Data mining and knowledge discovery: a guided approach base on monotone boolean functions

Vetle Ingvald Torvik
Louisiana State University and Agricultural and Mechanical College, vtorvik@uic.edu

Follow this and additional works at: https://digitalcommons.lsu.edu/gradschool_dissertations

Part of the Engineering Science and Materials Commons

Recommended Citation
https://digitalcommons.lsu.edu/gradschool_dissertations/1678

This Dissertation is brought to you for free and open access by the Graduate School at LSU Digital Commons. It has been accepted for inclusion in LSU Doctoral Dissertations by an authorized graduate school editor of LSU Digital Commons. For more information, please contact gradetd@lsu.edu.
DATA MINING AND KNOWLEDGE DISCOVERY: 
A GUIDED APPROACH BASED ON 
MONOTONE BOOLEAN FUNCTIONS

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Interdepartmental Program in Engineering Science

by
Vetle Ingvald Torvik
B.A., St. Olaf College, 1995
M.S., Oregon State University, 1997
May 2002
to Elizabeth

you are my source of inspiration
ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor, Professor Evangelos Triantaphyllou, for going above and beyond to make himself available to provide me with the support, guidance, and freedom in my research endeavors. This dissertation is a direct product of his resounding words of wisdom “Simplicity, simplicity, simplicity!”,”Never compromise the quality!””, and perhaps the most important “Enjoy the process!”.

This research was made possible by the support from the Office of Naval Research (ONR) Grant N00014-97-1-0632 through Dr. Donald Wagner, Director for Operations Research at the ONR. I also greatly acknowledge the following sources of funding: the Norwegian Government (Lånekassen) for scholarships, the Louisiana Engineering Foundation for the 2000-2001 Graduate Student Fellowship, and the LSU Graduate School and Industrial Engineering Department for a number of conference travel grants.

Thanks goes out to my committee members: Professors Manoj K. Chari, Jianhua Chen, Lynn R. LaMotte, T. Warren Liao, and Jerry L. Trahan, to my research collaborators: most notably Drs. Dean H. Judson and Ilya Shmulevich, to my M.S. advisor: Professor Jeffrey L. Arthur, to my family: Elizabeth, Are, John, Jenny, Julia, Dag, Judith, Ellen, Gordon, Andrea, Jim, Kathy, Katherine, and to my friends: to many to list, who have all contributed to this dissertation through ideas, encouragement, and support.
TABLE OF CONTENTS

ACKNOWLEDGMENTS ........................................................................................................ iii

LIST OF TABLES ............................................................................................................. vii

LIST OF FIGURES .......................................................................................................... viii

ABSTRACT ................................................................................................................... xi

CHAPTER 1. INTRODUCTION ...................................................................................... 1

CHAPTER 2. PRELIMINARY PROBLEM DESCRIPTIONS ........................................... 3
  2.1 Problem # 1: Inferring a Monotone Boolean Function from a Deterministic Oracle .... 3
  2.2 An Application of Problem # 1 to a Computer Sales Company ............................. 4
  2.3 Problem # 2: Inferring a Pair of Nested Monotone Boolean Functions ............... 5
    2.3.1 Problem # 2.1: Sequentially Inferring a Pair of Nested Monotone Boolean
      Functions from Two Deterministic Oracles ......................................................... 6
    2.3.2 An Application of Problem # 2.1 to Breast Cancer Diagnosis ...................... 6
    2.3.3 Problem # 2.2: Simultaneously Inferring a Pair of Nested Monotone
      Boolean Functions from a Single Three-valued Deterministic Oracle ............... 7
    2.3.4 An Application of Problem # 2.2 to Record Linkage in Databases .............. 8
    2.3.5 Problem # 2.3: Simultaneously Inferring a Pair of Nested Monotone
      Boolean Functions from Two Unrestricted, Deterministic Oracles .................. 10
    2.3.6 An Application of Problem # 2.3 to a College Acceptance Policy ............... 10
  2.4 Problem # 3: Inferring a Monotone Boolean Function from a Stochastic Oracle ...... 11
  2.5 Hierarchical Decomposition of Variables ......................................................... 12

CHAPTER 3. BACKGROUND INFORMATION ......................................................... 16
  3.1 Some Key Properties of Monotone Boolean Functions ........................................ 16
  3.2 Existing Approaches to Problem # 1 ................................................................. 19
    3.2.1 Illustration of the FIND-BORDER Algorithm for the Computer Sales
      Company Application ......................................................................................... 23
    3.2.2 Illustration of Hansel’s Algorithm for the Computer Sales
      Company Application ......................................................................................... 24
    3.2.3 Illustration of Sokolov’s Algorithm for the Computer Sales
      Company Application ......................................................................................... 25
  3.3 An Existing Approach to Problem # 2 ................................................................. 27
  3.4 Existing Approaches to Problem # 3 ................................................................. 27
  3.5 Stochastic Models for Monotone Boolean Functions .......................................... 27
  3.6 Unequal Probability Sampling ............................................................................. 29

CHAPTER 4. COMPLETE DESCRIPTIONS OF THE RESEARCH PROBLEMS ............ 31
  4.1 The Inference Objective for Problem # 1 .......................................................... 31
  4.2 The Inference Objective for Problem # 2 .......................................................... 31
### CHAPTER 5. METHODOLOGY

5.1 Methodology for Problem # 1 .................................35
   5.1.1 Minimizing the Average Inference Cost ..........................35
   5.1.2 Minimizing the Average Number of Queries ......................38
   5.1.3 Defining an Evaluative Criterion ..................................43
   5.1.4 Illustration of the Evaluative Criterion Approach to the Computer
         Sales Company Application .....................................47
   5.1.5 Some Other Evaluative Criteria ..................................48
5.2 Methodologies for Problems # 2.1, # 2.2, and # 2.3 ...............51
   5.2.1 Defining Evaluative Criteria .....................................51
   5.2.2 Illustration of the Evaluative Criterion Approach to the
         Breast Cancer Diagnosis Application ...............................52
   5.2.3 Illustration of the Evaluative Criterion Approach to the
         Record Linkage Application .........................................54
   5.2.4 Illustration of the Evaluative Criterion Approach to the
         College Acceptance Policy Application ..............................55
   5.2.5 Recursive Formulas for the Minimum Average Number of Queries
         on the Chain Poset ..................................................56
   5.2.6 Optimal and Evaluative Criterion Approaches to the Chain Poset ....58
5.3 Methodology for Problem # 3 .....................................61
   5.3.1 Error Minimization via Maximum Flow Computations ...............62
   5.3.2 An Incremental Maximum Flow Algorithm for Error Minimization ....67
   5.3.3 Defining an Evaluative Criterion ..................................71
5.4 An Algorithm for Randomly Generating Monotone Boolean Functions
       with Inclusion Probabilities ........................................74
   5.4.1 Computing the Inclusion Probabilities for Problem # 1 ..............76
   5.4.2 Computing the Inclusion Probabilities for Problem # 2 ..............77
   5.4.3 Approximating the Ratio $\Psi_k(n) / \Psi(n)$ ..........................77

### CHAPTER 6. EXPERIMENTAL RESULTS

6.1 Experimental Results for Problem # 1 ...............................81
6.2 Experimental Results for Problem # 2 ................................84
6.3 Experimental Results for Problem # 3 ................................87

### CHAPTER 7. CONCLUDING REMARKS

7.1 Summary of the Research Findings ..................................93
7.2 Significance of the Research Findings ................................94
7.3 Future Research Directions ...........................................95

REFERENCES PART I: BY THE AUTHOR ..............................97
LIST OF TABLES

3.1 History of monotone Boolean function enumeration for $n = 1, 2, \ldots, 8$ ...................... 19

5.1 The number of posets generated by MINIMIZE-AVE-Q($n$) ........................................ 45

5.2 Sample data for the example .............................................................. 63

5.3 Example likelihood values for all functions in $M_3$ ................................................... 72

5.4 Updated likelihood ratios for $m_\zeta(001) - m_\zeta(001) + 1$ ................................ 73

5.5 Some exact values of $\Psi_k(n)$ ................................................................. 78

5.6 Some exact values of $\alpha(n, k)$ ................................................................. 79

6.1 The representative functions used in the simulations of Problem # 3 ......................... 88

6.2 The average number of Stage 3 queries used by the evaluative criterion
$\max \Delta \lambda(v)$ to reach $\lambda > 0.99$ in Problem # 3 defined on $(0,1)^n$ with fixed
misclassification probability $q$ ............................................................... 90
# LIST OF FIGURES

2.1 Illustration of sequential oracles in breast cancer diagnosis ........................................6
2.2 Illustration of a single three-valued oracle in record linkage ........................................8
2.3 Illustration of two oracles for the college acceptance policy ..........................................10
2.4 Hierarchical decomposition of the variables for the breast cancer application ...............13
2.5 Hierarchical decomposition of the variables for the college acceptance policy application ........................................................14
3.1 The poset formed by \( \{0,1\}^3 \) and the \( \leq \) relation .........................................................17
3.2 Illustration of poset multiplication for two nested monotone Boolean functions defined on \( \{0,1\}^n \) ..........................................................20
3.3 The algorithm used to generate the Hansel chains in dimension \( n \) .............................21
3.4 The subroutine used to the find border vectors of a monotone Boolean function one at a time .........................................................22
3.5 The computer crash inference process using the FIND-BORDER algorithm .................24
3.6 The computer crash inference process using Hansel’s algorithm ................................25
3.7 The computer crash inference process using Sokolov’s algorithm ..........................26
4.1 The algorithm used to randomly generate monotone Boolean functions without inclusion probabilities .......................................................33
5.1 The exhaustive vector selection tree for \( \{0,1\}^2 \) .........................................................37
5.2 The vector selection tree for \( \{0,1\}^2 \) restricted to connected, non-dual, and non-isomorphic posets ..........................................................40
5.3 The algorithm used to compute \( Q(n) \) .................................................................41
5.4 Part of the tree traversed by the MINIMIZE-AVE-Q(3) ........................................43
5.5 All connected, non-dual, and non-isomorphic posets generated to compute \( Q(4) \), with the optimal vertices circled ..............................................44
5.6 Illustration of the sawtooth and chain posets .........................................................46
5.7 Average query complexity of Problem # 1 defined on the sawtooth poset 47

5.8 The computer crash inference process using the evaluative criterion $\min|K_1 - K_0|$ 48

5.9 Classification efficiency of two evaluative criteria over all functions in $M_5$ 50

5.10 Sequential inference of the two nested breast cancer diagnosis functions using the evaluative criterion $\min|K_1 - K_0|$ 53

5.11 Simultaneous inference of the two nested record linkage functions from a three-valued oracle using the evaluative criterion $\min|K_{11} - K_{00}|$ followed by $\min|K_1 - K_0|$ 54

5.12 Simultaneous inference of the two nested college acceptance policy functions using the evaluative criterion $\min|K_1 - K_0|$ 56

5.13 Constructing the query domain for a pair of nested monotone Boolean functions defined on a chain poset 57

5.14 The minimum average query complexity of Problem # 2 defined on the chain poset 59

5.15 Results from using the evaluative criteria for Problem # 2 defined on the chain poset 60

5.16 The algorithm used to construct the initial residual graph from the observed data 62

5.17 The graph used to construct the residual graph from the sample data 64

5.18 The residual graph constructed from the sample data 64

5.19 The algorithm used to traverse the residual graph 65

5.20 The algorithm used to update the capacities along an augmenting path 66

5.21 The iterations of the algorithm FLOW-INCREASE on the sample data 67

5.22 The algorithm used to update the residual graph when $f(v) = 1$ is observed 68

5.23 The residual graph for Example 5.1 70

5.24 The residual graph for Example 5.2 70

5.25 The residual graph for Example 5.3 70

5.26 A pair of nested monotone Boolean functions created from a non-monotone Boolean function defined on $\{0,1\}^4$ 75
5.27 The algorithm used to randomly generate monotone Boolean functions with inclusion probabilities .................................................. 75

5.28 Some exact and approximated values of the ratio $\Psi_k(n) / \Psi(n) ............................................. 79$

6.1 The query complexities of the inference algorithms used for Problem # 1 distributed over all monotone Boolean functions in the set $M_s .................. 82$

6.2 Average query complexities of the inference algorithms used for Problem # 1 defined on the poset $\{0,1\}^n ............................................. 83$

6.3 Average query complexities per border vector of the inference algorithms used for Problem # 1 defined on the poset $\{0,1\}^n ................................. 84$

6.4 Average query complexities of the evaluative criteria used for Problem # 2 defined on the poset $\{0,1\}^n ............................................. 85$

6.5 Increase in average query complexities due to the restricted access to the oracles of Problem # 2 defined on the poset $\{0,1\}^n ..................... 86$

6.6 Improvement in average query complexity of the evaluative criterion approach due to the nestedness assumption of Problem # 2 defined on the poset $\{0,1\}^n ................................. 87$

6.7 Average case behavior of various evaluative criteria used for Problem # 3 ................. 89

6.8 Average query complexity of the evaluative criterion $\max \Delta \lambda(v)$ used for Problem # 3 defined on the poset $\{0,1\}^n$ with fixed misclassification probability $q$ ..... 91

6.9 The unrestricted and regular maximum likelihood ratios simulated for Problem # 3 with expected $q = 0.2$ and $n = 3$ using the evaluative criterion $\max \Delta \lambda(v) ............... 92$
ABSTRACT

This dissertation deals with an important problem in Data Mining and Knowledge Discovery (DM & KD), and Information Technology (IT) in general. It addresses the problem of efficiently learning monotone Boolean functions via membership queries to oracles. The monotone Boolean function can be thought of as a phenomenon, such as breast cancer or a computer crash, together with a set of predictor variables. The oracle can be thought of as an entity that knows the underlying monotone Boolean function, and provides a Boolean response to each query. In practice, it may take the shape of a human expert, or it may be the outcome of performing tasks such as running experiments or searching large databases.

Monotone Boolean functions have a general knowledge representation power and are inherently frequent in applications. A key goal of this dissertation is to demonstrate the wide spectrum of important real-life applications that can be analyzed by using the new proposed computational approaches. The applications of breast cancer diagnosis, computer crashing, college acceptance policies, and record linkage in databases are here used to demonstrate this point and illustrate the algorithmic details. Monotone Boolean functions have the added benefit of being intuitive. This property is perhaps the most important in learning environments, especially when human interaction is involved, since people tend to make better use of knowledge they can easily interpret, understand, validate, and remember.

The main goal of this dissertation is to design new algorithms that can minimize the average number of queries used to completely reconstruct monotone Boolean functions defined on a finite set of vectors $V = \{0,1\}^n$. The optimal query selections are found via a recursive algorithm in exponential time (in the size of $V$). The optimality conditions are then summarized in the simple form of evaluative criteria, which are near optimal and only take polynomial time to compute. Extensive unbiased empirical results show that the evaluative criterion approach is far superior to any of the existing methods. In fact, the reduction in average number of queries increases exponentially with the number of variables $n$, and faster than exponentially with the oracle’s error rate.
CHAPTER 1. INTRODUCTION

It has been said many times that the current proliferation of cost efficient computing power and digital storage has led to a society which is increasingly becoming “a giant of information but a dwarf of new knowledge”. Huge databases are often described in terms of hundreds of terabytes according to current standards. It is not a coincidence that the term “teras” means monster in Greek!

The process of extracting new knowledge from large amounts of data is often called Knowledge Discovery or Data Mining. The knowledge discovery process is not always easy or efficient. In situations where one is able to obtain knowledge, it may be hard to interpret, understand, and validate it.

This dissertation addresses the problem of guided inference of monotone Boolean functions with the underlying objective to efficiently acquire simple and intuitive knowledge that can be validated and has a general representation power. The following key properties strengthen the argument in favor of this objective:

Key Property # 1: Monotone Boolean functions are inherently frequent in applications. The following three examples illustrate the versatility of the monotonicity property and how it applies to practical situations.

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).

If a keyword search in a database gives interesting hits, then hits for a proper superset of these keywords is also probably going to be interesting. On the other hand, if a keyword search in a database does not give interesting hits, then hits for a proper subset of these keywords is probably not going to be interesting either.

With all other factors constant, a student with a high Grade Point Average (GPA) is more likely to be accepted into a particular college than a student with a low GPA.

Key Property # 2: Monotone Boolean functions are simple and intuitive. This property is perhaps the most important one when human interaction is involved since people tend to make very good use of knowledge they can easily interpret, understand, validate, and remember. Due to the increasing computational efficiency and storage capacity, the recent trend has been to increase the knowledge representation power in order to capture more complex knowledge. For example, the popular neural networks are capable of representing very complex knowledge. Unfortunately, even small neural networks can be hard to interpret and validate.

Key Property # 3: Monotone Boolean functions can represent relatively complex knowledge and still be validated. Validating knowledge that is generalized from a set of specific observations, which may be noisy and incomplete, is based on philosophical arguments and assumptions. Traditional statistical approaches tend to require specific modeling in small dimensions, to gain a theoretical justification for the final model. This justification is obtained at the cost of eliminating the computational feasibility of learning higher dimensional rules. On the other hand, the more general the knowledge representation is, the more one tends to lose the handle on its validation.
Recent literature contains a plethora of 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 (Judson (2001)).

In practice, a great deal of time and effort is put into the knowledge discovery process. Software applications are tested, diseases are researched, database search engines are trained to be intelligent, and so on. The 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. Therefore, the main objective in this dissertation is to minimize the labor associated with gathering the data, as long as it is computationally feasible.

Monotone Boolean functions lay the ground for a simple and efficient 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 property, this learning strategy may significantly increase the learning rate, as an unguided learner might not receive the relevant pieces of information early enough in the inference process. Therefore, it is highly desirable not only to be able to pose questions, but to pose “smart” questions.

The main problem addressed in this dissertation is how to identify these “smart” questions in order to efficiently infer monotone Boolean functions. This dissertation focuses on the case when the monotone Boolean functions are defined on the set of $n$-dimensional Boolean vectors $\{0,1\}^n$. This does not necessarily limit the application domain as the methodology developed in this dissertation can be applied to any finite set of vectors $V \subset \mathbb{R}^n$, and any monotone function can be represented by a set of monotone Boolean functions.

Three special cases of the main problem are first loosely described in Chapter 2. After the background information and the relevant literature is reviewed in Chapter 3, a formal definition of these problems and their subproblems are given in Chapter 4. In Chapter 5, the methodologies for solving these problems are presented. Extensive computational results are provided in Chapter 6 and concluding remarks are given in Chapter 7.
CHAPTER 2. PRELIMINARY PROBLEM DESCRIPTIONS

Monotone Boolean functions lay the ground for a simple question asking strategy, which forms the basis of this dissertation. More specifically, the problem of inferring monotone Boolean functions by successive and systematic function evaluations (membership queries submitted to an oracle) is addressed.

The monotone Boolean function can be thought of as a phenomenon, such as breast cancer or a computer crash, together with a set of predictor variables. The oracle can be thought of as an entity that knows the underlying monotone Boolean function and provides a Boolean function value in response to each membership query. In practice, it may take the shape of a human expert, or it may be the outcome of performing tasks such as running experiments or searching large databases.

This inference problem is broken down by the nature of the oracle: whether it is deterministic or stochastic, and whether it is two-valued or three-valued. The simplest variant considers the guided inference of a deterministic monotone Boolean function defined on at most \( n \) Boolean variables. This case is referred to as Problem # 1 and is described in Section 2.1.

Problem # 1 is generalized into two different problems that are described in Sections 2.3 and 2.4. The first generalization includes a pair of nested monotone Boolean functions and is referred to as Problem # 2. Since this problem includes two oracles, it is further broken down into three subproblems # 2.1, # 2.2, and # 2.3, differing only in the manner in which these two oracles are accessed. The second generalization includes stochastic membership queries and is referred to as Problem # 3. For some of the problems described in this chapter, real world applications are provided. The wide variety of applications demonstrates the versatility of the monotonicity and nestedness assumptions and illustrates the main formulations they can take on.

2.1 Problem # 1: Inferring a Monotone Boolean Function from a Deterministic Oracle

Initially, the entire set of \( 2^n \) Boolean vectors \( \{0,1\}^n \) is considered to be unclassified. That is, the values of underlying monotone Boolean function \( f \) are all unknown and may be 0 or 1. A vector \( v \) is then selected from the set of unclassified vectors \( U \) and is submitted to an oracle as a membership query.

After the vector’s function value \( f(v) \) is provided by the oracle, the set of unclassified vectors is reduced according to the following monotonicity constraints:

\[
f(w) = 0, \quad \forall \ w \in U: \ w \preceq v, \text{ when } f(v) = 0,
\]

or the following monotonicity constraints:

\[
f(w) = 1, \quad \forall \ w \in U: \ v \preceq w, \text{ when } f(v) = 1.
\]

Here, the relationship \( v \preceq w \) holds if and only if \( v_i \leq w_i \), for \( i = 1, 2, ..., n \), where \( v_i \) and \( w_i \) denote the \( i \)-th Boolean elements of the vectors \( v \) and \( w \), respectively. Vectors are then repeatedly selected from the unclassified set until they are all classified (i.e., \( U = \oslash \)).

Given the classification of any unclassified vector other vectors may be concurrently classified if the underlying Boolean 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).

The cost of queries is in practice often associated with some sort of effort, such as consulting with experts, performing experiments or running simulations, and therefore far exceeds the computational cost. Therefore, this dissertation is focused on minimizing the query complexity as long as it is computationally feasible.
2.2 An Application of Problem # 1 to a Computer Sales Company
Consider a company that sells a particular computer. When a customer places an order, he or she
often selects a set of software applications to accompany the computer. Suppose that the company
has a set of five software applications including a web browser, an internet connector, an image
viewer, a CD player, and a word processor, that the customer can choose from. Further, suppose that
when the following combinations of software applications are run simultaneously, they tend to crash
the computer:

- (the image viewer),
- (the web browser and the internet connector),
- (the web browser and the word processor),
- (the internet connector and the word processor), or
- (the CD player and the word processor).

It is reasonable to assume that the computer tends to crash when additional applications are
run simultaneously. For example, if the web browser is run in addition to the image viewer, then the
computer tends to crash. Further, suppose that if none of these combinations are run, then the
computer does not tend to crash. That is, the monotonicity assumption is reasonable for the
computer’s crashing tendencies.

The monotone Boolean function that describes this phenomenon is given as follows:
\[ f(v) = v_3 \lor v_1 v_2 \lor v_1 v_3 \lor v_2 v_5 \lor v_4 v_5, \]
where \( f(v) = 1 \) if the computer tends to crash when the applications described by \( v \) are run
simultaneously, and \( f(v) = 0 \) otherwise. Here, the five corresponding Boolean variables are defined
as follows:

- \( v_1 = 1 \) if the web browser is running, 0 otherwise,
- \( v_2 = 1 \) if the internet connector is running, 0 otherwise,
- \( v_3 = 1 \) if the image viewer is running, 0 otherwise,
- \( v_4 = 1 \) if the CD player is running, 0 otherwise, and
- \( v_5 = 1 \) if the word processor is running, 0 otherwise.

Suppose this function is unknown to the company. To prevent the great costs associated with
unsatisfied customers, the company wants to build a system that warns the customer of which
software applications should not be run simultaneously. To that end, the company embarks on a
software testing project where various combinations of software applications are installed and run
on the computer. For each combination a worker runs a 10 minute test run to ensure a proper degree
of reliability.

The inference process can be viewed as a dialogue between the oracle (a 10 minute test run)
and the person posing the queries. The queries are described by using a Boolean vector of the form
\((v_1 \ v_2 \ v_3 \ v_4 \ v_5)\), where \( v_i = 0 \) or 1 for \( i = 1, 2, ..., 5 \). Initially, the set of unclassified vectors \( U \) is equal
to \( \{0,1\}^5 \). That is, there are \( 2^5 = 32 \) possible queries to choose from.

Suppose the vector \((00100)\) is selected for the first query. Then a 10 minute test is performed
with the image viewer running (i.e., \( v_3 = 1 \)). Once the test run results in a crash (i.e., \( f(00100) = 1 \)),
the 16 vectors of the form \((v_1 \ v_2 \ 1 \ v_4 \ v_5)\) can classified as 1 according to the monotonicity constraints:
\[ f(w) = 1, \forall w \in \{0,1\}^5: (00100) \preceq w. \]
This reduces the set of unclassified vectors to \( U = \{0,1\}^5 - \{(v_1 \ v_2 \ 1 \ v_4 \ v_5): v_i = 0 \text{ or } 1, \text{ for } i = 1, 2, 4, 5\}. \]

Suppose the vector \((10010)\) is selected for the second query. Then a 10 minute test is
performed with the web browser (i.e., \( v_1 = 1 \)) and the CD player running (i.e., \( v_4 = 1 \)). Once the test
run does not result in a crash (i.e., \( f(10010) = 0 \)), the 4 vectors of the form \((v_i \ 0 \ 0 \ v_4 \ 0)\) can classified
as 0 according to the monotonicity constraints: \( f(w) = 0, \forall w \in U: w \preceq (10010) \).
This leaves \(32 - 16 - 4 = 12\) unclassified vectors to choose from after just 2 queries. Similar queries are repeated until there are no unclassified vectors left (i.e., \(U = \{\}\)), at which time the monotone Boolean function is completely inferred.

If all the \(2^5\) combinations are to be tested, this project will take \(2^5 \times 10\) minutes, or 5 hours and 20 minutes. This project is even more time consuming when more software applications are involved. For example, if all combinations of 20 applications are to be tested, this project will take \(2^{20} \times 10\) minutes, or about 20 years! However, if the monotonicity property is properly utilized, the testing time can be significantly reduced.

2.3 Problem # 2: Inferring a Pair of Nested Monotone Boolean Functions

Two functions \(f_1\) and \(f_2\) are referred to as nested when the relationship \(f_1(v) \geq f_2(v)\) (or \(f_1(v) \leq f_2(v)\)) holds \(\forall v \in \{0,1\}^n\). In this dissertation, the case when \(f_1 \geq f_2\) is considered, as analogous results hold when \(f_1 \leq f_2\). A single monotone Boolean function does not capture the idea of a classification intermediate to 0 and 1. However, a pair of nested monotone Boolean functions can do so. For example, some vectors might belong to a class with a high probability (i.e., where \(f_1 = 1\) and \(f_2 = 1\)), and some might belong to the other class with a high probability (i.e., where \(f_1 = 0\) and \(f_2 = 0\)). Other instances might not be classifiable with a satisfactorily high probability. A pair of nested monotone Boolean functions allows for this intermediate classification (i.e., where \(f_1 = 1\) and \(f_2 = 0\)) to be incorporated. This makes the monotone Boolean function model more powerful.

