

Approximation
of some
AI Problems

by

Karsten A. Verbeurgt

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 1998

©Karsten A. Verbeurgt 1998



National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file *Votre référence*

Our file *Notre référence*

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-32866-X

The University of Waterloo requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

The work of this thesis is motivated by the apparent computational difficulty of practical problems from artificial intelligence. Herein, we study two particular AI problems: the constraint satisfaction problem of coherence, and the machine learning problem of learning a sub-class of monotone DNF formulas from examples. For both of these problems, we apply approximation techniques to obtain near-optimal solutions in polynomial time: thus trading off quality of the solution for computational tractability.

The constraint satisfaction problem we study is the coherence problem, which is a restricted version of binary constraint satisfaction. For this problem, we apply semidefinite programming techniques to derive a 0.878-approximation algorithm. We also show extensions of this result to the problem of settling a neural network to a stable state.

The approximation model we use for the machine learning problem is the Probably Approximately Correct (PAC) model, due to Valiant [Val 84]. This is a theoretical model for concept learning from examples, where the examples are drawn at random from a fixed probability distribution. Within this model, we consider the learnability of sub-classes of monotone DNF formulas on the uniform distribution. We introduce the classes of one-read-once monotone DNF formulas, and factorable read-once monotone DNF formulas, both of which are generalizations of the well-studied read-once DNF formulas, and give learnability results for these classes.

Acknowledgements

I would like to thank Ming Li for his support and encouragement of this work and Paul Thagard for his guidance and introducing me to the coherence problem. Most of all, I would like to thank my family for their love and support during this endeavour.

Contents

1	Part I - Approximating Binary CSPs	1
1.1	Introduction and Motivation	1
1.2	A Brief Overview of CSP Research	3
1.3	Approximation	5
1.4	The Coherence Problem	5
1.5	Applications of Coherence	7
1.6	The Constraint Satisfaction Problem	8
1.6.1	Exact Instances of Coherence as Constraint Satisfaction	10
1.7	The Partial Constraint Satisfaction Problem	10
1.7.1	Coherence as Partial Constraint Satisfaction	11
2	The Complexity of Coherence	14
2.1	A Proof of NP-completeness for Coherence	14
2.2	Solving Exact Instances of Coherence Efficiently	16

3	On-line and Connectionist 0.5-Approximations	18
3.1	An On-line 0.5-Approximation Algorithm	18
3.2	Optimality of the On-line 0.5-approximation	21
3.3	A Connectionist Algorithm for Coherence	22
3.3.1	A Neural Network for Coherence	23
3.3.2	Harmony and Stable States	24
3.3.3	Variations on the Update Rule with Provable Approximation Bounds	28
4	A 0.878-Approximation Algorithm	33
4.1	An Objective Function for Coherence	34
4.2	Positive Semidefinite Matrices	35
4.3	Semidefinite Programming	35
4.4	Semidefinite Relaxations	35
4.4.1	Randomized Partitioning	37
4.4.2	Expected Performance	37
5	An Approximation Algorithm for Neural Nets	41
5.1	A 0.878-Approximation Algorithm	41
5.1.1	Encoding Neural Networks as Coherence Problems	42
5.1.2	Harmony and Scaled Harmony	42
6	Future Work on Constraint Satisfaction	45

7	Part II - Learning Monotone DNF	48
7.1	Introduction	48
7.2	Previous Work	49
7.3	Definitions and Terminology	53
7.3.1	Functions and Classes	53
7.3.2	Learnability	55
7.3.3	Fourier Transform	56
7.4	Motivation for Spectral Analysis of MDNF	57
8	Approximation Results for Monotone DNF	59
9	Learning Sub-Classes of MDNF Formulas	69
9.1	Learning Read-Once MDNF	69
9.2	Learning Poly-disjoint One-read-once MDNF	76
9.3	Learning Read-once Factorable MDNF	80
10	Towards Learnability of Monotone DNF	82
10.1	Second order coefficients of MDNF	83
10.2	Third order coefficients of MDNF	84
10.3	Degenerate coefficients on Dense Terms	85
10.4	An Heuristic for Monotone Terms	86
10.5	Filtering Distributions	88
10.6	Adaptation of Freund's Boosting Algorithm	91

11 Empirical Results	96
11.1 Overview of Results	96
11.1.1 Dense Formulas	97
11.1.2 Sparse Formulas	101
11.1.3 Sparse and Dense Formulas	103
11.1.4 Majority Functions	105
11.2 Empirical Results on Real-World Databases	107
11.2.1 Tic-Tac-Toe Database	107
11.2.2 Mushroom Database	108
11.3 Summary of Empirical Results	111
12 Conclusions and Open Problems	113
Bibliography	115
A	125

List of Tables

11.1 Output Hypothesis Size and Accuracy on 50% of Data	109
11.2 Output Hypothesis Size and Accuracy on 20% of Data	110
11.3 Output Hypothesis Size and Accuracy on 10% of Data	110
11.4 Output Hypothesis Size and Accuracy on 1% of Data	111
A.1 Attributes for the Mushroom Database	133

List of Figures

1.1	Example of a Coherence Problem	7
1.2	Example of a Constraint Satisfaction Problem	9
1.3	Relationship between Coherence and Constraint Satisfaction Problems	12
3.1	An Approximation Algorithm for the Coherence Problem	19
3.2	Input with performance ratio of $\frac{1}{2}$	22
3.3	A Connectionist Algorithm for the Coherence Problem with Provable Approximation Bounds	29
9.1	Algorithm for Learning a Read-once MDNF Formula	72
9.2	Algorithm for Learning a Poly-disjoint One-read-once MDNF Formula	78
10.1	Heuristic Algorithm for Learning MDNF	93
10.2	Algorithm for Learning a Sparse Term	94
10.3	Algorithm for Learning a Dense Term	94

11.1 Running Time of Algorithm MDNF for a Dense Formula	98
11.2 Hypothesis Size for a Dense Formula	98
11.3 Running Time of Algorithm MDNF for a Random Dense Formula .	100
11.4 Hypothesis Size for a Random Dense Formula	100
11.5 Running Time of Algorithm MDNF for a Sparse Formula	102
11.6 Hypothesis Size for a Sparse Formula	102
11.7 Running Time of Algorithm MDNF for Majority	106

Chapter 1

Part I - Approximating Binary CSPs

1.1 Introduction and Motivation

Many important problems related to Artificial Intelligence have been shown to be NP-hard. The constraint satisfaction problem (CSP) is such a problem [Ku 92]. Tasks in areas such as cognitive science, computer vision, scheduling, and planning can be encoded as instances of the CSP. The pervasiveness of the CSP in the AI literature is testimony to its importance to the field. As such, much research has been devoted to determining sub-problems of the CSP that are tractable. Rather than asking for which instances exact solutions of the CSP can be computed efficiently, an alternate approach is to ask if there are good approximation algorithms that produce high quality solutions for all instances of the problem. The problem of approximation algorithms for variants of the CSP is the main focus of the first part of this thesis.

Several versions of the CSP have been studied. In the most general version, constraints may be multi-ary relations between variables, which have multi-valued domains. In the most basic version, the constraints are binary (i.e., constraints between two variables) and the domains of the variables are also binary. The coherence problem that we study in this thesis is a restricted version of the binary constraint satisfaction problem, with binary constraints and binary variables. We show in Chapter 2 that the coherence problem is NP-hard, and that its associated decision problem is NP-complete. Further, we show that this problem is complete for the class MAX SNP, which implies that there is some constant factor within which no approximation algorithm can guarantee a solution. We give two different approximation algorithms for maximizing the number of accepted constraints - an on-line algorithm that achieves a performance ratio of 0.5, and an algorithm that uses semidefinite programming techniques to achieve a performance ratio of 0.878.

Another problem in AI that is known to be NP-hard is to settle an arbitrary neural network to its state of optimal harmony (or energy). We show that this problem is related to the coherence problem discussed above, and give a 0.878-approximation algorithm for settling a neural network to a stable state. The solution produced by the algorithm comes within a factor of 0.878 of a "scaled" measure of the harmony of the network. This is the first result that gives a non-trivial performance guarantee on the quality of the stable state achieved in settling a neural network.

In this chapter, we introduce and define the constraint satisfaction problems (CSPs) studied in this thesis. In particular, we formulate the coherence problem as a special case of the CSP.

1.2 A Brief Overview of CSP Research

Algorithms for solving the CSP have been an important area of research for decades. Most CSP algorithms in the literature use some form of backtracking algorithm, which in the worst case takes exponential time. For an excellent survey of CSP algorithms, see [Ku 92]. To cull the search space of the backtracking algorithm, several authors have proposed techniques based on constraint propagation [Mac 77, DP 88, D 90], which eliminate impossible combinations of values from the search space. Other researchers have considered the variable ordering for value instantiation, and show that for almost all types of CSPs, the order in which variables are instantiated can greatly improve the efficiency of the backtracking algorithm. For tree-structured CSP's, the need for backtracking can be completely eliminated. For all but the tree-structured CSP's, backtracking is still used, resulting in an exponential time algorithm.

The CSP is defined such that solutions to the problem must satisfy all of the constraints. To address problems that are over-constrained, and there is no satisfying solution, or problems where it is difficult to entirely solve the CSP, Freuder and Wallace [FW 92] generalized the CSP to the partial constraint satisfaction problem (PCSP). The partial constraint satisfaction problem is to find an assignment to the variables of the CSP that maximize the number of constraints satisfied. Instead of using the classical backtracking algorithms for CSPs, Freuder and Wallace use branch-and-bound techniques to guide the search for a solution. As for the backtracking algorithms for CSPs, these algorithms are inherently exponential time.

The PCSP problem opens the door for the use of approximation in the search for a solution, since the solution is not required to satisfy all of the constraints, but rather to maximize the number of satisfied constraints. The maximization problem

is known to be NP-hard [FW 92]: however, an approximately maximal solution may be obtainable in polynomial time. It is this problem that we study in this thesis, and propose an algorithm that is guaranteed to find a solution that is within 87% of the weight of the optimal solution for the coherence problem.

The PCSP has been of interest in recent years in the theoretical computer science community in relation to approximation research. With the motivation of introducing a class of problems related by approximation-preserving reductions, Papadimitriou and Yannakakis [PY 91] defined the class MAX SNP. The results of Arora et. al. [ALMS 92] showed that for any problem that is MAX SNP-complete, there exists some constant factor c such that if any approximation algorithm exists that is guaranteed to produce a solution within a factor c of the optimal solution, then $P = NP$. Thus, for MAX SNP-hard problems, we assume that no approximation algorithm can guarantee a solution arbitrarily close (closer than some constant factor) to the optimal solution.

In a follow-up paper, Khanna, Motwani, Sudan and Vazirani [KMSV 94] show that the k -CSP problem (CSP with k -ary predicates) is in MAX SNP. Moreover, they show that k -CSP is a universal problem for MAX SNP: that is, any problem in MAX SNP can be encoded as an instance of the k -CSP. Assuming $P \neq NP$, this immediately implies there is some constant c such that no PCSP algorithm exists that guarantees to find a solution better than a factor c of optimal. In addition, they give an approximation algorithm for the k -CSP that guarantees a solution within $\frac{1}{2^k}$ of the optimal. For $k = 2$ (the binary CSP), this gives a $\frac{1}{4}$ -approximation algorithm. In this thesis, we first give a $\frac{1}{2}$ -approximation algorithm for coherence, and then give a 0.878-approximation algorithm. The 0.878-approximation algorithm is based on the approximation algorithm of Goemans and Williamson [GW 94] for the max cut problem, with the same performance guarantee.

