^1$ and $v^2, w^2 \in P^2$. Two monotone Boolean functions f and g are called *similar* either if $P_0(f)$ is isomorphic to $P_0(g)$ (and $P_1(f)$ is isomorphic to $P_1(g)$), or if $P_0(f)$ is isomorphic to $P_1^d(g)$ (and $P_1(f)$ is isomorphic to $P_0^d(g)$). The first isomorphism is obtained by permuting the variables, and the second isomorphism is obtained by switching the roles of the operations $\vee$ and $\wedge$. The sets of similar monotone Boolean functions of M_3 are: $\{F, T\}$, $\{v_1v_2v_3, v_1 \vee v_2 \vee v_3\}$, $\{v_1v_2, v_1v_3, v_2v_3, v_1 \vee v_2, v_1 \vee v_3, v_2 \vee v_3\}$, $\{v_1v_2 \vee v_1v_3, v_1v_2 \vee v_2v_3, v_1v_3 \vee v_2v_3, v_1 \vee v_2v_3, v_2 \vee v_1v_3, v_3 \vee v_1v_2\}$, $\{v_1, v_2, v_3\}$, and $\{v_1v_2 \vee v_1v_3 \vee v_2v_3\}$.

A useful fact about similar functions is that certain properties are preserved within the subset. For example, the average case behavior of the inference process is equivalent for similar functions as long as certain randomness precautions are taken. The fact that 6 non-similar functions represent the entire class of functions M_3, will reduce the computational burden of the associated experiments performed in Section 4 by a factor of $\Psi(3)/6 \approx 3.33$.

2.2 Stochastic Models for Monotone Boolean Functions

Suppose a set of observed vectors $V = \{v^1, v^2, ..., v^k\}$ is given. For a given number of queries m, let $m_z(v)$ be the number of times the oracle classified vector v as z (for $z = 0$ and 1, and $v \in V$). Associated with a monotone Boolean function f, the number of errors it performs on the set of observations is given by:

$$e(f) = \sum_{i=1}^{k} (f(v^i)m_0(v^i) + (1-f(v^i))m_1(v^i)).$$

It is assumed that the oracle misclassifies each query with a fixed probability $q \in (0, ½)$. That is, for a given monotone Boolean function f, the oracle returns 1 for vector v with probability $p(v) = q\times(1 - f(v)) + (1 - q)\times f(v)$, and 0 with probability $1- p(v)$. An estimate for the fixed misclassification probability associated with function f is simply the ratio of the number of errors over the total number of classifications performed, restricted by the maximum allowable value of ½ (i.e., $min\{e(f)/m, ½\}$). Note that the probability q is actually required to be strictly less than ½, but for practical purposes this estimate is sufficient when the observed error ratio is greater than or equal to ½.

If the sampled values are considered fixed, their joint probability distribution function can be thought of as the likelihood of function f matching the underlying function as follows:

$$L(f) = q^{e(f)}(1 - q)^{m - e(f)}.$$

The likelihood value of a particular monotone Boolean function decreases exponentially as more observations are added and therefore this value is generally very small. However, the likelihood ratio given by:

$$\lambda(f^*) = \frac{L(f^*)}{\sum_{f\in F(V)} L(f)},$$

measures the likelihood of a particular function f^* relative to the likelihood of all possible monotone Boolean functions $F(V)$, defined on the set of vectors V. Note that when the set of vectors V is equal to $\{0,1\}^n$, then the set of all possible monotone Boolean functions $F(V)$ is equal to M_n.

The goal of the maximum likelihood problem is to find a monotone Boolean function $f^*\in F(V)$, so that $L(f^*) \geq L(f) \; \forall \, f \in F(V)$. Assuming that the misclassification probability $q \in (0,½)$, this problem is equivalent to identifying a monotone Boolean function f^* that minimizes the number of errors $e(f^*)$ (Boros *et al.* (1995)). Note that if q can take on values greater than ½, then the maximum likelihood solution may maximize the number of errors, as demonstrated by Boros *et al.* (1995). In this paper, error maximization is avoided by restricting q to be less than ½.

It should be noted that an error minimizing solution with $q^* \leq ½$ always exists. One of the two uniform monotone Boolean functions T, or F satisfies this restriction as follows:

$$q(T) = \frac{e(T)}{m} = \frac{\sum_{i=1}^{k} m_0(v^i)}{\sum_{i=1}^{k} m_0(v^i)+m_1(v^i)}, \quad q(F) = \frac{e(F)}{m} = \frac{\sum_{i=1}^{k} m_1(v^i)}{\sum_{i=1}^{k} m_0(v^i)+m_1(v^i)}.$$

Therefore, $q(T) + q(F) = 1$ and $0 \leq q(T), q(F) \leq 1$. As a result, $q^* \leq min\{q(T), q(F)\} \leq ½$. Note that this does not

necessarily imply that one of the two functions T or F are error minimizing, but rather establishes the fact that an error minimizing function will indeed satisfy $q^* \leq ½$.

The error minimization problem can be converted into an integer maximization problem as follows:

$$\min e(f) =$$

$$\min \sum_{i=1}^{k} (f(v^i)m_0(v^i) + (1-f(v^i))m_1(v^i)) =$$

$$\min(-\sum_{i=1}^{k} (f(v^i)(m_1(v^i) - m_0(v^i)) + \sum_{i=1}^{k} m_1(v^i)).$$

Since the term $\sum_{i=1}^{k} m_1(v^i)$ is constant, it can be removed from the optimization objective. Furthermore, maximizing a particular objective function is equivalent to minimizing the negative of that objective function, resulting in the following simplified integer optimization problem:

$$\max \sum_{i=1}^{k} f(v^i)(m_1(v^i) - m_0(v^i))$$

subject to $f(v^i) \leq f(v^j)\ \forall\ v^i, v^j \in V: v^i \preceq v^j$, and

$$f(v^i) = 0 \text{ or } 1.$$

This problem is known as a maximum closure problem, which can be converted into a maximum flow problem as described in Section 3.1 (see also Picard (1976)). The most efficient algorithms developed for the maximum flow problem use the idea of preflows developed by Karzanov (1974). For example, the lift-to-front algorithm (e.g., Cormen *et al.* (1997)) takes $O(V^3)$ time. The fact that this problem can be solved in polynomial time is a nice property of the single q parameter model.

For two dimensional problems (i.e., $V \subset \mathbb{R}^2$), the minimum number of errors can also be guaranteed via a dynamic programming approach (Bloch and Silverman (1997)). This approach is also applicable when $(V, \preceq)$ forms a planar poset (i.e., a poset that can be drawn in a plane without crossing lines). The posets considered in this paper are of the form $(\{0,1\}^n, \preceq)$, and for n greater than 2 they are unfortunately not planar.

A more complex error model can potentially maintain as many parameters as the size of the domain V. That is, each vector v may have an associated unique parameter $p(v)$. In this case, minimizing the weighted least squares:

$$\min \sum_{i=1}^{k} (\bar{p}(v^i) - p(v^i))(m_1(v^i) + m_0(v^i))$$

subject to $p(v^i) \leq p(v^j)\ \forall\ v^i, v^j \in V: v^i \preceq v^j$,

where $$\bar{p}(v^i) = \frac{m_1(v^i)}{m_1(v^i) + m_0(v^i)}, \text{ for } i = 1, 2, \ldots, k,$$

yields a maximum likelihood solution (Robertson *et al.* (1988)). This is a hard optimization problem, and several algorithms have been developed to solve it optimally and near optimally. The Pooled Adjacent Violators Algorithm (PAVA) by Ayer *et al.* (1955) only guarantees optimality when $(V, \preceq)$ forms a chain poset (also referred to as a simple order). The Min-Max algorithm developed by Lee (1983) and the Isotonic Block Class with Stratification (IBCS)

algorithm by Block *et al.* (1994) guarantee optimality for the general poset but both algorithms can potentially consume exponential time. For other posets such as the simple tree order, the matrix order (Bloch and Silverman, 1997) and aligned orders (Boros *et al.* 1994), polynomial algorithms do exist. Unfortunately, no polynomial algorithm for the general poset was found in the literature.

In addition to the full parametric model, there are models of intermediate parametric complexity. One example is the logistic regression model with non-negativity constraints on its parameters, as used for record linkage in databases by Judson (2001). A monotone decision tree approach can be found in Makino *et al.* (1999), and a sequential monotone rule induction approach can be found in Ben-David (1992 and 1995).

It should be noted that the single parameter error model considered in this paper is somewhat restrictive. However, the goal is to efficiently uncover the underlying monotone Boolean function and not necessarily come up with accurate estimates for the errors. The fact remains that the error minimizing monotone Boolean function is also a maximum likelihood solution, as long as the errors occur according to a distribution that belongs to the exponential family (Robertson *et al.* (1988)). In other words, for a given set of observations the inferred function is the same for a very large class of error models. The error estimates are only used to measure the confidence (in terms of the maximum likelihood ratio) in having inferring the correct function. A more accurate estimate of the maximum likelihood ratio may require a substantial increase in computational complexity, as for the full parametric model described above.

2.3. Existing Approaches to Guided Inference

The problem of guided inference in the presence of stochastic errors is referred to as sequential design of experiments in the statistics community. The field of optimal experiment design (Federov (1972)) contains various optimality criteria that are applicable in a sequential setting. The most common vector selection criterion is based on instantaneous variance reduction. Other selection criteria, such as the maximum information gain used in MacKay (1992) and Tatsuoka (1999), have been studied. However, no guided inference studies using a maximum likelihood framework were found in the literature.

The theory of optimal experiment design is the most extensive for simple regression models (Federov (1972)). Fortunately, efficient guided inference for more complex models have been studied. Cohn *et al.* (1996) considered guided inference of feed forward neural networks. However, a sound theory has not been established. In fact, the same article reported a convergence problem for which a partial remedy was introduced in Cohn (1995).

3. The Methodology Involved in the Stochastic Guided Inference Problem

This section develops the details of the core methodologies that will be used for the stochastic guided inference problem. Section 3.1 presents a detailed overview of the a maximum flow algorithm for the general error minimization problem. This algorithm is used as a basis for the incremental version developed in Section 3.2. A sample dataset is used to illustrate the different aspects of these two algorithms, as well as the various likelihood

computations provided in Section 3.3. The goal of the paper is to define a vector selection criterion to guide the inference of stochastic monotone Boolean functions. Various criteria are further discussed in Section 3.3.