The nested inference problem is identical to Problem # 1, except that a pair of nested monotone Boolean functions \(f_1\) and \(f_2\), defined on the same set of \(n\) variables, are to be inferred. The manner in which the functions’ values are obtained (i.e., restrictions on the access to the oracles) depends on the application under study. Therefore, this problem is further broken down into Problems # 2.1, # 2.2, and # 2.3 (described in Sections 2.3.1, 2.3.3, and 2.3.5, respectively) by the manner in which the oracles are accessed. For each of these three problems, some associated real world applications are described in Sections 2.3.2, 2.3.4, and 2.3.6, respectively. The wide variety of applications demonstrates the versatility of the nestedness and monotonicity assumptions and illustrates the major formulations they can take on.

The inference process consists of the following steps. Initially, the entire set of \(2^n\) Boolean vectors \(\{0,1\}^n\) is considered to be unclassified for both functions \(f_1\) and \(f_2\). That is, the values of underlying monotone Boolean functions \(f_1\) and \(f_2\) are all unknown and may be 0 or 1. A vector \(v\) is then selected from the set of vectors \(U_i\) that are unclassified by function \(f_i\) and is submitted to oracle \(i\) as a membership query (where \(i = 1\) or 2).

After the vector’s function value \(f_i(v)\) is provided by the oracle, the set of unclassified vectors \(U_i\) is reduced according to the following monotonicity constraints:

\[f_i(w) = 0, \forall w \in U_i; w \leq v, \text{ when } f_i(v) = 0,\]

or the following monotonicity constraints:

\[f_i(w) = 1, \forall w \in U_i; v \leq w, \text{ when } f_i(v) = 1,\]

for \(i = 1\) or 2. Similarly, the other set of unclassified vectors may be reduced according to the following nestedness constraints:

\[f_1(w) = 1, \forall w \in U_1; v \leq w, \text{ when } f_2(v) = 1,\]

or the following nestedness constraints:

\[f_2(w) = 0, \forall w \in U_2; w \leq v, \text{ when } f_1(v) = 0.\]

Vectors are then repeatedly selected from the unclassified set until all of the vectors are classified by both functions (i.e., \(U_1 = U_2 = \{\}\)). As in Problem # 1, the key problem is to select “promising” vectors so as to reduce the total number of queries in this process.
2.3.1 Problem # 2.1: Sequentially Inferring a Pair of Nested Monotone Boolean Functions from Two Deterministic Oracles

This problem deals with applications where the costs associated with queries to one of the oracles far outweighs the costs associated with the queries to the other oracle, and situations where one has sequential access to the oracles. In such applications, it is beneficial to reconstruct the function associated with the least expensive oracle first, after which the reconstruction of the other function begins. In other words, two monotone Boolean functions are to be sequentially inferred from two oracles.

2.3.2 An Application of Problem # 2.1 to Breast Cancer Diagnosis

Breast cancer diagnosis will be used to illustrate the sequential inference problem. Figure 2.1 shows the two step sequential process in which breast cancer is diagnosed. In the first step, an X-ray image (called a mammogram) is studied by a radiologist who determines whether performing a biopsy is necessary. If the radiologist finds that a biopsy is not necessary, the diagnosis is concluded. Otherwise, a biopsy is performed to test for cancer.

![Figure 2.1 Illustration of sequential oracles in breast cancer diagnosis.](image)

The cost of the invasive biopsy procedure outweighs the cost of obtaining and analyzing a mammogram. Therefore, the radiologist can be viewed as the oracle governing function $f_1$, which should be completely restored first. The biopsy can be viewed as the oracle governing function $f_2$, whose restoration begins afterwards. The nature of the two functions $f_1$ and $f_2$ implies that they are nested (i.e., $f_1 \supseteq f_2$).

Consider the pair of nested monotone Boolean functions inferred by interviewing a radiologist in Kovalerchuk et al. (1996). The inferred function that describes their "biopsy subproblem" is defined as follows:

$$f_1(v) = v_1v_2 \lor v_3 \lor v_4v_5 \lor v_3v_4 \lor v_5,$$

where $f_1(v) = 1$ if a biopsy is recommended for a tumor with the features described by vector $v$, and $f_1(v) = 0$ otherwise. The inferred function that describes their "cancer subproblem" is defined as follows:

$$f_2(v) = v_1v_2 \lor v_3 \lor v_1v_3 \lor v_2v_3 \lor v_4v_5,$$

where $f_2(v) = 1$ if a tumor with the features described by vector $v$ is highly suspicious for malignancy, and $f_2(v) = 0$ otherwise. Here $v_i$ describes the $i$-th diagnostic feature which is 1 if it is "pro cancer" and 0 if it is "contra cancer". The five individual features are defined as follows:

$v_i = 1$ if amount and volume of calcifications is “pro cancer”, 0 otherwise,
$v_2 = 1$ if shape and density of calcifications is “pro cancer”, 0 otherwise,
$v_3 = 1$ if ductal orientation is “pro cancer”, 0 otherwise,
$v_4 = 1$ if comparison with previous exam is “pro cancer”, 0 otherwise, and
$v_5 = 1$ if associated findings is “pro cancer”, 0 otherwise.

It should be noted that the variables $v_1$ and $v_2$ are also monotone Boolean functions that have to be inferred prior to the inference of $f$. The details of this decomposition are temporarily left out for the purpose of simplifying this illustration. Section 2.5 addresses the issue of decomposition of variables and hierarchical inference, and provides detailed descriptions of the breast cancer diagnostic variables.

It should also be noted that the two functions $f_1$ and $f_2$ were both inferred by querying a radiologist. For the purpose of illustrating the sequential oracles, it is assumed here that the values of function $f_1$ are available via inquiries to the radiologist, and the values of function $f_2$ are available via the biopsy procedure. The interested reader is referred to Kovalerchuk et al. (1996) and Kovalerchuk et al. (2000a) for further information about this breast cancer application.

The inference process can be viewed as a dialogue between the oracles (the radiologist and the biopsy procedure) and the person posing the queries. The queries are described by using a vector of the form $(v_1 v_2 v_3 v_4 v_5 f_j)$, where $j = 1$ or 2 and $v_i = 0$ or 1 for $i = 1, 2, \ldots, 5$. Initially, the sets of unclassified vectors $U_1$ and $U_2$ are both equal to $\{0,1\}^5$. Since sequential inference only allows vectors from $U_2$ to be selected once $U_1$ is equal to $\{\}$, there are $2^5 = 32$ feasible queries to choose from.

Suppose the vector $(00010f_1)$ is selected for the first query. The radiologist is presented with the following query: “Is a biopsy recommended for a tumor with pro cancer feature in comparison with previous exam?” Once the radiologist answers “no” (i.e., $f_1(00010) = 0$), the 2 vectors of the form $(000v_40)$ can be classified as 0 or 1 for function $f_1$ according to the monotonicity constraints: $f_1(w) = 0$, $\forall w \in \{0,1\}^5: w \preceq (00010)$. This reduces the set of unclassified vectors to $U_1 = \{0,1\}^5 - \{(0 0 0 v_4 0): v_4 = 0 \text{ or } 1\}$. The 2 vectors of the form $(000v_40)$ can also be classified as 0 for function $f_2$ according to the nestedness constraints: $f_2(w) = 0$, $\forall w \in \{0,1\}^5: w \preceq (00010)$, since $f_1(00010) = 0$. This reduces the set of vectors unclassified by $f_2$ to $U_2 = \{0,1\}^5 - \{(0 0 0 v_4 0): v_4 = 0 \text{ or } 1\}$.

This leaves $64 - 4 = 60$ unclassified vectors after 1 query. Similar queries are repeated until there are no unclassified vectors left in $U_1$. Then, queries of the form $(v_1 v_2 v_3 v_4 v_5 f_2)$ are selected from $U_2$. When the set of unclassified vectors $U_2$ also becomes empty, the pair of nested monotone Boolean functions is completely inferred.

If all the combinations are to be queried by both oracles, $2^5 = 32$ queries to the radiologist and 32 biopsies need to be performed. However, if the nestedness and monotonicity assumptions are both properly utilized, the total number of queries can be significantly reduced.

2.3.3 Problem #2.2: Simultaneously Inferring a Pair of Nested Monotone Boolean Functions from a Single Three-valued Deterministic Oracle

This problem deals with applications where a single oracle knows both functions’ values. That is, the oracle answers each query with a pair of function values $(f_1, f_2) = (0,0), (1,0), (1,1)$. In other words, two monotone Boolean functions are to be simultaneously inferred from a single three-valued oracle. For this problem the two nested monotone Boolean functions are viewed as a single monotone function $f$ taking on the three values 0, 1, and 2, corresponding to $(f_1, f_2) = (0,0), (1,0), (1,1)$, respectively. Notice that $(f_1, f_2)$ cannot take on the values (0,1) due to the nestedness constraint $f_1 \geq f_2$. The single three-valued function is used to emphasize that the Boolean function values arrive in pairs, for each vector, from a single oracle.
This problem is similar to that of inferring a single monotone Boolean function in that a single oracle provides the answers. In contrast, some vectors may need to be evaluated by both oracles for Problems # 2.1 and # 2.3. In Problem # 2.2, vectors are not just considered classified or unclassified, but also partially classified. Some time into the inference process unclassified vectors may take on the values 0, 1 and 2 as function values while partially classified vectors may take on the values 0 and 1, or the values 1 and 2.

2.3.4 An Application of Problem # 2.2 to Record Linkage in Databases
The problem of merging a pair of databases with \( n \) common fields will be used to illustrate a three-valued oracle. Figure 2.2 shows a database administrator as an oracle who is asked whether a pair of records should be merged. The goal in the record linkage problem is to find which records in database A match the records in database B so that the newly created database does not have replications of the same records nor is missing any of the records from either of the two old databases. The interested reader is referred to Fellegi and Sunter (1969), Winkler (1995), or Judson (2001) for further details on the record linkage problem.

Record linkage can also be applied to a single database in order to avoid duplicate records. For example, when the terrorist attack took place at the World Trade Center in New York City on September 11, 2001, the number of reported missing people was around 6,500. Weeks later, that number shrunk to around 5,500, when it was discovered that some of the reports were duplicates of the same people.

Linking records across databases generalizes the single database problem by looking at fields that are common to the databases. Suppose the two databases have \( n \) common fields. Let variable \( v_i \) be defined as 1 if a pair of records from the two databases have the same value on the \( i \)-th field, and 0 otherwise, for \( i = 1, 2, ..., n \).

Consider a database A defined with the following fields: first name, last name, state of residence, email address, age, gender, salary, and a database B defined on the following fields: first name initial, middle name initial, last name, address, zip code, phone number, occupation, age, gender. For a pair of records \( r_A \) and \( r_B \), from databases A and B, respectively, the five variables can then be defined by the fields that are common to databases A and B, as follows:

\[
\begin{align*}
v_1 &= 1, \text{ if } \text{first name initial}(r_A) = \text{first name initial}(r_B), 0 \text{ otherwise}, \\
v_2 &= 1, \text{ if } \text{last name}(r_A) = \text{last name}(r_B), 0 \text{ otherwise}, \\
v_3 &= 1, \text{ if } \text{state of residence}(r_A) = \text{state of residence}(r_B), 0 \text{ otherwise},
\end{align*}
\]
where the common values of a pair of records are described by the vector

That is, the underlying functions to be inferred are given as follows:

Suppose, that a pair of records should be merged if they have the same values on at least:

Further, suppose that a pair of records should be given further consideration if they do not satisfy the conditions for merging and have the same values on at least:

That is, the underlying functions to be inferred are given as follows:

As an example consider the following record from database A: (Emily, Stark, MN, estrk@yehaa.net, 47, Female, 53k) and the following record from database B: (E, I, Stark, PO Box 3451, 55133, 612-345-6789, Teacher, 45, Male). These records have the same first name initial (i.e., \(v_1 = 1\)), the same last name (i.e., \(v_2 = 1\)), the same state of residence (i.e., \(v_3 = 1\), not the same age (i.e., \(v_4 = 0\)), and not the same gender (i.e., \(v_5 = 0\)). This pair of records are described by the vector \(v = (11100)\), for which \(f_1(v) = 1\) and \(f_2(v) = 0\). That is, they should be given further consideration (i.e., to be checked for errors, etc).

The inference process can be viewed as a dialogue between the database administrator and the person posing the queries. The queries are described by using a vector of the form \((v_1 v_2 v_3 v_4 v_5)\), where \(v_i = 0\) or \(1\) for \(i = 1, 2, \ldots, 5\). Initially, the sets of unclassified vectors \(U_1\) and \(U_2\) are both equal to \(\{0,1\}^5\), and there are \(2^5 = 32\) possible queries to choose from.

Selecting the vector \((11100)\) corresponds to the query: “Should a pair of records be merged if they have the same first name initial, last name and state of residence?” Once the database administrator answers “maybe” (i.e., \(f_1(11100) = 1\) and \(f_2(11100) = 0\)), the 4 vectors of the form \((11 1 v_4 v_5)\) can be classified as 1 for function \(f_1\) according to the monotonicity constraints: \(f_1(w) = 1\), \(\forall w \in \{0,1\}^5\); \((11100) \leq w\). Furthermore, the 8 vectors of the form \((v_1 v_2 v_3 0 0)\) can be classified as 0 for function \(f_2\) according to the monotonicity constraints: \(f_2(w) = 0\), \(\forall w \in \{0,1\}^5\); \(w \leq (11100)\).

This reduces the set of unclassified vectors to \(U_1 = \{0,1\}^5 - \{(11111)\}: v_i = 0\) or 1, for \(i = 4\) and 5\) and \(U_2 = \{0,1\}^5 - \{(v_1 v_2 v_3 0 0): v_i = 0\) or 1, for \(i = 1, 2, 3\}\.

The above considerations mean that 28 (= 32 - 4) and 24 (= 32 - 8) unclassified vectors are left in \(U_1\) and \(U_2\), respectively, after just 1 query. Similar queries are repeated until there are no unclassified vectors common to the sets \(U_1\) and \(U_2\). From that point and on, vectors are selected from either \(U_1\) or \(U_2\). Once no unclassified vectors are left in either \(U_1\) or \(U_2\), the pair of nested monotone Boolean functions is completely inferred.
2.3.5 Problem # 2.3: Simultaneously Inferring a Pair of Nested Monotone Boolean Functions from Two Unrestricted, Deterministic Oracles

This problem deals with applications where the costs associated with access to the two oracles are the same, and situations that allow for switching between the two oracles intermittently. At any inference step of such applications, it does not matter which oracle is queried. In other words, two monotone Boolean functions are to be simultaneously inferred with unrestricted access to two oracles.

This problem is similar to Problem # 2.1, in that two oracles are queried separately. Unlike Problem # 2.1, no restrictions are put on the manner in which the two oracles are queried. At each inference step, a vector can be submitted to any of the two oracles. In this sense, this is the least restrictive of the three problems, and it is therefore expected that this approach will be the most efficient one.

2.3.6 An Application of Problem # 2.3 to a College Acceptance Policy

Consider the evaluation process used for accepting students into a particular college illustrated in Figure 2.3. When a selection committee evaluates student applications they often place them into three ordered categories based on some performance variables such as their Grade Point Average (GPA), standardized test scores, quality of an essay, recommendations, etc. The bottom category consists of students that are not to be accepted. The top category consists of students that are to be accepted, while an intermediate category consists of applicants that need to be considered more carefully.

Suppose a college dean wishes to establish guidelines that the selection committee is to follow when placing applicants into these three categories. To that end, the dean asks two professors from the college, one that is known to be lenient and another that is known to be strict, to evaluate some applications. The lenient professor’s acceptance function is given by $f_1$, while the strict professor’s acceptance function is given by $f_2$. Here, $f_1(v) = 1$ if a student with criteria described by vector $v$ is accepted by the lenient professor, and $f_1(v) = 0$ if it is not accepted by the lenient professor. Also, $f_2(v) = 1$ if a student with criteria described by vector $v$ is accepted by the strict professor, and $f_2(v) = 0$ if it is not accepted by the strict professor.
It is assumed that a student with criteria \( v \) that is accepted by the strict professor will also be accepted by the lenient professor (i.e., \( f_2(v) = 1 \) implies that \( f_1(v) = 1 \)). It is also assumed that, if the student is not accepted by the lenient professor, the student will neither be accepted by the strict professor (i.e., \( f_1(v) = 0 \) implies that \( f_2(v) = 0 \)). This describes the nestedness assumption made of the two professors’ acceptance functions: \( f_1 \geq f_2 \).

The college dean has two oracles (i.e., the two professors) that govern the two functions, where the order in which the oracles are queried is not restricted. That is, any application can be evaluated by either of the two professors at any given time during the inference process.

Consider the following four simplified acceptance criteria for the college as follows:
\[
\begin{align*}
v_1 &= 1 \text{ for a high GPA}, \ 0 \text{ otherwise}, \\
v_2 &= 1 \text{ for a high Graduate Record Examination (GRE) score}, \ 0 \text{ otherwise}, \\
v_3 &= 1 \text{ for an essay of high quality}, \ 0 \text{ otherwise, and} \\
v_4 &= 1 \text{ for good recommendations}, \ 0 \text{ otherwise}.
\end{align*}
\]

Suppose, the lenient professor will accept students with the following minimum requirements:
\[\begin{align*}
(a \text{ high GPA}), \text{ or } \\
(a \text{ high GRE score and good recommendations}), \text{ or } \\
(an \text{ essay of high quality and good recommendations}),
\end{align*}\]
while the strict professor will accept students with the following minimum requirements:
\[\begin{align*}
(a \text{ high GPA, a high GRE score and good recommendations}), \text{ or } \\
(a \text{ high GPA, an essay of high quality and good recommendations}),
\end{align*}\]
That is, the two functions to be inferred (and are thus considered to be currently unknown) are:
\[
\begin{align*}
f_1(v) &= v_1 \lor v_2 v_4 \lor v_3 v_4, \text{ and} \\
f_2(v) &= v_1 v_2 v_4 \lor v_1 v_3 v_4.
\end{align*}
\]

The students that satisfy the strict professor’s minimum requirements will be accepted. The students that satisfy the lenient professor’s minimum requirements but not the strict professor’s are given further consideration. The students that do not satisfy either professor’s minimum requirements are not accepted. The queries are performed in a similar fashion as for Problem # 2.1. However, a vector from either \( U_1 \) or \( U_2 \) can be selected at any time of the inference process.

### 2.4 Problem # 3: Inferring a Monotone Boolean Function from a Stochastic Oracle

This problem is identical to Problem # 1, except that the membership values are now stochastic in nature. As in Problem # 1, vectors are selected from \( \{0,1\}^n \) and are submitted to an oracle as membership queries. Unlike Problem # 1, it is assumed that the oracle misclassifies each vector \( v \) with an unknown probability \( q(v) \in (0, \frac{1}{2}) \). That is, for a given monotone Boolean function \( f \), the oracle returns 1 for vector \( v \) with probability \( p(v) = q(v) \times (1 - f(v)) + (1 - q(v)) \times f(v) \), and it returns 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 one half.

The stochastic inference problem involves estimating the misclassification parameter \( q(v) \) for each vector \( v \), as well as reconstructing the underlying function \( f \). In this dissertation, these two tasks are based on a maximum likelihood framework. A 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 a function \( f^* \) are the estimated misclassification probabilities which are denoted by \( q^*(v) \) for each vector \( v \).

The inference process consists of two steps that are repeated successively. In the first step, a vector is submitted to the oracle as a query. After a vector’s function value is provided by the
oracle, both \(q^*(v)\) and \(f^*\) may have to be updated, according to the following monotonicity property:
\[ p(v) \leq p(w) \text{ if and only if } v \leq w, \forall v, w \in \{0,1\}^n. \]  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 \(f\). In other words, the underlying function is considered completely inferred when the maximum likelihood ratio for the inferred function, denoted by \(\lambda(f^*)\), reaches a value that is close to 1. Again, the key problem is to select “promising” vectors so as to reduce the total number of queries required in this process.

### 2.5 Hierarchical Decomposition of Variables

The variables used for the different applications described in this chapter are simplified in the illustrations. For example, a college applicant variable \(an\ essay\ of\ high\ quality\), (from Section 2.3.6) may consist of several factors such as whether the essay demonstrated good writing skills, a good vision, and the interests match the college’s program. These variables have to be defined prior to the inference of the college acceptance function.

In some applications, the variables may be monotone Boolean functions defined on set of Boolean diagnostic variables at a lower level. Kovalerchuk et al. (1996) decomposed the five breast cancer diagnostic variables (from Section 2.3.2.) in a hierarchical manner as follows. The first variable \(v_1\) is defined as 1 if the amount and volume of calcifications is “pro cancer”, and 0 if it is “contra cancer”. In reality, this variable was inferred (through queries to the radiologist) as the following monotone Boolean function:
\[
v_1(x_1, x_2, x_3) = x_2 \lor x_1 x_3.
\]
Here, the extra variables are defined as follows:
\[
\begin{align*}
x_1 &= 1 \text{ if the } \text{number of calcifications/cm}^2 \text{ is “large”}, \ 0 \text{ if “small”}, \\
x_2 &= 1 \text{ if the } \text{volume of calcifications (cm}^3\text{) is “small”}, \ 0 \text{ if “large”}, \text{ and} \\
x_3 &= 1 \text{ if the } \text{total number of calcifications is “large”}, \ 0 \text{ if “small”}.
\end{align*}
\]
The second variable \(v_2\) is defined as 1 if the shape and density of calcifications is “pro cancer”, and 0 if it is “contra cancer”. In reality, this variable was inferred (through queries to the radiologist) as the following monotone Boolean function:
\[
v_2(x_4, x_5, x_6, x_7, x_8) = x_4 \lor x_5 \lor x_6 x_7 x_8.
\]
Here, the extra variables are defined as follows:
\[
\begin{align*}
x_4 &= 1 \text{ if the } \text{irregularity in the shape of individual calcifications is “marked”}, \ 0 \text{ if “mild”}, \\
x_5 &= 1 \text{ if the } \text{variation in the shape of calcifications is “marked”}, \ 0 \text{ if “mild”}, \\
x_6 &= 1 \text{ if the } \text{variation in the size of calcifications is “marked”}, \ 0 \text{ if “mild”}, \\
x_7 &= 1 \text{ if the } \text{variation in the density of calcifications is “marked”}, \ 0 \text{ if “mild”}, \text{ and} \\
x_8 &= 1 \text{ if the } \text{density of calcifications is “marked”}, \ 0 \text{ if “mild”}.
\end{align*}
\]

In general, one can construct a hierarchy of the sets of variables, where each set of variables corresponds to an independent inference problem. Figure 2.4 shows this hierarchy for the breast cancer diagnostic variables. The upper level consists of the set \(\{v_1, v_2, v_3, v_4, v_5\}\) which is linked to the sets of variables \(\{x_1, x_2, x_3\}\), and \(\{x_4, x_5, x_6, x_7, x_8\}\) at the lower level. Here, the variables \(v_1\) and \(v_2\) have to be defined before the inference problem defined on the set variables \(\{v_1, v_2, v_3, v_4, v_5\}\) can begin. In general, the inference problems at the lower level have to be completed before the inference problems at the upper levels can begin.

The breast cancer inference problem is defined on the set of Boolean variables \(\{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, v_1, v_2, v_3, f\}\). This problem includes a total of \(2^{12} = 4,096\) vectors to choose from. However, it can be approached hierarchically, as three independent problems defined on the sets \(\{x_1, \ldots, x_8\}\), \(\{v_1, v_2\}\), and \(\{f\}\).
Figure 2.4 Hierarchical decomposition of the variables for the breast cancer diagnosis application.

\{v_1, v_2, v_3, v_4, v_5\}
\{x_1, x_2, x_3\}  \{x_4, x_5, x_6, x_7, x_8\}

\{x_2, x_3\} \{x_4, x_5, x_6, x_7, x_8\} \{v_1, v_2, v_3, v_4, v_5, f_i\}, respectively. These problems include a total of \(2^3 + 2^5 + 2^6 = 104\) possible vectors to choose from. The hierarchical approach to this problem reduces the number of possible vectors to choose from by a factor of \(4,096/104 \approx 39.4\).

Please notice that a single monotone Boolean function is to be inferred for each of the sets \{x_1, x_2, x_3\}, and \{x_4, x_5, x_6, x_7, x_8\}. This corresponds to Problem # 1 defined on the sets \{0,1\}^3 and \{0,1\}^5, respectively. In contrast, a pair of nested monotone Boolean functions defined on the set \{v_1, v_2, v_3, v_4, v_5\} are to be sequentially inferred. This corresponds to Problem # 2.1 and includes the query domain \{0,1\}^6.

Similar hierarchies can be constructed for the other applications described in this chapter. As an example consider the college acceptance policy application described in Section 2.3.6. The variable corresponding to the GRE score may be defined in terms of the individual verbal, quantitative and analytical scores. That is, the variable \(v_1\) may be defined as a monotone Boolean function defined on the following variables:
\[x_1 = 1 \text{ for a high verbal GRE score}, \ 0 \text{ otherwise},\]
\[x_2 = 1 \text{ for a high quantitative GRE score}, \ 0 \text{ otherwise},\]
\[x_3 = 1 \text{ for a high analytical GRE score}, \ 0 \text{ otherwise}.
\]
For example, a quantitative score of 700 or higher (out of 800) may be considered as a high score, implying that the variable \(x_2\) is equal to 1.