1.3 Approximation

Given an instance \mathcal{I} of a maximization problem¹, let $V(\mathcal{I}, S)$ be the value of a solution S and let $OPT(\mathcal{I})$ denote the value of the optimal solution. An algorithm A is said to have performance ratio $\mathcal{R}(n)$ with respect to instance \mathcal{I} , where $n = |\mathcal{I}|$ is the size of the problem description, if

$$\frac{V(\mathcal{I}, A(\mathcal{I}))}{OPT(\mathcal{I})} \geq \mathcal{R}(n) \quad (1.1)$$

If $\mathcal{R}(n)$ is bounded by a constant c for all instances \mathcal{I} , A is said to be a constant factor approximation algorithm. We also refer to A as a c -approximation algorithm. If $\mathcal{R}(n) = 1 - \epsilon$ for arbitrarily small ϵ , A is called an approximation scheme². If the approximation scheme runs in time polynomial in n , we refer to it as a polynomial-time approximation scheme (PTAS).

1.4 The Coherence Problem

Given a set of constraints that establish coherence relations between pairs of elements, the coherence problem is to partition the elements into two sets such that the overall coherence is maximized. We state the coherence problem as a graph problem, in which the elements are represented as vertices, and the constraints are

¹In recent works on approximation algorithms, the notion of performance ratio has been somewhat standardized so that maximization problems and minimization problems both have ratios normalized to be greater than one. Thus, in some works, the performance ratio is defined as the reciprocal of our definition. For example, see Khanna, Motwani, Sudan and Vazirani in [KMSV 94]. Here, we use the more traditional notation, since we consider only maximization problems.

²We refer to an approximation scheme rather than an approximation algorithm in this case, because an approximation algorithm is defined for each ϵ .

represented as edges. There are two edge sets for the graph, corresponding to the positive and negative constraints, respectively.

Coherence

Instance: Graph $G = (V, E)$, with vertex set V , a set E of weighted edges, and disjoint edge sets C^+ of positive constraints and C^- of negative constraints such that $C^+ \cup C^- = E$.

We denote a partition of the vertices V into two sets A (accepted) and R (rejected) as the ordered pair (A, R) . The weight of a partition, denoted $w(A, R)$, is the sum of the weights of the constraints that are satisfied, where a constraint is satisfied if either of the following holds:

1. if (e_i, e_j) is in C^+ , then e_i is in A iff e_j is in A .
2. if (e_i, e_j) is in C^- , then e_i is in A iff e_j is in R .

Problem: Partition V into sets (A, R) with maximum weight $w(A, R)$.

For example, consider the graph in Figure 1.1, in which negative constraints are indicated by bold lines, and positive constraints by thin lines. For this instance $C^- = \{(1, 2)\}$, and $C^+ = \{(1, 3), (2, 3)\}$. Thus, there is one negative constraint with weight 1 between vertices 1 and 2, and two positive constraints with weight 1 between vertices 1 and 3, and between vertices 2 and 3. Putting vertex 1 in set A , and vertices 2 and 3 into set R , we get $w(A, R) = 2$, since the constraint between vertices 1 and 2 is satisfied, and the constraint between vertices 2 and 3 is satisfied.

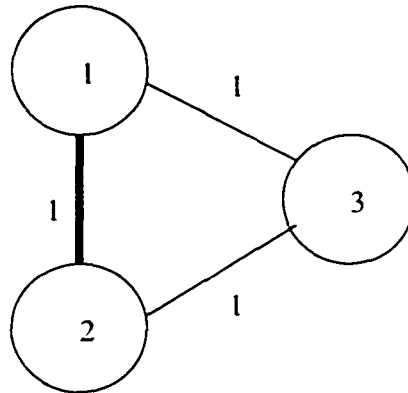


Figure 1.1: Example of a Coherence Problem

1.5 Applications of Coherence

Many problems in philosophy and cognitive science are naturally understood in terms of coherence. Philosophers have proposed coherence theories of truth, knowledge, and ethical justification. Thagard [Th 89, Th 92] showed how explanatory coherence could be computed using connectionist algorithms implemented in a program called ECHO that models choice between competing scientific theories. Thagard and Millgram [Th 94] showed how decision making can likewise be modeled using a program called DECO that computes deliberative coherence. Similar techniques have been used to model analogical mapping and retrieval: we can think of the problem of analogical mapping as involving finding a coherent correspondence between two analogs, and the more general problem of analog retrieval as finding which analog stored in memory is most coherent with a target analog (see Holyoak and Thagard, [HT 89, HT 95]; Thagard, Holyoak, Nelson, and Gochfeld, [THING 90]). In previous work, however, the notion of coherence has remained rather vague. In Section 1.4 we provided a precise definition of a coherence problem. In Chapter 2, we prove results concerning its computational complexity and

its relation to constraint satisfaction problems important in other areas of AI.

The definition of coherence is sufficiently general to cover many kinds of coherence that have been studied computationally. Thagard's theory of explanatory coherence proposes various constraints that impose coherence relations on hypotheses and evidence (see [Th 89, Th 92]). For example, if a hypothesis explains a piece of evidence or another hypothesis, then the two propositions cohere with each other: this is a positive constraint. On the other hand, if two hypotheses are inconsistent with each other or if they compete to explain the same evidence, then they are incoherent with each other: this is a negative constraint. Theory choice is a matter of accepting some hypotheses and rejecting others in a way that maximizes overall coherence. Maximizing explanatory coherence, which is efficiently computed using a connectionist algorithm, provides a flexible and powerful way of performing abductive inference (for other ways, see Josephson and Josephson, [JJ 94], and O'Rourke and Josephson, [OJ 94]). Decision making construed in terms of deliberative coherence is also a coherence problem as characterized above, with facilitation relations between actions and goals providing positive constraints, and incompatibility relations between pairs of actions or goals providing negative constraints (see Thagard and Millgram, [TM 95]). Analogical thinking is also characterizable in terms of constraint satisfaction (see Holyoak and Thagard, [HT 95]).

1.6 The Constraint Satisfaction Problem

Given a set of variables, and a set of constraints on the values these variables may take on, the constraint satisfaction problem (CSP) is to determine a set of values that satisfy all the constraints simultaneously. We use the formulation given by Dechter [D 90], and restate it in terms of graph theory (here, we will be concerned

only with binary constraints (i.e., constraints between two variables.)

[D 90] Constraint Satisfaction

Instance: Given a graph $G = (V, E)$, where the n nodes of G correspond to a set $\{X_1, \dots, X_n\}$ of variables with respective value domains $\{R_1, \dots, R_n\}$, where each R_i is a finite subset of integers. Each edge represents a constraint $C_i(X_j, X_k)$, which corresponds to the subset of the Cartesian product $R_j \times R_k$ that are compatible values for variables X_j and X_k .

Problem: Find an assignment of values to variables that satisfy all of the constraints.

Instances of the constraint satisfaction problem can also be represented as constraint graphs, with nodes representing the variables X_i , and edges between nodes X_i and X_j representing the set of allowable values for variables X_i and X_j . As an example of a constraint satisfaction problem, consider the constraint graph in Figure 1.2. Let the domain R_i of each variable X_i in the examples of Figure 1.2

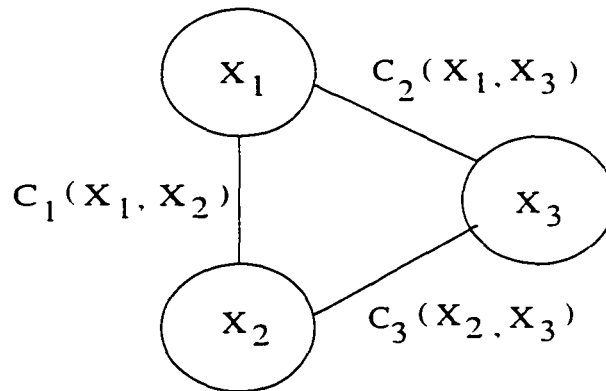


Figure 1.2: Example of a Constraint Satisfaction Problem

be defined as: $R_1 = R_2 = R_3 = \{1, 2, 3\}$, and let the constraints C_j be defined as:

$C_1(X_1, X_2) = C_2(X_1, X_3) = C_3(X_2, X_3) = \{(1, 2), (1, 3), (2, 1), (2, 3), (3, 1), (3, 2)\}$.

In this example, the constraints are defined so that no two variables can take on the same value.

The constraint satisfaction problem is known to be NP-hard [MF 93]. This is easy to see, since the graph k -coloring problem can be encoded as an instance of the constraint satisfaction problem [Ku 92]. For example, in Figure 1.2, we gave an encoding of the graph 3-coloring problem for a simple 3-node graph.

We explore the relationship between the constraint satisfaction problem and the coherence problem further in the following sections.

1.6.1 Exact Instances of Coherence as Constraint Satisfaction

We say that an instance of the coherence problem is exact if there is a solution that complies with all of the constraints in C^+ and C^- . It is then easy to see that exact instances of the coherence problem can be formulated as CSPs. If there is no exact solution to the coherence problem, then we require the notion of partial constraint satisfaction.

1.7 The Partial Constraint Satisfaction Problem

The first difference to note between the coherence problem and the constraint satisfaction problem is that the constraint satisfaction problem requires that a solution be consistent with all of the constraints, whereas the coherence problem maximizes

consistency with the constraints. Otherwise stated, the constraint satisfaction problem has hard constraints, whereas the coherence problem deals with soft constraints, which may be violated by the solution. Recent work by Freuder et. al. [FW 92] generalizes the constraint satisfaction problem to the partial constraint satisfaction problem (PCSP). Their formulation allows us to state the coherence problem as a partial constraint satisfaction problem.

[FW 92] Partial Constraint Satisfaction

Instance: As for the constraint satisfaction problem of Section 1.6.

Problem: Find an assignment of values to variables that maximizes the weight of satisfied constraints.

The partial constraint satisfaction problem is NP-hard, since the constraint satisfaction problem is a special case.

1.7.1 Coherence as Partial Constraint Satisfaction

The coherence problem can be encoded as an instance of the partial constraint satisfaction problem in which the value domain R_i of each variable X_i is the set $\{0, 1\}$. For each constraint $e = (v_i, v_j) \in C^+$ of the coherence problem, we have the constraint $C(X_i, X_j) = \{(0, 0), (1, 1)\}$ in the partial constraint satisfaction problem, and for each constraint $e = (v_i, v_j) \in C^-$, we have the constraint $C(X_i, X_j) = \{(0, 1), (1, 0)\}$. Let A be the set of vertices that are assigned the value 1 in the solution of the PCSP, and let R be the set of vertices that are assigned the value 0. The resulting partition then maximizes compliance with the constraints of the coherence problem.