3.1 Error Minimization via Maximum Flow Computations

As mentioned earlier, the error minimizing (and consequently the maximum likelihood) problem can be solved as a maximum flow problem. General purpose maximum flow algorithms often maintain a flow graph in addition to a so-called residual graph. The residual graph signifies how much additional flow is allowed along the edges, and is the focus of the algorithm's computations, while the flow graph is simply the solution to the problem. For the error minimizing problem considered in this paper, the flow graph is not needed since the optimal monotone Boolean function and the associated estimated misclassification probability can be found directly from the residual graph. Therefore, the flow graph is omitted in the following algorithms, while the concept of a flow is used to describe algorithmic features.

The initial residual graph for the error minimizing problem can be constructed using the algorithm CONSTRUCT-RESIDUAL-GRAPH shown in Figure 2 (see also Picard (1976)). The original vectors V together with the so-called source and sink vertices, labeled s and t, respectively, make up the set of vertices in the residual graph. The set of capacitated directed edges is constructed in three steps as follows.

In lines 1-3, edges are added for each pair of vectors $v \prec w \in V$. Here redundant precedence relations (i.e., relations where another vector u exists for which $v \prec u \prec w$) are omitted. The capacities along these edges can be thought of as infinite since they are used to allow free flow along preceding vertices. For practical purposes these capacities only have to be sufficiently large to be able to handle the maximum flow warranted. The fact that the maximum flow will never exceed M, given by:

$$M = min\{\sum_{v:\ m_1(v) > m_0(v)} (m_1(v) - m_0(v)),\ \sum_{v:\ m_1(v) < m_0(v)} (m_0(v) - m_1(v))\},$$

is evident from the construction of the rest of the capacitated directed edges. Therefore, capacities of any quantity greater than or equal to M, is sufficient for finding the maximum flow. However, for the purpose of finding the optimal monotone Boolean function f^* during the last execution of the algorithm FLOW-INCREASE shown in Figure 6, and during the execution of the algorithm MAX-FLOW-INCREMENT1 shown in Figure 8, $M + 1$ is used.

In lines 4-6, edges are added from the source vertex s to each vector v with a majority of 1-valued observations (i.e., $m_1(v) > m_0(v)$) with a capacity of $m_1(v) - m_0(v)$. Similarly, in lines 7-8, an edge is added to the sink vertex t with a capacity of $m_0(v) - m_1(v)$, from each vector v with a majority of 0-valued observations (i.e., $m_1(v) < m_0(v)$). Please note that the vectors with the same number of 0 and 1 valued observations (i.e., $m_1(v) = m_0(v)$) are the only vertices not directly connected to either the sink or the source.

```
CONSTRUCT-RESIDUAL-GRAPH(V, m₁, m₀)

1 for each v in V
2   for each u ≺ v in V
3     c(v,u) ← M+1
4 for each v in V
5   if m₁(v) > m₀(v)
6     c(s,v) ← m₁(v) - m₀(v)
7   elseif m₁(v) < m₀(v)
8     c(v,t) ← m₀(v) - m₁(v)
9 V ← V + {s,t}
```

Figure 2. The algorithm used to construct the initial residual graph from the observed data.

For the purpose of illustrating the graph construction process consider the sample of observations for the set of vectors $\{0,1\}^3$ given in Table 2. Each row in the table gives the observed values of a particular vector. For example, vector 100 was observed a total of 9 times of which 4 times it was 1-valued and 5 times it was 0-valued, making the value of $m_1(100) - m_0(100)$ equal to -1.

The vectors and their associated $m_1(v) - m_0(v)$ values together with their irredundant precedence relations are shown as a graph in Figure 3. This graph is used as a basis for constructing the residual graph for the maximum flow problem. The directed edges of the graph can be thought of as infinitely capacitated, while a capacity of $M + 1 = min\ \{1+2+1, 3+2+2+1\} + 1 = 5$ is sufficient for practical purposes.

Table 2. Sample data.

v	$m_1(v)$	$m_0(v)$	$m_1(v) - m_0(v)$
111	0	1	-1
110	3	5	-2
101	4	1	3
011	3	1	2
100	4	5	-1
010	2	0	2
001	3	3	0
000	1	0	1

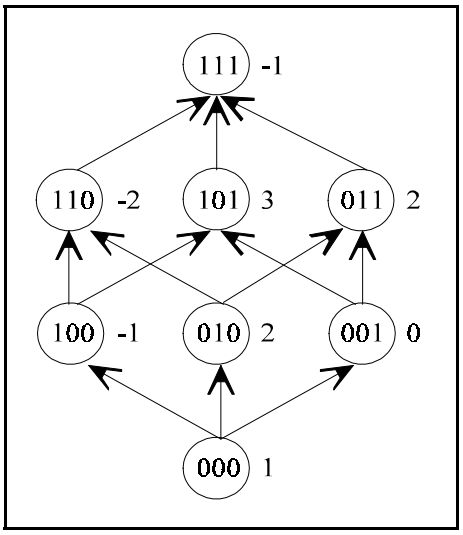

Figure 3. The graph used to construct the residual graph.

For the residual graph shown in Figure 4, these capacities are assigned to 6. This value is deliberately assigned a unit greater than the current set of observations dictate, since the illustrations in Section 3.2 include

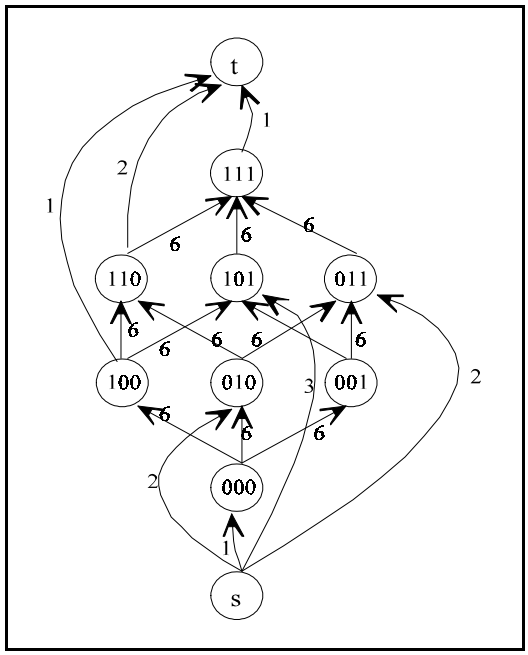

Figure 4. The residual graph constructed from the sample data.

different scenarios where a single observation is added. When working with an incremental problem where the m_0 and m_1 quantities are uncertain, a safe value for the capacities is simply the total number of observations when inference is terminated, which is 36 for the sample data.

The vertices {000, 010, 101, 011} all have a majority of 1-valued observations and their respective $m_1(v)$ - $m_0(v)$ values are given in Table 2 as 1, 2, 3 and 2, respectively. Edges are, therefore, added from the source vertex *s* to each of these vertices, with respective capacities of 1, 2, 3 and 2, to the graph in Figure 4. The vertices {100, 110, 111} all have a majority of 0-valued observations and their respective $m_1(v)$ - $m_0(v)$ values are given in Table 2 as -1, -2, and -1. Edges are, therefore, added from each of these vertices to the sink vertex *t*, with respective capacities of 1, 2, and 1, to the graph in Figure 4.

The vertex labeled 001 has the same number of 0-valued as 1-valued observations and is therefore not directly connected to either the sink or the source vertex. In fact, to find the maximum flow for the graph given in Figure 4, this vertex can be omitted. However, since the value of $m_1(001)$ - $m_0(001)$ may change as more observations are added, it will be kept. For the general incremental problem the $m_1(v)$ - $m_0(v)$ values may change from zero to a non-zero value (and vice-versa) several times. This is more likely to occur in the beginning of the inference process, and for a greater misclassification probability *q*. Regardless of what the value of *q* is, successively adding the corresponding vertex to (and removing it from) the residual graph is unnecessary and inefficient.

The Ford-Fulkerson (1962) maximum flow algorithm iteratively finds paths from the source vertex *s* to the sink vertex *t*. As an example of a path from *s* to *t*, consider the one going via 010 and 110 in Figure 4. For the general case, identifying such a path is not trivial, and an $O(V^2)$ method called a Breadth-First-Search (e.g., Cormen *et al.* (1997)) is often used. The algorithm BREADTH-FIRST-SEARCH(G,s,t) shown in Figure 5, performs Breadth-First-

```
BREADTH-FIRST-SEARCH(G,s,t)

1  e ← 0
2  Q ← s
3  r(s) ← infinite %larger than max c
4  for each v in V
5    f(v) ← 0
6    r(v) ← c(s,v)
7    e ← e + m1(v)
8  while Q ≠{}
9    v ← Q[1]
10   f(v) ← 1
11   Q ← Q - v
12   e ← e - m1(v) + m0(v)
13   for each u in adj(v)
14     if f(u) = 0
15       Π(u) ← v
16       if u = t
17         terminate
18       Q ← {Q,u}
19       f(u) ← ½
20       r(v) ← min{c(u,v),r(u)}
```

Figure 5. The algorithm used to traverse the residual graph.

Search modified to accommodate the maximum flow problem. The algorithm traverses a graph G starting at the source vertex s. It does so by iteratively (lines 9-20) taking the first vertex v out of a queue of vertices, denoted by Q (that initially only consists of s), and visits all its adjacent vertices, denoted by $adj[v]$. Once a vertex has been visited it is added to the end of the queue, unless it has already been in the queue (i.e., assigned $f(v)\leftarrow 1$ on line 10).

For the purpose of finding a path from s to t in the Ford-Fulkerson algorithm, the Breadth-First-Search is modified to additionally store the search tree (i.e., for each vertex v store its parent, denoted by $\pi(v)$) and its associated minimum capacity edge value, denoted by $r(v)$, along the path from s to any vertex v in the search tree. As the pseduo code in Figure 5 indicates, the modified algorithm also terminates whenever t has been visited.

When a path is found, the search tree will easily provide a path from s to t, by recursively tracing parents starting with $\pi(t)$, and the associated minimum capacity edge value, $r(t)$. When a path is not found the search tree contains each vertex v that is reachable from s, and is labeled $f(v) = 1$.