Similar decompositions can be created for the variables \(v_2, v_3,\) and \(v_4,\) as follows. The variable \(v_2\) defined in terms of the recommendations may be a monotone Boolean function defined on the following variables:
\[x_4 = 1 \text{ if highly motivated}, \ 0 \text{ otherwise},\]
\[x_5 = 1 \text{ if creative}, \ 0 \text{ otherwise},\]
\[x_6 = 1 \text{ if bright/smart}, \ 0 \text{ otherwise}.
\]
For example, if the people who wrote recommendations refer to the student as highly motivated, then the variable \(x_4\) is equal to 1. The variable \(v_3\) defined in terms of the essay may be a monotone Boolean function defined on the following variables:
\[x_7 = 1 \text{ for good writing skills}, \ 0 \text{ otherwise},\]
\[x_8 = 1 \text{ if the interests match the college program}, \ 0 \text{ otherwise},\]
\[x_9 = 1 \text{ for a good vision}, \ 0 \text{ otherwise}.
\]
The variable \(v_4\) defined in terms of the GPA may be a function defined on the following variables:
\[x_{10} = 1 \text{ for a high science GPA}, \ 0 \text{ otherwise},\]
\[x_{11} = 1 \text{ for a high arts GPA}, \ 0 \text{ otherwise},\]
\[x_{12} = 1 \text{ for a high language GPA}, \ 0 \text{ otherwise}.
\]
Here, the variable \(x_{10}\) can be further decomposed into the following variables:
\[ y_1 = 1 \text{ for a high applied math GPA}, 0 \text{ otherwise}, \]
\[ y_2 = 1 \text{ for a high theoretical math GPA}, 0 \text{ otherwise}, \]
\[ y_3 = 1 \text{ for a high physics or chemistry GPA}, 0 \text{ otherwise}. \]

Notice that the monotonicity assumption holds for each of the subsets of variables. For example, a high applied math GPA is more likely to result in a high GPA, than a low applied math GPA. Therefore, a high applied math GPA is more likely to result in acceptance than a low math GPA, since a high GPA is more likely to result in acceptance.

Figure 2.5 shows the hierarchical decomposition of these variables. The inference problem defined on the set of Boolean variables \( \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, y_1, y_2, y_3, x_{11}, x_{12}, f_i\} \) consists of \( 2^{15} = 32,768 \) possible vectors to choose from. Fortunately, it can be approached hierarchically, as six independent problems. Five are in the form of Problem # 1 defined on the set \( \{0,1\}^3 \), and one is in the form of Problem # 2.3 with the query domain \( \{0,1\}^5 \). Combined, these sets consist of \( 5 \times 2^3 + 2^5 = 72 \) vectors to choose from. The hierarchical approach to this problem reduces the number of possible vectors to choose from by a factor of \( 32,768/72 \approx 455 \).

As another illustrative example consider the record linkage application described in Section 2.3.3. Databases often contain errors. For example, the last name McGinniss may correspond to a person with the last name McGuinness. The variable \( v_2 \) is defined as 1 if a pair of last names are the same, and 0 otherwise. Hierarchical decomposition allows for similar last names to be matched, even when they are not an exact match. For example, the variable \( v_2 \) can be defined in terms of the following monotone Boolean function:
\[ v_2(x_1, x_2) = x_1 x_2, \]
where the variables are given by:
\[ x_1 = 1 \text{ if the last names are unique}, \]
\[ x_2 = 1 \text{ if there is an agreement in a large number of the characters}, \]
For example, the last names McGinniss and McGuinness may result in \( x_1 = 1 \) and \( x_2 = 1 \), for which \( v_3 = 1 \). Notice here that the monotonicity assumption holds since the more unique (i.e., rarer) a last name is and the greater agreements in the number of characters, the more likely the last names are to be the same.

It should also be noted that for any of the inference problems considered in this dissertation, it is a strict requirement that function(s) at the uppermost level is (are) defined on a set of Boolean variables. Therefore, each set of variables at the lower level(s) has to correspond to a single Boolean function. Otherwise, one or more of the variables at higher levels will not be Boolean. As a
consequence, a single two-valued oracle must be used for each of the inference problem(s) at the lower level(s) in Problems # 2.1, # 2.2, and # 2.3.

In the breast cancer application it seems reasonable to use the radiologist as the oracle when inferring the functions at the lower level. In the college acceptance policy, it seems reasonable to consider the lenient and the strict professor together as a single oracle for the inference problems at the lower levels. That is, each query is answered with a Boolean value that is in agreement with both professors. For the record linkage application, the database administrator is the sole oracle who should provide Boolean answers in the inference problems at the lower level(s).

The reduction in the query domain is not the only benefit of the hierarchical decomposition of the inference problems. For example, a tumor with characteristics described by the 11 element vector \((x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, v_3, v_4, v_5)\) may be hard for a radiologist to evaluate. In contrast, the queries for the three independent inference problems are of the much simpler forms \((x_1, x_2, x_3)\), \((x_4, x_5, x_6, x_7, x_8)\), and \((v_1, v_2, v_3, v_4, v_5)\). A college applicant given by the 14 element vector \((x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, y_1, y_2, y_3, x_{11}, x_{12})\) may be hard for the professors to evaluate. In contrast, the queries for the six independent inference problems are of the much simpler forms \((y_1, y_2, y_3)\), \((x_1, x_2, x_3)\), \((x_4, x_5, x_6)\), \((x_7, x_8, x_9)\), \((x_{10}, x_{11}, x_{12})\), and \((v_1, v_2, v_3, v_4)\).
CHAPTER 3. BACKGROUND INFORMATION

In this chapter, the literature pertaining to the different aspects of Problems # 1, # 2, and # 3 is reviewed. Definitions and properties of monotone Boolean functions that are utilized throughout this dissertation are presented in Section 3.1. The main existing approaches to Problems # 1, # 2, and # 3 are introduced in Sections 3.2, 3.3, and 3.4, respectively. In Section 3.5, several approaches to modeling the stochasticity of the oracle in Problem # 3 are discussed. The unequal probability sampling framework that is described in Section 3.6 will be used to unbiasedly evaluate the query complexity of the algorithms developed for Problems # 1 and # 2.

3.1 Some Key Properties of Monotone Boolean Functions

Let $V$ denote a finite set of vectors defined on $n$ variables. A vector $v \in V$ is said to precede another vector $w \in V$, denoted by $v \leq w$, if and only if (iff) $v_i \leq w_i$ for $i = 1, 2, ..., n$. Here, $v_i$ and $w_i$ denote the $i$-th element of vectors $v$ and $w$, respectively. Similarly, a vector $v \in V$ is said to succeed another vector $w \in V$, 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 $v > w$ (or $v < w$, respectively). If a vector $v$ either precedes or succeeds $w$, they are said to be related (or comparable).

An ordered set of related vectors $v^1 \leq v^2 \leq ... \leq v^p$ is sometimes called a chain, while an antichain (or layer) consists of a set of mutually unrelated vectors. When a set of vectors is partitioned into as few layers as possible, a layer partition is formed. Similarly, when a set of vectors is partitioned into as few chains as possible, a chain partition is formed. For a particular layer partition, the layers can be ordered as $L^1, L^2, ..., L^r$ so that a vector $v^i \in L^j$ cannot succeed another vector $v^j \in L^j$, if $i < j$.

Let $\{0,1\}^n$ denote the set of vectors defined on $n$ Boolean variables. 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. An example is the symmetric chain partition used by Hansel (1966) and Sokolov (1982) as described in Section 3.2.

A directed graph $G$ is often written in the form $(V, E)$, where $V$ denotes its set of vertices, and $E$ denotes its set of directed edges. Here, a directed edge from vertex $v$ to vertex $w$ is written as $(v, w)$. A directed graph $(V, E)$ is called cyclic if it has a sequence of edges that starts and ends with a vector $v$: $(v, v^1), (v^1, v^2), ..., (v^r, v) \in E$.

Figure 3.1 shows a partially ordered set (or poset for short). In general, posets can be formed by a set of vectors $V$ together with the precedence relation $\preceq$, and are written as $(V, \preceq)$. A poset can be viewed as a directed graph where each vertex corresponds to a vector and each directed edge $(v, w)$ represents the precedence relation $v \preceq w$.

When drawing a poset as a directed graph, its edges’ directions are often omitted without loss of information. The graph of a poset is acyclic and so all the directions can be forced upwards on a page by ordering the vertices by layers, as in Figure 3.1. The directions are included in this figure because they are useful for describing the maximum flow algorithms in Sections 5.3.1 and 5.3.2.

Precedence relations that are transitively implied by other relations are considered redundant. For example, the precedence relation $(000) \preceq (110)$ is redundant because it is implied by the two precedence relations $(000) \preceq (100)$ and $(100) \preceq (110)$. For the purpose of reducing storage and simplifying the visualization of posets, redundant precedence relations are generally omitted, as in Figure 3.1.
Two posets $P_1$ and $P_2$ are said to be isomorphic if there exists a one-to-one mapping of the vectors in $P_1$ to the vectors in $P_2$, where the precedence relations are preserved. That is, if $v^1 \rightarrow v^2$ and $w^1 \rightarrow w^2$, then $v^1 \preceq w^1$ iff $v^2 \preceq w^2$, $\forall v^1, w^1 \in P^1$ and $v^2, w^2 \in P^2$. For example, the poset formed by the vectors \{0000, 1001, 0100\} is isomorphic to the poset formed by the vectors \{1110, 1100, 1101\}. Here, one possible isomorphic mapping is as follows: (0000) $\rightarrow$ (1100), (1001) $\rightarrow$ (1110) and (0100) $\rightarrow$ (1101).

A poset $P$ is said to be symmetric if its layer partition $L^1, L^2, ..., L^r$ is unique, and there exists a one-to-one layerwise mapping $P \rightarrow P$, and all such mappings are isomorphic. Here, a layerwise mapping occurs when each vector $v^i \in L^i$ maps to a different vector $w^i$ within the same layer $L^i$, for $i = 1, 2, ..., r$. For example, the poset ({0,1}$^n$, $\preceq$) and the chain poset ({1, 2, ..., $r$}, $\preceq$) are both symmetric.

The dual of a poset $P$ is the poset $P^d$ resulting from reversing all the precedence relations in $P$. That is, $(V, \preceq)^d = (V, \succeq)$. For example, the dual of the poset formed by the vectors {0000, 1001, 0100} can be formed by the vectors {1111, 0110, 1011}. A poset $(V, \succeq)$ is called connected if for any pair of vectors $v$ and $w \in V$, either $v$ and $w$ are related, or there exists a sequence of vectors $v^1, v^2, ..., v^r \in V$ for which all the pairs $(v, v^1), (v^1, v^2), ..., (v^r, w)$ are related.

A Boolean function defined on the set of vectors {0,1}$^n$ is simply a mapping to {0,1}. A monotone Boolean function $f$ is called non-decreasing iff $f(v^*) = 0$ and $f(v) = 1$ $\forall v \in \{0,1\}^n : v \preceq v^*$. Similarly, a vector $v^*$ is called a lower unit if $f(v^*) = 1$ and $f(v) = 0$ $\forall v \in \{0,1\}^n : v \preceq v^*$. 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 in Conjunctive Normal Form (CNF) using the AND, OR, and NOT operations, denoted by $\land$, $\lor$, and $\neg$, respectively. A DNF or a CNF representation is minimal if removing any of its clauses results in a different mapping {0,1}$^n \rightarrow \{0,1\}$. For any monotone Boolean function $f$ there is a one-to-one relationship
between its lower units and its minimal DNF representation, as follows:

\[ f(v_1, v_2, \ldots, v_n) = \bigvee_{w \in \text{L}(f) \setminus \text{L}_c} (\bigwedge v_j). \]

Similarly, there is a one-to-one relationship between the upper zeros of a monotone Boolean function \( f \), and its minimal CNF representation as follows:

\[ f(v_1, v_2, \ldots, v_n) = \bigwedge_{w \in \text{U}(f) \setminus \text{L}_c} (\bigvee v_j). \]

For example, the monotone Boolean function defined by its lower units \{110, 101\} can be written in minimal DNF as \( v_1 v_2 \lor v_1 v_3 \). The corresponding upper zeros are \{011, 100\} and its minimal CNF representation is \( v_1 (v_2 \lor v_3) \). Often the operation \( \land \) is excluded when writing out Boolean functions, as in the previous two examples. Since the lower units and upper zeros are unique to a monotone Boolean function, so are its minimal representations in DNF and CNF. Another nice property of monotone Boolean functions is that they can be written in minimal CNF or DNF without using the NOT operation.

The set of all monotone Boolean functions defined on \{0,1\}^n is denoted by \( M_n \). For example, the set of all monotone Boolean functions defined on \{0,1\}^2 is given by \( M_2 = \{F, v_1 v_2, v_1, v_2, v_1 \lor v_2, T\} \). Here the functions \( T \) and \( F \) are defined by \( f(v) = 1, \forall v \in \{0,1\}^n \), and \( f(v) = 0, \forall v \in \{0,1\}^n \), respectively.

Let \( m(f) \) denote the number of border vectors associated with a Boolean function \( f \). It is well known (e.g., 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:

\[ \max_{f \in M_n} m(f) = \left( \begin{array}{c} n \\ \lfloor n/2 \rfloor \end{array} \right) + \left( \begin{array}{c} n \\ \lfloor n/2 \rfloor + 1 \end{array} \right). \]

The borders of any monotone Boolean function \( f \) are the only vectors that require evaluations in order to completely reconstruct the function. Therefore, the value of \( m(f) \) works as a lower bound on the number of queries for Problem # 1. The worst case query complexity for Problem # 1, is least as much as \( \max m(f) \) given by the equation above, but less than or equal to \( 2^n \). In other words, the worst case query complexity is exponential in \( n \).

The number of monotone Boolean functions defined on \{0,1\}^n is denoted by \( \Psi(n) \). That is, \( \Psi(n) = |M_n| \), where \(|X|\) is the number of elements in the set \( X \). All of the known values for \( \Psi(n) \) are given in Table 3.1. 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):

\[ \Psi(n) \sim \begin{cases} \binom{n}{n/2} e^{n/2 - 1} \left( \frac{1}{2^n} \frac{n!}{2^{n/2}} \frac{1}{2^{n/2}} \right), & \text{for even } n. \\ \binom{n}{n/2 - 1/2} e^{n/2 - 3/2} \left( \frac{1}{2^{n/2 - 1/2}} \frac{n!}{2^{n/2}} \frac{n/2 - 1/2}{2^{n/2}} \right) \cdot \left( \frac{1}{2^{n/2 - 1/2}} \frac{n/2 - 1/2}{2^{n/2}} \right), & \text{for odd } n. \end{cases} \]

Many monotone Boolean functions are of similar form. For example, by permuting the variables in the function \( v_1 \lor v_2 v_3 \), the two functions \( v_2 \lor v_1 v_3 \) and \( v_3 \lor v_1 v_2 \) are obtained. Also, by switching the roles of the operations \( \lor \) and \( \land \) in the function \( v_1 v_2 v_3 \), the function \( v_1 \lor v_2 \lor v_3 \) is obtained.
up a poset and the functions make up another poset \{0,1\}. Multiplying \{0,1\} is isomorphic to \{0,1\}.

Monotone Boolean functions defined on \{0,1\} 3. This reduces the computational burden of the associated simulations performed in Section 6.3 by a factor of \(\Psi(3)/6 = 3.33\).

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 for Problem # 3 is the same for similar functions, when certain randomness precautions are taken. The six non-similar functions \(F, v_1v_2v_3, v_1v_2 \lor v_1v_3, v_2v_3, v_1 \lor v_2, v_1 \lor v_3, v_2 \lor v_1\), \(v_1v_2 \lor v_1v_3, v_2v_3, v_1 \lor v_2, v_1 \lor v_3, v_2 \lor v_1\), \(v_1v_2 \lor v_1v_3, v_2v_3, v_1 \lor v_2, v_1 \lor v_3, v_2 \lor v_1\), \(v_1v_2 \lor v_1v_3, v_2v_3, v_1 \lor v_2, v_1 \lor v_3, v_2 \lor v_1\), \(v_1v_2 \lor v_1v_3, v_2v_3, v_1 \lor v_2, v_1 \lor v_3, v_2 \lor v_1\), and \(v_1v_2 \lor v_1v_3, v_2v_3, v_1 \lor v_2, v_1 \lor v_3, v_2 \lor v_1\), represent the entire class of similar monotone Boolean functions defined on \(\{0,1\}\). This reduces the computational burden of the associated simulations performed in Section 6.3 by a factor of \(\Psi(3)/6 = 3.33\).

A pair of monotone Boolean functions \(f_1\) and \(f_2\) are called nested when the following relationship holds: \(f_1(v) \geq f_2(v)\) (or \(f_1(v) \leq f_2(v)\)), \(\forall v \in \{0,1\}^n\). The case when \(f_1 \geq f_2\) is addressed in this dissertation as analogous results hold for the case when \(f_1 \leq f_2\). In other words, if \(f_2(v)\) is equal to 1, then \(f_1(v)\) must also be equal 1, and if \(f_1(v)\) is equal to 0, then \(f_2(v)\) must also be equal 0, for any vector \(v\). The latter definition gives meaning to the word nested, while the more succinct definition \(f_1 \geq f_2\) will be used throughout the dissertation.

The number of pairs of nested monotone Boolean functions defined on \(\{0,1\}^n\) is simply \(\Psi(n+1)\). This fact can be observed by constructing the poset connecting two posets \(P_1 = (\{0,1\}^n, \leq)\) and \(P_2 = (\{0,1\}^n, \leq)\) associated with functions \(f_1\) and \(f_2\) respectively, by adding the edges corresponding to the precedence relations \(f_1(v) \geq f_2(v), \forall v \in \{0,1\}^n\). The operation of constructing the new poset can be viewed as poset multiplication, denoted by \(\otimes\), where the domain \(\{0,1\}^n\) makes up a poset and the functions make up another poset \(\{0,1\}\). Multiplying \(\{0,1\}^n\) by \(\{0,1\}\) creates \(\{0,1\}^{n+1}\), as the example for \(n = 2\) in Figure 3.2 shows.

Poset multiplication can be generalized to any set of vectors \(V\) and any relation between the functions \(f_1, f_2, \ldots, f_m\). As an example consider the set of vectors \(V = \{1, 2, 3\}\), and three functions that satisfy the following relationships \(f_1 \leq f_2 \leq f_3\). Here, the set of vectors makes up a chain poset of length 3, and the functions make up a chain poset of length 3. The new poset created by multiplying the two posets is a 3x3 matrix poset.

### 3.2 Existing Approaches to Problem # 1

Let \(\Phi(A, f)\) denote the number of queries performed by an algorithm \(A\), when reconstructing the monotone Boolean function \(f\). A Teacher can be thought of as an inference algorithm that knows the
function ahead of time. It simply verifies that the function is correct by querying only the border vectors. Thus,

$$\varphi(Teacher, f) = m(f), \forall f \in M_n.$$ 

Please recall that $m(f)$ denotes the number of border vectors associated with a function $f$.

For any monotone Boolean function inference algorithm $A$, the value $m(f)$ can be considered as a lower bound on the number of queries. Thus,

$$\varphi(A, f) \geq m(f), \forall f \in M_n.$$ 

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 Section 3.1, for all monotone Boolean functions defined on $\{0,1\}^n$. One realization of this is based on partitioning the set of vectors into chains as described in Hansel (1966). Figure 3.3 presents a simple algorithm called GENERATE-HANSEL-CHAINS($n$) that can be used to generate this symmetric chain partition. This algorithm is used here to illustrate how the chains can be generated, while a more efficient algorithm can be found in Sokolov (1982).

The algorithm GENERATE-HANSEL-CHAINS($n$) starts with the set of chains in the first dimension $\{\{0, 1\}\}$ and grows the chains recursively for each additional dimension up to $n$. The key idea is to generate a pair of chains for the $(i+1)$-th dimension from each chain containing two or more vectors $\{v_1, v_2, ..., v_k\}$ of the $i$-th dimension. The first chain is made up of the vectors $\{(v_1, 0), (v_2, 0), ..., (v_k, 0), (v_k, 1)\}$ as shown in lines 5-11 in Figure 3.3. When $k > 1$, the second chain is made up of the vectors $\{(v_1, 1), (v_2, 1), ..., (v_k, 1)\}$ as shown in lines 13-17 in Figure 3.3.

The Hansel chains in two dimensions are: $\{\{00, 10, 11\}, \{01\}\}$, in three dimensions they are: $\{\{000, 100, 110, 111\}, \{001, 101\}, \{010, 011\}\}$, in four dimensions they are: $\{\{0000, 1000, 1100, 1110, 1111\}, \{0001, 1001, 1101\}, \{0010, 1010, 1011\}, \{0111\}, \{0100, 0110, 0111\}, \{0101\}\}$. In five dimensions they are: $\{\{00000, 10000, 11000, 11100, 11110, 11111\}, \{00001, 10001, 11001, 11101\}, \{00010, 10010, 11010, 11011\}, \{00101, 10101\}, \{00110, 00111\}, \{01000, 01100, 01110, 01111\}, \{01001, 01101, 01110, 01111\}, \{01010, 01011\}\}$.

Here, there are 2, 3, 6, and 10 Hansel chains in dimensions 2, 3, 4, and 5, respectively. In general, there are a total of $\left(\begin{array}{c} n \\ k \end{array}\right)$ chains in $n$ dimensions. An inference algorithm that searches these chains in increasing length is referred to as Hansel’s algorithm.

A key property of the Hansel chains is that once the function values are known for all the vectors in the chains of length $k$, the function values are unknown for at most two vectors in each
Figure 3.3 The algorithm used to generate the Hansel chains in dimension $n$.

```plaintext
GENERATE-HANSEL-CHAINS(n)
1  C_1 = \{(0), (1)\}
2  for i = 2, 3, ..., n
3    C_i = \{
4      for each c \in C_i
5        c_{\text{new}} = \{
6          for each v \in c
7            v_{\text{old}} = v
8            v_{\text{new}} = (v_{\text{old}}, 0)
9            c_{\text{new}} = \{c_{\text{new}}, v_{\text{new}}\}
10           v_{\text{new}} = (v_{\text{old}}, 1)
11           c_{\text{new}} = \{c_{\text{new}}, v_{\text{new}}\}
12        c_{\text{new}} = \{C_i, c_{\text{new}}\}
13    if length(c) > 1
14      c_{\text{new}} = \{
15        for each v \in c
16          v_{\text{new}} = (v, 1)
17          c_{\text{new}} = \{c_{\text{new}}, v_{\text{new}}\}
18      C_i = \{C_i, c_{\text{new}}\}
19  return C_n
```

chain of the next length $k+2$. Proof of this property can be found in both Hansel (1966) and Sokolov (1982). As a result, Hansel’s algorithm results in fewer or the same number of queries as the upper bound on $m(f)$ as follows.

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 Hansel’s algorithm is $2 \binom{n}{\lfloor n/2 \rfloor}$. When $n$ is even, there are $\binom{n}{\lfloor n/2 \rfloor} - \binom{n}{\lfloor n/2-1 \rfloor}$ chains of length one, and $\binom{n}{\lfloor n/2 \rfloor}$ chains of length greater than one. Therefore, the maximum number of queries used by Hansel’s algorithm is $\binom{n}{\lfloor n/2 \rfloor} - \binom{n}{\lfloor n/2-1 \rfloor} + 2 \binom{n}{\lfloor n/2-1 \rfloor} = \binom{n}{\lfloor n/2 \rfloor} + \binom{n}{\lfloor n/2-1 \rfloor}$. That is, the following inequality holds:

$$\varphi(\text{Hansel}, f) \leq \max_{g \in M_n} m(g) = \left(\frac{n}{\lfloor n/2 \rfloor}\right) + \left(\frac{n}{\lfloor n/2 \rfloor + 1}\right), \quad \forall f \in M_n.$$  

Hansel’s algorithm is further illustrated in Section 3.2.2.

The algorithm described in Sokolov (1982) is also based on the Hansel chains. In contrast to Hansel’s algorithm, it considers the chains in the reverse order (i.e., in decreasing length) and performs the binary search within 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 monotone Boolean function $T$. This function has only one border vector $(00...0)$, which is located in the longest chain. For this function, Sokolov’s algorithm performs at most $\lceil \log_2(n) \rceil + 1$ evaluations, while Hansel’s algorithm needs at least $\binom{n}{\lfloor n/2 \rfloor}$ evaluations. For instance, when $n = 20$ this translates into at least 184,756 evaluations performed by Hansel’s algorithm and at most 5 evaluations performed by Sokolov’s algorithm.

Sokolov’s algorithm does not satisfy the upper bound, 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 $v \in \{0,1\}^n : \|v\| \geq n/2$, and 0 otherwise. Then the set of border vectors is $\{v : \|v\| = \lfloor n/2 \rfloor \text{ or } \lfloor n/2 \rfloor - 1\}$ and
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 \( |w^2| = \lfloor 3n/4 \rfloor \neq n/2 \) and \( n/2-1 \). Therefore, the following inequality holds:

\[
\Phi(\text{Sokolov}, f) > \{w^1\} \cup \{w^2\}, \text{ for at least one } f \in M_n.
\]

Sokolov’s algorithm is further illustrated in Section 3.2.3.

In an attempt to provide a unified efficiency testing platform, Gainanov (1984) proposed to compare inference algorithms based on the number of evaluations needed for each border vector. To that end, he presented an algorithm that searches for border vectors one at a time. 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. An implementation of this subroutine, called FIND-BORDER(\( v \)), is presented in Figure 3.4. This subroutine is also used in the algorithms of Boros et al. (1997), Makino and Ibaraki (1995), and Valiant (1984).

**Figure 3.4** The subroutine used to find the border vectors of a monotone Boolean function one at a time.

The first query in FIND-BORDER(\( v \)) is performed on line 1. If the oracle responds with \( f(v) = 1 \), the algorithm executes lines 2-5, where it will find a lower unit by queries to the oracle performed on line 4. If the oracle responds with \( f(v) = 0 \), the algorithm executes lines 7-10, where it will find an upper zero by queries to the oracle performed on line 9. On lines 3 and 8, the vector \( e^i \) denotes a vector with a 1 at the \( i \)-th position and 0’s at the other \( n-1 \) positions. An implementation of the FIND-BORDER algorithm is further illustrated in Section 3.2.1.