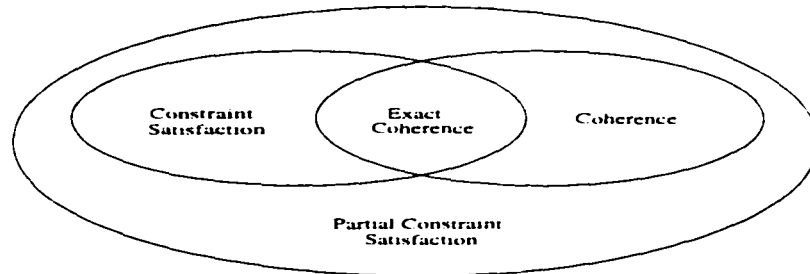


Figure 1.3: Relationship between Coherence and Constraint Satisfaction Problems

In Figure 1.3, we show graphically the relationship between coherence, exact coherence, constraint satisfaction, and partial constraint satisfaction. In this section, we showed that coherence problems can be encoded as partial constraint satisfaction problems. In Section 1.6.1, we showed that exact instances of the coherence problem can be encoded as instances of the constraint satisfaction problem. In the next chapter, we will show that exact instances of coherence can be solved efficiently.

In Chapter 3, we give an on-line 0.5-approximation algorithm for the coherence problem (an on-line algorithm is one in which the vertices of the constraint graph are presented one at a time to the algorithm.) We show that 0.5 is the optimal approximation ratio for an on-line approximation algorithm for coherence. In Chapter 4, we give a 0.878-approximation algorithm, based on the 0.878-approximation algorithm of Goemans and Williamson for MAX CUT. In Chapter 5, we apply the coherence approximation algorithm to the problem of settling a neural network to

a stable state.

Chapter 2

The Complexity of Coherence

2.1 A Proof of NP-completeness for Coherence

In Section 1.4, we stated the coherence problem as an optimization problem. In the following proof, we show that the coherence problem is NP-hard. It remains NP-hard even when there are only negative constraints, and no positive constraints at all. The associated decision problem to the coherence problem is clearly in the class NP, since a partition (A, R) is a certificate that is verifiable in polynomial time, hence the following theorem implies that the decision problem is NP-complete. We will show subsequently that the coherence problem is also MAX SNP-complete.

Theorem 2.1 *The coherence problem is NP-hard.*

Proof: The proof is a transformation of the max cut problem to the coherence problem. For completeness, we state the max cut problem [Ka 75, GJ 79] (denoted MAX CUT) here: given a graph $G = (V, E)$, weight $w(e) \in Z^+$ for each $e \in E$, and positive integer K , is there a partition of V into disjoint sets

V_1 and V_2 such that the sum of the weights of the edges from E that have one endpoint in V_1 and one endpoint in V_2 is at least K ? The max cut problem remains NP-hard even if all weights $w(e)$ are constrained to be 1 [GJ 79]. Here we show that the same holds for coherence.

For a graph G where all edges have weight 1, as an instance of the max cut problem, construct an instance of the coherence problem by putting all of the edges of G in C^- . We show that any solution to the coherence problem is then a solution to the max cut problem.

The solution to the coherence problem will partition the vertices of G into two subsets A and R such that compliance with the constraints is maximized (i.e., if (e_i, e_j) is in C^+ , then e_i is in A iff e_j is in A , and if (e_i, e_j) is in C^- , then e_i is in A iff e_j is in R .) Since the only constraints are negative ones, $C^+ = \emptyset$, and $C^- = E$, the edge set of G . The partition of vertices into sets A and R must be such that compliance with the constraints of C^- (i.e., if (e_i, e_j) is in C^- , then e_i is in A iff e_j is in R) is maximized. This partition therefore maximizes the number of edges with one endpoint in A and one endpoint in R . Let $V_1 = A$ and $V_2 = R$. Since C^- contains all edges of G , and since each edge weight is 1, the partition into sets V_1 and V_2 must maximize the sum of the weights of edges such that one endpoint is in V_1 and one endpoint is in V_2 . Hence, we have a solution to the max cut problem. ■

Theorem 2.2 *The coherence problem is complete for MAX SNP.*

Proof: The max cut problem was shown to be MAX SNP-complete by Papadimitriou and Yannakakis, in [PY 91]. Thus, by the same reduction as in the proof of the previous theorem, the coherence problem is MAX SNP-hard. To complete the proof, we just need to show that the coherence problem is in

the class MAX SNP, and to show this, we just need to show that a constant factor approximation algorithm exists for coherence. We give such algorithms in Chapters 3 and 4. ■

In this section we have shown that instances of the coherence problem where there are negative constraints and no positive constraints are as hard as solving the max cut problem, which is both NP-hard and MAX SNP-hard. We note here that if there are only positive constraints, and no negative constraints at all, then the problem is trivially solvable by placing all vertices in the accepted set A .

2.2 Solving Exact Instances of Coherence Efficiently

Theorem 2.3 *Exact instances of the coherence problem are solvable in polynomial time.*

Proof: Construct a graph $G'' = (V'', E'')$ with vertex set $V'' = V$, and edge set $E'' = C^+$. Find each connected component of G'' , and let m be the number of connected components in G'' . Construct a graph $G' = (V', E')$, where $|V'| = m$, and each vertex of G' corresponds to a connected component of G'' . Now, we 2-color G' , which can be done in polynomial time [GJ 79]. Color G'' by assigning the color of the corresponding vertex in G' to each vertex of the connected component in G'' . Now, assign all of the vertices of one color to A , and all of the vertices with the other color to R to obtain a solution to the coherence problem. ■

The above proof shows that the instances of the coherence problem that are also instances of the constraint satisfaction problem can be solved in polynomial time. This is interesting, in that the constraint satisfaction problem is NP-hard, yet exact instances of the coherence problem, which are a subset of the CSP, are efficiently solvable.

We have shown in Section 2.1 that the coherence problem is NP-hard as well as MAX SNP-hard, even when there are only negative constraints. In contrast, we also note in Section 2.2 that the problem is trivially solvable if there are only positive constraints. Thus, it would seem that it is the negative constraints that make the problem hard when both types of constraints are allowed. In Section 1.6, we showed the relationship between constraint satisfaction and coherence: namely, that exact instances of coherence can be encoded as instances of the constraint satisfaction problem. In Section 2.2, we showed that these instances are solvable in polynomial time. In Section 1.7, we showed that the coherence problem can be encoded as an instance of the partial constraint satisfaction problem.

Chapter 3

On-line and Connectionist

0.5-Approximations

3.1 An On-line 0.5-Approximation Algorithm

In this section, we give an on-line approximation algorithm for the coherence problem that produces a solution within a factor $\frac{1}{2}$ of the weight of the optimal solution. For an on-line algorithm, the vertices of the graph are presented one at a time, along with the edges joining the vertex presented to all the previous vertices [LST 89]. The reason that we distinguish on-line algorithms from algorithms where all vertices may be considered in parallel is that on-line algorithms often have performance bounds worse than their unrestricted counterparts. For example, Lovasz, Saks, and Trotter give an algorithm for the graph coloring problem that has an $O(\frac{n}{\log n})$ performance ratio, whereas the algorithm of Blum [B 89] achieves a bound of $O(n^{1-1/(k-4/3)} \log^{8/5} n)$ (i.e., $\tilde{O}(n^{0.4})$ for 3-colorable graphs).

We show in Section 3.2 that $\frac{1}{2}$ is the best approximation ratio achievable by

an on-line algorithm for coherence. This implies that the approximation algorithm that we give here is optimal. Algorithms that are not restricted to being on-line can achieve better bounds. In Chapter 4, we give an algorithm that achieves a 0.878 performance ratio. In Section 3.3, we will consider a connectionist algorithm used by Thagard [Th 89, THNG 90, Th 94] that performs well in practice, although we are unable to prove performance guarantees.

Algorithm Approx-Coherence(G)

1. Let v_1, v_2, \dots, v_n be an arbitrary ordering of the vertices of G .
2. Let $A = R = \emptyset$
3. for $i = 1$ to n
4. if $w(A \cup v_i, R) \geq w(A, R \cup v_i)$
5. $A = A \cup v_i$
6. else
7. $R = R \cup v_i$
8. end if
9. end for
10. output (A, R)

Figure 3.1: An Approximation Algorithm for the Coherence Problem

The analysis of the algorithm of Figure 3.1 follows the technique of Sahni and Gonzalez for their approximation algorithm to the max cut problem [SG 76], and an approximation algorithm given by Kececioğlu and Myers for DNA sequence assembly [KM 92].

Theorem 3.1 *Algorithm Approx-Coherence() is an approximation algorithm for the coherence problem that produces a solution with weight within a factor $\frac{1}{2}$ of the weight of the optimal solution.*

Proof: Let $w(v_i, v_j)$ denote the weight of edge (v_i, v_j) . and let $w(A, R)$ denote the weight of the partition (A, R) . For each vertex v_i , step 4 of the algorithm tests the increase in the weight of the partial solution depending on whether v_i is added to set A or R . Let $\Delta w_i(A)$ denote the change in weight by adding v_i to A . and $\Delta w_i(R)$ denote the change in weight by adding v_i to R . Now.

$$\Delta w_i(A) = \sum_{\substack{\{v_j \in A \mid j < i\} \\ (v_i, v_j) \in C^+}} w(v_i, v_j) + \sum_{\substack{\{v_j \in R \mid j < i\} \\ (v_i, v_j) \in C^-}} w(v_i, v_j)$$

and

$$\Delta w_i(R) = \sum_{\substack{\{v_j \in R \mid j < i\} \\ (v_i, v_j) \in C^+}} w(v_i, v_j) + \sum_{\substack{\{v_j \in A \mid j < i\} \\ (v_i, v_j) \in C^-}} w(v_i, v_j)$$

Hence, $\Delta w_i(A) + \Delta w_i(R) = \sum_{j < i} w(v_i, v_j)$. Let

$$\Delta w_i = \max\{\Delta w_i(A), \Delta w_i(R)\}$$

It follows that $\Delta w_i \geq \frac{1}{2} \sum_{j < i} w(v_i, v_j)$.

Let (A^*, R^*) be the partition with optimal weight. Since the weight of the optimal solution must be bounded above by the weight of the entire graph. we have that $w(A^*, R^*) \leq \sum_i \sum_{j < i} w(v_i, v_j)$.

Hence,

$$w(A, R) = \sum_i \Delta w_i \geq \frac{1}{2} \sum_i \sum_{j < i} w(v_i, v_j) \geq \frac{1}{2} w(A^*, R^*)$$

■

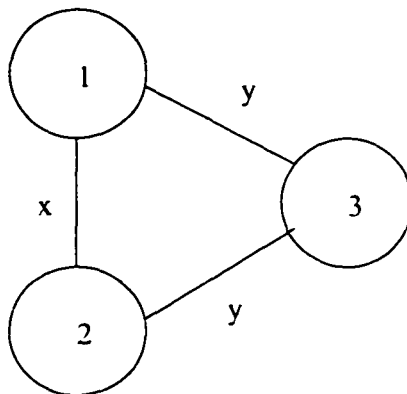
3.2 Optimality of the On-line 0.5-approximation

In the previous section we have given an on-line 0.5-approximation algorithm for coherence. We now show that this ratio is optimal for any on-line approximation algorithm for this problem.