The residual graph's edge capacities corresponding to the precedence relations are always positive since they are initially set to a value greater than the maximum flow value. As a result, if a vertex v is reachable from s, each vertex w succeeding v is also reachable from s, and hence labeled $f(w) = 1$. Therefore, the variable f used in BREADTH-FIRST-SEARCH, defines a monotone Boolean function, and the variable e holds the associated number errors $e(f)$.

The algorithm FLOW-INCREASE shown in Figure 6 performs a single iteration of the Ford-Fulkerson (1962) algorithm. FLOW-INCREASE works on the residual graph which starts out as the graph created by algorithm CONSTRUCT-RESIDUAL-GRAPH. For the sample data in Table 2 the initial residual graph is shown in Figure

4. During each execution of FLOW-INCREASE, a path from the source vertex *s* to the sink vertex *t*, with a positive capacity (i.e., a path that can be augmented with increased flow) is found using the algorithm BREADTH-FIRST-SEARCH. An augmenting path $(v^k, v^{k-1}, ..., v^1)$, where $v^k = s$ and $v^1 = t$, for which the minimum capacity edge value is $min\{c(v^k, v^{k-1}),\ c(v^{k-1}, v^{k-2}), ...,\ c(v^2, v^1)\} = r > 0$, is found in line 1 of FLOW-INCREASE. The notation is reversed by *k*, *k*-1, ..., 1 in order to be consistent with the implementation given in Figure 6.

```
FLOW-INCREASE(G,s,t)

1 BREADTH-FIRST-SEARCH(G,s,t)
2 if Π(t) ≠ {}
3   v(1) ← t
4   r ← r(t)
5   i ← 2
6   while v(i) ← Π(v(i-1)) ≠ {}
7     c(v(i),v(i-1)) ← c(v(i),v(i-1)) - r
8     c(v(i-1),v(i)) ← c(v(i-1),v(i)) + r
9     i ← i + 1
```

Figure 6. An algorithm used to update the capacities along an augmenting path.

If a path is found in line 1 of FLOW-INCREASE, the flow along the augmenting path is increased as much as possible in lines 3-9. That is, the capacities along the path are reduced as much as possible without making any of the capacities negative: $c(v^{k-1}, v^{k-2}) \leftarrow c(v^{k-1}, v^{k-2}) - r$, $c(v^{k-1}, v^{k-2}) \leftarrow c(v^{k-1}, v^{k-2}) - r$, ..., $c(v^2, v^1) \leftarrow c(v^2, v^1) - r$. Furthermore, the capacities along the same path in the opposite direction are increased by the same amount. That is, $c(v^1, v^2) \leftarrow c(v^1, v^2) + r$, $c(v^2, v^3) \leftarrow c(v^2, v^3) + r$, ..., $c(v^{k-1}, v^k) + r$. If a path from the source *s* to the sink vertex *t* is not found on line 1 of FLOW-INCREASE, the BREADTH-FIRST-SEARCH algorithm terminates with a monotone Boolean function defined by the variable *f*, and *e* contains the corresponding number of errors.

Figure 7 shows the graphs resulting from three executions of the FLOW-INCREASE algorithm, starting out with the residual graph created in Figure 4. The left most graph is the result of the first execution, and successive graphs are shown to the right of it. The bold edges give the layouts of the paths that were augmented and indicate how much the capacities changed. For example, the path (*s*, 010, 110, *t*) in the initial residual graph was augmented with 2 units. That is, the capacities along (*s*, 010, 110, *t*) were reduced by 2 units, while the capacities along the path (*t*, 110, 010, *s*) were increased by 2 units, creating the leftmost graph in Figure 7.

The residual graph after the third iteration (i.e., the rightmost graph in Figure 7) does not contain a path from the source vertex *s* to the sink vertex *t*. The three vertices labeled 101, 011 and 111 are reachable from *s*. That is, the optimal monotone Boolean function found by the maximum flow algorithm is defined by $f^*(111) = f^*(101) = f^*(011) = 1$, and $f^*(000) = f^*(100) = f^*(010) = f^*(001) = f^*(110) = 0$. This function can also be defined by its lower units $LU(f^*) = \{101, 011\}$, and consequently as $f^* = v_1v_3 \vee v_2v_3$, and yields a total of $e(f^*) = m_0(111) + m_0(101) + m_0(011) + m_1(000) + m_1(100) + m_1(010) + m_1(001) + m_1(110) = 1 + 1 + 1 + 1 + 3 + 2 + 4 + 3 = 16$ errors. Therefore, $q^* = e(f^*)/m = 16/36 \approx 0.444$.

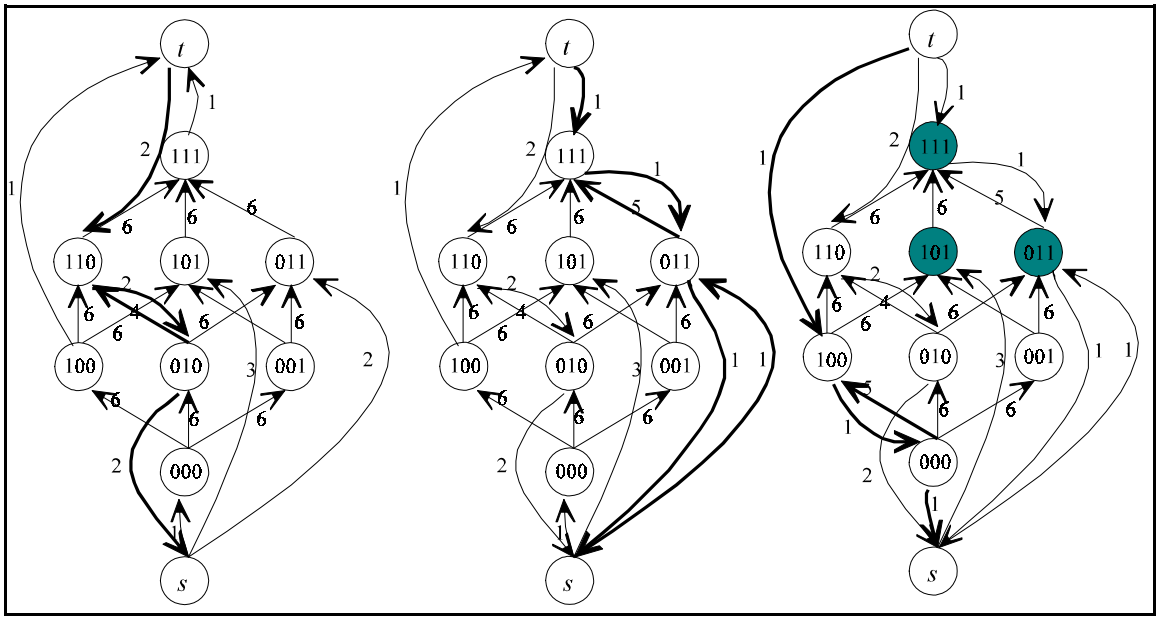

Figure 7. The iterations of the algorithm FLOW-INCREASE on the sample data.

3.2 An Incremental Maximum Flow Algorithm for Error Minimization

Suppose the error minimizing function f_{old}^* and its misclassification parameter q_{old}^*, associated with a set of vectors $V = \{v^1, v^2, ..., v^k\}$ and their $m_0(v)$ and $m_1(v)$ values, are given. When a new vector is classified by the oracle (i.e., $m_z(v) \leftarrow m_z(v) + 1$), the function f_{old}^* and its misclassification parameter q_{old}^* may have to be updated. Since the new error minimizing function is likely to be close to the old function it may be inefficient to solve the entire problem over again.

Simply stated the incremental problem consists of finding f_{new}^* and consequently q_{new}^* when $m_z(v) \leftarrow m_z(v) + 1$. If the new classification is consistent with the old function (i.e., $f_{old}^*(v) = z$), then the old function remains error minimizing (i.e., $f_{old}^* = f_{new}^*$). Therefore, the number of errors remains the same and the misclassification estimate is reduced to $q_{new}^* = e(f_{old}^*)/(m_{old} + 1)$. Note that this case is the most likely one since it occurs with an estimated probability of $1 - q_{old}^* \geq ½$.

On the other hand, if the new classification is inconsistent with the old function (i.e., $f_{old}^*(v) = 1 - z$), the old function may or may not remain error minimizing. The only case in which the old function does not remain error minimizing is when there is an alternate error minimizing function f_a^* on the old data for which $f_a^*(v) = z$. In this case f_a^* is error minimizing for the new data. However, the number of possible error minimizing functions may be exponential in V, and therefore storing all them may not be an efficient solution to this problem. To avoid this computational burden an incremental maximum flow algorithm is developed.

The algorithm MAX-FLOW-INCREMENT1 described in Figure 8 shows the details of the update $m_z(v) \leftarrow m_z(v) + 1$ when $z = 1$. An analogous algorithm can be created for the case when $z = 0$. Therefore, only the case when $z = 1$ is considered here. The updates performed in algorithm MAX-FLOW-INCREMENT1 are very different depending on the value of $m_1(v) - m_0(v)$, because of the manner in which the original residual graph was constructed.

```
MAX-FLOW-INCREMENT1(G,f,e,m1,m0,v)

1  if m1(v) ≥ m0(v)
2    c(s, v) ← c(s, v) + 1
3    if c(s, v) = 1
4      if path from v to t exists
5        increase flow by 1 along (s, v, ..., t)
6        e ← e + 1 - f(v)
7      else
8        if f(v) = 0
9          f(w) ← 1 for w in {v and x reachable from v}
10 else
11   if c(v, t) > 0
12     c(v, t) ← c(v, t) - 1
13     e ← e + 1
14   else (c(v, t) = 0)   % flow on (v,t) > 0 was max
15     c(t, v) ← c(t, v) - 1
16     increase flow by 1 on path from v to s
17     if path from s to t exists
18       increase flow by 1 on path from s to t
19     update f and e if necessary
```

Figure 8. The algorithm used to update the residual graph when $f(v) = 1$ is observed.