Suppose the subroutine FIND-BORDER is executed once given an unclassified vector \( v \) as input. If the result of the first query is \( f(v) = 1 \), then an additional \( |v| \) queries are performed. Similarly, if the result of the first query is \( f(v) = 0 \), then an additional \( n - |v| \) queries are performed. In either case, at least one of the queries will hit a border vector. In other words, the maximum number of queries per border vector is \( \max\{|v| + 1, n - |v| + 1\} = n + 1 \). As a result, any inference algorithm \( A \) (FIND-BORDER) that feeds unclassified vectors to the subroutine FIND-BORDER, satisfies the following upper bound:

\[
\Phi(A \text{ (FIND-BORDER)}, f) \leq m(f)(n+1), \forall f \in M_n.
\]

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.
Earlier work on monotone Boolean function inference (such as Hansel (1966), Sokolov (1982), and Gainanov (1984)) focuses on reducing the query complexity. More recent work (like Boros et al. (1997), Makino and Ibaraki (1997), and Fredman and Khachiyan (1996)) considers both the query complexity and the computational complexity. The problem of inferring a monotone Boolean function via membership queries is equivalent to many other computational problems in a variety of fields (see, for instance, Bioch and Ibaraki (1995), and Eiter and Gottlob (1995)). In 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, performing experiments or running simulations. For such applications, queries far surpass computations in terms of cost. Therefore, this dissertation focuses on minimizing the query complexity as long as it is computationally feasible.

For the purpose of contrasting the existing algorithms for Problem # 1, the monotone Boolean function for the computer sales company described in Section 2.2 will be used. Please recall that the function was defined as follows: \( f(v) = v_1 v_2 \lor v_3 v_5 \lor v_2 v_5 \lor v_4 v_5. \) The set of eight border vectors for this function is \{11000, 00100, 10001, 01001, 00011, 10010, 01010, 00001\}. That is, the number of queries used by Teacher is \( m(f) = 8, \) since it represents the minimum number of queries required to infer the function. The inference processes for the subroutine FIND-BORDER, Hansel’s algorithm, and Sokolov’s algorithm are described next in Sections 3.2.1, 3.2.2, and 3.2.3, respectively.

### 3.2.1 Illustration of the FIND-BORDER Algorithm for the Computer Sales Company Application

Figure 3.5 shows the sequence of queries and their answers for the computer sales application, when the unclassified vectors are arbitrarily fed to the algorithm FIND-BORDER. Below each query, the remaining unclassified vectors are shown in the form of a poset with the selected vector circled.

Initially, all the vectors in \{0,1\}^5 are unclassified. The vector (00000) is arbitrarily selected and fed to the FIND-BORDER algorithm. This vector corresponds to a 10 minute test run with no applications running. Since the computer does not tend to crash in this situation (i.e., \( f(00000) = 0 \)), the set of unclassified vectors is reduced and forms the poset shown below the 2nd query.

Next, the FIND-BORDER algorithm queries the vector (10000). A 10 minute test with only the web browser running (i.e., \( v_1 = 1 \)) is performed. After the test run does not result in a crash (i.e., \( f(10000) = 0 \)), the set of unclassified vectors is further reduced and forms the poset shown below the 3rd query. This process continues for the additional 16 queries shown in Figure 3.5, after which the computer’s crashing function is inferred.

The first sequence of queries (00000, 10000, 11000, 10100, 10010, 10011) are generated by the first execution of the algorithm FIND-BORDER\((v)\) using \( v = (00000) \) as input, where it returns the upper zero (10010). The second vector fed to FIND-BORDER is (00001), where it returns the upper zero (00001) from the following sequence of queries (00001, 10001, 01001, 00101, 00011). Similarly, the third and fourth sequence of queries are (01000, 00000), and (01000, 11000, 01100, 01010, 01011), where the lower unit (00100) and the upper zero (01010) are found, respectively. The subsets of lower units and upper zeros \{(00100)\}, and \{(00001), (10010), (01010)\}, respectively, can be used to represent the uncovered monotone Boolean function by \( f(v) = v_3 \lor (v_1 \lor v_2 \lor v_3 \lor v_4)(v_2 \lor v_3 \lor v_5)(v_1 \lor v_3 \lor v_5) = v_1 v_2 \lor v_3 \lor v_2 v_5 \lor v_4 v_5. \)

Please notice that each sequence of queries contains at most \( n + 1 = 6 \) queries. The upper bound on the total number of queries given in Section 3.2 by \( m(f)(n+1) = 8 \times 6 = 48, \) is indeed greater than 18. However, this upper bound is trivial since it is greater than the maximum number
of queries \(2^5 = 32\). The FIND-BORDER algorithm may query vectors that are already classified. The posets in Figure 3.5 only contain the unclassified vertices, and therefore some of the posets do not contain circled vertices (queries 13, 15, 16, and 18). In a practical situation, these queries can be answered by the function inferred thus far, and are not posed as queries to the oracle. In other words, the total number of queries to the oracle is 18 - 4 = 14.

3.2.2 Illustration of Hansel’s Algorithm for the Computer Sales Company Application

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).

Figure 3.6 shows the sequence of queries and their answers for the computer sales application, when the queries are selected using Hansel’s algorithm. Below each query, the remaining unclassified vectors are shown in the form of a poset with the selected vector circled. Initially, all of the vectors \(\{0,1\}^5\) are unclassified. The vector (00011) is arbitrarily selected from the first Hansel chain \{00011, 10011\}. A 10 minute test is performed with the CD player (i.e., \(v_4 = 1\)) and the word processor (i.e., \(v_5 = 1\)) running. After the test does result in a crash (i.e., \(f(00011) = 1\)), the set of unclassified vectors is reduced and forms the poset shown below the 2nd query.

As a result of the first query, both vectors in the first chain are classified and Hansel’s algorithm proceeds to the second chain \{00101, 10101\}. The vector (00101) is arbitrarily selected.
A 10 minute test is performed with the image viewer (i.e., \( v_3 = 1 \)) and the word processor (i.e., \( v_5 = 1 \)) running. After the test does result in a crash (i.e., \( f(00101) = 1 \)), the set of unclassified vectors is reduced and forms the poset shown below the 3rd query. This process continues for the additional 9 queries shown in Figure 3.6. After these 11 queries, the value of the computer’s crash function \( f \) is known for all the vectors in \( \{0,1\}^5 \).

As noted in Section 3.2, the number of queries per chain is guaranteed to be less than or equal to 2. In this application, the respective number of queries per chain are 1, 1, 1, 1, 1, 2, 2, 1, 0, 1, for a total of 11 queries.

### 3.2.3 Illustration of Sokolov’s Algorithm for the Computer Sales Company Application

Sokolov’s algorithm searches the 10 Hansel chains in decreasing size: \( \{00000, 10000, 11000, 11100, 11110, 11111\}, \{01000, 01100, 01110, 01111\}, \{00100, 10100, 10110, 10111\}, \{00010, 10010, 11010, 11011\}, \{00001, 10001, 11001, 11101\}, \{01010, 01011, 01011\}, \{01001, 01101\}, \{00110, 00111\}, \{00101, 10101\}, \{00011, 10011\}\). It performs the binary search within each of these chains.

Figure 3.7 shows the sequence of queries and their answers for the computer sales application, when the queries are selected using Sokolov’s algorithm. Below each query, the remaining unclassified vectors are shown in the form of a poset with the selected vector circled.

Initially, all of the vectors \( \{0,1\}^5 \) are unclassified. The two vectors \( (11000) \) and \( (11100) \) are both middle vectors of the first chain. The vector \( (11000) \) is arbitrarily selected. A 10 minute test is performed with the web browser (i.e., \( v_1 = 1 \)) and the internet connector (i.e., \( v_2 = 1 \)) running. After the test does result in a crash (i.e., \( f(11000) = 1 \)), the set of unclassified vectors is reduced and forms the poset shown below the 2nd query. The two vectors \( (00000) \) and \( (10000) \) are the only remaining
Query 1: $f(11000)=1$
Query 2: $f(00000)=0$
Query 3: $f(10000)=0$
Query 4: $f(01100)=1$
Query 5: $f(01000)=0$
Query 6: $f(10100)=1$
Query 7: $f(00100)=1$
Query 8: $f(00010)=0$
Query 9: $f(10010)=0$
Query 10: $f(00001)=0$
Query 11: $f(10001)=1$
Query 12: $f(01010)=0$
Query 13: $f(01011)=1$
Query 14: $f(01001)=1$
Query 15: $f(00011)=1$

**Figure 3.7** The computer crash inference process using Sokolov’s algorithm.

vectors of the first chain. Both vectors are middle vectors, and the vector (00000) is arbitrarily selected for the next query. A 10 minute test is performed with no applications running. After the test does not result in a crash (i.e., $f(00000) = 0$), the set of unclassified vectors is reduced and forms the poset shown below the 3rd query.

This process continues for the additional 13 queries shown in Figure 3.7. After the 15 queries, the value of the computer’s crash function $f$ is known for all the vectors $v$ in $\{0,1\}^5$. As was noted in Section 3.2, the number of queries per chain for Sokolov’s algorithm is not necessarily less than or equal to 2, as for Hansel’s algorithm. The respective number of queries per chain are here 3, 2, 2, 2, 2, 1, 0, 0, 1, for a total of 15 queries.

The number of queries performed by each of the three methods Hansel, FIND-BORDER, and Sokolov for the computer sales application were 11, 14, and 15, respectively. The number of queries per border vector are $11/8 = 1.375$, $14/8 = 1.75$ and $15/8 = 1.875$ for the Hansel, 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. Section 6.1 provides the results from more extensive comparisons.
3.3 An Existing Approach to Problem # 2
Kovalerchuk et al. (1996) considered the problem of inferring a pair of nested monotone Boolean functions. Their algorithm, which exhibited a promising efficiency in their cancer diagnosis application, is an extension of Hansel’s inference algorithm for a single monotone Boolean function. However, the algorithm performance analysis is far from conclusive as a single application represents a single pair of nested monotone Boolean functions.

3.4 Existing Approaches to Problem # 3
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, such as the feed forward neural networks in Cohn et al. (1996), even though 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).

3.5 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:

\[
e(f) = \sum_{z=1}^{2} (f(v)^z)m_z(v) + (1 - f(v)^z)m_1(v^z).
\]

It is assumed that the oracle misclassifies each vector \( v \) with a probability \( q(v) \in (0, \frac{1}{2}) \). That is, for a given monotone Boolean function \( f \), the oracle returns 1 for vector \( v \) with probability \( p(v) = q(v)\times(1 - f(v)) + (1 - q(v))\times f(v) \), and 0 with probability \( 1 - p(v) \). A key assumption is that the misclassification probabilities are all less than \( \frac{1}{2} \), otherwise it would not be possible to infer the correct monotone Boolean function.

When the misclassification probabilities are all fixed (i.e., \( q(v) = q \forall v \in V \)), an estimate for \( p \) 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 \( \frac{1}{2} \) (i.e., \( \min\{e(f)/m, \frac{1}{2}\} \)). Note that the probability \( q \) is actually required to be strictly less than \( \frac{1}{2} \), but for practical purposes this estimate is sufficient when the observed error ratio is greater than or equal to \( \frac{1}{2} \).

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:
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 probabilities $q(v)$ are all less than $\frac{1}{2}$, 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 $\frac{1}{2}$, then the maximum likelihood solution may maximize the number of errors, as demonstrated by Boros et al. (1995). In this dissertation, error maximization is avoided by restricting $q$ to be less than $\frac{1}{2}$.

It should be noted that an error minimizing solution with $q^* \leq \frac{1}{2}$ always exists. One of the two uniform monotone Boolean functions, $T$ or $F$, satisfies this restriction as follows:

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

Therefore, $q(T) + q(F) = 1$ and $0 \leq q(T), q(F) \leq 1$. As a result, $q^* \leq \min\{q(T), q(F)\} \leq \frac{1}{2}$. 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 $q^* \leq \frac{1}{2}$.

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 $\sum_{i} m_0(v^i)$ 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 $f(v^i) \leq f(v^i) \forall v^i, v^i \in V: v^i \leq v^i$, and $f(v^i) = 0$ or $1$.

This problem is known as a maximum closure problem, which can be converted into a maximum flow problem as described in Section 5.3.1 (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
The posets considered in this dissertation are of the form \(\{0,1\}^n\), 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:

\[
\min \sum_{i=1}^{k} (\tilde{p}(v^i) - p(v^i))(m_1(v^i) + m_0(v^i))
\]

subject to \(p(v) \leq p(v') \forall v', v' \in V: v' \leq v'\),

where

\[
\tilde{p}(v^i) = \frac{m_1(v^i)}{m_1(v^i) + m_0(v^i)}, \text{ for } i = 1, 2, ..., k,
\]

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 dissertation is somewhat restrictive. However, the goal of this dissertation is to efficiently uncover the underlying monotone Boolean function and not necessarily come up with accurate estimates for the errors. The fixed misclassification probability assumption does not affect the capability of the inference methodology as will be demonstrated in the subsequent chapters. The assumption is simply used to estimate the error rate and the confidence in having inferred the correct function.

The fact remains that the error minimizing monotone Boolean function also maximizes the likelihood (Boros et al. (1995)). 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.

### 3.6 Unequal Probability Sampling

Often a random sample of test cases is generated to estimate parameters such as the average algorithm complexity over the entire universe of test cases. For an estimator to be unbiased, its expected value has to be equal to the actual parameter value it is estimating. If some test cases are more likely to end up in the sample, then the average complexity of the sample puts too much weight on the more likely test cases, and too little weight on the less likely test cases. If the probability of
each test case being included in the sample (i.e., its inclusion probability) is known, then an unbiased estimator can be obtained.

The estimators that are presented next can be found in most textbooks on sampling (e.g., Thompson (1992)). Consider the finite universe of fixed quantities \( \{ \Phi_1, \Phi_2, \ldots, \Phi_{\Psi(n)} \} \) 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:

\[
\hat{t} = \sum_{i=1}^{\Psi(n)} \frac{\Phi_i}{p_i}
\]

where \( \{ \Phi_1, \Phi_2, \ldots, \Phi_{\Psi(n)} \} \) are the quantities corresponding to the set of \( \Psi \) 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 \( \hat{t} \) requires the joint pairwise inclusion probabilities. Let \( p_{ij} \) be the probability that elements \( i \) and \( j \) are included together in the sample. Then, an unbiased estimator is given by:

\[
\hat{\text{Var}}(\hat{t}) = \sum_{i=1}^{\Psi(n)} \frac{1 - p_{ii} \Phi_i^2}{p_i^2} + \sum_{i=1}^{\Psi(n)} \sum_{j \neq i} \frac{(p_{ij} - p_{ii} p_{jj}) \Phi_i \Phi_j}{p_i p_j p_{ij}}
\]

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

\[
\hat{\mu} = \frac{\hat{t}}{\Psi(n)},
\]

and a corresponding unbiased estimate of its variance is:

\[
\hat{\text{Var}}(\hat{\mu}) = \frac{\hat{\text{Var}}(\hat{t})}{\Psi(n)^2}.
\]

At a first glance it may seem like the only benefit of using the inclusion probabilities are unbiased estimators. If the sampling technique allows for adjusting the inclusion probabilities, however, it may be possible to adjust them so as to reduce the variance of the estimators. This issue is further discussed in Section 5.4.
CHAPTER 4. COMPLETE DESCRIPTIONS OF THE RESEARCH PROBLEMS

In the earlier chapters the goal for each of the three inference problems was loosely described as minimizing the query complexity as long as it is computationally feasible. The purpose of this chapter is to provide exact definitions of the inference objectives and their associated research problems.

The objectives for Problems # 1, # 2, and # 3 are defined in Sections 4.1, 4.2, and 4.3, respectively. The algorithmic problem of finding the maximum likelihood monotone Boolean function incrementally for Problem # 3 is described in Section 4.4. The problem of randomly generating monotone Boolean functions is described in Section 4.5.

4.1 The Inference Objective for Problem # 1
An inference algorithm that performs fewer queries than another algorithm when reconstructing a particular deterministic monotone Boolean function is considered more efficient on that particular function. However, it has not been clear how to compare algorithms on the entire class of monotone Boolean functions defined on \{0,1\}^n.

The existing algorithms (e.g., Hansel (1966), Sokolov (1982), Gainanov (1984)) focus on 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 the algorithm on an arbitrary monotone Boolean function.

For example, algorithms that implement the subroutine referred to as FIND-BORDER in Section 3.2 indirectly suggest minimizing the upper bound on the number of evaluations 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}{2} + \binom{n}{3} + \cdots + \binom{n}{k} border vectors). Kovalerchuk et al. (1996) demonstrated promising results for a Hansel based inference algorithm on a real world application. However, their performance analysis is far from conclusive as a single application represents a single monotone Boolean function.

With no prior knowledge (other than monotonicity) about the inference application, each function is equally likely to be encountered and should therefore carry the same weight in the objective. The objective for this problem is to develop an algorithm that minimizes the average number of queries over the entire class of monotone Boolean functions defined on the set \{0,1\}^n.

This objective can be expressed mathematically as follows:

\[
Q(n) = \min_{A} \frac{\sum_{f \in M_n} \Phi(A, f)}{\Psi(n)}.
\]

The objective \(Q(n)\) represents the entire class of monotone Boolean functions \(M_n\). As such, it provides a better indication of what to expect when executing an algorithm on an arbitrary monotone Boolean function.

4.2 The Inference Objective for Problem # 2
The approach taken to this problem is analogous to that of Problem # 1. The minimum average number of queries for Problem # 2.\(k\) (for \(k = 1, 2, \text{and } 3\)) can be expressed mathematically as follows:

\[
Q_k(n) = \min_{A_k} \frac{\sum_{f_1, f_2} \Phi(A_k, f_1, f_2)}{\Psi(n+1)}.
\]

Here, \(\Phi(A_k, f_1, f_2)\) denotes the number of queries performed by algorithm \(A_k\), in reconstructing the
pair of nested monotone Boolean functions \( f_1 \) and \( f_2 \) defined on the set \( \{0,1\}^n \). Here, \( A_1, A_2, \) and \( A_3 \) denote algorithms designed for Problems # 2.1, # 2.2, and # 2.3, respectively. Please recall from Section 3.1 that the number of pairs of nested monotone Boolean functions defined on the set \( \{0,1\}^n \) is equal to \( \Psi(n+1) \), the number of monotone Boolean functions defined on the set \( \{0,1\}^{n+1} \).

Since these three problems differ in the way the oracles are queried, it should be clarified that a query unit pertains to the membership value from one of the two functions \( f_1 \) and \( f_2 \). This definition is intuitive for Problems # 2.1 and # 2.3, where two oracles are accessed individually. For Problem # 2.2, the membership values are provided in pairs from a single three valued oracle. To make the definition of \( Q_3(n) \) comparable to \( Q_1(n) \) and \( Q_3(n) \), each query to the three valued oracle will be counted as two queries.

### 4.3 The Inference Objective for Problem # 3

The approach taken to Problem # 3 is similar to that of Problems # 1 and # 2. The goal is to minimize the average number of queries needed to completely reconstruct the underlying monotone Boolean function, expressed mathematically as follows:

\[
\min_{\mathbf{A}} \frac{\sum_{f \in \mathcal{M}_n} \Phi(A, f, q)}{\Psi(n)}.
\]

Here, \( \Phi(A, f, q) \) denotes the expected number of queries performed by algorithm \( A \) in completely reconstructing the underlying monotone Boolean function \( f \) from an oracle with a fixed misclassification probability \( q \). Completely reconstructing the underlying function translates into making the likelihood ratio \( \lambda(f^*) \) for the inferred function \( f^* \) reach a sufficiently high value, of say 0.99.

It should be stressed that the misclassification probability \( q \) is unknown and ranges from 0 up to \( \frac{1}{2} \). However, it is expected that the average number of queries will increase significantly with \( q \), since it by definition is infinite when \( q \) is equal to \( \frac{1}{2} \), and it is finite when \( q \) is equal to 0. Therefore, the average over a large range \( q \) may not be an accurate prediction of how many queries to expect for a particular application. The average query complexity will therefore be evaluated as a function of \( n \) and \( q \), even though \( q \) is unknown.

### 4.4 Incremental Updates for the Fixed Misclassification Probability Model

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

If, on the other hand, the new classification is inconsistent with the old function (i.e., \( f_{\text{old}}^*(v) = 1 - z \)), 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.
The number of possible error minimizing functions may be exponential in the size of the set \( V \), and therefore storing all of them may not be an efficient solution to this problem. To avoid this computational burden an incremental algorithm is needed.

### 4.5 Randomly Generating Monotone Boolean Functions

The problem addressed in this section is how to generate monotone Boolean functions in order to unbiasedly compare the inference algorithms. 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. Please refer to the number of monotone Boolean functions \( \Psi(n) \) given in Table 3.1.

As a remedy, a sample of functions can be generated. However, a sample that is not drawn randomly is subject to conditional results. Also, if one generates monotone Boolean functions that contain a certain number of border vectors (e.g., Makino et al. (1999)), for example, then one simply cannot make valid claims towards the entire class of monotone Boolean functions.

It is practically impossible to generate monotone Boolean 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 example, consider the algorithm GENERATE-MBF outlined in Figure 4.1.

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

**Figure 4.1** The algorithm used to randomly generate monotone Boolean functions without inclusion probabilities.

This algorithm starts with all the vectors \( \{0,1\}^n \) as the unclassified set, denoted by \( U \). Then (on line 3) it randomly selects a vector from the unclassified vectors so that each one is selected with equal probability. In the last step (lines 4-12), it classifies the selected vector as 0 or 1 with (equal) probability of \( \frac{1}{2} \) 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. Here the subroutine SELECT-AT-RANDOM(\( \{v^1, v^2, ..., v^i\}, \{p^1, p^2, ..., p^i\} \)), selects the \( i \)-th element \( v^i \) from the set \( \{v^1, v^2, ..., v^i\} \) with probability \( p^i \), where \( \sum p^i = 1 \).

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 can vary significantly.
For example, the probability that the zero vector is classified as 1 is greater than $\frac{1}{2^{(n+1)}}$. This value corresponds to the probability of selecting the zero vector during the first iteration and classifying it as 1. This case corresponds to the monotone Boolean function $T$. Thus, the inclusion probability for the function $T$ is much greater than what it would be in the uniform sampling as the following inequality indicates:

$$\Pr\{f = T\} \geq 2^{-(n+1)} \gg \Psi(n)^{-1}.$$ 

Korshunov’s asymptotic to $\Psi(n)$ given in Section 3.1 provides an idea of the magnitude of the right hand side of this inequality.

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. The algorithm GENERATE-MBF 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 this algorithm is used and the number of queries for a particular inference algorithm significantly increases with the number of border vectors. Then an estimate of the average query complexity which ignores the inclusion probabilities is seriously biased downwards.

A lower bound on, instead of the exact value of, the exact inclusion probability of the function $T$ was given above. This is due to the fact that inclusion probabilities are hard to compute for GENERATE-MBF, because there are so many different ways of generating the same function.

Since the inclusion probabilities are essential to unbiased estimators, they need to be easy to compute. If nothing is known a priori about the number of queries, then it is also desirable for the inclusion probabilities to be close to the uniform case, which will lower the variance of the estimators given in Section 3.6.

To evaluate an inference algorithm for any of the three nested Problems # 2.1, # 2.2, and # 2.3, pairs of nested monotone Boolean functions have to be randomly generated in a similar fashion as for Problem # 1. Fortunately, slight modifications to the manner in which single monotone Boolean functions are generated will provide pairs of nested monotone Boolean functions and their inclusion probabilities.
CHAPTER 5. METHODOLOGY

The purpose of this chapter is to develop algorithms for the research problems that were described in Chapter 4. Sections 5.1, 5.2, and 5.3 develop the methodologies used for Problems #1, #2, and #3, respectively. Section 5.4 provides an algorithm for randomly generating monotone Boolean functions with inclusion probabilities.

5.1 Methodology for Problem #1

The objective $Q(n)$ for Problem #1 is given in Section 4.1. Before the solution strategy is developed, a generalized version of this objective is described and solved in Section 5.1.1 via recursion. In Section 5.1.2, this recursive solution methodology is then used as a model for computing $Q(n)$ with an improved efficiency due to overlapping subproblems. After the optimal query selections are known, Section 5.1.3 summarizes the optimality conditions in the form of an evaluative criterion.

5.1.1 Minimizing the Average Inference Cost

In some applications the cost of inferring a particular monotone Boolean 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. The monotonicity property holds for keyword searches as the following examples illustrate.

Define a Boolean function $f$ as taking on the value of 0 when at least one article is found, and 1 otherwise. If at least one article is found containing the keywords $k_1$ and $k_3$ (i.e., $f(10100\ldots0) = 0$), then there are articles containing proper subsets of the keywords $k_1$ and $k_3$, (e.g., $f(10000\ldots0) = 0$). If no articles are found containing the keywords $k_1$, $k_2$, and $k_3$ (i.e., $f(11100\ldots0) = 1$), then there are no articles containing proper supersets of these keywords (e.g., $f(11110\ldots0) = 1$). Therefore, the Boolean function describing whether articles are found is monotone (i.e., $f(v) \leq f(w) \forall v, w \in \{0,1\}^n : v \preceq w$).

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 when searching several articles. Suppose $b$ articles containing the keywords $k_1$ and $k_3$ (i.e., query $(10100\ldots0)$) are found. Then searching for articles containing the keywords $k_1$, $k_2$, and $k_3$ (i.e., query $(11100\ldots0)$) among the $b$ articles found from the query $(10100\ldots0)$ is less expensive than searching all of the $a$ articles over again. Thus, the total cost for evaluating the vector $(10100\ldots0)$ followed by the vector $(11100\ldots0)$ is $2a + 3b$.

If, on the other hand, the query order is reversed, and the query $(11100\ldots0)$ does not result in any articles, then all the $a$ articles have to be searched again for the query $(10100\ldots0)$. Now the total cost for evaluating vectors $<(11100\ldots0), (10100\ldots0)>$ is $5a (= 2a + 3a)$, which is greater than the cost $2a + 3b$ of the queries $<(10100\ldots0), (11100\ldots0)>$.

The point made through this example was that the order of the queries may have an effect on the total inference cost even though the conclusion is potentially the same. Here articles containing the keywords $k_1$ and $k_3$ were found (i.e., $f(10100\ldots0) = 0$), while no articles containing keywords the $k_1$, $k_2$, and $k_3$ were found (i.e., $f(11100\ldots0) = 1$).