Theorem 3.2 *No on-line approximation algorithm for coherence can achieve a performance ratio better than $\frac{1}{2}$.*

Proof: Consider the graph given of Figure 3.2. The nodes are labelled 1, 2 and 3, with weights denoted by x and y , where x and y are integers to be specified. Suppose without loss of generality that the on-line algorithm receives vertex 1 first, then 2, and finally three. Assume also that the algorithm places vertex 1 in the accepted set, and vertex 2 in the rejected set. Now, an adversary could set x to be 1, and y to be an arbitrarily large integer. Vertex 3 can now be placed in either the accepted or rejected set. In either case, the value of the solution will be $x + y$, whereas the value of the optimal solution is $2y$, giving an approximation ratio arbitrarily close to $\frac{1}{2}$. If the algorithm chooses to place the first two vertices differently (both in A or both in R) an adversary could similarly set the weights of the unseen edges so that the ratio is arbitrarily close to $\frac{1}{2}$.

■

Figure 3.2: Input with performance ratio of $\frac{1}{2}$

3.3 A Connectionist Algorithm for Coherence

The formulation of the coherence problem, as stated in Section 1.4, emerged as an abstraction of the cognitive models: ACME, ARCS, DECO and ECHO. ARCS is a program for Analog Retrieval by Constraint Satisfaction that is designed to retrieve analogs from long term memory for analogical reasoning systems, and ECHO (Explanatory Coherence by Harmany Optimization) is designed to model theory choice (in particular, how conceptual revolutions take place in the scientific community.) All of these systems are based upon constraint networks with positive (excitatory) and negative (inhibitory) links between the elements of the network. These type of networks bear some similarity to neural networks, with neurons representing elements of the constraint network, and excitatory and inhibitory links between the neurons representing the positive and negative constraints in the network. Thus, neural networks are a natural choice for solving coherence problems. Thagard's empirical results [Th 89, THNG 90, Th 94] show that connectionist networks do indeed produce good solutions to a variety of coherence problems.

In this section, we analyze some properties of these networks. In particular,

we show the relationship between the energy states of the neural network and the quality of the solution to the coherence problem. Secondly, we show some relationships between the neural network algorithm and the approximation algorithm we give in Section 3.1.

3.3.1 A Neural Network for Coherence

We represent a neural network as a graph, where the vertices represent nodes of the neural network, and weighted edges represent links (excitatory or inhibitory) between nodes of the network. Given an instance G of a coherence problem, we construct an n -node neural network $N = (A, W)$, with nodes $A = \{a_1, \dots, a_n\}$, and weight set

$$W = \begin{cases} w_{ij} = w(v_i, v_j) & \text{if } (v_i, v_j) \in C^+ \\ w_{ij} = -w(v_i, v_j) & \text{if } (v_i, v_j) \in C^- \\ w_{ij} = 0 & \text{otherwise} \end{cases}$$

Associated with each node a_i is an activation value, which for these networks will be constrained to the range $[min, max]$, where $min = -1$ and $max = 1$. We denote the activation value of node a_i at time t as $a_i(t)$. We will use $a_i(\infty)$ to denote the state of the neural network after it has settled to a stable state. The activation value at time $t + 1$ is a function of the activation value at time t , and the activation values of neighboring nodes at time t . The activation function used here is:

$$a_i(t + 1) = a_i(t) + \begin{cases} net_i(t)(max - a_i(t)) & \text{if } net_i(t) > 0 \\ net_i(t)(a_i(t) - min) & \text{otherwise} \end{cases}$$

where $net_i(t) = \sum_j w_{ij}a_j(t)$. This activation function is similar to that of Hopfield networks [H 82, RM 86], except that Hopfield networks use a threshold activation function rather than a continuous function.

3.3.2 Harmony and Stable States

The harmony of network N at time t is defined as:

$$H(N, t) = \sum_i \sum_j w_{ij} a_i(t) a_j(t)$$

Note that the harmony as defined above “double-counts” in that the weight contribution of each pair of vertices is counted twice in $H(N, t)$. For our purposes, we will use the following definition of harmony:

$$H_2(N, t) = \sum_{i < j} w_{ij} a_i(t) a_j(t) \quad (3.1)$$

It can be easily verified that $H_2(N, t) = \frac{1}{2}H(N, t)$, hence optimizing $H_2(N, t)$ also optimizes $H(N, t)$.

We show two properties of these neural networks here: first, that the points of maximal harmony occur when all of the vertices are in state -1 or 1 , or the net input of the node is zero; and secondly, that when the network is in a stable state, the harmony of the network corresponds to the quality of the solution represented by that state: in other words, maximizing harmony maximizes coherence.

Theorem 3.3 *The states of maximal harmony occur when for all i , either $a_i(\infty) = 1$, $a_i(\infty) = -1$, or $net_i(\infty) = 0$.*

Proof: Suppose that some state is a point of maximal harmony, such that for some node a_i , $net_i(\infty) \neq 0$, $a_i(\infty) \neq 1$, and $a_i(\infty) \neq -1$. The contribution of unit i to the harmony of the network is:

$$H_i(N, \infty) = \sum_j w_{ij} a_i(\infty) a_j(\infty) = net_i(\infty) a_i(\infty)$$

Now, suppose that $net_i(\infty)$ is negative. Then by reducing the value of $a_i(\infty)$, the harmony would be increased. Conversely, if $net_i(\infty)$ is positive, then increasing the value of $a_i(\infty)$ would increase the harmony. Hence, the network cannot be in a state of maximal harmony if $net_i(\infty)$ is non-zero for any a_i , and the activation values are not at their extremes. ■

This theorem shows that the stable states (states of maximal harmony) all have activation values of 1 or -1 for every unit, unless the net input is zero. We associate the nodes with the activation value 1 to the accepted set A of the coherence problem, and the nodes with the activation value -1 to the rejected set R . The nodes with net input zero, which may not settle into either state 1 or -1 , can be placed in either set without affecting the harmony of the network. It turns out that the vertices corresponding to these nodes can be placed in either set without effecting the quality of the solution to the coherence problem. In the next theorem, we show the correspondence between the harmony of the stable states of the neural network and the quality of the associated solution to the coherence problem.

Theorem 3.4 *Let N be the neural network after having settled into a stable state, and let (A, R) be the associated solution to the coherence problem. Let $w(A, R)$ be the weight of the solution, and let $w(G)$ be the weight of the entire graph. Then $H_2(N, \infty) = 2 * w(A, R) - w(G)$.*

Proof: Suppose that the network reaches a stable state with harmony $H(N, \infty)$.

By the previous theorem, if a state is stable, then all nodes must have an activation value of 1 or -1 , unless the net input to the node is zero. We map the state of the network onto a solution (A, R) of the coherence problem as follows: nodes with an activation values of 1 are mapped to the accepted set

A . and nodes with an activation value of -1 are mapped to the rejected set R .

First, we show that for all nodes with net input zero, the value of the harmony is not affected by the activation value. Suppose for unit a_i that $net_i(\infty) = 0$. The contribution of unit a_i to $H(N, \infty)$ is $\sum_j w_{ij} a_i(\infty) a_j(\infty) = a_i(\infty) (\sum_j w_{ij} a_j(\infty)) = a_i(\infty) * net_i(\infty) = 0$, since $net_i(\infty) = 0$. Thus, we can consider all of the nodes with $net_i(\infty) = 0$ to be in A (i.e., to have value 1), since the activation value does not effect the harmony.

First, we compute $w(A, R)$, the weight of the solution to the coherence problem. Consider a node $v_i \in A$. The contribution of v_i to the weight of the solution is

$$\sum_{\substack{v_j \in A \\ (v_i, v_j) \in C^+}} w(v_i, v_j) + \sum_{\substack{v_j \in R \\ (v_i, v_j) \in C^-}} w(v_i, v_j)$$

Similarly, for a node $v_i \in R$, the contribution of v_i to the weight of the solution is

$$\sum_{\substack{v_j \in R \\ (v_i, v_j) \in C^+}} w(v_i, v_j) + \sum_{\substack{v_j \in A \\ (v_i, v_j) \in C^-}} w(v_i, v_j)$$

Summing over all vertices in A and R , we get the weight of the solution (A, R) .

$$w(A, R) = \sum_{v_i \in A} \left(\sum_{\substack{v_j \in A \\ (v_i, v_j) \in C^+}} w(v_i, v_j) + \sum_{\substack{v_j \in R \\ (v_i, v_j) \in C^-}} w(v_i, v_j) \right) + \sum_{v_i \in R} \left(\sum_{\substack{v_j \in R \\ (v_i, v_j) \in C^+}} w(v_i, v_j) + \sum_{\substack{v_j \in A \\ (v_i, v_j) \in C^-}} w(v_i, v_j) \right)$$

Now, we compute the harmony of the state of the neural network that produced the solution (A, R) . Assume for node a_i that $v_i \in A$. We will compute the contribution of node a_i to the harmony. Since $v_i \in A$, $a_i(\infty) = 1$. For all edges in $(v_i, v_j) \in C^+$, we have $w_{ij} = w(v_i, v_j)$ in N . Now, since all $v_j \in A$ also have $a_j(\infty) = 1$, we have a harmony contribution of

$$\sum_{\substack{r_j \in A \\ (v_i, v_j) \in C^+}} a_j(\infty)w_{ij} = \sum_{\substack{r_j \in A \\ (v_i, v_j) \in C^+}} (1)(w(v_i, v_j)) = \sum_{\substack{r_j \in A \\ (v_i, v_j) \in C^+}} w(v_i, v_j)$$

Similarly, for all edges $(v_i, v_j) \in C^-$, we have $w_{ij} = -w(v_i, v_j)$, and for all $v_i \in R$, $a_i(\infty) = -1$. Thus, the harmony contribution is

$$\sum_{\substack{r_j \in R \\ (v_i, v_j) \in C^-}} a_j(\infty)w_{ij} = \sum_{\substack{r_j \in R \\ (v_i, v_j) \in C^-}} (-1)(-w(v_i, v_j)) = \sum_{\substack{r_j \in R \\ (v_i, v_j) \in C^-}} w(v_i, v_j)$$

Thus, the contribution to harmony of all satisfied constraints containing node a_i is

$$\sum_{\substack{r_j \in A \\ (v_i, v_j) \in C^+}} w(v_i, v_j) + \sum_{\substack{r_j \in R \\ (v_i, v_j) \in C^-}} w(v_i, v_j)$$

A similar analysis holds if $v_i \in R$. Summing over all vertices v_i for $1 \leq i \leq n$, we get $w(A, R)$. Thus, the contribution to the harmony of all vertices that satisfy constraints is just $w(A, R)$.

There is also a contribution to harmony from the constraints that are violated. Assume again that $v_i \in A$. The harmony contribution of violated constraints is

$$\sum_{\substack{r_j \in A \\ (v_i, v_j) \in C^-}} -w(v_i, v_j) + \sum_{\substack{r_j \in R \\ (v_i, v_j) \in C^+}} -w(v_i, v_j)$$

Summing over all nodes gives

$$\sum_{v_i \in A} \left(\sum_{\substack{v_j \in A \\ (v_i, v_j) \in C^-}} -w(v_i, v_j) + \sum_{\substack{v_j \in R \\ (v_i, v_j) \in C^+}} -w(v_i, v_j) \right) +$$

$$\sum_{v_i \in R} \left(\sum_{\substack{v_j \in R \\ (v_i, v_j) \in C^-}} -w(v_i, v_j) + \sum_{\substack{v_j \in A \\ (v_i, v_j) \in C^+}} -w(v_i, v_j) \right)$$

Which is just $-(w(G) - w(A, R))$. Hence, the harmony is

$$w(A, R) - (w(G) - w(A, R)) = 2 * w(A, R) - w(G)$$

■

It follows from this theorem that the higher the harmony achieved in the neural network N , the better the solution to the coherence problem.