The first case is determined by $m_1(v) \geq m_0(v)$. Here the vertex v will be (and may have been) directly connected to s by $c(s, v)$ being positive in the initial residual graph. When $m_1(v) \leftarrow m_1(v) + 1$, this fact remains true. That is, $m_1(v) \geq m_0(v)$ still holds, and the position of the edge does not change, while its capacity is merely increased by one unit as follows: $c(s, v) \leftarrow c(s, v) + 1$. This step (line 2 in of MAX-FLOW-INCREMENT1) takes a total of O(1) time.

If $c(s, v) = 1$ after this update takes place, then this edge was operating at its maximum capacity (i.e., $c(s, v) = 0$) before this update. Adding another unit of capacity along the edge (s, v) may allow another unit of flow along a path from the source vertex s to the sink vertex t via the vertex v. If such a path exists, none of the vertices along this path were reachable from s before the update (if so additional flow could go from s to t before update, which contradicts the maximum flow assumption). Hence, none will be reachable after the flow has been increased along this path. That is, f^* remains the same, while the number of errors remains the same if $f^*(v) = 1$, and is increased by 1 unit if $f^*(v) = 0$ (i.e., $e(f^*) \leftarrow e(f^*) + f^*(v)$, as on line 6). If such a path does not exist and $f^*(v) = 0$, then vertex v has changed from not being reachable to being reachable from s. Therefore, any vertex w reachable from v is now also reachable from s. If $f^*(v) = 0$, then by the definition of f^*, it is updated as follows: $f^*(w) \leftarrow 1$ for each vertex w reachable from v (including v). These steps (lines 4-9) take a total time of $O(V^2)$.

The second case is identified by $m_1(v) < m_0(v)$. Here the vertex v was directly connected to t by $c(v, t)$ being positive in the initial residual graph. If the current value of $c(v, t)$ is positive, then the capacity on the edge (v, t) was not fully utilized in the maximum flow. Consequently, reducing its capacity by 1 unit does not change the maximum flow or the residual graph other than as follows: $c(v, t) \leftarrow c(v, t) - 1$. That is, the optimal monotone Boolean function f^* remains the same. Since vertex v is not reachable from the source vertex s (otherwise additional flow from s to

t would be allowed, which is a contradiction of the maximum flow assumption), we can further deduce that $f^*(v) = 0$. That is, the number of errors is increased by one unit. These steps (lines 12-13) take a total of O(1) time.

The most computationally expensive case is when $c(v, t)$ is zero. Here, the edge (v, t) was operating with a positive flow (since its initial capacity was assigned to $m_0(v) - m_1(v) > 0$) at its maximum capacity. Therefore, the flow along a path from vertex s to vertex t that goes via edge (v, t), which is guaranteed to exist, has to be reduced by 1 unit and the capacity is updated as follows: $c(v, t) \leftarrow c(v, t) - 1$. This update is equivalent to $c(t, v) \leftarrow c(t, v) - 1$, and increasing the flow by 1 unit along a path from vertex v to vertex s, shown in lines 15-16. The single unit of flow that was retracted along the path $(s, ..., v, t)$, may be pushed through another path from vertex s to vertex t. The algorithm FLOW-INCREASE can be used to perform this operation and the algorithm BREADTH-FIRST-SEARCH is needed afterwards to potentially update f^* and e^*. These steps (lines 15-19) take a total of $O(V^2)$ time. In summary, the total computational complexity of algorithm MAX-FLOW-INCREMENT1 is $O(V^2)$.

For the purpose of illustrating algorithm MAX-FLOW-INCREMENT1, consider Examples 1, 2, and 3, describing three different increment scenarios for the data given in Table 2. The initial residual graph for the MAX-FLOW-INCREMENT1 algorithm is given as the right most of the graphs in Figure 3. Please recall that the associated values are: $m = 36$, $f^* = v_1v_3 \vee v_2v_3$, $e^* = 16$, and $q^* = 16/36 \approx 0.444$. Now consider the three following examples when another 1 valued vector (i.e., $f(v) = 1$) is observed.

Example 1. $f(011) = 1$ is observed:

$m_1(011) = 3 \geq m_0(011) = 1$ and so $c(s, 011) \leftarrow c(s, 011) + 1 = 1 + 1 = 2$. The capacity was not fully utilized, therefore f^* and e^* remain the same, while q^* is updated to $16/37 \approx 0.432$. Only lines 1 and 2 were executed for this update. The total time used is O(1), and the updated graph is shown in Figure 9.

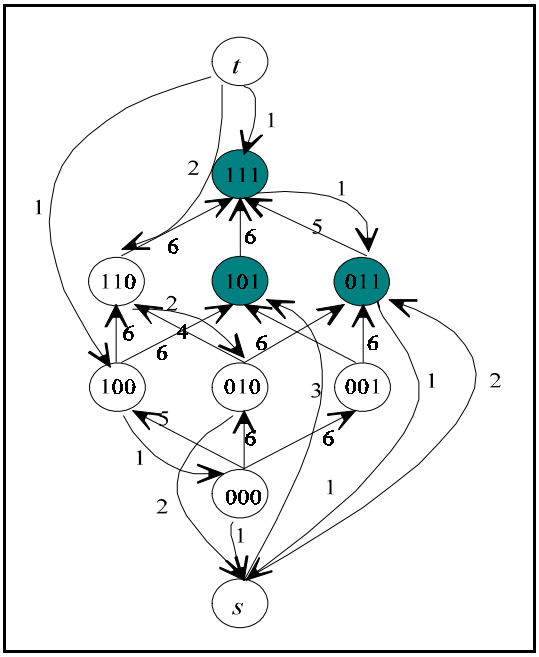

Figure 9. The residual graph for Example 1.

Example 2. $f(010) = 1$ is observed:

$m_1(010) = 2 \geq m_0(010) = 0$ and so $c(s, 010) \leftarrow c(s, 010) + 1 = 0 + 1 = 1$. The capacity was fully utilized but no new path is created along $(s, 010, ..., t)$. Since $f^*(010) = 0$, a new set of vertices that are reachable from s is created as follows: f(010) ← 1, and f(110) ← 1. As a result, the new optimal monotone Boolean function is $f^* = v_2 \vee v_1 v_3$, e^* remains the same at 16, and finally q^* is updated to $16/37 \approx 0.432$. Lines 1-4, 8, and 9 were executed using a total of $O(V^2)$ time. The updated graph is shown in Figure 10.

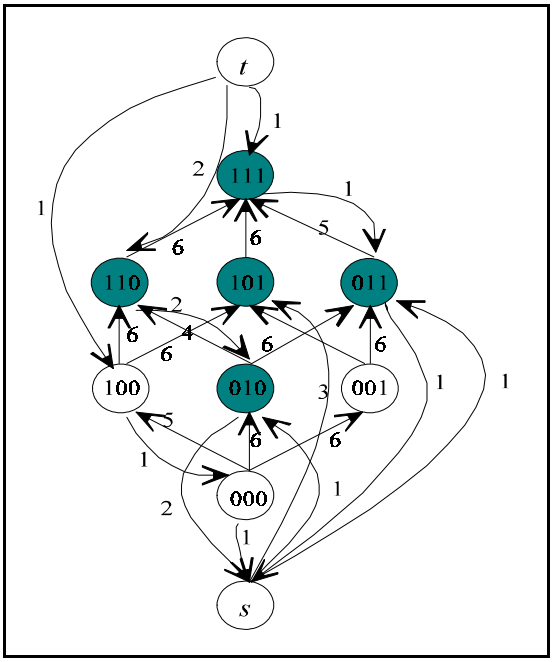

Figure 10. The residual graph for Example 2.

Example 3. $f(100) = 1$ is observed:

$m_1(100) = 4 < m_0(100) = 5$ and $c(100, t) = 0$, so the edge is fully utilized and $c(t, 100)$ is reduced by 1 unit: $c(t, 100) \leftarrow c(t, 100) - 1 = 1 - 1 = 0$, and along the path $(100, 000, s)$ the flow is increased by 1 unit as follows: $c(100, 000) \leftarrow c(100, 000) - 1 = 1 - 1 = 0$, $c(000, 100) \leftarrow c(000, 100) + 1 = 5 + 1 = 6$, $c(000, s) \leftarrow c(000, s) - 1 = 1 - 1 = 0$, $c(s, 000) \leftarrow c(s, 000) + 1 = 0 + 1 = 1$. No path from s to t exists, so f^* is updated as follows: $f(000) \leftarrow 1$, $f(100) \leftarrow 1$, $f(010) \leftarrow 1$, $f(001) \leftarrow 1$, $f(110) \leftarrow 1$, making the new optimal monotone Boolean function $f^* = T$, for which the number of errors e^* remains the same at 16, and finally $q^* = 16/37 \approx 0.432$. Lines 1, 11, 15, 16, 17 and 19 were executed using a total of $O(V^2)$ time. The updates are shown in Figure 11.

As mentioned earlier the fastest maximum flow algorithms are of the preflow type (Karzanov (1974)) and take $O(V^3)$ time. However, the incremental algorithm developed in this section uses the idea of augmenting paths which was conceived earlier by Ford and Fulkerson (1962). The fastest known maximum flow algorithms based on augmenting paths use $O(VE^2)$ time (Edmonds and Karp (1972)), where E denotes the set of edges which can be of size $O(V^2)$. The incremental algorithm developed here uses $O(V^2)$ time for each observation, and hence takes a total

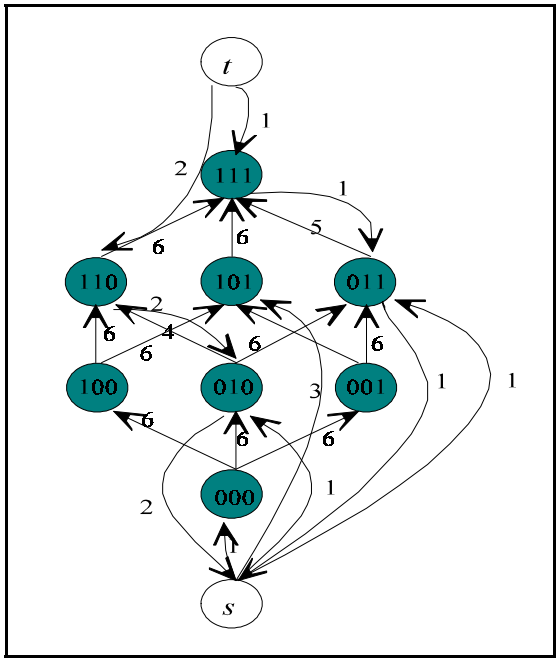

Figure 11. The residual graph for Example 3.

of $O(mV^2)$ time. Here m denotes the number of observations which is generally greater than the number of vectors V. That is, using the incremental algorithm for a non-incremental problem does not improve upon the existing algorithm taking $O(V^3)$ time. However, the incremental algorithm reduces the complexity of solving the incremental problem by a factor of $O(V)$.