To describe this general cost situation, let $C(A, f)$ denote 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:
This objective translates into the intuitive notion of minimizing the average inference cost. As such, it generalizes the objective $Q(n)$, which considers the queries themselves as the cost unit.

The tree in Figure 5.1 illustrates an approach for computing $C(n)$ 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 $<(11), (10), (01), (00)>$, performed in inferring the function $T$.

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 computing $C(2)$ associates a cost with each leaf in this tree. Let $c_i$ denote the cost of the $i$-th leaf from the top, for $i = 1, 2, ..., 76$. 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 5.1. If vector (01) is queried next, the costs $c_1$, $c_2$, and $c_3$ associated with the three topmost leaves will be used in the objective $C(2)$. If instead the vector (00) is queried next, the costs $c_4$, $c_5$, and $c_6$ associated with the next three leaves, will be used in the place of $c_1$, $c_2$, and $c_3$ in the objective. Therefore, vector (01) should be selected if $c_1 + c_2 + c_3 < c_4 + c_5 + c_6$, 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 $V = \{v^1, v^2, ..., v^p\}$ corresponding to the prior vector selections $W = <w^1, w^2, ..., w^r>$ and their classifications $F = <f^1, f^2, ..., f^r>$, define the cost function as follows:

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

where

$c(W, F)$ is the fixed cost associated with inferring the monotone function $f$ defined by $f(w^1) = f^1$, $f(w^2) = f^2$, ..., $f(w^r) = f^r$ via the sequence of queries $W$.
Figure 5.1 The exhaustive vector selection tree for \([0,1]^2\).
$V_z$ is the set of unclassified vectors of $V$ remaining after $f(v^i) = z$ has been established for $i = 1, 2, ..., p$ and $z = 0, 1$, and

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

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

**Proof:** The correctness 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.

---

### 5.1.2 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. The objective $Q(n)$ possesses several 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 $v \in V$. In other words, 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 query 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 $V = \{v^1, v^2, ..., v^p\}$, define the cost function as follows:

$$C_2(V) = \begin{cases} 0, & \text{if } |V| = 0, \\ \min_{l=1,2, ..., p} \left\{ \frac{N(V'_0)(C_2(V'_0) + c_q(v^l)) + N(V'_1)(C_2(V'_1) + c_q(v^l))}{N(V'_0) + N(V'_1)} \right\}, & \text{otherwise,} \end{cases}$$

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

**Lemma 2.** The recursive function $C_2(\{0,1\}^n)$ yields the minimum average inference cost $C(n)$ when the individual query costs are fixed.

**Proof:** Suppose the cost of queries are fixed and given by $c_q(v^i)$ for $i = 1, 2, ..., p$. 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 $c(<w^1, w^2, ..., w^p>, F)$ equals $c_q(w^1) + c_q(w^2) + ...+ c_q(w^p)$. As a result, $C_2(V)$ equals $C_1(<,>,\{0,1\}^n)$. From Lemma 1, $C_1(<,>,\{0,1\}^n)$ equals $C(n)$, and hence $C_2(\{0,1\}^n)$ equals $C(n)$.

---

Lemma 2 provides a way to compute the minimum average query cost in a more efficient manner than Lemma 1. In particular, branches out of nodes corresponding to unclassified vector subsets are required only for unique vector subsets. As an example, consider the node corresponding to the set $\{10,01\}$ which appears twice in the tree in Figure 5.1. 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 of vectors $V$ consists of a set of mutually unrelated subsets $\{V^1, V^2, ..., V^p\}$, then the following equation holds:

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

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

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

Proof: For any monotone Boolean function $f$ defined on $V$, fixing the value of $f(v')$ for $v' \in V^i$ does not restrict the value of $f(w)$ for $w \in W = \{V^1, V^2, ..., V^{i'-1}, V^{i'+1}, ..., V^p\}$. 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 $N(W)N(V^i)$. By further reducing $W$ in a similar manner for the other unrelated subsets, this expression can be reduced to $N(V^1)N(V^2) \ldots N(V^p)$.

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

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

Another apparent property of the objective $Q(n)$ 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 resulting vector subsets be denoted by $V^i_z$, for $z = 0$ and 1, when $v^i$ is selected from the set of vectors $V^i$ (for $i = 1, 2$).

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

$$N(V^i_0) = N(V^z_0) \land N(V^i_1) = N(V^z_1).$$

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

$$C_2(V^i_0) = C_2(V^z_0) \land C_2(V^i_1) = C_2(V^z_1).$$

Proof: This lemma can be proved by induction. As a basis of the induction consider the two non-isomorphic posets: a single vertex $v$ (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 $v^i$ to $v^z$. If $v^i$ is selected from $V^i$, then it results in a pair of posets that are isomorphic to the pair of posets that result from selecting $v^z$ 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 analogous 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 5.2 shows the resulting reduced search tree when these bounding criteria are applied to the tree shown in Figure 5.1.

"Figure 5.2 The vector selection tree for $\{0,1\}^2$ restricted to connected, non-dual, and non-isomorphic posets."

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 5.2. A size comparison of the trees in Figures 5.1 and 5.2 indicates the potential effectiveness of these bounding criteria. The original tree contained 143 nodes (including the leaves) while the reduced tree only has 31 nodes.

To simplify the implementation, the algorithm was divided into two steps as shown in the algorithm MINIMIZE-AVE-Q given in Figure 5.3. 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 by the algorithm CREATE-POSET-LIBRARY and subsequently stored in library $L$. This is realized in lines 1 and 2 of MINIMIZE-AVE-Q. 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 algorithm CREATE-POSET-LIBRARY actually generates 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 vectors $V$ are computed for all of the posets stored in library $L$. Since the parameters of a poset of size $k$ (i.e., one with $k$ vectors) 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 to find the poset in library $L$ that is isomorphic to $P$ or its dual. For the implementation of this search, the size $k$ (the simplest isomorphism invariant) and an index number $i$ are used to uniquely identify posets in library $L$. 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 a single sweep by accessing $Q_k(i)$, $N_k(i)$, and $V_k(i)$.

**Figure 5.3** The algorithm used to compute $Q(n)$.
There are five subroutines in Figure 5.3 whose details have been left out for the purpose of brevity. The subroutine CONNECTED($P$) should return TRUE if 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 poset $P$, and TRUE otherwise. The subroutine REMOVE-CONE($P$, $v$, $f$) should return the poset formed by removing from poset $P$ the vector $v$ and the vertices that precede $v$ if $f = 0$ (or the vertices that succeed $v$ if $f = 1$). The subroutine MAX-UNRELATED-POSETS($P$) should return the largest partition of poset $P$ into a set of mutually unrelated posets. The subroutine EDGE-VERTICES($P$) should return the set containing all the edge vertices of poset $P$.

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 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 5.4 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 a pair of posets that are isomorphic to the posets resulting from branching on 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 5.4. In general, only one vector from each layer of a symmetric poset needs to be branched upon. Since the initial poset $(\{0,1\}^n, \preceq)$ is known to be symmetric, only $\lceil n/2 \rceil$ out of the $2^n$ branches are needed for its corresponding node.

At first glance, it may 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 5.1.3 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 \(\{0,1\}^n\). It also finds all the optimal vertices. Figure 5.5 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 5.1.3 addresses the issue of summarizing the optimality conditions for the posets in Figure 5.5 and for some posets observed for \(n > 4\).

### 5.1.3 Defining an Evaluative Criterion

The total numbers of connected, non-dual, and non-isomorphic posets generated by MINIMIZE-AVE-Q(n) for \(n\) equal to 1, 2, ..., 5 are given in Table 5.1. There are at least half as many posets as the number of monotone Boolean functions for \(n\) equal to 1, 2, ..., 5 (given in Table 3.1). 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.
Figure 5.5 All connected, non-dual, and non-isomorphic posets generated to compute $Q(4)$, with the optimal vertices circled.

When computing $Q(n)$ for $n$ greater than 6, the optimal approach presented in Section 5.1.2 is currently computationally infeasible. However, the optimality conditions for the posets generated for $n$ up to and including 4 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.
Table 5.1 The number of posets generated by MINIMIZE-AVE-Q(n).

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

When computing $Q(4)$, many different and complex posets were encountered. The optimal vectors of these posets seemed to display two general properties. First, the optimal vectors tend to be in the *vertical middle*. More specifically, all posets in Figure 5.5 have at least one optimal vector in the most populous layer. This observation alone is not sufficient to pinpoint an optimal vector.

The second property observed is that the optimal vectors also tend to be *horizontal end points*. In particular, vectors that are related to only one other vector are optimal for all the posets given in Figure 5.5. However, some of these posets do not contain such a vector. In this case, vectors that are related to fewer vectors tend to be optimal within a layer.

To study the idea of vertical middle and horizontal end point, consider two specific posets, namely a *chain* and a *sawtooth*, as shown in Figure 5.6. Examples of these posets can also be seen in Figure 5.5. 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 poset with minimum width. For a chain with $h$ vertices the number of monotone Boolean functions $N^c(h)$ equals $h+1$. The minimum average number of queries is given by:

$$Q^c(h) = Q^c\left(\lceil h/2 \rceil \right) \left(\lceil h/2 \rceil + 1\right) + \frac{Q^c(\lceil h/2 - 1/2 \rceil) \left(\lceil h/2 - 1/2 \rceil + 1\right)}{h+1} + 1$$

where $Q^c(0) = 0$. This value is obtained by invariably selecting a middle vertex (i.e., the $\lceil h/2 \rceil$-th vertex).
A sawtooth is an arbitrarily wide poset with minimum height, where each vertex has 2 related vertices, except the two end vertices which are related to only 1 vertex. Consider a sawtooth poset with \( w \) vertices. The number of possible monotone Boolean functions \( N^s(w) \) equals \( N^s(w-1) + N^s(w-2) \). The minimum average number of queries is given by

\[
Q^s(w) = \frac{Q^s(w-1)N^s(w-1) + Q^s(w-2)N^s(w-2)}{N^s(w-1) + N^s(w-2)} + 1 \approx 0.7236068 \times w + 0.2291796,
\]

where \( Q^s(1) = 1, Q^s(0) = 0, N^s(0) = 1 \) and \( N^s(1) = 2 \). This value is obtained by invariably selecting one of the end vertices.

The surprising result about the sawtooth poset is that consistently selecting the vertex adjacent to the end vertex maximizes the average number of queries. Figure 5.7 shows the average number of queries divided by \( w \) for the sawtooth poset of with \( w \) vertices. Here, the vertices are invariably selected in three different ways: an end vertex (the lower curve in Figure 5.7), a middle vector (the oscillating curve), and a vertex adjacent to an end vertex (the upper curve). For large values of \( w \), 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, \( V = \{ v_1, v_2, ..., v^p \} \) is given. Let \( K_1(v^i) \) and \( K_0(v^i) \) be the number of vectors that are concurrently classified when \( f(v^i) = 1 \) and \( 0 \), respectively. Invariably selecting a vector \( v \) with the minimum \( |K_1(v) - K_0(v)| \) 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 posets encountered by the criterion \( \text{min} |K_1 - K_0| \). 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), (10,2) starting from the upper left corner of Figure 5.5.

Unfortunately, this evaluative criterion is not optimal for all the posets generated for \( n \) equal to 4. It is only optimal for the subset of posets encountered when using the criterion \( \text{min} |K_1 - K_0| \). Another drawback is that it is not optimal for the inference problem when \( n \) is equal to 5. This fact is shown in Section 5.1.5 where another evaluative criterion is shown to have a slightly lower average query complexity, at the cost of a great increase in computational complexity. However, the criterion \( \text{min} |K_1 - K_0| \) is probably close to optimal since the larger posets eventually decompose into smaller posets. Before this issue is further discussed in Section 5.1.5, the next section illustrates how the criterion \( \text{min} |K_1 - K_0| \) can be used in practice.
Figure 5.7 Average query complexity of Problem # 1 defined on the sawtooth poset.

5.1.4 Illustration of the Evaluative Criterion Approach to the Computer Sales Company Application

To demonstrate how the inference process works using the evaluative criterion \( \min |K_1 - K_0| \) consider the computer sales company application described in Section 2.2. This example was also used to illustrate the FIND-BORDER, Hansel’s, and Sokolov’s algorithms in Sections 3.2.1, 3.2.2, and 3.2.2, respectively. Please recall that the computer crash function is defined as follows:

\[
f(v) = v_1 v_2 \lor v_3 \lor v_1 v_5 \lor v_3 v_5 \lor v_4 v_5,
\]

where \( f(v) = 1 \) if the computer tends to crash with the set of software applications \( v \) are running, and 0 otherwise.

Figure 5.8 shows the sequence of queries and their answers for the computer sales application, when the queries are selected according to the criterion \( \min |K_1 - K_0| \). Below each query, the remaining unclassified vectors are shown in the form of a poset with the selected vector circled.

Initially, all 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. A 10 minute test is performed with the CD player (i.e., \( v_4 = 1 \)) and the word processor (i.e., \( v_5 = 1 \)) running. The computer tends to crash in this situation (i.e., \( f(00011) = 1 \)). The set of unclassified vectors is reduced accordingly and forms the poset shown below the 2nd query.

In this poset, several vectors possess the minimum \( |K_1 - K_0| \) value of 0. One of these vectors (00101) is arbitrarily selected for the next query. A 10 minute test is performed with the image viewer (i.e., \( v_3 = 1 \)) and the word processor (i.e., \( v_5 = 1 \)) running. Since the computer tends to crash in this situation (i.e., \( f(00011) = 1 \)), the set of unclassified vectors is reduced and forms the poset shown below the 3rd query. This process continues for the additional 10 queries shown in Figure 5.8.

After these 12 queries, the value of the computer crash 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. This function has a total of 8 border vectors. Hence, the number of queries used by the evaluative criterion \( \min |K_1 - K_0| \) per border vector is \( 12/8 = 1.5 \).
In contrast, the Hansel, FIND-BORDER, and Sokolov algorithms performed $11/8 = 1.375$, $14/8 = 1.75$ and $15/8 = 1.875$ queries per border vector, respectively, to infer the same function. This comparison is far from conclusive as it is performed on one of the 7,581 functions defined on $\{0,1\}^5$. More extensive comparisons are presented in Section 6.1.

5.1.5 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)). However, 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 5.5. 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 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 5.5. 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 $(1, 12), (4, 2), (5, 2), (1, 7), (1, 8)$. 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 row and columns (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (2, 2), (2, 3), (3, 1), and (4, 3), starting from the upper left corner of Figure 5.5. 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 5.5. 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 5.5. 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 $\min|K_1(v) - K_0(v)|$ 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 posets in their entireties.

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 selection criterion should be modified. Figure 5.9 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 an algorithm that sacrifices earlier to perform better in the end.

The greedy approach (the solid line in Figure 5.9) maximizes the instantaneous reduction in the average number of remaining vectors between the vector selections. The other approach (the dotted line in Figure 5.9) 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 5.9, 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 those two evaluative criteria, the criterion $\min |K_1 - K_0|$ queries 13.7 vectors on the average. It is therefore not optimal for $n$ equal to 5. 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 3.1). Since there are about $10^7$ monotone Boolean functions for $n$ equal to 6, and about $10^{12}$ for $n$ equal to 7, these two 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.
5.2 Methodologies for Problems # 2.1, # 2.2, and # 2.3

The minimum average number of queries for the unrestricted problem $Q_3(n)$ is equal to that of the single function case in one dimension higher $Q(n+1)$. To see this connection consider a pair of nested monotone Boolean functions $f_1$ and $f_2$ defined on $\{0,1\}^n$. The query domain for the nested case can be viewed as the product: $\{0,1\}^n \times \{f_2,f_1\}$. Each of the vertices in the resulting poset ($\{0,1\}^{n+1}$, $\preceq$), may take on function values of 0 or 1, where the monotonicity property is preserved. In other words, a pair of nested monotone Boolean functions defined on $\{0,1\}^n$ are equivalent to a single monotone Boolean function defined on $\{0,1\}^{n+1}$.

As an example, consider inferring a pair of nested monotone Boolean functions defined on $\{0,1\}^2$. Figure 3.2 shows the query domain ($\{0,1\}^3$, $\preceq$) where the vertices are labeled $(v,f_i)$, for $i = 1$ and 2, and $v \in \{0,1\}^2$. Each vertex in this poset is in the form of a query, where a sample query now looks like $(01f_1)$. If the answer to that query is 0, then the vertices strictly preceding $(01f_1)$ (i.e., $(00f_1)$, $(01f_2)$, and $(00f_2)$) are also assigned the value 0. This leaves the vertices $(11f_1)$, $(10f_1)$, $(11f_2)$, $(10f_2)$ unclassified.

As a related note, consider a set of functions making up a poset of the form ($\{0,1\}^m$, $\preceq$) where each function is defined on the set of vectors of the form ($\{0,1\}^n$, $\preceq$). The poset that is subject to queries is then of the form ($\{0,1\}^n$, $\preceq$) $\oplus$ ($\{0,1\}^m$, $\preceq$) = ($\{0,1\}^{n+m}$, $\preceq$). For example, inferring four functions from four unrestricted oracles, defined on $\{0,1\}^5$, satisfying the following relations $f_1 \preceq f_2$, $f_1 \preceq f_3$, $f_2 \preceq f_4$, and $f_3 \preceq f_4$, is equivalent to inferring a single monotone Boolean function defined on $\{0,1\}^{2+5} = \{0,1\}^7$.

5.2.1 Defining Evaluative Criteria

The evaluative criterion $\min\{K_1(v)-K_0(v)\}$ was shown to be very efficient in minimizing the average number of queries in Problem # 1. It will therefore be used for the three nested problems with a slight modification.

The query domain for the nested case is made up of the set of vectors $\{0,1\}^n \times \{f_2,f_1\}$. For a vertex labeled $(v,f_i)$, let $K_i(v,f_i)$ be the number of vertices that are concurrently classified when the value of $f_i(v)$ is queried and the answer is $f_i(v) = z$, for $z = 0$ and 1. When the access to the oracles is unrestricted (i.e., Problem # 2.3), vertices are selected based on the criterion $\min\{K_1(v,f_i) - K_0(v,f_i)\}$. This criterion is equivalent to inferring a single monotone Boolean function defined on $\{0,1\}^{2+5} = \{0,1\}^7$.

For sequential oracles (i.e., Problem # 2.1), queries of the form $f_2(v)$ are infeasible until all of the queries of the form $f_1(v)$ are classified. In this case, the criterion used during the first phase is $\min\{K_1(v,f_i) - K_0(v,f_i)\}$, after which the criterion $\min\{K_1(v,f_j) - K_0(v,f_j)\}$ is used.

For the three-valued oracle (i.e., Problem # 2.2), the queries are of the form $(f_1(v), f_2(v))$ and are selected using the criterion $\min\{K_{11}(v) - K_{00}(v)\}$. Here the value of the function $K_{11}(v)$ equals the number of vertices concurrently classified when vertex $v$ is queried and the result of the query is $f_1(v) = f_2(v) = z$, for $z = 0$ and 1. Once there are no pairs of vertices of the form $(f_1(v), f_2(v))$ left unclassified, the criterion $\min\{K_1(v,f_j) - K_0(v,f_j)\}$ is used for the remaining of the query selections.

These evaluative criteria are further illustrated on the chain poset in Section 5.2.5. The inference applications described in Sections 2.3.2, 2.3.4, and 2.3.6 will be solved using these evaluative criteria next in Sections 5.2.2, 5.2.3, and 5.2.4, respectively.
5.2.2 Illustration of the Evaluative Criterion Approach to the Breast Cancer Diagnosis Application

Please recall from Section 2.3.2 that the two nested monotone Boolean functions for the breast cancer diagnosis applications were \(f_1(v) = v_1 v_2 \lor v_3 \lor v_4 v_5\) and \(f_2(v) = v_1 v_2 \lor v_3 \lor v_4 v_5 \lor v_2 v_3 \lor v_4 v_5\). Also recall that the five individual features are defined as follows:

- \(v_1 = 1\) if amount and volume of calcifications is “pro cancer”, 0 otherwise,
- \(v_2 = 1\) shape and density of calcifications is “pro cancer”, 0 otherwise,
- \(v_3 = 1\) ductal orientation is “pro cancer”, 0 otherwise,
- \(v_4 = 1\) comparison with previous exam is “pro cancer”, 0 otherwise, and
- \(v_5 = 1\) associated findings is “pro cancer”, 0 otherwise.

Figure 5.10 shows the sequence of queries and their answers, when the queries are selected based on the criterion \(\text{min}|K_1-K_0|\). Below each query, the remaining unclassified vectors are shown in the form of a poset with the selected vector circled. A vector of the form \((v_1, v_2, v_3, v_4, v_5)\) corresponds to a query of the form \(f_1(v_1, v_2, v_3, v_4, v_5)\), and a vector of the form \((v_1, v_2, v_3, v_4, v_5)\) corresponds to a query of the form \(f_2(v_1, v_2, v_3, v_4, v_5)\). During the inference of the biopsy recommendation function \(f_1\), only vectors of the form \((v_1, v_2, v_3, v_4, v_5)\) are feasible.

Initially, all the vectors in \(\{0,1\}^5 \otimes \{f_1, f_2\} = \{0,1\}^6\) are unclassified and several vectors on the middle layer possess the same minimum feasible value \(|K_1-K_0|=0\). One of these vectors \((000111) = (000111)\) is arbitrarily selected for the first query. The radiologist is asked whether a biopsy is recommended for a tumor with “pro cancer” features in comparison with previous examination \((v_4 = 1)\) and in associated findings \((v_5 = 1)\). After the radiologist answers “yes” (i.e., \(f_1(000111) = 1\)), the set of unclassified vectors is reduced and forms the poset shown below the 2nd query.

In this poset the minimum \(|K_1-K_0|\) value of 1 belongs to the vector \((111000) = (111000)\) which is an infeasible query at this point. However, there are several vectors with the minimum feasible \(|K_1-K_0|\) value of 2. The vector \((011001) = (011000)\) is one of them. The radiologist is asked whether a biopsy is recommended for a tumor with “pro cancer” features in shape and density of calcifications \((v_2 = 1)\) and in ductal orientation \((v_3 = 1)\). After the radiologist answers “yes” (i.e., \(f_1(011001) = 1\)), the set of unclassified vectors is further reduced and forms the poset shown below the 3rd query. This process continues and after the 10th query the biopsy recommendation function \(f_1\) is completely restored (i.e., all of the vectors of the form \((v_1, v_2, v_3, v_4, v_5)\) are classified).

At this point the inference of the cancer function \(f_2\) from the biopsy oracle begins. The feasible vectors are now of the form \((v_1, v_2, v_3, v_4, v_5)\), and make up the poset shown below the 11th query. The minimum \(|K_1-K_0|\) value of 0 belongs to vector \((110100) = (111000)\). A biopsy is performed on a tumor with “pro cancer” features in amount and volume of calcifications \((v_1 = 1)\) in shape and density of calcifications \((v_2 = 1)\), and in comparison with previous examination \((v_4 = 1)\). After the cancer is detected (i.e., \(f_2(110100) = 1\)), the set of unclassified vectors is further reduced and forms the poset shown below the 12th query. The querying continues for a total of 23 queries; the first 10 for the radiologist and the last 13 in the form of biopsies.

Kovalerchuk et al. (1996) reported a total of 26 queries when sequentially inferring \(f_1\) followed by \(f_2\), using a Hansel chains based approach. Their first 13 questions were used to infer \(f_1\) and their last 13 were used to infer \(f_2\). Thus, the evaluative criterion reduced the number of questions to the radiologist by 3, while it needed the same number of biopsies.

Please take notice of the vector \((111000) = (111000)\) corresponding to the vertex immediately to the right of the circled vertex \((110100)\) in the poset below the 11th query. It yielded the minimum \(|K_1-K_0|\) value for the 2nd query at which point it was infeasible. Immediately following the 10th query, it was still unclassified and became feasible. However, it yielded a \(|K_1-K_0|\) value of

52
Figure 5.10 Sequential inference of the two nested breast cancer diagnosis functions using the evaluative criterion $\min|K_1-K_0|$.

which was not the minimum. That is, the optimality of infeasible vertices is not necessarily preserved.
5.2.3 Illustration of the Evaluative Criterion Approach to the Record Linkage Application

Please recall from Section 2.3.4 that the pair of nested record linkage functions were given as follows:\( f_1(v) = v_1v_2v_3 \lor v_1v_2v_4 \lor v_2v_4v_5 \) and \( f_2(v) = v_1v_2v_3v_4 \lor v_1v_2v_3v_5 \lor v_1v_2v_4v_5 \lor v_2v_3v_4v_5 \). Also recall that the five variables were defined as follows:

\[
\begin{align*}
    v_1 &= 1, \text{ if } \text{first name initial}(r_A) = \text{first name initial}(r_B), \ 0 \text{ otherwise,} \\
    v_2 &= 1, \text{ if } \text{last name}(r_A) = \text{last name}(r_B), \ 0 \text{ otherwise,} \\
    v_3 &= 1, \text{ if } \text{state of residence}(r_A) = \text{state of residence}(r_B), \ 0 \text{ otherwise,} \\
    v_4 &= 1, \text{ if } \text{age}(r_A) = \text{age}(r_B), \ 0 \text{ otherwise, and} \\
    v_5 &= 1, \text{ if } \text{gender}(r_A) = \text{gender}(r_B), \ 0 \text{ otherwise.}
\end{align*}
\]

Figure 5.11 shows the sequence of queries and their answers for the record linkage application, when the queries are selected based on the criterion \( \min|K_{11} - K_{00}| \). Below each query, the remaining unclassified vectors are shown in the form of a poset with the selected vector circled.

**Figure 5.11** Simultaneous inference of the two nested record linkage functions from a three-valued oracle using the evaluative criterion \( \min|K_{11} - K_{00}| \) followed by \( \min|K_1 - K_0| \).
Initially, all of the vectors \( \{0,1\}^5 \otimes \{f_2, f_1\} = \{0,1\}^6 \) are unclassified. A query of the form \( f(v_j v_2 v_3 v_4 v_5) \) corresponds to the pair of vectors \( (v_j v_2 v_3 v_4 v_5 0) \) and \( (v_j v_2 v_3 v_4 v_5 1) \). When there are no unclassified pairs left, the queries are given in the form \( f_1(v_j v_2 v_3 v_4 v_5) \) or \( f_2(v_j v_2 v_3 v_4 v_5) \) to show what information was gained by the query \( f(v_j v_2 v_3 v_4 v_5) \).