3.3.3 Variations on the Update Rule with Provable Approximation Bounds

We have seen in the previous section that by using a Hopfield-type update rule, the network must settle into a state of maximal harmony. This does not, however, guarantee that it will settle into a state of maximum harmony. Indeed, achieving the state of maximum harmony is NP-hard, since it would amount to finding the optimal solution to the coherence problem. It would be nice to know how good the solution produced by the neural network is. While we have no general answer to this question for the update rule discussed above, in this section we show that by restricting the update order of the nodes, the network will produce the same solution as the approximation algorithm given in Section 3.1.

Instead of updating the activation values of the nodes asynchronously, as is done for the activation function given above, or synchronously as is done in Thagard's networks [Th 89], suppose that we update the nodes in order. That is to say, for each node i , where $1 \leq i \leq n$, we update node i until it reaches a stable state before updating node $i + 1$ (see Figure 3.3). We claim that the solution obtained will correspond exactly to the solution produced by the approximation algorithm of Section 3.1.

Algorithm Ordered-Connect-Coherence(N)

1. Let a_1, a_2, \dots, a_n be an arbitrary ordering of the nodes of N .
2. Let $t = 1$
3. Let $a_1(t) = 1$, and $a_j(t) = 0$ for all $j \neq 1$
4. for $i = 2$ to n
5. while $a_i(t) \neq 1$ and $a_i(t) \neq -1$ and $net_i(t) \neq 0$
6. set $a_i(t + 1) = a_i(t) + \begin{cases} net_i(t)(max - a_i(t)) & \text{if } net_i(t) > 0 \\ net_i(t)(a_i(t) - min) & \text{otherwise} \end{cases}$
7. for all $j \neq i$, $a_j(t + 1) = a_j(t)$
8. $t = t + 1$
9. end while
10. end for
11. output N

Figure 3.3: A Connectionist Algorithm for the Coherence Problem with Provable Approximation Bounds

Theorem 3.5 *Algorithm Ordered-Connect-Coherence() is equivalent to Algorithm Approx-Coherence().*

Proof: To show that these algorithms are equivalent, we show that for all inputs, and for each vertex v_i , they make the same choice as to which set, A or R , v_i is placed in. The proof is by induction on i .

Let A_i denote the set A for each i at step 3 of algorithm Approx-Coherence(), and similarly for R_i .

For Algorithm Ordered-Connect-Coherence(), we will consider all updates of the activation value of unit a_i to occur at time i . This makes sense because net_i remains constant for all updates of node a_i , hence all updates will be in the direction of the sign of net_i .

Inductive Hypothesis: For $1 \leq i \leq k$, $v_i \in A_i$ iff $a_i(i) = 1$, and $v_i \in R_i$ iff $a_i(i) = -1$.

Basis: $i = 1$ - For algorithm Approx-Coherence(), $A = R = \emptyset$, so $w(A \cup v_1) = w(R \cup v_1) = 0$, and v_1 is placed in A at step 4. For algorithm Ordered-Connect-Coherence(), by definition, the activation value of the first node considered is set to 1 at step 3. Thus, the claim holds for the base case.

Induction Step: Consider any fixed value of i at step 3 of algorithm Approx-Coherence(). Suppose that $w(A \cup v_i) \geq w(R \cup v_i)$ at step 4. Then at step 5, v_i will be added to A .

By the inductive hypothesis, we can conclude that for all vertices $v_j \in A_{i-1}$, $a_j(i-1) = 1$, and for all vertices $v_j \in R_{i-1}$, $a_j(i-1) = -1$. Now, we show that $a_i(i)$ must tend to 1. To show this, it is sufficient to show that $net_i(i)$ is positive.

Let

$$s_1 = \sum_{\substack{v_j \in A \\ (v_i, v_j) \in C^+}} w(v_i, v_j)$$

$$s_2 = \sum_{\substack{v_j \in A \\ (v_i, v_j) \in C^-}} w(v_i, v_j)$$

$$s_3 = \sum_{\substack{v_j \in R \\ (v_i, v_j) \in C^+}} w(v_i, v_j) \text{ and}$$

$$s_4 = \sum_{\substack{v_j \in R \\ (v_i, v_j) \in C^-}} w(v_i, v_j).$$

Since $w(A \cup v_i) \geq w(R \cup v_i)$, $s_1 + s_4 \geq s_2 + s_3$, so $s_1 + s_4 - (s_2 + s_3)$ is positive.

Now.

$$\begin{aligned} net_i(i) &= \sum_{\{j|v_j \in A\}} w_{ij}a_j(i) + \sum_{\{j|v_j \in R\}} w_{ij}a_j(i) \\ &= \sum_{\substack{\{j|v_j \in A\} \\ (v_i, v_j) \in C^+}} w_{ij}a_j(i) + \sum_{\substack{\{j|v_j \in A\} \\ (v_i, v_j) \in C^-}} w_{ij}a_j(i) + \\ &\quad \sum_{\substack{\{j|v_j \in R\} \\ (v_i, v_j) \in C^+}} w_{ij}a_j(i) + \sum_{\substack{\{j|v_j \in R\} \\ (v_i, v_j) \in C^-}} w_{ij}a_j(i) \\ &= \sum_{\substack{\{j|v_j \in A\} \\ (v_i, v_j) \in C^+}} w(v_i, v_j)(1) + \sum_{\substack{\{j|v_j \in A\} \\ (v_i, v_j) \in C^-}} -w(v_i, v_j)(1) + \\ &\quad \sum_{\substack{\{j|v_j \in R\} \\ (v_i, v_j) \in C^+}} w(v_i, v_j)(-1) + \sum_{\substack{\{j|v_j \in R\} \\ (v_i, v_j) \in C^-}} -w(v_i, v_j)(-1) \\ &= s_1 - s_2 - s_3 + s_4 \\ &= s_1 + s_4 - (s_2 + s_3) \end{aligned}$$

Thus, $net_i(i)$ must be positive, and it follows that $a_i(i) \rightarrow 1$. The proof is similar for the case when $w(A \cup v_i) < w(R \cup v_i)$ at step 4. ■

Thus, we have shown that by restricting the update order of the nodes, the network produces a solution within a factor one-half of the weight of the optimal solution. Intuitively, by removing the restriction on the update order, we should get a strictly better approximation. This does not follow immediately, however, since it may be the case that by removing the restriction on the update order, the network can settle into a state that is worse than a factor one-half of optimal.

In Section 3.3, we discussed connectionist algorithms for the coherence problem. For a variation on the update rule, we showed that the connectionist algorithm given is equivalent to the approximation algorithm of Section 3.1. Without restricting the update rule, we are currently unable to show that the connectionist algorithm produces approximations with guaranteed performance bounds: the derivation of such bounds remains an important open question. We conjecture that the fully parallel connectionist algorithm will perform strictly better than the on-line approximation algorithm given in this chapter.

Chapter 4

A 0.878-Approximation Algorithm

For twenty years, the best approximation algorithm known for the max cut problem was the 0.5-approximation algorithm of Sahni and Gonzalez [SG 76]. In a recent breakthrough in approximation theory, Goemans and Williamson [GW 94] gave a 0.878-approximation algorithm using semidefinite programming. In this chapter, we apply their results to obtain a 0.878-approximation algorithm for the coherence problem. The techniques we use here are essentially the same as those [GW 94] for MAX CUT, except that the objective function for the coherence problem is more general than the objective function for MAX CUT. The objective function for the coherence problem draws on both the objective functions for MAX CUT and MAX 2SAT.

4.1 An Objective Function for Coherence

We define a variable y_i corresponding to each vertex v_i . For a partition (A, R) , let $y_i = 1$ if $v_i \in A$, and $y_i = -1$ if $v_i \in R$. Let $w_{ij} = w(v_i, v_j)$. The weight of the partition (A, R) can be expressed as an integer quadratic program, as follows:

$$\begin{aligned} \text{Maximize: } & \frac{1}{2} \left(\sum_{\substack{i < j \\ (v_i, v_j) \in C^+}} w_{ij}(1 + y_i y_j) + \sum_{\substack{i < j \\ (v_i, v_j) \in C^-}} w_{ij}(1 - y_i y_j) \right) \quad (4.1) \\ \text{subject to: } & y_i \in \{-1, 1\} \quad \forall i \text{ such that } v_i \in V \end{aligned}$$

That the above equation encodes the weight of the partition (A, R) can be seen as follows. Suppose that $(v_i, v_j) \in C^+$, and v_i and v_j are both placed in A (the constraint is satisfied). Then $y_i = 1$ and $y_j = 1$, so $w_{ij}(1 + y_i y_j) = 2w_{ij}$, which when multiplied by $\frac{1}{2}$ gives w_{ij} . Similarly, if both v_i and v_j are placed in R , we get w_{ij} . Now, if v_i is placed in A and v_j is placed in R (the constraint is not satisfied), we get $y_i = 1$ and $y_j = -1$, so $w_{ij}(1 + y_i y_j) = 0$. Similarly, if $y_i = -1$ and $y_j = 1$, $w_{ij}(1 + y_i y_j) = 0$. A similar analysis for the second term of (4.1) shows that if the negative constraint is satisfied, the net contribution to the weight of the partition is w_{ij} , or 0 if it is not satisfied.

Thus, the objective function of (4.1) encodes the coherence problem, and an optimal solution to this integer programming problem would give an optimal solution to the coherence problem. As would be expected, solving this problem is NP-hard. In the following, we show how to relax the optimization problem to obtain a semidefinite program, which can be solved efficiently. The vectors obtained by solving the semidefinite program will then be randomly partitioned to obtain an approximate solution to the coherence problem. First, we discuss briefly semidefinite programming.

4.2 Positive Semidefinite Matrices

An $n \times n$ matrix A is said to be positive semidefinite if for every vector $x \in R^n$, $x^T Ax \geq 0$. For a symmetric matrix A the following are equivalent: (i) A is positive semidefinite; (ii) all eigenvalues of A are nonnegative; and (iii) there exists a matrix B such that $A = B^T B$.

4.3 Semidefinite Programming

In [Ali 92], an efficient algorithm is given for solving semidefinite programming problems. The standard semidefinite programming problem is defined as follows:

$$\min_X \{C \bullet X : A_i \bullet X = b_i \text{ for } i = 1, \dots, m \text{ and } X \text{ is positive semidefinite}\} \quad (4.2)$$

where C, A_i and X are $n \times n$ matrices, the b_i s are integers, and X is symmetric and positive semidefinite. For matrices A and B , the operation $A \bullet B$ is the inner product, defined as $A \bullet B = \sum_{i,j} A_{ij} B_{ij}$.