3.3 Defining an Evaluative Criterion

The status of the inference process will be considered to be in one of three stages. Stage 1 starts with the first question, and lasts until a deterministic monotone Boolean function is obtained. During Stage 1 only vectors that may take on both 0 and 1 values are queried. As a result, no (identifiable) errors are observed in Stage 1, and thus the monotone Boolean function inferred during Stage 1 is deterministic. This function, however, may or may not be the correct function. In fact, the probability that it is the correct function is equal to the probability that no misclassifications were made: $(1- q)^m$, where m is the number of questions used during Stage 1 and q is the true misclassification probability. This probability decreases rapidly with m, regardless of the value of q. Therefore, the queries performed after Stage 1 will benefit greatly from a reduction in the number of Stage 1 queries. Please note that since no inconsistencies have been observed, there is no way to estimate q at this point.

After a deterministic monotone Boolean function is obtained in Stage 1, the inference process enters Stage 2. At this point it is unclear as to how to select queries for Stage 2, so a random selection procedure will be used for this stage. After the first error occurs in Stage 2, the inference process enters Stage 3, in which it will remain until termination. Stage 3 is the focus of this paper, because it is the only stage in which the likelihood ratio can be properly evaluated and q can be estimated based on the observed vectors.

Please recall that the likelihood function is given by:

$$L(f) = q^{e(f)}(1 - q)^{m - e(f)},$$

and the likelihood ratio is given by:

$$\lambda(f^*) = \frac{L(f^*)}{\sum_{f \in F(V)} L(f)}.$$

As an example of the likelihood ratio computations consider the example data given in Table 2. The function $f^* = v_1v_3 \vee v_2v_3$ found in Section 3.1 produces 16 errors. Its associated estimated misclassification probability q^* is 16/36 = 4/9, since the total number of observations is m = 36. Therefore, the likelihood value of this function $L(f^*)$ is $(4/9)^{16}(1 - 4/9)^{36-16} = 1.818\times10^{-11}$. Notice how small this value is after only 36 observations. The likelihood values for the other functions are given in Table 3. Adding up all the likelihood values yields $(13\times1.455 + 2\times1.536 + 5\times1.818)\times10^{-11} = 3.107 \times10^{-10}$. Then the maximum likelihood ratio is computed as follows: $\lambda(f^*) = 1.818\times10^{-11}/3.107 \times10^{-10} = 0.0585$.

Now let us return to the vector selection (or guided inference) problem. The probability that the correct function is inferred during Stage 1 decreases rapidly with the number of queries used during that stage. As mentioned earlier, Stage 1 has been studied extensively in the past. Torvik and Triantaphyllou (2001a) suggested selecting the vector v that minimizes the quantity $|K_0(v) - K_1(v)|$, where $K_z(v)$ is the number of unclassified vectors preceding v if z = 0, or succeeding v if z = 1. The vector selection criterion minmizes the average number of queries used for Stage 1 for n up to and including 4. According to the empirical results of Torvik and Triantaphyllou (2001a), this is currently the best known criterion and is probably very close to optimal for n greater than 4.The probability that the correct function is inferred during Stage 1 decreases rapidly with the number of queries used during that stage. Therefore, the selection criterion $min|K_0(v) - K_1(v)|$ will be used as a standard for Stage 1, when comparing different approaches for the following Stage 3. This avoids bias in the sense that all Stage 3 approaches will benefit from using $min|K_0(v) - K_1(v)|$ during Stage 1.

One important property of the evaluative criterion for Stage 3 is that the maximum likelihood ratio converges to 1. It is possible to define selection criteria that do not converge. If, for example, the same vector is invariably selected, the estimated value of q will converge to its true value. In this case, the likelihood values may remain equal for several monotone Boolean functions and hence the maximum likelihood ratio will never converge to 1.

As was demonstrated in Torvik and Triantaphyllou (2001a), intuition may lead to an inefficient selection criterion for Stage 1. The same holds true for Stage 3. For example, let $E_z(v)$ be defined by the number of errors associated with assigning the function value $f(v)$ to z, as follows:

$$E_0(v) = \sum_{w \le v} m_1(w) - m_0(w), \qquad E_1(v) = \sum_{v \le w} m_0(w) - m_1(w).$$

Then, consider defining the vector v which contributes with the most to errors by $max(E_0(v)+E_1(v))$. This vector selection criterion may lead to the same vector being invariable queried and hence it might suffer from convergence

problems, as will be demonstrated empirically in Section 4.

Table 3. Example likelihood values for all functions in M_3.

f	$e(f)$	$q(f)$	$L(f)$	$\lambda(f)$
F	20	½	1.455×10^{-11}	0.0468
$v_1v_2v_3$	21	½	1.455×10^{-11}	0.0468
v_1v_2	23	½	1.455×10^{-11}	0.0468
v_1v_3	18	½	1.455×10^{-11}	0.0468
$v_1v_2 \vee v_1v_3$	20	½	1.455×10^{-11}	0.0468
v_1	21	½	1.455×10^{-11}	0.0468
v_2v_3	19	½	1.455×10^{-11}	0.0468
$v_1 \vee v_2v_3$	19	½	1.455×10^{-11}	0.0468
$v_1v_3 \vee v_2v_3$	16	4/9	1.818×10^{-11}	0.0585
$v_1v_2 \vee v_1v_3 \vee v_2v_3$	18	½	1.455×10^{-11}	0.0468
$v_1v_2 \vee v_2v_3$	21	½	1.455×10^{-11}	0.0468
v_2	19	½	1.455×10^{-11}	0.0468
$v_1 \vee v_2$	17	17/36	1.536×10^{-11}	0.0495
$v_2 \vee v_1v_3$	16	4/9	1.818×10^{-11}	0.0585
v_3	16	4/9	1.818×10^{-11}	0.0585
$v_2 \vee v_3$	16	4/9	1.818×10^{-11}	0.0585
$v_1 \vee v_2 \vee v_3$	17	17/36	1.536×10^{-11}	0.0495
$v_1 \vee v_3$	19	½	1.455×10^{-11}	0.0468
$v_3 \vee v_1v_2$	18	½	1.455×10^{-11}	0.0468
T	16	4/9	1.818×10^{-11}	0.0585

The likelihood framework seems to form a great basis for defining a Stage 3 vector selection criterion. Since the goal is to make the likelihood ratio converge to 1 as fast as possible, a reasonable approach would be to select the vector that maximizes the expected maximum likelihood ratio at each inference step. To do this, the expected maximum likelihood ratio $\Delta\lambda(v) = p(v)\lambda_1(v) + (1 - p(v))\lambda_1(v)$ has to be estimated for each vector v. Here $\lambda_z(v)$ denotes the resulting maximum likelihood ratio when $f(v) = z$ is observed. Please recall that $p(v)$ is the probability of observing $f(v) = 1$. That is, it can be estimated by $p^*(v) = q^*(1- f^*(v)) + (1 - q^*)f^*(v)$.

As an example consider observing the vector (001). Table 4 gives the updated likelihood ratios for each monotone Boolean function in M_3 when $m_z(001) = m_z(001) + 1$, for $z = 0$ and 1. For a monotone Boolean function

f, and a classification z, $e_z(001, f)$ and $\lambda_z(001, f)$ here denote the updated number of errors and the likelihood ratio, respectively. The updated maximum likelihood ratios are $\lambda_1(001, T) = 0.0649$ and $\lambda_0(001, v_1v_3 \vee v_2v_3) = 0.0657$. Since the optimal function assigns the vector (001) to 0 (i.e., $f^*(001) = 0$), the estimated probability of observing $f(001) = 1$ is given by $p^*(001) = q^* = 4/9$. Therefore, the expected maximum likelihood ratio when querying vector 001 is given by $\Delta\lambda(001) = p^*(001)\lambda_1(001, T) + (1 - p^*(001))\lambda_0(001, v_1v_3 \vee v_2v_3) = 4/9 \times 0.0649 + 5/9 \times 0.0657 = 0.0653$. Similar computations for the other vectors yield $\Delta\lambda(000) = 0.0651$, $\Delta\lambda(010) = 0.0654$, $\Delta\lambda(011) = 0.0592$, $\Delta\lambda(100) = 0.0652$, $\Delta\lambda(101) = 0.0592$, $\Delta\lambda(110) = 0.0654$, and finally $\Delta\lambda(111) = 0.0592$. The vectors with the largest expected likelihood ratio value are 010 and 110. Since no further improvements of the selection criterion is immediately obvious, ties are broken arbitrarily.

Table 4. Updated likelihood ratios for $m_z(001) = m_z(001) + 1$.

f	$\lambda(f)$	$e_1(001, f)$	$\lambda_1(001, f)$	$e_0(001, f)$	$\lambda_0(001, f)$
F	0.0468	21	0.0462	20	0.0468
$v_1v_2v_3$	0.0468	22	0.0462	21	0.0468
v_1v_2	0.0468	24	0.0462	23	0.0468
v_1v_3	0.0468	19	0.0462	18	0.0474
$v_1v_2 \vee v_1v_3$	0.0468	21	0.0462	20	0.0468
v_1	0.0468	22	0.0462	21	0.0468
v_2v_3	0.0468	20	0.0462	19	0.0468
$v_1 \vee v_2v_3$	0.0468	20	0.0462	19	0.0468
$v_1v_3 \vee v_2v_3$	0.0585	17	0.0522	16	0.0657
$v_1v_2 \vee v_1v_3 \vee v_2v_3$	0.0468	19	0.0462	18	0.0474
$v_1v_2 \vee v_2v_3$	0.0468	22	0.0462	21	0.0468
v_2	0.0468	20	0.0462	19	0.0468
$v_1 \vee v_2$	0.0495	18	0.0469	17	0.0529
$v_2 \vee v_1v_3$	0.0585	17	0.0522	16	0.0657
v_3	0.0585	16	0.0649	17	0.0529
$v_2 \vee v_3$	0.0585	16	0.0649	17	0.0529
$v_1 \vee v_2 \vee v_3$	0.0495	17	0.0522	18	0.0474
$v_1 \vee v_3$	0.0468	19	0.0462	20	0.0468
$v_3 \vee v_1v_2$	0.0468	18	0.0469	19	0.0468
T	0.0585	16	0.0649	17	0.0529

The simulations in Section 4 reveal the efficiency of the evaluative criterion $max\ \Delta\lambda(v)$ in terms of the query complexity. It may take an exponential time (in the size of V) to compute $max\ \Delta\lambda(v)$. Since the computational time for incrementally finding the inferred function is of $O(V^2)$, it would be nice to find an evaluative criterion that does not take more time than this and still makes the likelihood converge to 1 at a faster rate than randomly selecting vectors. One such possibility may be based on the inferred border vectors.