Initially, several pairs of vectors possess the same minimum feasible value \( |K_{11111}K_{00000}| = 8 \). One of these pairs \( \{000110, 000111\} \) is arbitrarily selected for the first query. The database administrator is asked whether two records with the same age \( (v_3 = 1) \), and gender \( (v_5 = 1) \) should be merged. After the database administrator answers “no” \( (i.e., f(000110) = 0) \), the set of unclassified vectors is reduced and then forms the poset shown below the 2\sup{nd} query.

In this poset the minimum \( |K_{11111}K_{00000}| \) value of 0, belongs to the pair of vectors \( \{001110, 001111\} \). The database administrator is asked whether two records with the same state of residence \( (v_3 = 1) \), age \( (v_4 = 1) \), and gender \( (v_5 = 1) \) should be merged. After the database administrator answers “no” \( (i.e., f(001111) = 0) \), the set of unclassified vectors is reduced and then forms the poset shown below the 3\sup{rd} query. This process continues and after the 13\sup{th} query, there are no pairs of unclassified vectors of the form \( \{(v_j v_2 v_3 v_4 v_5 0), (v_j v_2 v_3 v_4 v_5 1)\} \).

From there on the evaluative criterion \( min|K_{11111} - K_{00000}| \) is used. From Figure 5.11, it is obvious that the remaining vectors \( (011110, 110110, 111100) \) are unrelated. As a result, all of them have to be queried and the query order does not matter. For the 13\sup{th} query, the database administrator is asked whether two records with the same last name \( (v_3 = 1) \), state of residence \( (v_3 = 1) \), age \( (v_4 = 1) \) and gender \( (v_5 = 1) \) should be merged. After the database administrator answers “yes” \( (i.e., f(011111) = 2) \), the set of unclassified vectors is reduced and then forms the poset shown below the 14\sup{th} query. After 15 queries to the database administrator, the record linkage function is completely restored.

Please note that it is known from the first 12 queries that \( f_1(011111) = f_1(110111) = f_1(111100) = 1 \). In other words, the information gained by the 13\sup{th}, 14\sup{th}, and 15\sup{th} queries are that \( f_2(011111) = f_2(110111) = f_2(111100) = 1 \), respectively, which are shown over the respective posets in Figure 5.11.

**5.2.4 Illustration of the Evaluative Criterion Approach to the College Acceptance Policy**

**Application**

Please recall from Section 2.3.4 that the lenient and the strict professors’ acceptance functions were given by: \( f_1(v) = v_1 \lor v_2 v_4 \lor v_3 v_4 \), and \( f_2(v) = v_1 v_2 v_4 \lor v_1 v_3 v_4 \), where the four variables were defined by:

\[
\begin{align*}
v_1 & = 1 \text{ for a high GPA}, \ 0 \text{ otherwise}, \\
v_2 & = 1 \text{ for a high GRE score}, \ 0 \text{ otherwise}, \\
v_3 & = 1 \text{ for an essay of high quality}, \ 0 \text{ otherwise}, \ \text{and} \\
v_4 & = 1 \text{ for good recommendations}, \ 0 \text{ otherwise}.
\end{align*}
\]

Figure 5.12 shows the sequence of queries and their answers for the student evaluation application, when the queries are selected based on the criterion \( min|K_{11111} - K_{00000}| \). Below each query, the remaining unclassified vectors are shown in the form of a poset with the selected vector circled.

Initially, all the vectors in \( \{0,1\}^4 \otimes \{f_2, f_1\} = \{0,1\}^5 \) are unclassified. A vector of the form \( (v_j v_2 v_3 v_4 1) \) corresponds to a query of the form \( f_1(v_j v_2 v_3 v_4) \), and a vector of the form \( (v_j v_2 v_3 v_4 0) \) corresponds to a query of the form \( f_2(v_j v_2 v_3 v_4) \). Several vectors on the two middle layers possess the same minimum feasible value \( |K_{11111} - K_{00000}| = 4 \). One of these vectors \( (00011) = (0001f_1) \), is arbitrarily selected for the first query. The lenient professor evaluates whether a student with good recommendations \( (v_4 = 1) \) should be accepted. After the lenient professor answers “no” \( (i.e., f_1(0001) = 0) \), the set of unclassified vectors is reduced and then forms the poset shown below the 2\sup{nd} query.
Figure 5.12 Simultaneous inference of the two nested college acceptance policy functions using the evaluative criterion $\min |K_1 - K_0|$. 

In this poset the minimum $|K_1 - K_0|$ value of 0, corresponds to the vectors (0011), (0101), and (1001). The vector (0011) = (0011 $f_1$) is arbitrarily selected. The lenient professor evaluates whether a student with an essay of high quality ($v_3 = 1$) and good recommendations ($v_4 = 1$) should be accepted. After the lenient professor answers “yes” (i.e., $f_1(0011) = 1$), the set of unclassified vectors is further reduced and forms the poset shown below the 3rd query. This process continues for an additional 7 queries to the lenient professor and 5 queries to the strict professor (not in that order) as shown in Figure 5.12. After, the 14th query, both professors’ acceptance functions, and hence the college acceptance policy, are completely restored.

5.2.5 Recursive Formulas for the Minimum Average Number of Queries on the Chain Poset

When a pair of nested monotone Boolean functions are defined on the chain poset, the query domain shown on the right hand side of Figure 5.13 is no longer a chain poset. Let the number of pairs of nested monotone Boolean functions defined on the chain poset be denoted by $N_2^c(h)$. It is easy to verify that $N_2^c(h) = N_2^c(h-1) + h + 1 = 1/2h^2 + 3/2h + 1$. Furthermore, let $Q_k^c(h)$ denote the minimum average number of queries used to infer a pair of nested monotone Boolean functions defined on a chain poset of height $h$, for $k = 1$ (i.e., sequential inference), $k = 2$ (i.e., three-valued oracle), and $k = 3$ (i.e., unrestricted inference).
Figure 5.13 Constructing the query domain for a pair of nested monotone Boolean functions defined on a chain poset.

The expressions of \( Q_k^c(h) \) for \( k = 1, 2, \) and 3 can be viewed as recursive functions. The simplest problem is when the oracle is three-valued (i.e., \( k = 2 \)):

\[
Q_2^c(h) = \min_{i \in \{1, 2, \ldots, h\}} \frac{Q_2^c(i-1)N_2^c(i-1) + Q_2^c(h-i)N_2^c(h-i) + 2Q^c(i-1) + Q^c(h-i)N^c(i-1)N^c(h-i)}{N_2^c(h)} + 2.
\]

The equation for \( Q_2^c(h) \) is obtained from the fact that an answer to querying the \( i \)-th vector from a three-valued oracle (i.e., query \( f_1(i) \) and \( f_2(i) \)) results in either:

- \( f_1(i) = f_2(i) = 1 \): a pair of nested functions defined on a chain of length \( i-1 \),
- \( f_1(i) = f_2(i) = 0 \): a pair of nested functions defined on a chain of length \( h-i \), or
- \( f_1(i) = 1, f_2(i) = 0 \): two unrelated functions defined on chains of lengths \( h-i \) and \( i-1 \).

A single access to this three-valued oracle counts as two queries since a pair of function values are provided. Therefore, the number of queries for the independent functions, \( Q_1^c(i) \) and \( Q_2^c(i-1) \), are multiplied by 2, and 2 is added at the end of the expression for \( Q_2^c(h) \).

A nice property of this recursive equation is that \( Q_2^c(h) \) can be computed for increasing \( h \), starting with \( Q_2^c(0) = 0 \) and \( Q_2^c(1) = 2 \), and searching among vertices \( i = 1, 2, \ldots, \lfloor h/2 \rfloor \). The total time it takes to compute \( Q_2^c(h) \) is therefore \( O(h^2) \).

For sequential inference (i.e., \( k = 1 \)), the minimum average number of queries is a bit more complicated to compute. In this case, \( f_1 \) has to be completely inferred before the inference of \( f_2 \) can begin. Let \( Q_1^c(h_1, h_2) \) denote the minimum average number of queries remaining when there are \( h_1 \) vertices left unclassified by \( f_1 \), and \( h_2 \) vertices left unclassified by \( f_2 \). Similarly, let \( N_1^c(h_1, h_2) \) denote the number of pairs of functions remaining for sequential inference when there are \( h_1 \) vertices left unclassified by \( f_1 \), and \( h_2 \) vertices left unclassified by \( f_2 \). Since the isomorphism of the posets encountered can be determined by the two invariants \( h_1 \) and \( h_2 \), the minimum average number of queries can be written as a recursive function as follows:

\[
Q_1^c(h_1, h_2) = \min_{i \in \{1, 2, \ldots, h_1\}} \frac{Q_1^c(h_1-i, h_2)N_1^c(h_1-i, h_2) + Q_1^c(i-1, h_2)N_1^c(i-1, h_2)}{N_1^c(h_1, h_2)} + 1,
\]

where \( h_2 \geq h_1 \).

The values for \( Q_1^c(h_1, h_2) \) can be computed using nested loops with \( h_2 = 0, 1, \ldots, h \), \( h_1 = 1, 2, \ldots, h_2 \), and \( i = 1, 2, \ldots, h_1 \), starting with \( Q_1^c(0, 0) = 0 \) and \( Q_1^c(0, h_2) = Q^c(h_2) \). Afterwards, \( Q_1^c(h) \) can be extracted from \( Q_1^c(h, h) \). The total time it takes to compute \( Q_1^c(h) \) is therefore \( O(h^3) \).
The recursive function for $Q_3^c(h)$ is more complex since more complex posets are encountered. The isomorphism of these posets can be determined by using a third invariant, namely the number of unclassified vertices that are common to both functions $f_1$ and $f_2$. Let $Q_3^c(h_1, h_2, h_3)$ denote the minimum average number of queries, when there are $h_1$ and $h_2$ vertices left unclassified by functions $f_1$ and $f_2$, respectively, and $h_3$ of these vertices are common to both functions $f_1$ and $f_2$. The values for $Q_3^c(h_1, h_2, h_3)$ can be computed in a recursive fashion as follows:

$$Q_3^c(h_1, h_2, h_3) = \min\{Q_{3,1}^c(h_1, h_2, h_3), Q_{3,2}^c(h_1, h_2, h_3)\} + 1,$$

where $h_1 \geq h_2 \geq h_3$, and $Q_{3,j}^c(h_1, h_2, h_3)$ denotes the average number of queries when an optimal vertex is selected for evaluation by $f_j$, for $j = 1$ and 2. As a result,

$$Q_{3,1}^c(h_1, h_2, h_3) = \min_{i \in \{1, 2, \ldots, h_1\}} Q_{3,1,0}^c(h_1, h_2, h_3) N_3^c(h_1, h_2, h_3) + Q_{3,1,1}^c(h_1, h_2, h_3) N_3^c(h_1, h_2, h_3),$$

where

$$Q_{3,1,0}^c(h_1, h_2, h_3) = Q_{3,1,0}^c(\max\{h_1, h_2\}, \min\{h_1, h_2\}, h_3),$$

and

$$Q_{3,1,1}^c(h_1, h_2, h_3) = Q_{3,1,1}^c(h_1, h_2, h_3).$$

The recursive equation for $Q_3^c(h)$ can be used to compute its values in a forward manner by nested loops increasing in the order $h_1 = 0, 1, 2, \ldots, h$; $h_2 = 0, 1, 2, \ldots, h_1$, followed by $h_3 = 0, 1, 2, \ldots, h_2$. Here, the optimal vertex must be selected from a set of $h_1 + h_2$ vertices (the vertices unclassified for $f_1$ and the vertices unclassified for $f_2$). The loops start with $Q_3^c(0,0,0) = 0$. After this three dimensional array is computed, $Q_3^c(h, h, h)$ can be extracted from $Q_3^c(h)$.

5.2.6 Optimal and Evaluative Criterion Approaches to the Chain Poset

The minimum average number of queries $Q_1^c(h)$ for $h = 1, 2, \ldots, 256$; $Q_2^c(h)$ for $h = 1, 2, \ldots, 1024$; and $Q_3^c(h)$ for $h = 1, 2, \ldots, 128$ were computed. Figure 5.14 shows these values divided by the number of vertices (i.e., $2^h$) as a function of the height of the chain poset, $h$.

As it turns out, the two curves corresponding to $Q_1^c(h)$ and $Q_3^c(h)$ overlap each other for $h = 1, 2, \ldots, 128$. That is, sequential access to the two oracles does not restrict the inference process from minimizing the average number of queries. The minimum average number of queries for the three-valued oracle is greater than that for sequential and unrestricted inference, as can be seen in Figure 5.14.

The matrix of plots in Figure 5.15 summarizes the results when the vertices are selected optimally and using an evaluative criterion for the three different types of oracles. For sequential and unrestricted oracles, the criterion $\min|K_1 - K_0|$ is used. For the three-valued oracle the criterion $\min|K_1 - K_0|$ is used until no more vertex pairs are available. For the remaining queries the criterion $\min|K_1 - K_0|$ is used.
Figure 5.14 The minimum average query complexity of Problem #2 defined on the chain poset.

Starting from the top, the first row of plots in Figure 5.15 shows the results for sequential oracles ($k=1$), the second row show the results for a three-valued oracle ($k=2$), and the third row shows the results for unrestricted oracles ($k=3$).

Starting from the left, the first column shows the average number of queries used by the evaluative criterion divided by the minimum average number of queries. For sequential oracles, the average number of queries added due to the evaluative criterion fluctuates between two curves that tend to increase with the height of the chain poset. These two curves begin at 0 for $h=1$, and level off for $h>175$ at about 2.3% and 2.8%, respectively. For a three-valued oracle, the average number of queries added due to the evaluative criterion, oscillates between 0% (when $\log_2(h)$ is an integer) and a curve that tends to decrease with the height of the chain poset. This curve levels off at about 0.7% for $h>750$. For the unrestricted oracle, the average number of queries added due to the evaluative criterion fluctuates between two curves that tend to increase with the height of the chain poset. These curves level off for $h>750$ at about 0.6% and 0.8%, respectively.

The second column shows the optimal vertices (solid lines) and the vertices selected by an evaluative criterion (dotted lines). For sequential oracles it is assumed, without loss of generality, that function $f_1$ is to be inferred first. The corresponding optimal vertices (first row of plots) precede the ones selected by the evaluative criterion. The optimal vertices tend to fluctuate to and from the vertices selected by using the evaluative criterion. At certain points the optimal vertices actually coincide with the ones selected by the evaluative criterion, and they are the furthest apart when $\log_2(h)$ is an integer.
For the three-valued oracle (second row of plots), there are potentially two pairs of optimal vertices, due to symmetry. For example, if a pair of vertices at position $x \leq \lfloor h/2 \rfloor$ is optimal, then the pair of vertices at position $h - x$ is also optimal. This is due to the fact that querying the $x$-th vertex may result in three different posets that are pairwise isomorphic to the three possible posets resulting from querying the $(h - x)$-th vertex. The symmetry property also holds for the evaluative criterion. The plot shows these curves, which are symmetric about $h/2$. For the sake of simplifying the discussion, consider the case when the pair of vertices is selected from the lower part of the poset (i.e., from $\{1, 2, ..., \lfloor h/2 \rfloor\}$). The optimal pair of vertices precedes the pair selected by the evaluative criterion. Furthermore, the optimal vertices fluctuate approximately opposite of the sequential oracle. That is, the optimal vertices coincide with the vertices selected using the evaluative criterion when $\log_2(h)$ is an integer, and they are the furthest apart when $h$ assumes some value in between.
For the two unrestricted oracles (third row of plots), there are potentially two optimal vertices, again due to symmetry. For example, if the vertex at position $x \leq \lfloor h/2 \rfloor$ is an optimal query for the 1st oracle, then the vertex at position $h - x$ is optimal for the 2nd oracle. The symmetry property also holds for the evaluative criterion. Therefore, the plot shows the vertex selections for both oracles. For the sake of simplifying the discussion, consider the case when the pair of vertices is selected from the lower part of the poset (i.e., from $\{1, 2, ..., \lfloor h/2 \rfloor\}$). The optimal vertices fluctuate similarly to those of the sequential oracle. The only difference is that the optimal vertices for the unrestricted problem both succeed and precede the vertices selected by the evaluative criterion.

The third column shows the reduction in efficiency due to the restriction on the oracles (in the 1st and 2nd rows) and the improvement in efficiency due to the nestedness assumption (in the 3rd row). The solid lines in these three plots correspond to the minimum averages, whereas the dotted lines correspond to the averages obtained by using an evaluative criterion.

For the sequential oracles (1st row of plots), there is no reduction in efficiency due to the restriction on the oracles since the solid line is horizontal at value 1. When the evaluative criterion approaches are used for sequential inference and unrestricted inference, there is a slight increase in the average number of queries due to the sequential restriction. This reduction in efficiency tends to fluctuate between two curves that tend to increase with the height of the poset. These two curves level off at 1.2% and 2.0%, respectively.

For the three-valued oracle (2nd row of plots), there is a significant increase in the minimum average number of queries due to the restriction on the oracle (solid curve). This increase grows rapidly from 20% to about 80% as the height of the chain poset grows from 1 to about 50. As the chain poset grows even larger, this increase tends to level off at about 84%. When the average number of queries used by the evaluative criterion for the three-valued oracle is compared to that of two unrestricted oracles (dotted curve), similar results hold. That is, the solid curve lies close to the dotted curve.

The lower plot in the 3rd column shows the reduction in the average number of queries due to the nestedness assumption. If the inference of $f_1$ and $f_2$ were treated independently, then it would take $2Q(h)$ queries to infer both of them on the average. However, if the access to the two oracles is unrestricted and the nestedness assumption is taken advantage of, then the minimum average number of queries is reduced to $Q_3(h)$. Therefore, the fraction $Q_3(h)/2Q(h)$ quantifies the reduction in the average number of queries due to the nestedness assumption. The solid curve corresponding to this fraction oscillates between two curves. The lower curve increases rapidly from 83.3% to about 91%, for $h = 1, 2, ..., 40$, and levels off at about 92% for larger chain posets. The upper curve increases rapidly from 83.3% to about 92%, for $h = 1, 2, ..., 30$, and levels off at about 94% for larger chain posets. In other words, there is a 6-8% decrease in the average number of queries due to the nestedness assumption.

The dotted curve shows the fraction of the average number of queries when the evaluative criterion is used for the unrestricted case over the average number of queries used by the evaluative criterion for the functions treated independently. Please note that the evaluative criterion is optimal for the independent functions, while it is not optimal for the nested functions. As a result, the dotted curve lies approximately 0.7% above the solid line.

5.3 Methodology for Problem # 3
This section develops the details of the core methodologies that will be used for the stochastic guided inference problem. Section 5.3.1 presents a detailed overview of the maximum flow algorithm used for the general error minimization problem. This algorithm is used as a basis for the incremental
5.3.1 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 dissertation, 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 5.16 (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.

\[
\text{CONSTRUCT-RESIDUAL-GRAPH}(V, m_1, m_0)
\]

1. For each \( v \in V \)
2. For each \( u < v \in V \), and \( \exists w \in V: u < w < v \)
3. \( c(v,u) \leftarrow M+1 \)
4. For each \( v \in V \)
5. If \( m_1(v) > m_0(v) \)
6. \( c(s,v) \leftarrow m_1(v) - m_0(v) \)
7. Elseif \( m_1(v) < m_0(v) \)
8. \( c(v,t) \leftarrow m_0(v) - m_1(v) \)
9. \( V \leftarrow V \cup \{s,t\} \)

**Figure 5.16** The algorithm used to construct the initial residual graph from the observed data.

In lines 1-3, edges are added for each pair of vectors \( v \) and \( u \in V \) where vector \( v \) strictly precedes \( u \), omitting the redundant precedence relations (i.e., relations where another vector \( w \) exists for which \( v < w < u \)). 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 5.20, and during the execution of the algorithm MAX-FLOW-INCREMENT1 shown in Figure 5.22, $M + 1$ is used.

In lines 5-6, 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.

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 5.2. 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.

<table>
<thead>
<tr>
<th>$v$</th>
<th>$m_1(v)$</th>
<th>$m_0(v)$</th>
<th>$m_1(v) - m_0(v)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>0</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>110</td>
<td>3</td>
<td>5</td>
<td>-2</td>
</tr>
<tr>
<td>101</td>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>011</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>010</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>001</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>000</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

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 5.17. 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 capacititated, while a capacity of $M + 1 = \min \{1+2+1, 3+2+2+1\} + 1 = 5$ is sufficient for practical purposes.

For the residual graph shown in Figure 5.18, 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 5.3.2 include 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 $m_1(v) - m_0(v)$ values are given in Table 5.2 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 5.18. The vertices $\{100, 110, 111\}$ all have a majority of 0-valued observations and their $m_1(v) - m_0(v)$ values are given in Table 5.2 as -1, -2, and -1, respectively. 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 5.18.
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 5.18, 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 \( t \) in the residual graph. As an example of a path from \( s \) to \( t \), consider the one going via 010 and 110 in Figure 5.18. 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 \textsc{BREADTH-FIRST-SEARCH}(G,s,t) shown in Figure 5.19 performs Breadth-First-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 \( \text{adj}[v] \). 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 \( f(v) = 1 \) on line 10).

\begin{center}
\begin{verbatim}
BREADTH-FIRST-SEARCH(G,s,t)
1 e ← 0
2 Q ← s
3 r(s) ← infinite %larger than max c
4 for each v ∈ V
5   f(v) ← 0
6   r(v) ← c(s,v)
7   e ← e + m_1(v)
8 while Q ≠ ()
9   v ← Q[1]
10  f(v) ← 1
11  Q ← Q − v
12  e ← e − m_1(v) + m_0(v)
13  for each u ∈ 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)}
\end{verbatim}
\end{center}

\textbf{Figure 5.19} The algorithm used to traverse the residual graph.

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 \( π(v) \)) 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 pseudocode in Figure 5.19 indicates, the modified algorithm also terminates whenever the sink \( t \) 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 \( π(t) \), 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 it is
labeled \( f(w) = 1 \). Therefore, the variable \( f \) used in BREADTH-FIRST-SEARCH, defines a monotone Boolean function, and the variable \( e \) holds the number errors \( e(f) \).

The algorithm FLOW-INCREASE shown in Figure 5.20 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 5.2 the initial residual graph is shown in Figure 5.18. 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 5.20.

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: 

\[
\begin{align*}
c(v(i-1), v(i-2)) &\rightarrow c(v(i-1), v(i-2)) - r, \\
c(v(i-2), v(i-3)) &\rightarrow c(v(i-2), v(i-3)) - r, \\
&\vdots \\
c(v^2, v^1) &\rightarrow c(v^2, v^1) - r.
\end{align*}
\]

Furthermore, the capacities along the same path in the opposite direction are increased by the same amount. That is, 

\[
\begin{align*}
c(v^1, v^2) &\rightarrow c(v^1, v^2) + r, \\
c(v^2, v^3) &\rightarrow c(v^2, v^3) + r, \\
&\vdots \\
c(v^{k-2}, v^{k-1}) &\rightarrow c(v^{k-2}, v^{k-1}) + r.
\end{align*}
\]

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 5.21 shows the graphs resulting from three executions of the FLOW-INCREASE algorithm, starting out with the residual graph created in Figure 5.18. The leftmost 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 5.21.

The residual graph after the third iteration (i.e., the rightmost graph in Figure 5.21) does not contain a path from the source vertex \( s \) to the sink vertex \( t \). The three vertices labeled 101, 011, and
Figure 5.21 The iterations of the algorithm FLOW-INCREASE on the sample data.

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 $f^* = v_1 v_3 \lor v_2$, and yields a total of $e(f^*) = m_0(111) + m_0(101) + m_0(011) + m_1(000) + m_1(100) + m_1(010) + m_1(001) + m_1(110) = 1 + 1 + 1 + 3 + 2 + 4 + 3 = 16$ errors. Therefore, $q^* = e(f^*)/m = 16/36 = 0.444$.

5.3.2 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., $m_0(v) \leftarrow m_0(v) + 1$), 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 $m_0(v) \leftarrow m_0(v) + 1$. 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 $1 - q_{old}^* \geq 1/2$.

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 of 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 5.22 shows the details of the update \( m_1(v) \rightarrow m_1(v) + 1 \) 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.

**Figure 5.22** 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 \( m_1(v) \rightarrow m_1(v) + 1 \), 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: \( c(s, v) \rightarrow c(s, v) + 1 \). 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., \( e(f^*) \rightarrow e(f^*) + 1 - f^*(v) \), 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: \( f^*(w) \rightarrow 1 \) for each vertex \( w \) reachable from \( v \) (including \( v \)). These steps (lines 4-9) take a total time of \( O(V^2) \).
The capacity on the edge (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) ← 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) ← c(v, t) - 1. This update is equivalent to c(t, v) ← c(t, v) - 1, and increasing the flow by 1 unit along a path from vertex v to vertex s, as shown in lines 15-16. The 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 the algorithm MAX-FLOW-INCREMENT1, consider Examples 5.1, 5.2, and 5.3, describing three different increment scenarios for the data given in Table 5.2. The initial residual graph for the MAX-FLOW-INCREMENT1 algorithm is given as the rightmost of the graphs in Figure 5.17. Please recall that the associated values are: m = 36, f* = v_1 v_3 \lor v_2 v_3, e* = 16, and q* = 16/36 = 0.444. Now consider the three following examples when another 1 valued vector (i.e., f(v) = 1) is observed.

Example 5.1. f(011) = 1 is observed:

m_1(011) = 3 \geq m_0(011) = 1 and so c(s, 011) - 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 ≈ 0.432. Only lines 1 and 2 were executed for this update using a total of O(1) time. The updated residual graph is shown in Figure 5.23.

Example 5.2. f(010) = 1 is observed:

m_1(010) = 2 \geq m_0(010) = 0 and so c(s, 010) - 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 \lor v_1 v_3, e* remains the same at 16, and finally q* is updated to 16/37 ≈ 0.432. Lines 1-4, 8, and 9 were executed, using a total of O(V^2) time. The updated residual graph is shown in Figure 5.24.