The algorithm of Alizadeh [Ali 92] solves any semidefinite program, as defined in (4.2), for variable X to within an additive error ϵ of optimality in $O(\sqrt{n} |\log \frac{1}{\epsilon}|)$ iterations, where each iteration requires time polynomial in n . Note that while the solution produced is a symmetric positive semidefinite matrix X , the coefficient matrices C and A_i are not required to be either symmetric or positive semidefinite.

4.4 Semidefinite Relaxations

In this section, we show how to relax the objective function for the coherence problem to a semidefinite programming problem. Consider the y_i variables of (4.1)

to be unit norm vectors in one dimension. Now, suppose that we allow y_i to be an n -dimensional vector z_i of unit norm. Let S_n denote the n -dimensional unit sphere in R_n . Then $z_i \in S_n$. The program of (4.1) then becomes:

$$\begin{aligned} \text{Maximize:} \quad & \frac{1}{2} \left(\sum_{\substack{i < j \\ (v_i, v_j) \in C^+}} w_{ij}(1 + z_i \cdot z_j) + \sum_{\substack{i < j \\ (v_i, v_j) \in C^-}} w_{ij}(1 - z_i \cdot z_j) \right) \quad (4.3) \\ \text{subject to:} \quad & z_i \in S_n \quad \forall i \text{ such that } v_i \in V \end{aligned}$$

To see that the program of (4.3) is a relaxation of (4.1), note that when the z_i 's are one-dimensional unit vectors, $(1 - z_i \cdot z_j)$ reduces to $(1 - y_i y_j)$.

The objective in (4.3) is not yet in the form of a semidefinite program, since the z_i 's are unit vectors. However, the dot product of z_i and z_j is a scalar value in the range $[-1, 1]$, which we denote by z_{ij} . Hence, we can rewrite (4.3) as:

$$\begin{aligned} \text{Maximize:} \quad & \frac{1}{2} \left(\sum_{\substack{i < j \\ (v_i, v_j) \in C^+}} w_{ij}(1 + z_{ij}) + \sum_{\substack{i < j \\ (v_i, v_j) \in C^-}} w_{ij}(1 - z_{ij}) \right) \quad (4.4) \\ \text{subject to:} \quad & z_{ii} = 1 \quad \forall i \text{ such that } v_i \in V \end{aligned}$$

Restricting z_{ii} to 1 forces the z_i 's to be unit vectors, since $z_{ii} = z_i \cdot z_i = 1$ if and only if the norm of z_i is 1.

The program of (4.4) is now in the form of a semidefinite program. In order to transform the solution of this program back to the form of (4.3), we use the following property of symmetric positive semidefinite matrices (property (iii) of Section 4.2): if A is a symmetric positive semidefinite matrix, then there exists a matrix B such that $A = B^T B$, and the matrix B can be computed in polynomial time using incomplete Cholesky decomposition. Now, let $Z = (z_{ij})$. Using Cholesky

decomposition, a matrix Z' can be computed such that $Z = Z'^T Z'$. Let the columns of Z' be denoted z_1, \dots, z_n . As noted above, the z_i 's are vectors on the unit sphere.

4.4.1 Randomized Partitioning

In the above section, we have shown how to solve the relaxation of the coherence problem given in (4.3): namely to solve the semidefinite program of (4.4) for the matrix Z , and then to compute a matrix Z' whose columns comprise a set of unit vectors z_1, \dots, z_n on the n -dimensional sphere via incomplete Cholesky factorization. Now, we show how to compute an approximate solution to the coherence problem using vectors z_1, \dots, z_n . The technique is to draw a vector r uniformly at random from the unit sphere S_n , and to assign vertices to the accepted set A or rejected set R as follows:

$$A = \{v_i | z_i \cdot r \geq 0\}$$

$$R = \{v_i | z_i \cdot r < 0\}$$

Taking the dot product of each vector z_i with the random vector r partitions the vectors with a random hyperplane through the origin with normal r into the set A of vectors that lie above the hyperplane, and the set R that lie below it.

We show in the following section that the solution to the coherence problem produced by this randomized partitioning technique has expected weight within 0.878 of optimal.

4.4.2 Expected Performance

Let $E[w(A, R)]$ be the expected value of the weight of partition (A, R) produced by the randomized algorithm (where the weight of a partition is as defined in Section

1.4). The following theorem gives a performance ratio for $E[w(A, R)]$.

Theorem 4.1

$$E[w(A, R)] \geq \alpha \frac{1}{2} \left(\sum_{\substack{i < j \\ (v_i, v_j) \in C^+}} w_{ij}(1 + z_i \cdot z_j) + \right. \quad (4.5)$$

$$\left. \sum_{\substack{i < j \\ (v_i, v_j) \in C^-}} w_{ij}(1 - z_i \cdot z_j) \right) \quad (4.6)$$

where $\alpha = \min_{0 < \theta \leq \pi} \frac{2}{\pi} \frac{\theta}{1 - \cos \theta} > 0.87856$.

The approximation algorithm for coherence draws a random vector r , and partitions the vectors z_1, \dots, z_n obtained from the solution of the semidefinite program into those above the hyperplane whose normal is r , and those below it. Thus, the expected weight of the partition (A, R) is given by the following lemma:

Lemma 4.2

$$E[w(A, R)] = \sum_{\substack{i < j \\ (v_i, v_j) \in C^+}} w_{ij} \cdot \Pr[\text{sgn}(z_i \cdot r) = \text{sgn}(z_j \cdot r)] + \quad (4.7)$$

$$\sum_{\substack{i < j \\ (v_i, v_j) \in C^-}} w_{ij} \cdot \Pr[\text{sgn}(z_i \cdot r) \neq \text{sgn}(z_j \cdot r)] \quad (4.8)$$

where $\text{sgn}(x) = 1$ if $x \geq 0$, and -1 otherwise.

Proof: Recall that a positive constraint is satisfied if both vertices are placed in A , or both in R . Recall also that vertex v_i is placed in A if $z_i \cdot r \geq 0$. Hence, the summation of (4.7) is the expected weight of positive constraints that are satisfied by the solution. Similarly, the summation of (4.8) is the expected weight of negative constraints that are satisfied by the solution. ■

The following two lemmas are proven in [GW 94]. We include the proofs here for completeness.

Lemma 4.3 ([GW 94])

$$\Pr[\text{sgn}(z_i \cdot r) \neq \text{sgn}(z_j \cdot r)] = \frac{1}{\pi} \arccos(z_i \cdot z_j) \quad \text{and} \quad (4.9)$$

$$\Pr[\text{sgn}(z_i \cdot r) = \text{sgn}(z_j \cdot r)] = 1 - \frac{1}{\pi} \arccos(z_i \cdot z_j) \quad (4.10)$$

Proof: We first show that $\Pr[\text{sgn}(z_i \cdot r) \neq \text{sgn}(z_j \cdot r)] = \frac{1}{\pi} \arccos(z_i \cdot z_j)$. Since r is a vector drawn uniformly at random from the unit sphere, the probability that $\text{sgn}(z_i \cdot r) \neq \text{sgn}(z_j \cdot r)$ is simply the probability that the hyperplane whose normal is r separates vectors z_i and z_j : thus, it is proportional to the angle θ between z_i and z_j (given by $\arccos(z_i \cdot z_j)$). By symmetry, $\Pr[\text{sgn}(z_i \cdot r) \neq \text{sgn}(z_j \cdot r)] = 2 \Pr[z_i \cdot r \geq 0, z_j \cdot r < 0]$. The set $\{r : z_i \cdot r \geq 0, z_j \cdot r < 0\}$ is the intersection of two half-spaces whose angle is θ , and the intersection of this set with the unit sphere is a spherical digon of angle θ . Thus, the measure of $\Pr[z_i \cdot r \geq 0, z_j \cdot r < 0]$ is $\frac{\theta}{2\pi}$, and $\Pr[\text{sgn}(z_i \cdot r) \neq \text{sgn}(z_j \cdot r)] = \frac{\theta}{\pi}$.

Now, $\Pr[\text{sgn}(z_i \cdot r) \neq \text{sgn}(z_j \cdot r)] + \Pr[\text{sgn}(z_i \cdot r) = \text{sgn}(z_j \cdot r)] = 1$, so it follows that $\Pr[\text{sgn}(z_i \cdot r) = \text{sgn}(z_j \cdot r)] = 1 - \frac{\theta}{\pi}$. ■

Lemma 4.4 ([GW 94]) For $-1 \leq y \leq 1$, $\frac{1}{\pi} \arccos(y) \geq \alpha \frac{1}{2}(1 - y)$, and $1 - \frac{1}{\pi} \arccos(y) \geq \alpha \frac{1}{2}(1 + y)$, where $\alpha = \min_{0 < \theta \leq \pi} \frac{2}{\pi} \frac{\theta}{1 - \cos \theta} > 0.87856$.

Proof: We first show that $\frac{1}{\pi} \arccos(y) \geq \alpha \frac{1}{2}(1 - y)$. Let $y = \cos \theta$. Substituting gives $\frac{\theta}{\pi} \geq \alpha \frac{1}{2}(1 - \cos \theta)$. Using calculus, we get $\frac{2}{\pi} \theta > .87856(1 - \cos \theta)$.

That $1 - \frac{1}{\pi} \arccos(y) \geq \alpha \frac{1}{2}(1 + y)$ follows from the above, by substituting $-y$ for y , and by the fact that $\pi - \arccos(y) = \arccos(-y)$. ■

The proof of Theorem 4.1 follows from Lemmas 4.2, 4.3 and 4.4.

In this chapter, we have applied the semidefinite programming approximation technique of [GW 94] to the coherence problem, to obtain a 0.878-approximation algorithm. The best performance guarantee previously known for this problem was the 0.5-approximation algorithm of the previous chapter.

It would be interesting to explore the application of semidefinite programming to certain minimization problems. For example, rather than approximating the maximum weight of satisfied constraints in the coherence problem, it would be interesting to approximate the minimum weight of rejected constraints. These are dual problems when referring to exact solutions, but it may be possible to get better approximations in certain cases by using an approximation to the dual. In practice, the optimal coherence will generally satisfy a large fraction of the constraints, perhaps in the order of 70%; thus an algorithm that approximates the rejected constraints (30% of the constraints) may perform better. We have not been successful at applying the semidefinite programming technique to this dual formulation. The best algorithm known for minimizing the number of rejected constraints is an $O(\log k)$ -approximation algorithm, where k is the number of constraints, which follows from the work of Klein et. al. [KARR 90].

Chapter 5

An Approximation Algorithm for Neural Nets

5.1 A 0.878-Approximation Algorithm

In [Th 89], Thagard showed how to implement coherence problems using neural networks. In Chapter 4, we showed the relationship between the weight of the solution to the coherence problem and the harmony of the neural network used to encode the problem. In this section, the theorem of Chapter 3 is restated and generalized to show the relationship between arbitrary neural networks and coherence problems. We use this relationship to get an approximation algorithm for settling neural networks. We use the terminology introduced in Section 3.3.1 for describing the neural networks.