For the sake of argument suppose that the underlying monotone Boolean function f to be inferred is known. Then randomly selecting vectors from its corresponding border vectors will make the maximum likelihood ratio converge to 1. As the number of queries m goes to infinity, the ratios $m_0(v)/(m_0(v) + m_1(v)) \ \forall\ v \in \mathrm{LU}(f)$ and $m_1(w)/(m_0(w) + m_1(w)) \ \forall\ w \in \mathrm{UZ}(f)$ all converge to q. The number of errors performed by any other monotone Boolean function g is at least $x = min\{min\{m_1(v) - m_0(v), v \in \mathrm{LU}(f)\}, \{m_0(w) - m_1(w),\ w \in \mathrm{UZ}(f)\}$ greater than the number of errors performed by function f. Furthermore, $x \approx qm - (1\text{-}q)m = m(2q - 1)$ for large m. That is, the number of additional errors increases at least linearly with m. Then, as m goes to infinity, so does the number of additional errors performed by each of the other monotone Boolean functions. That is, the relative likelihoods $L(f)/L(g) > (q/(1\text{-}q))^x$ converge to 0 as m goes to infinity. Since the number other monotone Boolean functions is a finite number that does not depend on m, the likelihood ratio $\lambda(f) = \mathrm{L}(f) / (\mathrm{L}(f) + \sum L(g))$ converges to 1 as m goes to infinity.

Focusing the queries at the border vectors of the underlying function probably allows this convergence to occur at a faster rate than randomly selecting from all the vectors. In situations where the underlying function is unknown, it may be that focusing the queries on the border vectors of the inferred function (i.e., $v \in \mathrm{LU}(f^*) \cup \mathrm{UZ}(f^*)$) is better than completely random selection. In the long run, an inferred border vector will not prevail if it is not an underlying border vector. Since the misclassification rate is less than ½, the rate at which the incorrectly classified inferred border vectors become correctly classified is greater the rate at which correctly classified inferred border vectors become incorrectly classified. Therefore, in the long run all the classifications become correct when the queries are selected from the set of border vectors of the inferred function.

Notice that this convergence holds even if the misclassification probability is different for each vector, as long as they are all less than ½. Another added benefit is that finding the border vectors is easy, since they are readily available from the inferred function f^*. In fact, a simple modification of the incremental maximum flow algorithm can store each of these vectors as they are found. For each monotone Boolean function there are at most $O(V)$ border vectors in a set of vectors V. During the inference process the inferred function may take on any of these monotone Boolean functions. Therefore, randomly selecting one of the border vectors takes $O(V)$ time.

4. Experimental Results

For the purpose of comparing the efficiency of the different selection criteria for Stage 3 on the same basis, ties resulting from the selection criteria ($min|K_0(v) - K_1(v)|$ for Stage 1, and $max(E_0(v) + E_1(v))$, $max\ \Delta\lambda(v)$, and $v \in \mathrm{LU}(f^*) \cup \mathrm{UZ}(f^*)$ for Stage 3) were broken randomly. The four different inference processes using $max\ \Delta\lambda(v)$, $v \in$

LU(f^*) ∪ UZ(f^*), $max(E_0(v) + E_1(v)$, or random selection for Stage 3 were simulated on the set of vectors $\{0,1\}^n$.

For all three Stage 3 selections criteria, the criterion $min|K_0(v) - K_1(v)|$ was used for Stage 1 and random selection was used for Stage 2. The resulting simulations were repeated 100, 50, 25, and 10 times for each of 6 representative functions of M_n, with misclassification probabilities 0.1, 0.2, 0.3, and 0.4, for n = 2, 3, 4 and 5, respectively. For n equal to 6, the number of monotone Boolean functions is 7,828,354 and generating all of them to evaluate their likelihoods became too computationally burdensome for the 600 Mhz Pentium III based personal computer with 384 Mbytes of RAM used in these experiments. This limitation is also be due to the fact that the programs were run in interpreting mode for MATLAB 5.3 under the Windows 98 operating system.

The representative functions are given in Table 5. For n = 4 and 5, the representative functions were randomly generated from a uniform distribution with individual probabilities of $1/\Psi(n)$ = 1/168 and 1/7581, respectively. For n = 3, the representative functions consist of non-similar functions (one from each similar subset of M_3). These functions represent all the functions in M_3, since the average case behavior is the same for a pair of similar monotone Boolean functions. To compute the overall average for a given q, the individual curves were weighted by the number of similar functions the representative function has (including itself) in M_3. The individual curves for the monotone Boolean functions F, $v_1v_2v_3$, v_1v_2, $v_1v_2 \vee v_1v_3$, v_1, and $v_1v_2 \vee v_1v_3 \vee v_2v_3$, were therefore weighted by 2, 2, 6, 6, 3 and 1, respectively. For n = 2, 4, and 5, the overall averages were computed without weights. The overall averages for n = 2 and 3 benefit from a reduced variance, since no additional errors are added due to sampling of functions as done for n = 4 and 5.

Table 5. The representative functions used in the simulations.

$n = 2$	$n = 3$	$n = 4$	$n = 5$
F	F	$v_1v_2 \vee v_2v_4 \vee v_1v_3v_4$	$v_1v_4 \vee v_1v_5 \vee v_2v_4 \vee v_2v_5$
v_1v_2	$v_1v_2v_3$	$v_1v_2 \vee v_1v_3 \vee v_2v_3 \vee v_2v_4 \vee v_3v_4$	$v_1v_3 \vee v_2v_3 \vee v_2v_4 \vee v_1v_2v_5$
v_1	v_1v_2	$v_2v_3 \vee v_2v_4$	$v_2 \vee v_1v_3v_4 \vee v_1v_4v_5$
v_2	$v_1v_2 \vee v_1v_3$	$v_1v_2v_3 \vee v_1v_3v_4 \vee v_2v_3v_4$	$v_1v_3 \vee v_2v_4 \vee v_3v_5 \vee v_1v_4v_5$
$v_1 \vee v_2$	v_1	$v_1v_2 \vee v_2v_4 \vee v_3v_4$	$v_2v_4 \vee v_2v_5 \vee v_3v_5 \vee v_4v_5$
T	$v_1v_2 \vee v_1v_3 \vee v_2v_3$	$v_3 \vee v_1v_2 \vee v_1v_4$	$v_2v_5 \vee v_1v_2v_3 \vee v_1v_3v_4 \vee v_1v_4v_5 \vee v_3v_4v_5$

Figure 12 shows the resulting average maximum likelihood curves for the inference problem defined on n = 2, 3, 4, and 5, and q = 0.1, 0.2, 0.3, and 0.4. Each curve is the average of 600, 300, 150, and 60 simulated inference processes observed for n = 2, 3, 4, and 5, respectively. In each plot, the horizontal axis corresponds to the number of Stage 3 queries, and the vertical axis corresponds to the maximum likelihood ratio. The curves are shown for the range of Stage 3 queries where the curves for the evaluative criterion $max\ \Delta\lambda(v)$ has a maximum likelihood ratio that is less than 0.99.