Example 5.3. 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) ← 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) - c(100, 000) - 1 = 1 - 1 = 0, c(000, 100) ← c(000, 100) + 1 = 5 + 1 = 6, c(000, s) - c(000, s) - 1 = 1 - 1 = 0, c(s, 000) ← c(s, 000) + 1 = 0 + 1 = 1. No path from s to t
exists, so $f^*$ is updated as follows: $f(000) \rightarrow 1, f(100) \rightarrow 1, f(010) \rightarrow 1, f(001) \rightarrow 1, f(110) \rightarrow 1$, 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 updated residual graph is shown in Figure 5.25.

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 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 $O(V)$.

5.3.3 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 properly 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 dissertation, 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:

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

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 5.2. The function $f^* = v_1v_3$ found in Section 5.3.1 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 $L(f^*) = (4/9)^{16}(1 - 4/9)^{36-16} = 1.818 \times 10^{-11}$. Notice how small this value is after only 36 observations. The likelihood values for the other functions are given in Table 5.3. Adding up all the likelihood values yields $(13 \times 1.455 + 2 \times 1.536 + 5 \times 1.818) \times 10^{-11} = 3.107 \times 10^{-10}$. Then the maximum likelihood ratio is computed as follows: $\hat{\lambda}(f^*) = 1.818 \times 10^{-11} / 3.107 \times 10^{-10} = 0.0585$.

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. Therefore, the selection criterion $min_K |K_0(v) - K_1(v)|$ 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 $min_K |K_0(v) - K_1(v)|$ 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
Table 5.3 Example likelihood values for all functions in $M_3$. 

<table>
<thead>
<tr>
<th>$f$</th>
<th>$e(f)$</th>
<th>$q(f)$</th>
<th>$L(f)$</th>
<th>$\lambda(f)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$</td>
<td>20</td>
<td>1/2</td>
<td>$1.455 \times 10^{-11}$</td>
<td>0.0468</td>
</tr>
<tr>
<td>$v_1 v_2 v_3$</td>
<td>21</td>
<td>1/2</td>
<td>$1.455 \times 10^{-11}$</td>
<td>0.0468</td>
</tr>
<tr>
<td>$v_1 v_2$</td>
<td>23</td>
<td>1/2</td>
<td>$1.455 \times 10^{-11}$</td>
<td>0.0468</td>
</tr>
<tr>
<td>$v_1 v_3$</td>
<td>18</td>
<td>1/2</td>
<td>$1.455 \times 10^{-11}$</td>
<td>0.0468</td>
</tr>
<tr>
<td>$v_1 v_2 \lor v_1 v_3$</td>
<td>20</td>
<td>1/2</td>
<td>$1.455 \times 10^{-11}$</td>
<td>0.0468</td>
</tr>
<tr>
<td>$v_1$</td>
<td>21</td>
<td>1/2</td>
<td>$1.455 \times 10^{-11}$</td>
<td>0.0468</td>
</tr>
<tr>
<td>$v_2 v_3$</td>
<td>19</td>
<td>1/2</td>
<td>$1.455 \times 10^{-11}$</td>
<td>0.0468</td>
</tr>
<tr>
<td>$v_1 \lor v_2 v_3$</td>
<td>19</td>
<td>1/2</td>
<td>$1.455 \times 10^{-11}$</td>
<td>0.0468</td>
</tr>
<tr>
<td>$v_1 v_3 \lor v_2 v_3$</td>
<td>16</td>
<td>4/9</td>
<td>$1.818 \times 10^{-11}$</td>
<td>0.0585</td>
</tr>
<tr>
<td>$v_1 v_2 \lor v_1 v_3 \lor v_2 v_3$</td>
<td>18</td>
<td>1/2</td>
<td>$1.455 \times 10^{-11}$</td>
<td>0.0468</td>
</tr>
<tr>
<td>$v_1 v_2 \lor v_2 v_3$</td>
<td>21</td>
<td>1/2</td>
<td>$1.455 \times 10^{-11}$</td>
<td>0.0468</td>
</tr>
<tr>
<td>$v_2$</td>
<td>19</td>
<td>1/2</td>
<td>$1.455 \times 10^{-11}$</td>
<td>0.0468</td>
</tr>
<tr>
<td>$v_1 \lor v_2$</td>
<td>17</td>
<td>17/36</td>
<td>$1.536 \times 10^{-11}$</td>
<td>0.0495</td>
</tr>
<tr>
<td>$v_2 \lor v_1 v_3$</td>
<td>16</td>
<td>4/9</td>
<td>$1.818 \times 10^{-11}$</td>
<td>0.0585</td>
</tr>
<tr>
<td>$v_3$</td>
<td>16</td>
<td>4/9</td>
<td>$1.818 \times 10^{-11}$</td>
<td>0.0585</td>
</tr>
<tr>
<td>$v_2 \lor v_3$</td>
<td>16</td>
<td>4/9</td>
<td>$1.818 \times 10^{-11}$</td>
<td>0.0585</td>
</tr>
<tr>
<td>$v_1 \lor v_2 \lor v_3$</td>
<td>17</td>
<td>17/36</td>
<td>$1.536 \times 10^{-11}$</td>
<td>0.0495</td>
</tr>
<tr>
<td>$v_1 \lor v_3$</td>
<td>19</td>
<td>1/2</td>
<td>$1.455 \times 10^{-11}$</td>
<td>0.0468</td>
</tr>
<tr>
<td>$v_3 \lor v_1 v_2$</td>
<td>18</td>
<td>1/2</td>
<td>$1.455 \times 10^{-11}$</td>
<td>0.0468</td>
</tr>
<tr>
<td>$T$</td>
<td>16</td>
<td>4/9</td>
<td>$1.818 \times 10^{-11}$</td>
<td>0.0585</td>
</tr>
</tbody>
</table>

In 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 Section 5.1, 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:

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

Then, consider defining the vector $v$ which “contributes the most errors” by $\max(E_0(v) + E_1(v))$. This vector selection criterion may lead to the same vector being invariably queried and hence it might suffer from convergence problems, as will be demonstrated empirically in Section 6.3.

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 $\Delta \lambda(v) = p(v)\lambda_z(v) + (1 - p(v))\lambda_0(v)$ 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 5.4 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.
a monotone Boolean function $f$, and a classification $z$, $e_z(001, f)$ and $\hat{\lambda}_z(001, f)$ here denote the updated number of errors and the likelihood ratio, respectively. The updated maximum likelihood ratios are $\hat{\lambda}_1(001) = \lambda_1(001, T) = 0.0649$ and $\hat{\lambda}_0(001) = \lambda_0(001, v_1v_3 \lor v_2v_3) = 0.0657$. 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 $\Delta \hat{\lambda}(001) = p^*(001)\hat{\lambda}_1(001) + (1- p^*(001))\hat{\lambda}_0(001) = 4/9 \times 0.0649 + 5/9 \times 0.0657 = 0.0653$.

Similar computations for the other vectors yield $\Delta \hat{\lambda}(000) = 0.0651$, $\Delta \hat{\lambda}(010) = 0.0654$, $\Delta \hat{\lambda}(011) = 0.0592$, $\Delta \hat{\lambda}(100) = 0.0652$, $\Delta \hat{\lambda}(101) = 0.0592$, and $\Delta \hat{\lambda}(111) = 0.0592$. 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.

The simulations in Section 6.3 reveal the efficiency of the evaluative criterion $max \Delta \hat{\lambda}(v)$ in terms of the query complexity. In terms of computational complexity it may take an exponential time (in the size of $V$) to compute $max \Delta \hat{\lambda}(v)$. 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.

<table>
<thead>
<tr>
<th>$f$</th>
<th>$\hat{\lambda}(f)$</th>
<th>$e_z(001, f)$</th>
<th>$\hat{\lambda}_1(001, f)$</th>
<th>$e_0(001, f)$</th>
<th>$\hat{\lambda}_0(001, f)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$</td>
<td>0.0468</td>
<td>21</td>
<td>0.0462</td>
<td>20</td>
<td>0.0468</td>
</tr>
<tr>
<td>$v_1v_2v_3$</td>
<td>0.0468</td>
<td>22</td>
<td>0.0462</td>
<td>21</td>
<td>0.0468</td>
</tr>
<tr>
<td>$v_1v_2$</td>
<td>0.0468</td>
<td>24</td>
<td>0.0462</td>
<td>23</td>
<td>0.0468</td>
</tr>
<tr>
<td>$v_1v_3$</td>
<td>0.0468</td>
<td>19</td>
<td>0.0462</td>
<td>18</td>
<td>0.0474</td>
</tr>
<tr>
<td>$v_1v_2 \lor v_1v_3$</td>
<td>0.0468</td>
<td>21</td>
<td>0.0462</td>
<td>20</td>
<td>0.0468</td>
</tr>
<tr>
<td>$v_1$</td>
<td>0.0468</td>
<td>22</td>
<td>0.0462</td>
<td>21</td>
<td>0.0468</td>
</tr>
<tr>
<td>$v_2v_3$</td>
<td>0.0468</td>
<td>20</td>
<td>0.0462</td>
<td>19</td>
<td>0.0468</td>
</tr>
<tr>
<td>$v_1 \lor v_2v_3$</td>
<td>0.0468</td>
<td>20</td>
<td>0.0462</td>
<td>19</td>
<td>0.0468</td>
</tr>
<tr>
<td>$v_1v_3 \lor v_2v_3$</td>
<td>0.0585</td>
<td>17</td>
<td>0.0522</td>
<td>16</td>
<td>0.0657</td>
</tr>
<tr>
<td>$v_1v_2 \lor v_1v_3 \lor v_2v_3$</td>
<td>0.0468</td>
<td>19</td>
<td>0.0462</td>
<td>18</td>
<td>0.0474</td>
</tr>
<tr>
<td>$v_1v_2 \lor v_2v_3$</td>
<td>0.0468</td>
<td>22</td>
<td>0.0462</td>
<td>21</td>
<td>0.0468</td>
</tr>
<tr>
<td>$v_2$</td>
<td>0.0468</td>
<td>20</td>
<td>0.0462</td>
<td>19</td>
<td>0.0468</td>
</tr>
<tr>
<td>$v_1 \lor v_2$</td>
<td>0.0495</td>
<td>18</td>
<td>0.0469</td>
<td>17</td>
<td>0.0529</td>
</tr>
<tr>
<td>$v_2 \lor v_1v_3$</td>
<td>0.0585</td>
<td>17</td>
<td>0.0522</td>
<td>16</td>
<td>0.0657</td>
</tr>
<tr>
<td>$v_3$</td>
<td>0.0585</td>
<td>16</td>
<td>0.0649</td>
<td>17</td>
<td>0.0529</td>
</tr>
<tr>
<td>$v_2 \lor v_3$</td>
<td>0.0585</td>
<td>16</td>
<td>0.0649</td>
<td>17</td>
<td>0.0529</td>
</tr>
<tr>
<td>$v_1 \lor v_2 \lor v_3$</td>
<td>0.0495</td>
<td>17</td>
<td>0.0522</td>
<td>18</td>
<td>0.0474</td>
</tr>
<tr>
<td>$v_1 \lor v_2 \lor v_3$</td>
<td>0.0468</td>
<td>19</td>
<td>0.0462</td>
<td>20</td>
<td>0.0468</td>
</tr>
<tr>
<td>$v_3 \lor v_1v_2$</td>
<td>0.0468</td>
<td>18</td>
<td>0.0469</td>
<td>19</td>
<td>0.0468</td>
</tr>
<tr>
<td>$T$</td>
<td>0.0585</td>
<td>16</td>
<td>0.0649</td>
<td>17</td>
<td>0.0529</td>
</tr>
</tbody>
</table>
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 \( m_v(v) / (m_v(v) + m_i(v)) \) \( \forall v \in \text{LU}(f) \) and \( m_w(w) / (m_w(w) + m_i(w)) \) \( \forall w \in \text{UZ}(f) \) all converge to \( q \). The number of errors performed by any other monotone Boolean function \( g \) is at least \( x = \min \{ \min \{ m_v(v) - m_o(v), v \in \text{LU}(f) \}, \min \{ m_w(w) - m_o(w), w \in \text{UZ}(f) \} \} \) greater than the number of errors performed by function \( f \). Furthermore, \( x = qm - (1-q)m = m(2q - 1) \) 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 \( L(f) / L(g) > (q/(1-q))^m \) converge to 0 as \( m \) goes to infinity. Since the number of other monotone Boolean functions is a finite number that does not depend on \( m \), the likelihood ratio \( \lambda(f) = L(f) / (L(f) + \sum L(g)) \) 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., \( v \in \text{LU}(f^*) \cup \text{UZ}(f^*) \)) 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 \( 1/2 \), 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 \( 1/2 \). 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.

### 5.4 An Algorithm for Randomly Generating Monotone Boolean Functions with Inclusion Probabilities

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

Inclusion probabilities are essential to unbiased estimators (see Section 3.6) and they need to be easy to compute. 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 unique pair of nested monotone Boolean functions, \( f_1 \) and \( f_2 \), that sandwich the original function by \( f_1 \geq f \geq f_2 \).

Figure 5.26 illustrates how this pair of functions is created from a non-monotone function. The shaded vectors carry a function value of 1, while the non-shaded vectors carry a function value
of 0. The non-monotone function $f$ shown in the top of the figure has upper zeros $\text{UZ}(f) = \{(1110), (1101), (0011)\}$ and lower units $\text{LU}(f) = \{(0100), (1001)\}$. The lower units of function $f$ uniquely define the first target monotone function $f_1$ shown in the bottom right of the figure. The upper zeros of $f$ uniquely define the second target monotone function $f_2$ shown on the bottom left.

**Figure 5.26** A pair of nested monotone Boolean functions created from a non-monotone Boolean function defined on $\{0,1\}^4$.

```
GENERATE-MBF-P(n,p)
1  LU = {}, UZ = {}
2  for each $v \in \{0,1\}^n$
3    $f(v)$ ← SELECT-AT-RANDOM($\{0,1\}$, $(1-p(v), p(v))$)
4    if $f(v) = 1$
5      $W$ ← $\{w: w \in \text{LU}, v \leq w\}$
6      if $W = \{}$
7        $\text{LU} = \text{LU} \cup \{v\}$
8      else
9        $\text{LU} = \text{LU} - W$
10     else
11       $W$ ← $\{w: w \in \text{UZ}, v \geq w\}$
12      if $W = \{}$
13        $\text{UZ} = \text{UZ} \cup \{v\}$
14      else
15        $\text{UZ} = \text{UZ} - W$
16  return $\text{LU}, \text{UZ}$
```

**Figure 5.27** The algorithm used to randomly generate monotone Boolean functions with inclusion probabilities.
The algorithm GENERATE-MBF-P\((n, p)\) shown in Figure 5.27 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 \(UZ\), and all the lower units into the set \(LU\). The first target monotone function \(f_1\) is then defined by the lower units \(LU\), while the second target monotone function \(f_2\) is defined by the upper zeros in \(UZ\).

5.4.1 Computing the Inclusion Probabilities for Problem # 1
After the two nested monotone Boolean functions have been generated by the algorithm GENERATE-MBF-P, their individual inclusion probabilities have to be computed. To that end, define the two random monotone Boolean functions \(F_1\) and \(F_2\) 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 or the greatest function, given by:

\[
\Pr\{F_1 = f \lor F_2 = f\} = \Pr\{F_1 = f\} + \Pr\{F_2 = f\} - \Pr\{F_1 = f \land F_2 = f\},
\]

where

\[
\Pr\{F_2 = f\} = \prod_{v \in LU(f)} p(v) \prod_{v \in \{w : f(w) = 0\}} (1 - p(v)).
\]

Similarly,

\[
\Pr\{F_1 = f\} = \prod_{v \in UZ(f)} (1 - p(v)) \prod_{v \in \{w : f(w) = 1\}} p(v),
\]

and finally

\[
\Pr\{F_1 = f \land F_2 = f\} = \prod_{v \in \{w : f(w) = 1\}} p(v) \prod_{v \in \{w : f(w) = 0\}} (1 - p(v)).
\]

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

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

The reasoning behind this particular definition of \(p(v)\) is provided in Section 5.4.3. Suppose the general function shown in the top of Figure 5.27 was generated as a result. Consider computing the inclusion probabilities for the function \(f_1\) shown on the bottom right of Figure 5.27. The lower units of \(f_1\) 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 by:

\[
q_k = 1 - (1 - q_1)^k.
\]
Here the inclusion probability for a single execution can be computed as follows:

\[
q_1 = p(0100)p(0101)(1-p(1010))(1-p(0011))(1-p(1000))(1-p(0010)) \times \\
(1-p(0001))(1-p(0000)) + (1-p(1010))(-p(0011))p(1000)p(1100) \times \\
p(1001)p(0110)(1-p(1110)p(1011)p(0111)p(1011)p(0111) \times \\
p(1111)p(1110)p(1111)p(0111)p(1011)p(0111)p(0101)p(1000)) \\
(1-p(1010))(1-p(0011))(1-p(1000))(1-p(0001))(1-p(0000)) \\
\times (1-p(0001))(1-p(0000)) + (1-p(1010))(-p(0011))p(1000)p(1100) \times \\
p(1001)p(0110)(1-p(1110)p(1011)p(0111)p(1011)p(0111) \times \\
p(1111)p(1110)p(1111)p(0111)p(1011)p(0111)p(0101)p(1000)) \\
(1-p(1010))(1-p(0011))(1-p(1000))(1-p(0001))(1-p(0000)) \\
= \left( \frac{5}{42} \right)^3 \left( \frac{1}{2} \right)^3 \left( \frac{37}{42} \right) \left( \frac{167}{168} \right) + \left( \frac{1}{2} \right)^4 \left( \frac{37}{42} \right) \left( \frac{167}{168} \right) - \left( \frac{167}{42} \right)^2 \left( \frac{37}{42} \right) \left( \frac{5}{42} \right) \approx 0.01047
\]

For example, if GENERATE-MBF-P was executed 20 times, the inclusion probability would be equal to \(q_{20} \approx 1 - (1 - 0.01047)^{20} \approx 0.1898\) for function \(f_G\). This value is fairly close to the ideal uniform case where the inclusion probabilities are \(1 - (1 - 1/\Psi(4))^{20} \approx 0.1126\). The inclusion probability for other monotone Boolean functions can be computed in a similar fashion.

### 5.4.2 Computing the Inclusion Probabilities for Problem # 2

To compute the inclusion probability of a pair of nested monotone Boolean functions generated by GENERATE-MBF-P, define the two random monotone Boolean functions \(F_1\) and \(F_2\) by the output of a single execution of the algorithm. The inclusion probability for the pair of functions \(f_1\) and \(f_2\) is simply the product of the individual vector assignment probabilities, excluding the vectors that lie in between the \(LU(f)\) and \(UZ(f)\), given by, \(S = \{0,1\}^n - \{w : w^1 \prec w \prec w^2, w^1 \in LU(f), w^2 \in UZ(f)\}\). The inclusion probability is given by the following equation:

\[
Pr\{F_1 = f_1 \wedge F_2 = f_2\} = \prod_{v \in \{w : f(w) = 1, w \in S\}} p(v) \prod_{v \in \{w : f(w) = 0, w \in S\}} (1-p(v)).
\]

Suppose the pair of nested monotone Boolean functions \(f_1\) and \(f_2\) shown at the bottom of Figure 5.27 were generated using the same definition for \(p(v)\) as in Section 5.4.1. The upper zeros are given by the set \(UZ(f) = \{(1110), (1101), (0011)\}\) and the lower units are given by the set \(LU(f) = \{(0100), (1001)\}\). As a result, the set of fixed vectors are given by \(S = \{0,1\}^n - \{1100, 0110, 0101\}\). The inclusion probability from a single execution of the algorithm GENERATE-MBF-P can then be computed as follows:

\[
q_1 = p(1111)p(1010)p(0110)p(0010)(1-p(1000))(1-p(1010))(1-p(0011))(1-p(1000))(1-p(0001))(1-p(0000)) \times \\
(1-p(1010))(1-p(0011))(1-p(1000))(1-p(0001))(1-p(0000)) \\
= \left( \frac{167}{168} \right)^2 \left( \frac{37}{42} \right)^5 \left( \frac{1}{2} \right)^3 \left( \frac{5}{42} \right)^3 \approx 0.0001106.
\]

Now, if the algorithm GENERATE-MBF-P was executed 20 times, the inclusion probability would be equal to \(q_{20} \approx 1 - (1 - 0.0001106)^{20} \approx 0.002210\) for the pair of functions \(f_1\) and \(f_2\). This value is fairly close to the ideal uniform case where the inclusion probabilities are \(1 - (1 - 1/\Psi(5))^{20} \approx 0.002635\). The inclusion probability for other pairs of nested monotone Boolean functions can be computed in a similar fashion.

### 5.4.3 Approximating the Ratio \(\Psi_k(n) / \Psi(n)\)

The goal of this section is to come up with a definition for \(p(v)\) so that the algorithm GENERATE-MBF-P generates monotone Boolean functions from a distribution that is as close to uniform as possible. To that end, \(p(v)\) is defined as the fraction of monotone Boolean functions in \(M_n\) that
classify the vector \( v \in \{0,1\}^n \) as 1. For example, the inclusion probabilities for the \( T \) and \( F \) functions in Problem # 1, are very close to what they would be in the uniform case since \( \Pr\{F_1 = T\} = \Pr\{F_1 (000...0) = 1\} = 1/\Psi(n) \) and \( \Pr\{F_2 = F\} = \Pr\{F_2 (111...1) = 0\} = 1/\Psi(n) \). This procedure does not result in uniform inclusion probabilities for all monotone Boolean functions. However, it is probably as close to uniform as possible using the independent classifications of the algorithm GENERATE-MBF-P. It should be noted that no definition for \( p(v) \) can make the algorithm GENERATE-MBF-P generate monotone Boolean functions from a uniform distribution.

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 \( v \) should be classified 1 with probability \( \Psi_{|v|}(n) / \Psi(n) \). The following equation gives the known generalizations of \( \Psi_k(n) \):

\[
\Psi_k(n) = \begin{cases} 
1, & k=0 \\
\Psi(n-1), & k=1 \\
\Psi(n)/2, & k=n/2, \text{ and } n \text{ is even} \\
\Psi(n)-\Psi(n-1), & k=n-1 \\
\Psi(n)-1, & k=n 
\end{cases}
\]

This equation indicates that computing \( \Psi_k(n) \) is just as hard as computing \( \Psi(n) \). Some of the exact values for \( \Psi_k(n) \) were computed and are listed in Table 5.5.

<table>
<thead>
<tr>
<th>( k )</th>
<th>( n = 1 )</th>
<th>( n = 2 )</th>
<th>( n = 3 )</th>
<th>( n = 4 )</th>
<th>( n = 5 )</th>
<th>( n = 6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>20</td>
<td>168</td>
<td>7,581</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>14</td>
<td>84</td>
<td>2,008</td>
<td>595,649</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>19</td>
<td>148</td>
<td>5,573</td>
<td>3,914,177</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>167</td>
<td>7,413</td>
<td>7,232,705</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7,580</td>
<td>7,828,353</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>7,828,353</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The solid curves in Figure 5.28 correspond to \( \Psi_k(n) / \Psi(n) \) for \( n = 1, 2, ..., 6 \). These curves exhibit symmetric S-shapes that change rapidly as \( n \) increases. This is the motivation behind the transformation of the distribution function into a sigmoid function as follows:

\[
\frac{\Psi_k(n)}{\Psi(n)} = \frac{\alpha(n,k)(k-n/2)}{1 + \alpha(n,k)(k-n/2)}.
\]

Since \( \alpha(n,k) \) is a function of \( n \) and \( k \), no information is lost or gained by this transformation. The problem of approximating \( \Psi_k(n) / \Psi(n) \) is merely transformed into the equivalent problem of approximating the function \( \alpha(n,k) \).

Some of the exact values of \( \alpha(n,k) \) are given in Table 5.6. An approximate relationship between the known \( \alpha(n,k) \) values is more transparent than between the known values for \( \Psi_k(n) \) given in Table 5.5.
Solving the sigmoid equation for $\alpha(n, k)$ yields,

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

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

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

The relationship $\Psi_i(n) = \Psi(n-1)$ also holds, which further implies the following equation:

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

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 3.1. For $n$ greater than 8 Korshunov’s approximation given in Section 3.1 can be used. However, to compute $\alpha(n, k)$ for $k > 1$, a
generalization is needed. Based on the observed values in Table 5.6, it seems reasonable to use the following approximation:

\[ \alpha(n, k) \approx \alpha(n-1, k-1) \text{ for } k = 1, 2, \ldots, \left\lfloor \frac{n}{2} \right\rfloor - 1, \]

and then use the fact that \( \alpha(n, k) = \alpha(n-k, k) \) to find the \( \alpha(n, k) \) values for \( k = \left\lceil \frac{n}{2} \right\rceil + 1, \ldots, n. \)

Figure 5.28 shows the approximated curves, for \( n = 1, 2, \ldots, 6, \) resulting from using this approximation and generalizing with \( \alpha(n, k) \approx \alpha(n-1, k-1) \) and \( \alpha(n, k) = \alpha(n-k, k) \) for \( k = 1, 2, \ldots, \left\lfloor \frac{n}{2} \right\rfloor - 1. \) The approximations based on \( \alpha(n, k) \approx \alpha(n-1, k-1) \) are probably worse for larger \( k. \)

Approximations of \( \Psi(n)/\Psi(n) \) that are off by a few percentage points will not have a major effect on the experiments described in Sections 6.1 and 6.2. Fortunately, these approximations are only needed for \( n \) up to 13 (i.e., for \( k \) up to 6). 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, \( \Psi(11) > 10^{144} \) by using Korshunov’s approximation given in Section 3.1.
CHAPTER 6. EXPERIMENTAL RESULTS

The goal of this chapter is to compare and contrast the efficiency of existing inference algorithms to the ones developed in this dissertation. Sections 6.1, 6.2, and 6.3 provide extensive computational results for Problems # 1, # 2, and # 3, respectively.