5.1.1 Encoding Neural Networks as Coherence Problems

In the Chapter 3, we showed how, given an arbitrary instance of the coherence problem, to construct a neural network that encodes the problem instance. It should be evident that the converse holds as well: an arbitrary neural network, with bounded maximum and minimum activation values, can be encoded as an instance of the coherence problem. We assume here that the neural network has maximum and minimum activation values bounded by 1 and -1 respectively: then the neural network can be encoded as an instance of coherence simply by placing all links with positive weight in the set C^+ of positive constraints, and all links with negative weight in the set C^- of negative constraints. The vertices that are placed in the accepted set A correspond to nodes of the neural network that settle into the 1 state, and those that are placed in R correspond to nodes that settle into the -1 state.

5.1.2 Harmony and Scaled Harmony

Recall from Section 3.3.2 that the Harmony is defined as:

$$H_2(N, t) = \sum_{i < j} w_{ij} a_i(t) a_j(t) \quad (5.1)$$

The value of $H_2(N, t)$ will be in the range $[-w(G), w(G)]$, where $w(G) = \sum_{i < j} |w_{ij}|$. The possibility of a negative or near-zero value for the optimal harmony leads to problems of unbounded performance ratios, since the denominator in the approximation ratio (see equation 1.1) will be negative or near zero. Thus, in order to talk about approximating harmony, we introduce the notion of *scaled harmony*, which we define as:

$$SH_2(N, t) = H_2(N, t) + w(G)$$

The following corollary to Theorem 3.4 shows the correspondence between the scaled harmony of the stable states of the neural network and the quality of the associated solution to the coherence problem.

Corollary 5.1 *Let N be the neural network after having settled into a stable state, and let (A, R) be the associated solution to the coherence problem. Let $w(A, R)$ be the weight of the solution, and let $w(G) = \sum_{i < j} |w_{ij}|$ be the weight of the entire graph. Then $SH_2(N, \infty) = 2 * w(A, R)$.*

It follows from Corollary 5.1 that the higher the harmony achieved in the neural network N , the better the solution to the coherence problem, and vice-versa. We will use this corollary to show that by encoding a neural network as a coherence problem, the solution obtained using the 0.878-approximation algorithm for coherence gives an approximation algorithm for settling neural networks.

Theorem 5.2 *Let N be a neural network with activation values bounded by $\max = 1$ and $\min = -1$, and G be the associated graph for the coherence problem. Then the solution obtained by running the 0.878-approximation algorithm for coherence on G gives a state for N for which $SH_2(N, \infty)$ is within a factor 0.878 of optimal.*

Proof: Let $w(A^*, R^*)$ be the weight of the optimal solution for an instance of the coherence problem, and let N^* denote the neural network N after having settled into the state with the optimal harmony. By Corollary 5.1, $SH_2(N^*, \infty) = 2 * w(A^*, R^*)$. Running the 0.878-approximation algorithm for coherence on G will give a solution (A, R) with $w(A, R) \geq 0.878w(A^*, R^*)$, and by Corollary 5.1, $SH_2(N, \infty) \geq 2 * (0.878w(A^*, R^*))$. Thus, the performance ratio in approximating $SH_2(N^*, \infty)$ is bounded by:

$$\frac{2 * (0.878w(A^*, R^*))}{2 * w(A^*, R^*)} = 0.878$$

Hence, the value of $SH_2(N, \infty)$ produced by the approximation algorithm is within 0.878 of optimal. ■

We make one final note on Theorem 5.2: the solution produced by applying the approximation algorithm for coherence to a neural network will produce a state that is within 0.878 of the scaled harmony of the optimal stable state. The state produced, however, is not necessarily stable itself. A stable state is easily obtained using this state as the start state, though, by applying a standard Hopfield-type algorithm to settle the network into a stable state with scaled harmony at least as good as that of the start state.

In this chapter, we showed the correspondence between the weight of a solution to the coherence problem and the harmony of the neural network encoding of the problem, and used this correspondence to obtain a 0.878-approximation algorithm for settling neural networks with bounded activation values. The 0.878 performance guarantee applies to a scaled version of harmony: namely the harmony plus the sum of the absolute values of the weights in the network. We argue in Section 3.3 that this is a reasonable quantity to approximate, due to the technical difficulty of approximating the harmony itself. Nevertheless, it would be interesting to investigate different notions of approximating optimal harmony.

Chapter 6

Future Work on Constraint Satisfaction

In [GW 94], Goemans and Williamson gave 0.878-approximation algorithms for both MAX CUT and MAX 2SAT. No significant improvements have been made since then for the MAX CUT problem, but for MAX 2SAT, Feige and Goemans [FG 95] improve the 0.878-approximation result to 0.931. While the formulation of the coherence Problem is not equivalent to MAX 2SAT, the objective function that we use for coherence does bear some resemblance to that used in [GW 94] for MAX 2-SAT. Thus, it would be interesting to see if these results can be applied to obtain an improved approximation algorithm for coherence.

Since we have shown that the coherence problem is MAX SNP complete, there exists some constant upper bound on the approximation ratio achievable. For MAX CUT, the best upper bound currently known is 0.9813, due to Trevisan, Sorkin, Sudan and Williamson [TSSW 96], which is a slight improvement to the 0.988 result of [BGS 95] (we refer the reader to [CK 95] for an excellent survey of the current

state of the art in approximation ratios.) The best upper bound known for MAX 2SAT is 0.9872 [BGS 95]. Since coherence solves MAX CUT as a special case, the upper bound for MAX CUT applies also to coherence. It may well be the case, however, that tighter upper bounds can be shown for coherence than for MAX CUT. Obtaining tighter bounds for coherence is an interesting open problem.

Klein, Agrawal, Ravi and Rao [KARR 90] have studied a problem they call 2-CNF \equiv , which is to determine the minimum number of clauses that can be deleted from a 2-CNF formula to make it satisfiable. The coherence problem can be encoded as a 2-CNF formula in an approximation-preserving reduction. In this transformation, minimizing the number of clauses to be deleted to produce a satisfying formula is equivalent to minimizing the number rejected constraints in the coherence problem. The algorithm given in [KARR 90] yields an $O(\log^3 n)$ -approximation algorithm. This has been improved to $O(\log n)$ in [GVY 93]. It remains an open question whether a constant-factor approximation algorithm exists for this problem.

The version of the CSP that we have considered has binary constraints, and the only cases we have given approximation algorithms for also have binary domains. There are two directions in which many open problems lie: the generalization from binary to k -ary constraints; and the generalization from binary to multi-valued domains. The case of k -ary constraints has been studied by [KMSV 94], and they give an approximation algorithm with performance ratio $\frac{1}{2^k}$. The algorithm they give uses non-oblivious local search. The semidefinite approximation technique we use to approximate the binary case uses a global search technique. It would be interesting to study whether the global search techniques can be extended to the k -ary CSP, or whether local search algorithms can achieve better approximation ratios.

The second direction in which there are many interesting open problems is the

generalization of the binary CSP to multi-valued domains. Note that this problem contains as a special case the graph-coloring problem, which has been well-studied for decades. The results of Karger, Motwani and Sudan [KMS 94] give an algorithm with approximation ratio $n^{(1-\frac{3}{k+1})} \log n$, for instances of the CSP with unweighted constraints, where all constraints are “NOT-EQUAL” (i.e., constraints specify that two variables cannot have the same value.) Their algorithm is to date the strongest result on graph coloring.

In this thesis, we have given an approximation algorithm for the coherence problem, a restricted version of the binary-valued binary CSP. Due to its generality, the multi-valued multi-ary CSP contains as special cases many NP-hard optimization problems, hence generalizing the binary problem to multi-ary constraints and multi-valued variables will contain many important but challenging problems.

Chapter 7

Part II - Learning Monotone DNF

7.1 Introduction

In this thesis, we give positive learnability results for sub-classes of monotone DNF formulas on the uniform distribution. We define the classes of *one-read-once monotone DNF*, for which each term of the formula has at least one attribute that is read-once and *read-once factorable monotone DNF*. Both classes are generalizations of read-once monotone DNF formulas, where every attribute in the formula must be read-once. In Chapter 9, we give the main learnability results of this part of the thesis: we show that poly-disjoint one-read-once monotone DNF formulas, where for each term of the formula, the set of examples that satisfy only that term has polynomial probability weight, are learnable on the uniform distribution; and we show that read-once factorable monotone DNF formulas are learnable on the uniform distribution. We use spectral analysis to show the correspondence between the Fourier coefficients of terms of a one-read-once formula and the probability weight of the set of vectors that satisfy exactly one term of the one-read-once for-

mula. More precisely, we show that if the positive Fourier coefficients of a term of a one-read-once formula is large then the term can be constructed in polynomial time, and if it is small, then the term is insignificant in the approximation of the formula.

Our learnability result for read-once factorable monotone DNF formulas is based on a lemma that we prove in Chapter 8, which we dub the Diffraction Lemma. In this lemma, we show that if a set of terms all have the property that the probability weight of the set of vectors that satisfy exactly one term is small, then the entire set of terms can be approximated by their greatest common factor. This, together with the relationship between the weight of the disjoint vectors and the Fourier coefficients of terms of a one-read-once monotone DNF formula gives the learnability result. We now consider briefly previous work on learnability relevant to our results.

7.2 Previous Work

Since the inception of computational learning theory in the PAC (Probably Approximately Correct) learning model due to Valiant [Val 84], the problem of the learnability of DNF has received much attention. The primary reason for this is the potential of DNF as a form of knowledge representation, with applications in expert systems and, more recently, data mining. DNF is also interesting in that it appears to be near the boundary of learnability. Learning general Boolean formulas and log-depth circuits is known to be as hard as factoring [KV 88]. The results of Lund and Yannakakis [LY 93] on the hardness of approximating the k -coloring problem and the results of Pitt and Valiant [PV 86] reducing the coloring problem to the DNF learning problem show that s -term DNF formulas are not learnable by $n^\epsilon s$ -term DNF hypotheses for some ϵ , unless $\text{NP} = \text{RP}$. In contrast, several sub-

classes of DNF formulas are known to be learnable. For an excellent survey of the DNF learning problem, we refer the reader to [AP 95].

The difficulty of learning DNF in the distribution-independent PAC model seems to be due to the robustness of the model. Under distribution-independence, many variants of the PAC model turn out to be equivalent. Kearns, Li, Pitt and Valiant [KLPV 87, KLV 94] showed that if a monotone class of Boolean formulas is learnable in the distribution-independent model, then so is the non-monotone class formed by allowing negated variables in the original class. The same authors introduced the notions of weak learning and group learning. In the weak-learning PAC model, the accuracy of the hypothesis is only required to be a polynomial fraction better than random guessing (i.e., the error of the hypothesis is bounded by $\frac{1}{2} - \frac{1}{p(s,n)}$, where s and n are size parameters of the target function). In the group learning model, the learner receives a set of examples, either all positive examples or all negative examples, and has to determine whether the examples are positive or negative. Kearns, Li, Pitt and Valiant showed the equivalence of the weak-learning model and the group learning model. In a surprising result, Schapire [S 90] showed that weak learning is as hard as PAC learning in the distribution-independent model. Thus, PAC learning, weak learning, and group learning are all equivalent.