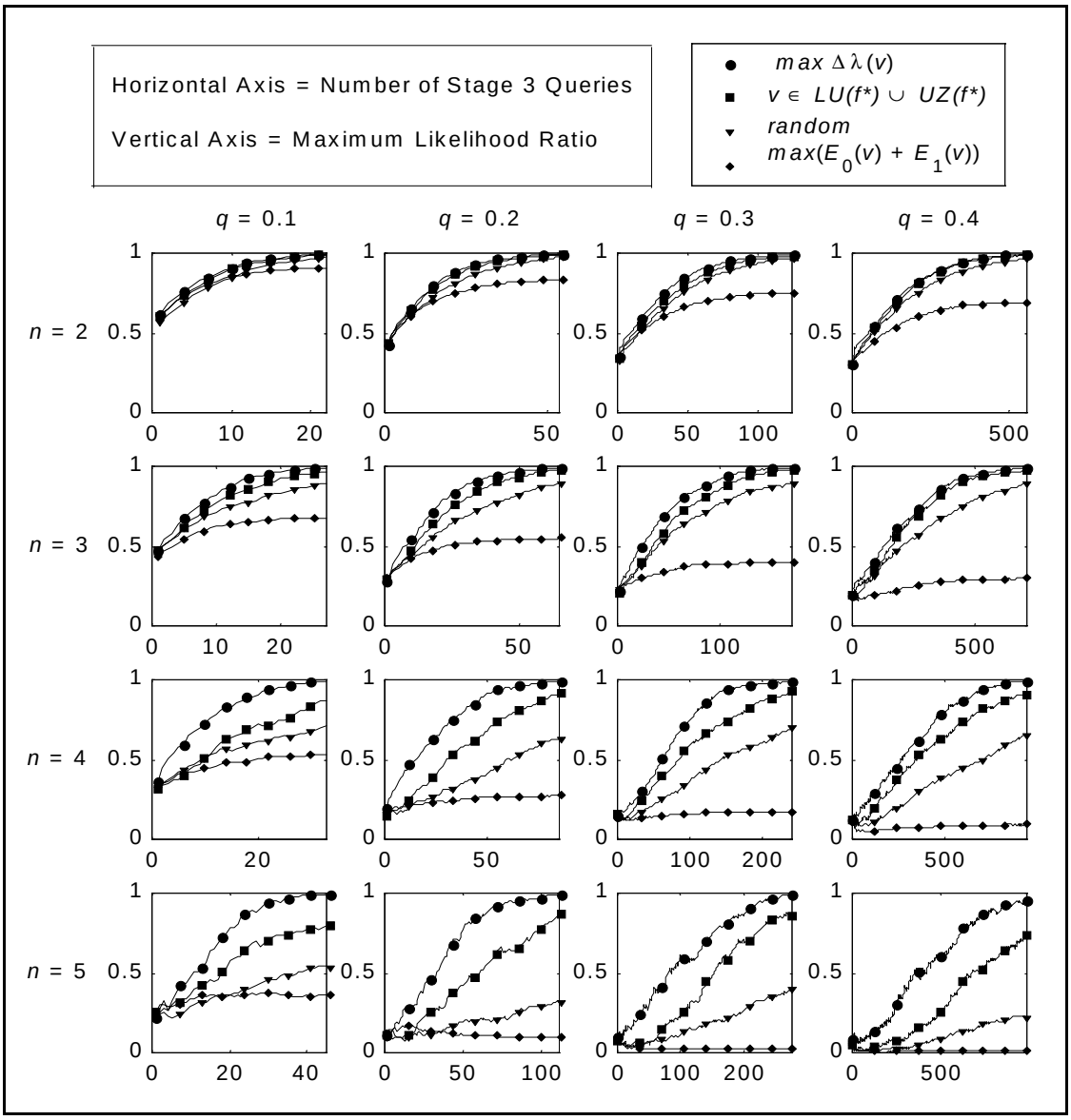

Figure 12. Average case behavior for various selection criteria.

Not only do the curves corresponding to the guided selection criteria *max* $\Delta\lambda(v)$ and $v \in \mathrm{LU}(f^*) \cup \mathrm{UZ}(f^*)$ converge to 1 but they do so at a much faster rate than the curves corresponding to unguided random selection. In fact, the random selection achieves a maximum likelihood ratio of only about 0.7 after the same number of queries as the criterion *max* $\Delta\lambda(v)$ uses to reach 0.99, and the criterion $v \in \mathrm{LU}(f^*) \cup \mathrm{UZ}(f^*)$ uses to reach about 0.9, for $n = 4$.

The difference between the curves for unguided selection and these two guided selections grows for higher values of the misclassification probability and for higher dimensions. That is, the benefits from actively selecting vectors over passively receiving observations, are greater for larger values of q and n. In other words, the higher the misclassfication probability and the dimension of the problem are, the greater the benefits of guiding the inference process become.

The curves associated with criterion $max(E_0(v) + E_1(v))$ seems to converge to a value significantly less than 1. For example, when $n = 3$ and $q = 0.3$, the maximum likelihood ratio converges to about 0.4, and this value decreases as the values of q and n increase. Therefore, the larger error rate and vector domain is, the more important it becomes to define an appropriate vector selection criterion.

Table 6 gives the average number of queries needed by the selection criterion $max\ \Delta\lambda(v)$ to converge to a maximum likelihood ratio of 0.99 for $n = 2$, 3, 4, and 5, and for $q = 0.1$, 0.2, 0.3, and 0.4. For a given n, these numbers increase dramatically as q increases. In fact, there seems to be more than a doubling in the numbers for fixed increments of q. For a given q, these numbers do not increase in such a dramatic fashion when n increases. However, they do increase faster than linearly with n.

Table 6. The average number of Stage 3 queries used by $max\ \Delta\lambda(v)$ to reach $\lambda > 0.99$.

	$q = 0.1$	$q = 0.2$	$q = 0.3$	$q = 0.4$
$n = 2$	22	54	125	560
$n = 3$	27	65	170	710
$n = 4$	33	85	241	951
$n = 5$	45	111	277	1167

Figure 13 further illustrates how the average number of queries increases with q and n. The vertical axis corresponds to the number of queries and are shown on a $\log_{10}$ scale. The horizontal axis corresponds to the dimension n, and each curve corresponds to a particular misclasssification probability q. These curves exhibit three significant properties. First, they all tend to increase linearly with n, indicating that the number of queries tend to increase exponentially with n. Second, the curves seem to be parallel, indicating that this exponential increase is fixed regardless of what the value of q is. Third, the distance between the parallel lines tend to increase with fixed increments of q, indicating that the number of queries increases faster than exponentially with q. That is, the number of queries can be approximated by a function of the parameters n and q in the form $a2^{bn} + c(q)2^{dq} + e$, where a, b, d, and e are constants and $c(q)$ is an increasing function of q. It should be noted that this approximation may not be appropriate in greater dimensions (i.e., for $n > 5$), or outside this range of misclassification probabilities (i.e, for $q < 0.1$ or $q > 0.4$).

This approximation also sheds some light on the total computational complexity of finding the error minimizing function. Please recall from Section 3.2 that the time used by the incremental maximum flow algorithm per observation was $O(V^2)$. Therefore, the total time spent refitting the model is $O(mV^2)$ where m is the total number of queries needed to make the maximum likelihood ratio reach 0.99. The number of queries divided by $|V|$ decreases with $|V|$, according the data in Table 6. Thus, the total time used by the incremental algorithm is less than $O(V^3)$. In contrast, the fastest known general purpose maximum flow algorithm is of $O(V^3)$. This is the complexity of the

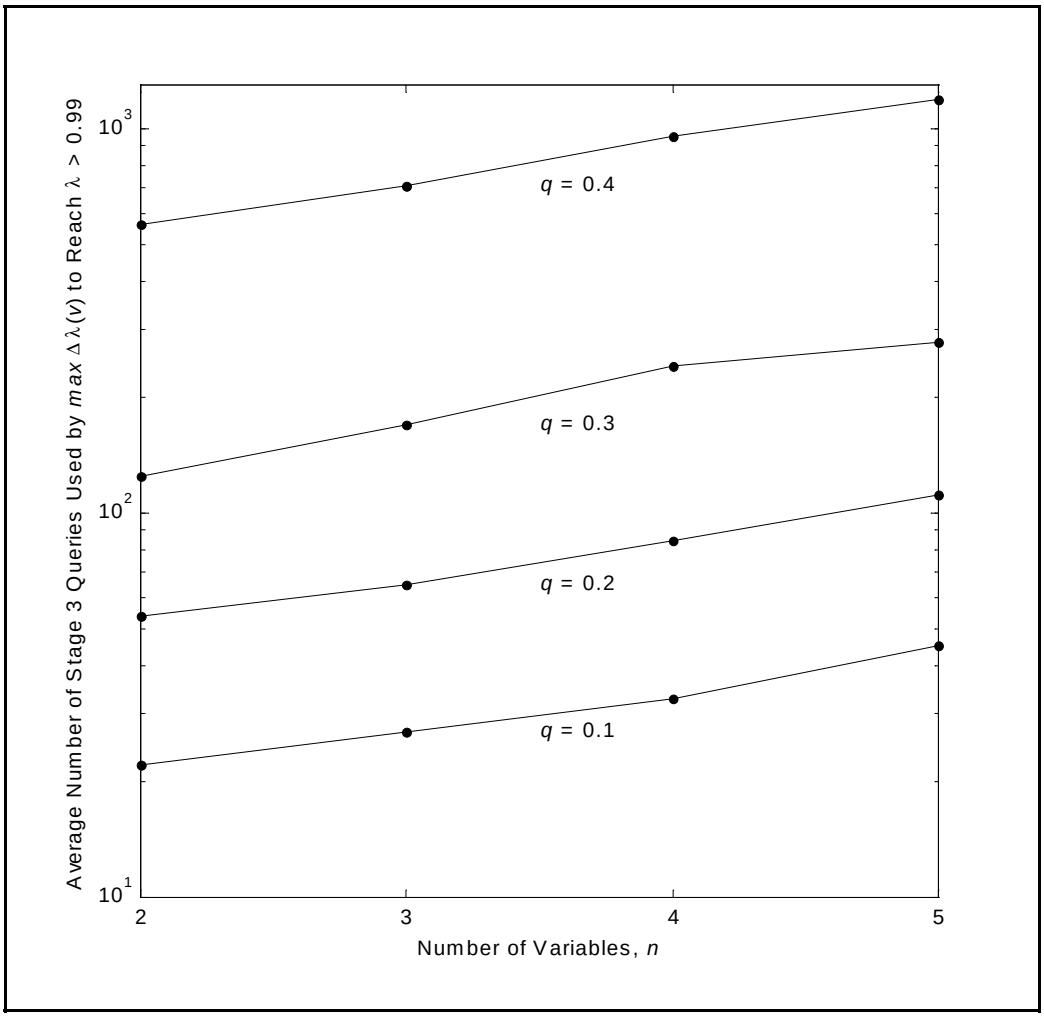

Figure 13. Average query complexity of $max\ \Delta\lambda(v)$ on the poset $\{0,1\}^n$ with misclassification probability q.

problem if all the data were gathered before the error minimizing monotone Boolean function was found. Therefore, the time used fitting the model is not increased by using the incremental approach.

It should be noted that the computational complexity of evaluating the maximum likelihood ratio, and hence the evaluative criterion, is exponential in the size of the set V. In some applications where the set V is large, computing the criterion $max\ \Delta\lambda(v)$ may be an infeasible task. However, randomly selecting a border vector (i.e., $v \in LU(f^*) \cup UZ(f^*)$) takes at most $O(V)$ time. Even though focusing on border vectors does not reduce the query complexity as much as the criterion $max\ \Delta\lambda(v)$, it performs much better than random selection.

Please recall from Section 3.3 that randomly selecting the inferred border vectors (i.e., $v \in LU(f^*) \cup UZ(f^*)$) makes the maximum likelihood ratio converge to 1, as long as the misclassification probabilities are all less than ½. That is, the misclassification probabilities do not necessarily have to be fixed. To see whether this holds for the selection criterion $max\ \Delta\lambda(v)$, consider an unrestricted model where the misclassification probability $q(v)$ is a random variable distributed uniformly on the interval $[q(1-\delta), q(1+\delta)]$, where $\delta \in [0,1]$, for each vector $v \in \{0,1\}^n$.