6.1 Experimental Results for Problem # 1

The preexisting inference algorithms described in Section 3.2 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, the subroutine FIND-BORDER needs to be fed unclassified vectors, of which there may be many. Even the evaluative criterion \( \min|K_1-K_0| \) which was described in Section 5.1.3 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 6.1 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 horizontal axes of the respective histograms. The average number of queries for an inference algorithm \( A \) is given by

\[
\sum_{f \in M_5} \varphi(A, f) / \Psi(5).
\]

For the Teacher, the evaluative criterion \( \min|K_1-K_0| \), Hansel’s algorithm, Sokolov’s algorithm, and an 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 minimum average.

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 6.2 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 \( \Psi_p(n) / \Psi(n) \) constructed in Section 5.4.3 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, 2,000 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 sizes were generally close to these maxima.

The Horvitz-Thompson estimator given in Section 3.6 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 magnitudes of the averages in Figure 6.2 are not overshadowed by the large values obtained for \( n \) equal to 11. As a consequence, two algorithms that result in parallel curves in such a plot, have an exponential (in \( n \)) difference in the average number of queries. Also, the gap between the curves in Figure 6.2 and the horizontal line \( \text{Average Number of Queries} / 2^n = 1 \) (not shown in the figure) can be thought of as the benefit of the monotone assumption. This is due to the
The query complexities of the inference algorithms used for Problem # 1 distributed over all monotone Boolean functions in the set $M_5$.

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 6.2 paints a clear picture of how the preexisting inference algorithms fare against each other. Hansel’s algorithm was the best performer by far, Sokolov’s came in second, and an algorithm using the subroutine FIND-BORDER (which is also used by Gainanov (1984), Valiant (1984), Makino and Ibaraki (1995), and Boros et al. (1997)) was a distant third. In fact, since the curve differences between Hansel and Sokolov, and Sokolov and the subroutine FIND-BORDER implementation, seem to increase with $n$, the corresponding difference in the average number of queries increases at rate greater than exponentially with $n$. 
The difference between the curves for Hansel and “Teacher” decreases as \( n \) increases. The algorithm based on the criterion \( \min|K_1-K_0| \) has a curve that is almost parallel to Hansel’s curve, indicating that the evaluative criterion performs about 2% better than Hansel’s algorithm. This decrease is especially clear in Figure 6.2 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 6.3 shows the number of queries each of the inference algorithms uses per border vector on the average. That is, for each inference algorithm \( A \), the quantity given by:

\[
\sum_{f \in \mathcal{M}_n} \frac{\varphi(A, f)}{\varphi(Teacher, f)}
\]

is computed for \( n = 1, 2, ..., 11 \). Again, the average values for \( n = 1, 2, ..., 5 \) are exact values, and the average values for \( n = 6, 7, ..., 11 \) are estimated. The same ranking as for the average query complexity (given in Figure 6.2) holds for the average number of queries per border vector when \( n = 1, 4, 5, ..., 11 \).

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. In other words, on the average the inference algorithms tend to perform a constant number of queries per border vector. In contrast, the worst case complexity of \((n+1)m(f)\) given for FIND-BORDER in Section 3.2, indicates that the maximum number of queries per border vector increases linearly with \( n \). In Section 3.2 it was also shown that Hansel’s algorithm may perform an exponential number of queries per border vector in the worst case. On the average it tends to perform a constant number of queries per border vector as indicated by Figure 6.3.

6.2 Experimental Results for Problem # 2
The different evaluative criteria were described in Section 5.2.1 without specifying which vector to select when there are ties. For the purpose of comparing the algorithms on the same ground and without introducing another aspect of randomness, ties were broken by selecting the first vector in the list of tied vectors.

The results in Figures 6.4, 6.5, and 6.6 are based on an exhaustive analysis (i.e., all the monotone functions were generated) for \( n \) up to and including 4. For \( n = 4, 5, ..., 12 \) random samples of functions were generated as described in Section 5.4. The number of pairs of nested monotone Boolean functions generated were 2,000 for \( n = 5, 6, 7 \), and 200 for \( n = 8, 9, 10 \), and 100 for \( n = 11 \) and 12. This is the maximum number of pairs of functions used in the estimate, because the functions were 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.

Since Problem \# 2.3 with \( n \) variables is equivalent to Problem \# 1 with \( n+1 \) variables, the following relationship holds: \( Q_3(n) = Q(n+1) \). As a result, the evaluative criterion \( \min |K_1 - K_0| \) is optimal for Problem \# 2.3 with 3 or fewer variables and probably close to optimal for 4 or more variables.

The Horvitz-Thompson (1952) estimator is used to compute the averages for \( n \) greater than 4. The average number of queries is normalized by the maximum possible number of queries \( 2^{n+1} \) so that the magnitudes of the averages in Figure 6.4 are not overshadowed by the large values obtained for \( n \) equal to 12. 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.

The gap between the curves in Figure 6.4 and the horizontal line \( \text{Average Number of Queries} / 2^{n+1} = 1 \) (the uppermost line of the box around the curves) can be thought of as the benefit of the monotone and nestedness assumptions together. This is due to the fact that \( 2^{n+1} \) is the number of required queries when the underlying pair of functions are neither nested or monotone. For example, when \( n = 12 \) in the unrestricted problem \( (k = 3) \) the average number of queries is reduced to about 20% of the maximum number of queries \( 2^{13} = 8,192 \) due to the monotone and nestedness assumptions.
Figure 6.5 Increase in average query complexities due to restricted access to the oracles of Problem # 2 defined on the poset \( \{0,1\}^n \).

The evaluative criterion \( \min|K_1-K_0| \) used for the unrestricted problem achieves the minimum average number of queries for \( n \) up to 3. Its curve (i.e., lower curve in Figure 6.4) can therefore be thought of as a lower bound on the average number of queries for \( n \) up to 3, and is slightly above the minimum for \( n \) greater than 3.

Figure 6.4 shows the average number of queries for Problem # 2 when using the evaluative criteria. The lower curve corresponds to the unrestricted case, which achieves the fewest number of queries on the average. In contrast to the chain poset, the sequential case (i.e., the middle curve) is not as efficient as the unrestricted oracles, though they are very close for \( n = 1, 2, 3, \) and 4. The least efficient of the three types of oracles is the three-valued one (i.e., the upper curve).

Figure 6.5 quantifies the increase in the average number of queries due to the two restrictions on the oracles. As mentioned earlier, the sequential oracles are not very restrictive for \( n = 1, 2, 3, \) and 4, with the greatest increase in the average number of queries at about 1% occurring for \( n = 3 \), and 0 for \( n = 1, 2 \) and 4. For \( n \) greater than 4, the increase in average query complexity due to sequential oracles fluctuates between two curves that increase with the number of variables. The lower curve corresponds to the even number of variables and it levels off at about 12% for \( n \) greater than 9. The upper curve correspond to the odd number of variables and it levels off at about 33% for \( n \) greater than 6. That is, the increase in the average number of queries due to the sequential restriction is between 12% and 33% for \( n \) greater than 6.
Figure 6.6 Improvement in average query complexity of the evaluative criterion approach due to the nestedness assumption of Problem #2 defined on the poset \( \{0,1\}^n \).

In contrast, the three-valued oracle is much more restrictive. The increase in the average number of queries due to the three-valued oracle, fluctuates between 35% and 55%. The increases in the average number of queries for the sequential and three-valued cases are dramatic. This is probably due to the fact that the average number of queries increases exponentially with the number of variables.

If the nested property of the two functions defined on \( \{0,1\}^n \) is ignored, the minimum total number of questions is on the average \( 2Q(n) \). The benefit from the nestedness assumption for Problem #2 is quantified by the ratio of \( Q_3(n)/2Q(n) \) which is given in Figure 6.6 for \( n = 1, 2, \ldots, 12 \). Therefore, the curves given in Figure 6.6 show the reduction in the average number of queries due to the nestedness assumption. This reduction decreases with the number of variables. It starts out at 20% for \( n = 1 \), and oscillates between 1% and 10% for \( n \) greater than 7.

6.3 Experimental Results for Problem #3
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 \( v \in LU(f^\#) \cup UZ(f^*) \) for Stage 3 were broken randomly. The four different inference processes using \( \max(\Delta \lambda(v)) \), \( v \in LU(f^\#) \cup UZ(f^*) \), \( \max(E_0(v) + E_1(v)) \), or random selection for Stage 3 were simulated on the set of vertices \( \{0,1\}^n \).
For all three Stage 3 selection criteria, the criterion \( \min(K_n(v) - K_n(v)) \) 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 computationally too burdensome for the 600 Mhz Pentium III based personal computer with 384 Mbytes of RAM used in these experiments. This limitation is also 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 6.1. 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.

Table 6.1 The representative functions used in the simulations of Problem # 3.

<table>
<thead>
<tr>
<th>( n = 2 )</th>
<th>( n = 3 )</th>
<th>( n = 4 )</th>
<th>( n = 5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( F )</td>
<td>( F )</td>
<td>( v_1v_2 \lor v_3v_4 \lor v_1v_4v_4 )</td>
<td>( v_1v_4 \lor v_1v_5 \lor v_2v_4 \lor v_2v_5 )</td>
</tr>
<tr>
<td>( v_1v_2 )</td>
<td>( v_1v_3 )</td>
<td>( v_1v_2 \lor v_1v_3 \lor v_1v_3 \lor v_2v_4 \lor v_3v_4 )</td>
<td>( v_1v_3 \lor v_2v_3 \lor v_2v_4 \lor v_1v_2v_5 )</td>
</tr>
<tr>
<td>( v_1 )</td>
<td>( v_2 )</td>
<td>( v_2v_3 \lor v_1v_4 )</td>
<td>( v_2 \lor v_1v_3v_4 \lor v_2v_5 )</td>
</tr>
<tr>
<td>( v_2 \lor v_1v_2 )</td>
<td>( v_1v_3 \lor v_1v_4v_4 \lor v_2v_4v_4 )</td>
<td>( v_1v_3 \lor v_2v_4 \lor v_3v_5 \lor v_4v_5 )</td>
<td></td>
</tr>
<tr>
<td>( v_1 \lor v_2 )</td>
<td>( v_1 )</td>
<td>( v_1v_2 \lor v_2v_4 \lor v_3v_4 )</td>
<td>( v_2v_4 \lor v_2v_5 \lor v_3v_5 \lor v_4v_5 )</td>
</tr>
<tr>
<td>( T )</td>
<td>( v_1v_2 \lor v_1v_3 \lor v_2v_3 )</td>
<td>( v_3 \lor v_1v_2 \lor v_1v_4 )</td>
<td>( v_3v_5 \lor v_1v_2v_3 \lor v_1v_3v_4 \lor v_1v_4v_5 \lor v_2v_5v_4 )</td>
</tr>
</tbody>
</table>

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 \( F, v_1v_2v_3, v_1v_2, v_1v_2 \lor v_1v_3, v_2, \) and \( v_1v_2 \lor v_1v_3 \lor v_2v_3 \), 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 the sampling of functions as done for \( n = 4 \) and 5.

Figure 6.7 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 corresponding to the evaluative criterion \( \max \Delta\lambda(v) \) has a maximum likelihood ratio that is less than 0.99.

Not only do the curves corresponding to the guided selection criteria \( \max \Delta\lambda(v) \) and \( v \in LU(f^*) \cup UZ(f^*) \) 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 \( v \in LU(f^*) \cup UZ(f^*) \) uses to reach about 0.9, for \( n = 4 \).

The difference between the curves for unguided selection and these two guided selections grows with the misclassification probability \( q \) and with the dimension \( n \). That is, the benefits from actively selecting vectors over passively receiving observations are greater when the values of \( q \) and
Figure 6.7 Average case behavior of various evaluative criteria used for Problem # 3.

$n$ are large. In other words, the higher the misclassification probability and the dimension of the problem are, the greater become the benefits of guiding the inference process.

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 the vector domain is, the more important it becomes to define an appropriate vector selection criterion.
Table 6.2 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 6.2: The average number of Stage 3 queries used by the evaluative criterion \( \max \Delta \lambda(v) \) to reach \( \lambda > 0.99 \) in Problem # 3 defined on \( \{0,1\}^n \) with fixed misclassification probability \( q. \)

<table>
<thead>
<tr>
<th></th>
<th>( q = 0.1 )</th>
<th>( q = 0.2 )</th>
<th>( q = 0.3 )</th>
<th>( q = 0.4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n = 2 )</td>
<td>22</td>
<td>54</td>
<td>125</td>
<td>560</td>
</tr>
<tr>
<td>( n = 3 )</td>
<td>27</td>
<td>65</td>
<td>170</td>
<td>710</td>
</tr>
<tr>
<td>( n = 4 )</td>
<td>33</td>
<td>85</td>
<td>241</td>
<td>951</td>
</tr>
<tr>
<td>( n = 5 )</td>
<td>45</td>
<td>111</td>
<td>277</td>
<td>1167</td>
</tr>
</tbody>
</table>

Figure 6.8 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 misclassification 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 \( a2^{bn} + c(q)2^{dn} + e, \) 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 5.3.1 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 to the data in Table 6.2. Thus, the total time used by the incremental algorithm is less than \( O(V^3). \) The fastest known general purpose maximum flow algorithm is of \( O(V^3) \) time complexity. This is the complexity of the problem if all the data was 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 computing the evaluative criterion \( \max \Delta \lambda(v), \) 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 completely random selections.
Please recall from Section 5.3.3 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 \( \frac{1}{2} \). 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 for \( q(v) \) increases with \( \delta \), 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 \( \delta \). To compare this estimate to an unrestricted estimate, the inference
Figure 6.9 The unrestricted and regular maximum likelihood ratios simulated for Problem # 3 with expected \( q = 0.2 \) and \( n = 3 \) using the evaluative criterion \( \max \Delta \lambda(v) \).

The process was simulated 200 times for each \( \delta = 0, 0.5, \) and \( 1, \) when \( q = 0.2, \) and \( n = 3 \). Figure 6.9 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 \( \delta \) 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. For the case when \( q(v) \) is fixed at 0.2 (i.e., \( \delta = 0 \)), the regular maximum likelihood ratio should be used, and when \( \delta > 0 \) it is an overestimate of the true maximum likelihood ratio. For the case when \( \delta = 1 \), the unrestricted maximum likelihood ratio should be used, and when \( \delta < 1 \) it may be an underestimate. The true likelihood ratio lies somewhere in between the two.
CHAPTER 7. CONCLUDING REMARKS

This chapter is organized as follows. Section 7.1 summarizes the computational results given in Chapter 6. Section 7.2 highlights the significance and potential impact these results may have on the fields of Knowledge Discovery and Data Mining, and Information Technology in general. In the end, section 7.3 proposes the beginning of a new wave of research expanding the scope of this dissertation.

7.1 Summary of the Research Findings

The recent focus on the computational complexity has come at the expense of a drastic increase in the query complexity for Problem # 1. In fact, the more recent the inference algorithm is, the worse it performs in terms of the average query complexity (and the average query complexity per border vector for \( n > 3 \))! The subroutine, here referred to as FIND-BORDER, is the most commonly used in the recent literature (Gainanov (1984), Valiant (1984), Makino and Ibaraki (1995), and Boros et al. (1997)), and its performance was by far the worst. Therefore, the framework for unbiased empirical comparison of inference algorithms developed in this dissertation 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 used for Problem # 1 revealed the non-intuitive nature of the optimal solutions. These solutions paved the way for the new evaluative criterion \( \min |K_1-K_0| \). This criterion would probably not have been developed (due to its non-intuitive nature) without the consultation of the optimal solutions.

The inference algorithm based on this evaluative criterion extends the feasible problem sizes to up to about 20 variables (which involves about 1 million vectors) for Problem # 1. 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 at a time by using the subroutine FIND-BORDER is the last but perhaps still computationally feasible resort.

Problem # 2 focused on the extension of the single monotone Boolean function inference problem to the inference of a pair of nested monotone Boolean functions. The benefits of this research are manyfold. First, it shows how the optimal and evaluative criterion approach to minimizing the average query complexity is extended to three different inference applications using a pair of nested monotone Boolean functions. The evaluative criteria seem to be good choices for the nested inference problem. They result in a slight increase in the average query complexity for the chain poset. For the poset \( \{0,1\}^n \), they are optimal for \( n = 1, 2, 3 \) and are probably very close to optimal for \( n \) greater than 3.

Second, it demonstrates how the nested monotone Boolean function model often is sufficient (i.e., a more complex model is not needed) and necessary (i.e., simpler models are not sufficient) for a wide variety of real world applications. Suppose a simpler model, such as a single monotone Boolean function, is used for these applications. At best, the simpler model will provide a poor approximation of the phenomenon under study. At worst, it will be unable to model the phenomenon. Suppose a more complex model, such as two independent monotone Boolean functions, is used for similar applications. Then, at the very least, the query complexity will increase. In addition, the inferred functions may lead to conflicting knowledge and are more likely to contain errors.
Third, it quantifies the reduction in query complexity due to the nestedness assumption. The improvement due to the nestedness assumption is between 6% and 8% for larger chain posets ($h > 50$). This improvement is greater for smaller chain posets, reaching its maximum of 20% for $h = 2$. In general, the average query complexity on the chain poset is $O(\log(h))$, so this improvement is not very significant. For the poset $\{0,1\}^n$, this improvement is a few percent points for $n > 8$. This improvement decreases with the number of variables, reaching its maximum of 20% for $n = 1$. The average query complexity on the poset $\{0,1\}^n$ is exponential in $n$. This fact makes this improvement far more dramatic than for the chain poset.

Fourth, it compares the efficiency of the three major types of oracles. The three-valued oracle provides the most significant restriction on the oracles. It causes up to 84% and 55% increase in the average number of queries for the chain poset and the poset $\{0,1\}^n$, respectively. It is interesting to observe that the sequential oracles are just as efficient as the unrestricted oracles on the chain poset and for the poset $\{0,1\}^n$ for $n$ up to 4. This implies that the pair of nested monotone Boolean functions defined on these posets can be inferred sequentially without losing optimality. For the poset $\{0,1\}^n$ with $n > 7$, the sequential oracle causes a significant increase in the average query complexity of 12-33%.

The maximum likelihood ratio approach to modeling the inference process of Problem # 3 yielded a number of benefits. It was demonstrated that an appropriately defined guided learner, such as maximizing the expected maximum likelihood ratio ($\max \Delta\hat{\lambda}(v)$) 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 $\max(E_0(v) + E_1(v))$ which is based on the number of errors.

For complete reconstruction of monotone Boolean functions, the guided approach showed a dramatic reduction in the average number of queries over a passive learner. 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.

For large problems (i.e., $n > 5$), it may not be possible to compute the evaluative criterion $\max \Delta\hat{\lambda}(v)$ 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.

When the hierarchical decomposition described in Section 2.5 is appropriate, it provides a way to address a large inference problem as a set of smaller independent inference problems. Even though it was not mentioned earlier, this decomposition is applicable to all three Problems #1, #2, and #3 where it can dramatically reduce the query complexity. Perhaps the greatest benefit of this decomposition is its simplified queries. This fact may not only improve the efficiency but also reduce the number of human errors, and hence increase the likelihood of inferring the correct function.

7.2 Significance of the Research Findings

The single most important discovery in this dissertation is the near optimal evaluative criteria which take polynomial time to evaluate. This leads to the efficient inference of monotone Boolean functions. The significance of these criteria is further strengthened by the scope of real-life problems that can be modeled by using monotone Boolean functions. Even though only one or (two nested) monotone Boolean function(s) defined on the set of Boolean vectors $\{0,1\}^n$ were studied here, the
evaluative criterion approach to guiding the learner is appropriate for any monotone mapping \( V \to F \), where the sets \( V \subset \mathbb{R}^n \) and \( F \subset \mathbb{R}^r \) are both finite. The query domain can be viewed as a finite poset by using the monotonicity constraints: \( f_i(v) \leq f_i(w) \) iff \( v \preceq w \), for \( i = 1, 2, \ldots, r \), and whatever the relationships between the functions are, such as the nestedness constraints: \( f_i(v) \geq f_i(v) \forall v \in V \). The selection criteria can be evaluated for any such poset in order to pinpoint “smart” queries.

Once the border vectors have been established for each monotone function, they can be used to classify new observations. In addition, they can be represented by a (set of) monotone Boolean function(s) defined on set of Boolean variables. Representing the inferred knowledge in this intuitive manner is perhaps the most important aspect of this problem when human interaction is involved since people tend to make better use of knowledge they can easily interpret, understand, validate, and remember.

The use of Boolean functions for analyzing fixed datasets has recently gained a momentum due their simple representation of intuitive knowledge. See Triantaphyllou and Soyster (1996b), Boros et al. (1995), Torvik et al. (1999), and Yilmaz et al. (2001) for example. Boolean models are also becoming more popular because methods for solving their related hard logical optimization problems are emerging (e.g., Triantaphyllou (1994), Chandra and Hooker (1999), and Hooker (2000)). Some initial studies on guided inference of Boolean functions from fixed datasets are provided in Triantaphyllou and Soyster (1996a) and Nieto et al. (2001).

The narrow vicinity hypothesis proposed by Kovalerchuk et al. (2000a) suggests that the use of the monotonicity assumption is often necessary and sufficient. As such, it can greatly improve upon knowledge representations that are too simple or too complex. This dissertation demonstrated that the problem of guided inference in the presence of monotonicity can be of great benefit in a wide variety of important real-life applications.

It has been mentioned several times throughout the dissertation that the monotonicity assumption is appropriate in many applications. One of the goals of this dissertation research is to “spread the word” about the benefits of guided inference. Parts of this dissertation have been summarized in the form of six articles (most of which are in refereed journals), and they have been presented at several national and international conferences. For the details please see Part I of the reference list.

7.3 Future Research Directions

As mentioned in Section 7.2 the evaluative criterion approach to learning monotone Boolean functions defined on \( \{0,1\}^n \) is applicable in the much more general monotone setting: \( V \to F \), where the sets \( V \subset \mathbb{R}^n \) and \( F \subset \mathbb{R}^r \) are both finite. The monotone mapping \( V \to F \), where the set \( V \subset \mathbb{R}^n \) is infinite and the set \( F \subset \mathbb{R}^r \) is finite, forms another intriguing problem. It is well known that binary search is optimal when the query domain \( V \) is a bounded subset of the real line, and \( F = \{0,1\} \). However, when the set \( V \) is multidimensional and infinite (e.g., \( V = [a,b]^2 \)), pinpointing the optimal queries is a much more complex problem. The evaluative criterion \( \min |K_1 - K_0| \) can be modified to accommodate this case also. Let the \( U \) denote the unclassified set (i.e., \( v \notin V \)) and let the parameters \( K_0(v) \) and \( K_1(v) \) now denote the size of the subsets \( \{w \in U : w < v\} \) and \( \{w \in U : v < w\} \), respectively. For example, \( K_1(v) \) is measured in terms of distance, area, volume, etc. when \( n = 1, 2, 3, \) etc., respectively. The evaluative criterion \( \min |K_1 - K_0| \) is then optimal for \( n = 1 \). How well this criterion performs when \( n > 1 \), is an open question.

For the problems considered in this dissertation, the evaluative criteria attempt to minimize the average query costs. This objective is based on the certain assumptions of the query costs (fixed cost of querying an oracle in Problems # 1, # 2, and # 3, and highly disproportionate or equal query
costs for the two oracles in Problems # 2.1 and # 2.3, respectively). It would be interesting to see how the dialogue with the oracle(s) changes as these assumptions are modified. When dealing with two oracles, it may be that the cost of querying the first oracle may be less than, yet of similar magnitude as, the cost of querying the second oracle. In this case, the first few queries should be directed at the first oracle. After a few queries it may be cost beneficial to begin alternating between the two oracles. It could also be that the order of the queries has an effect on the total inference cost, as for the keyword search problem described in Section 5.1.1. In some applications, additional properties may be known about the underlying function. Some applications 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 may be that the underlying function belongs to a subclass of monotone Boolean functions, such as threshold functions, 2-monotonic functions, etc.

The methodologies presented in this dissertation provide a framework for solving diverse and potentially very important real-life problems that can be modeled as guided inference problems in the presence of monotonicity. The benefits of these methodologies were shown to be dramatic for the specific problems studied here. However, these research findings are just the tip of the iceberg.
REFERENCES PART I: BY THE AUTHOR
(some of the papers are available at the website http://arrowsmith2.psych.uic.edu/torvik)


Torvik, V.I. and E. Triantaphyllou. 1999. Inference of Monotone Boolean Functions and Their Applications to Data Mining. 5th International Conference of The DSI in Athens, Greece, July 5-8.


REFERENCES PART II: BY OTHER AUTHORS


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


Torvik, V.I., please see Part I of the references on page 97.


VITA

Vetle Ingvald Torvik was born in Porsgrunn, Norway, on April 7, 1973. His passion for interdisciplinary science started early with majors in mathematics, physics, chemistry, and computer science at Porsgrunn High School in Norway. Following graduation in June of 1992, he enrolled at St. Olaf College in Northfield, Minnesota, with a scholarship to study mathematics and physics. After receiving his Bachelor of Arts in Mathematics in May of 1995, he joined the Department of Statistics at Oregon State University to study operations research. It was the research on a genetic algorithm for model selection in linear regression performed under the direction of Professor Jeffrey L. Arthur that sparked Vetle’s interest in applying mathematical programming and statistical modeling to artificial intelligence problems. This research led to a Master of Science in Operations Research in June of 1997.

As a doctoral student, Vetle joined the Operations Research and Artificial Intelligence research group at Louisiana State University headed by Professor Evangelos Triantaphyllou in the Department of Industrial and Manufacturing Systems Engineering in August of 1997. While working as a research assistant funded by the Office of Naval Research Grant N00014-97-1-0632, Vetle has presented his research findings at several national and international conferences, has published several articles, has been invited to participate in the doctoral student colloquia of the Decision Sciences Institute (DSI) and the Institute for Operations Research and Management Science (INFORMS), and received the Louisiana Engineering Foundation's 2000-2001 Vincent A. Forte Graduate Student Fellowship. He maintains active student memberships in the INFORMS, the Society for Industrial and Applied Math (SIAM), and the DSI. On October 3, 1998, he married the actress Elizabeth Christine Tanner.

Vetle is currently a research assistant professor in the Department of Psychiatry at the University of Illinois at Chicago where he is working on the knowledge discovery project called Arrowsmith (http://arrowsmith.psych.uic.edu) jointly funded by the National Institute of Mental Health and the National Library of Medicine. For more details please consult his personal web page at http://arrowsmith2.psych.uic.edu/torvik.