The case when $\delta = 0$, corresponds to the fixed misclassification probability model (i.e., $q(v) = q \; \forall \; v \in \{0,1\}^n$). The range of values that $q(v)$ can take on increases with δ, but the expected value of $q(v)$ is equal to q. Therefore, the estimate of the maximum likelihood ratio based on the fixed q model is worse for larger values of δ. To compare this estimate to an unrestricted estimate, the inference process was simulated 200 times for each $\delta = 0$, 0.5, and 1, when $q = 0.2$, $n = 3$. Figure 14 shows the average maximum likelihood ratio curves for the unrestricted model (dotted curves) and the fixed model (solid curves) when using the selection criterion $max \; \Delta\lambda(v)$.

The regular and the unrestricted maximum likelihood ratios both converge to 1, though at slower rates as δ increases. In other words, evaluative criterion $max \; \Delta\lambda(v)$ is appropriate in situations where the misclassification probability is not necessarily fixed. In general, the unrestricted maximum likelihood ratio is much smaller than the regular one. The case when $q(v)$ is fixed at 0.2 (i.e., $\delta = 0$), the regular maximum likelihood ratio should be used, and for $\delta > 0$ it is an overestimate of the true maximum likelihood ratio. The case when $\delta = 1$, the unrestricted maximum likelihood ratio should be used, and for $\delta < 1$ it may be an underestimate. The true likelihood ratio lies somewhere in between the two.

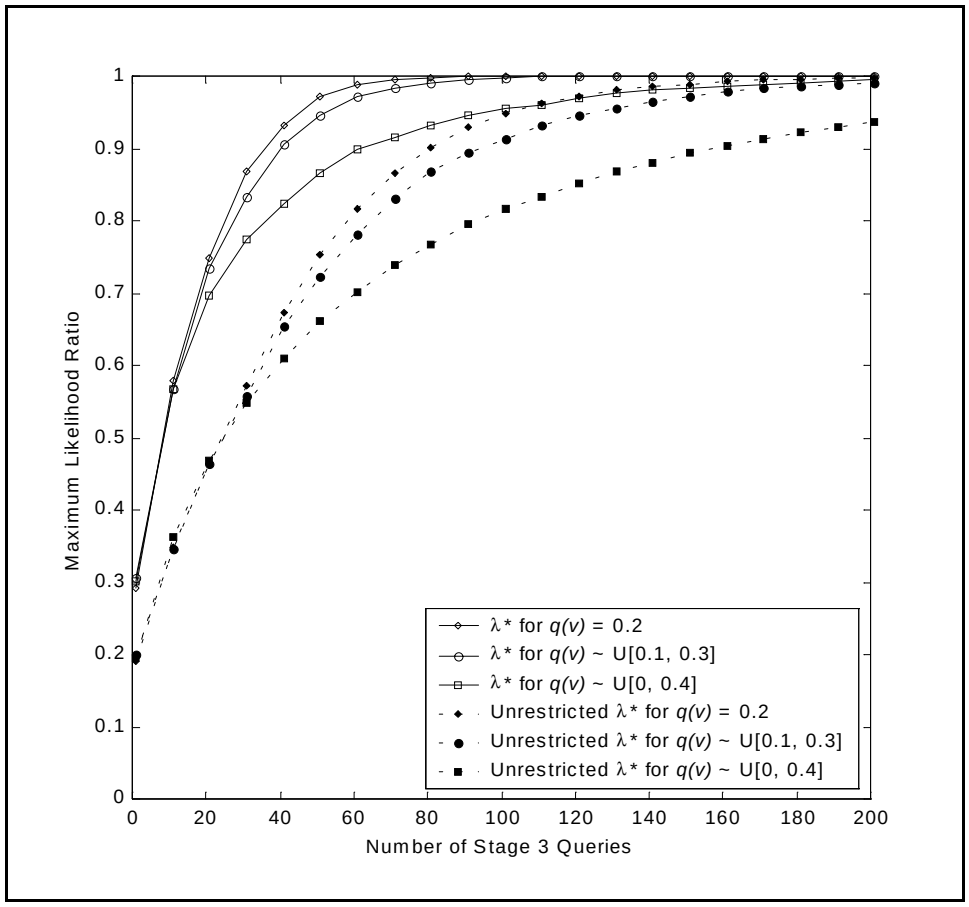

Figure 14. The unrestricted and regular maximum likelihood ratios simulated with expected $q = 0.2$ and $n = 3$.

5. Conclusion

The maximum likelihood ratio approach to modeling the inference process of monotone Boolean functions yielded a number of benefits. It was demonstrated that an appropriately defined guided learner, such as maximizing the expected maximum likelihood ratio ($max\ \Delta\lambda(v)$) or randomly selecting inferred border vectors ($v \in LU(f^*) \cup UZ(f^*)$), allowed the maximum likelihood ratio to converge to 1, even when the misclassification probability was not fixed. This avoids the bias problems associated with the variance approach reported in Cohn *et al.* (1996), and also observed with the selection criterion based on the number of errors ($max(E_0(v) + E_1(v))$).

For large problems (i.e., $n > 5$), it may not be possible to compute the evaluative criterion $max\ \Delta\lambda(v)$ since it takes exponential time (in the size of the query domain V) to do so. For such problems, queries can be selected randomly from the border vectors ($v \in LU(f^*) \cup UZ(f^*)$). This only takes $O(V)$ time, and results in much fewer queries than completely random selection on the average.

For complete reconstruction of monotone Boolean functions, the guided approach showed a dramatic reduction in the average number of queries over a passive learner on the average. The simulations also indicated that this improvement grows at least exponentially as the number of variables n and the error rate q increase. Thus, defining an appropriate and efficient evaluative criterion is even more beneficial for large problems and applications with a high error rate.

Acknowledgements

The authors would like to thank Professor Jerry L. Trahan in the Electrical Engineering Department at Louisiana State University for stimulating discussions of the incremental maximum flow algorithm. Also, the authors gratefully acknowledge the support from the Office of Naval Research Grant N00014-97-1-0632.

References

Ayer, M., Brunk, H.D., Ewing, G.M., Reid, W.T. Silverman, E. 1955. An Empirical Distribution Function for Sampling with Incomplete Information. *Annals of Mathematical Statistics* 26 641-647.

Ben-David, A. 1992. Automatic Generation of Symbolic Multiattribute Ordinal Knowledge-Based DSSs: Methodology and Applications. *Decision Sciences* 23(6) 1357-1372.

Ben-David, A. 1995. Monotonicity Maintenance in Information-Theoretic Machine Learning Algorithms. *Machine Learning* 19(1) 29-43.

Bloch, D.A., Silverman, B.W. 1997. Monotone Discriminant Functions And Their Applications in Rheumatology. *Journal of the American Statistical Association* 92(437) 144-153.

Block, H., Qian, S., Sampson, A. 1994. Structure Algorithms for Partially Ordered Isotonic Regression. *Journal of Computational and Graphical Statistics* 3(3) 285-300.

Boros, E., Hammer, P.L., Hooker, J.N. 1994. Predicting Cause-Effect Relationships from Incomplete Discrete Observations. *SIAM Journal on Discrete Mathematics* 7(4) 531-543.

Boros, E., Hammer, P.L., Hooker, J.N. 1995. Boolean Regression. *Annals of Operations Research* 58 201-226.

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.

Cohn, D.A. 1996. Neural Network Exploration Using Optimal Experiment Design. *Neural Networks* 9(6) 1071-1083.

Cohn, D.A. 1995. Minimizing Statistical Bias with Queries. A.I. Memo No. 1552, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA.

Cormen, T.H , Leiserson, C.H., Rivest, R.L. 1997. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, USA.

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

Edmonds, J. Karp, R.M. 1972. Theoretical Improvements in the Algorithmic Efficiency for Network Flow Problems. *Journal of the ACM*, 19 248-264.

Federov, V.V. 1972. *Theory of Optimal Experiments*. Academic Press, New York, NY, USA.

Ford, L.R. Fulkerson, D.R. 1962. *Flows in Networks*. Princeton University Press, Princeton, NJ, USA.

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

Judson, D.H. 2001. A partial order approach to record linkage. Federal Committee on Statistical Methodology conference, November 14-16, Arlington, VA, USA.

Karzanov, A.V. 1974. Determining the Maximal Flow in a Network by the Method of Preflows. *Soviet Mathematics Doklady* 15 434-437.

Korshunov, A.D. 1981. On the Number of Monotone Boolean Functions. *Problemy Kibernetiki* 38 5-108 (in Russian).

Kovalerchuk, B., Triantaphyllou, E. Vityaev, E. 1995. Monotone Boolean Function Learning Techniques Integrated with User Interaction. Proceedings of Workshop "Learning from Examples vs. Programming by Demonstration", 12th International Conference on Machine Learning, Lake Tahoa, CA, USA, 41-48.

Kovalerchuk, B., Triantaphyllou, E. Deshpande, A.S. 1996. Interactive Learning of Monotone Boolean Functions. *Information Sciences*, 94 87-118.

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

Lee, C.I.C. 1983. The min-max Algorithm and Isotonic Regression. *The Annals of Statistics*. 11 467-477.

MacKay, D.J.C. 1992. Information-based Objective Functions for Active Data Selection. *Neural Computation* 4(4) 589-603.

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

Picard, J.C. 1976. Maximal Closure of a Graph and Applications to Combinatorial Problems. *Management Science* 22 1268-1272.

Robertson, T., Wright, F.T., Dykstra, R.L. 1988. *Order Restricted Statistical Inference*. John Wiley & Sons, New York, NY, USA.

Shmulevich, I. 1997. Properties and Applications of Monotone Boolean Functions and Stack Filters. A Ph.D. Dissertation, Department of Electrical Engineering, Purdue University, West Lafayette, IN, USA.

Tatsuoka, C., Ferguson, T. 1999. Sequential Classification on Partially Ordered Sets. Technical Report 99-05, Department of Statistics, The George Washington University, Washington D.C., USA.

Torvik, V.I., Triantaphyllou, E. 2001a. Minimizing the Average Query Complexity of Learning Monotone Boolean Functions. To appear in *INFORMS Journal on Computing*.

Torvik, V.I., Triantaphyllou, E. 2001b. Guided Inference of Nested Monotone Boolean Functions. Pending journal review.

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

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