Due to the apparent difficulty of learning DNF for arbitrary distributions, research efforts have focussed on learning this class for particular distributions, in particular for the uniform distribution. Kearns, Li, Pitt and Valiant [KLPV 87, KLV 94] gave the first such distribution-specific algorithm for learning μ -DNF formulas, in which every attribute occurs at most once in the formula. These authors also gave a weak-learning algorithm for the entire class of Boolean formulas, again for the uniform distribution. Note that Schapire's results, discussed above, on the

equivalence of weak learning to PAC learning cannot be applied to the distribution-specific case, so the weak-learning algorithm for DNF on the uniform distribution does not imply a DNF learning algorithm for the uniform distribution.

Hancock [H 92] further studied restricted-read DNF, which are sub-classes of DNF in which the number of occurrences of each attribute in the formula is bounded by a constant. The most general class of formulas for which he gives polynomial time algorithms is the $k\mu$ -DNF formulas (or equivalently, the read- k DNF formulas) where each attribute occurs at most k times in the formula. Hancock's results apply also to the uniform distribution.

There have been several positive partial results for the learnability of DNF on the uniform distribution, although no polynomial time algorithms are known. Linial, Mansour and Nisan [LMN 89] showed how to learn AC^0 circuits on the uniform distribution by learning the Fourier coefficients in $O(n^{\log n})$ time. The hypothesis output by this algorithm is an approximation to the Fourier transform of the target function. In [Ver 90], we showed that DNF is learnable under the uniform distribution with a similar time bound, but for which the output hypothesis is a DNF formula.

The learnability of read-once Boolean formulas has been studied by several authors in the membership query and equivalence query models. Angluin, Hellerstein and Karpinski [AHK 93] show that monotone read-once formulas can be learned from membership queries alone, and that read-once formulas can be learned using membership and equivalence queries. In the PAC-learning model, Goldman, Kearns and Schapire show in [GKS 90] that read-once formulas can be learned on the uniform distribution in time $O(\frac{n^9}{\epsilon^6})$, using $O(\frac{n^6}{\epsilon^6})$ examples. This result is generalized to product distributions in [S 92], giving an algorithm with time complexity $O(\frac{n^{14}}{\epsilon^6})$ and sample complexity $O(\frac{n^{12}}{\epsilon^6})$.

Kushilevitz and Mansour [KM 93] gave an algorithm for learning decision trees on the uniform distribution via Fourier coefficients. Mansour [Man 92] used this result to show that when membership queries are allowed, DNF can be learned on the uniform distribution in $O(n^{\log \log n})$ time. A polynomial time algorithm is given by Khardon [Khar 94] for learning Disjoint-DNF, where every example satisfies at most one term. This result uses Fourier analysis of the DNF formula to construct terms of the target formula, and in that respect is in a similar vein to the results of this thesis.

In a break-through result, Jackson [J 94] showed that DNF can be learned in polynomial time on the uniform distribution when membership queries are allowed. The results of [BFJK 94] give a lower bound of $\Omega(n^{\log n})$ for learning DNF on any distribution, including uniform, in the statistical query model due to Kearns [Ke 93], where the learner is allowed only to make queries about the statistics of an arbitrary Boolean function on the input distribution. It remains an open question whether monotone DNF is learnable on the uniform distribution in polynomial time using only examples and no membership queries.

In this thesis, we introduce the classes of one-read-once monotone DNF formulas, and read-once factorable monotone DNF formulas. We give positive learnability results for poly-disjoint one-read-once monotone DNF and read-once factorable monotone DNF, on the uniform distribution. The class of read-once factorable monotone DNF is a superclass of read-once monotone DNF, but a sub-class of read-once monotone Boolean formulas. Thus, the learnability of this class is implied by [GKS 90] and [S 92]; however, the complexity of our algorithm is of a lower order: time and sample complexity $\tilde{O}(n \frac{s^5}{\epsilon})$, where n is the number of attributes in the learning domain, and s is the number of terms in the formula. This complexity is not directly comparable to the algorithms of [GKS 90], since the complexity of their

algorithm is in terms of n , and ours is in terms of s . However, our results give a lower complexity for small formulas. The class of one-read-once monotone DNF is also a generalization of read-once monotone DNF, but since the algorithms we give here work only for poly-disjoint formulas from this class, this result is incomparable to the results for read-once formulas discussed above.

7.3 Definitions and Terminology

7.3.1 Functions and Classes

Let $X = \{x_1, x_2, \dots, x_n\}$ be the set of Boolean attributes in the learning domain. A Boolean function is a function $f : \{0, 1\}^n \rightarrow \{-1, 1\}$, where -1 represents "false", and $+1$ represents "true". A (monotone) *term* t_i is a conjunction of attributes in X , none of which appear negated. Let m_i denote the set of indices of attributes in t_i , and let function t map the sets of indices onto terms; thus $t_i = t(m_i)$. Let function m be the inverse of t , mapping a term onto the set of indices of attributes in the term. Thus $m(t_i) = m_i$. We will sometimes use set notation for terms where the context is clear, for example, $x_i \in t_i$ to imply that $i \in m_i$. For $x \in \{0, 1\}^n$, and a term t_i , we use $x \Rightarrow t_i$ to indicate that vector x satisfies t_i . A *cross-term* of a formula f is a term that contains attributes from more than one term of f .

We consider Boolean functions that can be represented as sub-classes of monotone DNF formulas, and use f as well to denote the representation of the Boolean function by a formula. A Boolean formula f is a monotone DNF (hereafter MDNF) formula if f is of the form $f = t_1 + t_2 + \dots + t_s$, where each t_i is a monotone term. The size of the formula is s , the number of terms in the formula. A *concept* is the set of examples satisfying a *target formula* f . We will sometimes refer to this set of

examples as the *satisfying set* of f . A *concept class* is the set of concepts defined by a class of formulas. For example, the set of all Boolean formulas defines all possible concepts in the Boolean space.

A formula f is *read-once* if no variable appears more than once in f . We define generalizations of the well-studied read-once formulas. An attribute x_i is *read-once* if it occurs exactly once in f . A formula f is *one-read-once* if for every term of f , at least one attribute is read-once.

Let $S(t_i) = \{x|x \Rightarrow t_i\}$ be the set of vectors that satisfy term t_i . Let $\mathcal{DS}(t_i)$ denote the set of vectors that satisfy term t_i , but do not satisfy any other term t_j , $i \neq j$. We will refer to the set $\mathcal{DS}(t_i)$ as the disjoint satisfying set of term t_i .

For a term t and a formula $f = t_1 + \dots + t_s$, we define the restriction of f to t , f_t , by $f_t = t \cdot t_1 + \dots + t \cdot t_s$.

For a set of terms T we define the greatest common factor of T to be the largest term t that is contained in every term in T .

We define the factorization of an MDNF formula recursively. As the base case, a read-once formula is a factorization (with the trivial factor of the empty set of attributes). Any formula that is formed as the sum of products of terms (factors) and a factorization of an MDNF formula is also a factorization. That is, if f_1 and f_2 are factorizations of an MDNF formula then $t_1 \cdot f_1 + t_2 \cdot f_2$, where t_1 and t_2 are terms whose attributes do not already occur in f_1 or f_2 , is a factorization. A monotone DNF formula is said to be *read-once factorable* if there exists a factorization of f such that no attribute occurs more than once in the factorization. For example, for the formula $f = x_1x_2 + x_1x_3 + x_2x_3$, a factorization of f is $f = x_1(x_2 + x_3) + x_2(x_1 + x_3) + x_3(x_1 + x_2)$. Such a representation is called a factored form of f . This formula is not, however, a read-once factorization of f , and indeed, no read-once

factorization exists for f . However, the formula $g = x_1x_2x_3 + x_1x_2x_4 + x_1x_5x_6$ is read-once factorable, and $g = x_1(x_2(x_3 + x_4) + x_5x_6)$ is a factorization.

We refer to the set F containing the factors of a read-once factorization of a formula f as the *set of maximal factors* of f . Thus, if $f = t_1(t_2(t_3 + t_4) + t_5)$, where the t_i s are read-once terms, the maximal factor set consists of $\{t_1, t_1t_2\}$, since t_1 is a factor of $t_2(t_3 + t_4) + t_5$, and t_1t_2 is a factor of $(t_3 + t_4)$. Formula g above has maximal factor set $\{x_1, x_1x_2\}$.

Each of monotone DNF, read-once MDNF, one-read-once MDNF and read-once factorable MDNF is referred to as a *class* of formulas. For each of the classes discussed above, we may use the qualifier “poly-disjoint” to refer to the set of all formulas in the class for which every term t_i in the formula has $Pr_D[\mathcal{DS}(t_i)] \geq \frac{1}{p(s, \frac{1}{\epsilon})}$, where p is a polynomial, s is the size of the formula, $0 \leq \epsilon \leq 1$, and D is a probability distribution.

7.3.2 Learnability

We use the standard definitions for PAC-learnability in this paper. We assume that the reader is familiar with these (see [HKLW 88] for an excellent description). Here, we give only the following definitions.

Let D^+ denote the uniform distribution over the positive examples of the target formula f , and D^- be the uniform distribution over the negative examples. Let D denote the uniform distribution over the entire example space, $\{0, 1\}^n$. We will require error functions for both positive and negative examples. Let the positive error, $e^+(h)$ of hypothesis h with respect to the target formula f be the probability that h miss-classifies a positive example drawn according to distribution D . Similarly, let the negative error, $e^-(h)$, be the probability that the hypothesis h

miss-classifies a negative examples drawn according to distribution D . We use $e(h)$ without the superscripts “+” or “-” to denote the total error on both the positive and negative examples. We also use $\Pr[f \Delta g]$ to denote probability that $f \neq g$ over distribution D .

For a hypothesis h , and for $0 \leq \alpha \leq \frac{1}{2}$, we say that h is an α -good hypothesis, or α -approximate hypothesis, if $e(h) \leq \alpha$.

Let \mathbf{C} and \mathbf{H} be concept classes. For concept class \mathbf{C} , let C_n be the set of concepts in \mathbf{C} with domain size n , and let $C_{n,s}$ be the set of concepts in C_n of size at most s . \mathbf{C} is *polynomially learnable by \mathbf{H}* on the uniform distribution iff there exists an algorithm A with inputs ϵ , δ , s , and n , which $\forall \epsilon, \delta \leq 1, \forall s, n \geq 1$, and all target functions $f \in C_{n,s}$, outputs a representation of a hypothesis $h \in H_n$ which with probability $\geq 1 - \delta$ has $e^+(h) \leq \epsilon$ and $e^-(h) \leq \epsilon$, and the run-time of A is bounded by a polynomial in $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, s , and n .

If the hypothesis output by algorithm A has $e(h) \leq \frac{1}{2} - \frac{1}{p(s,n)}$ for some polynomial p , then we say that A is a *weak polynomial learning algorithm*.

In [Ke 93], Kearns introduced the statistical query model, where the learner makes queries about the statistics of an arbitrary Boolean function over the probability distribution on the examples, rather than drawing individual examples. The algorithms we develop in this work use only statistical queries, hence the results apply to the Statistical Query Model as well as to the traditional PAC model.

7.3.3 Fourier Transform

We use the definitions of the Fourier transform given in [LMN 89] and [J 94]. For every subset $A \subseteq \{1, \dots, n\}$ and for $x \in \{0, 1\}^n$, we define the parity function,

$\chi_A : \{0, 1\}^n \rightarrow \{-1, +1\}$. by:

$$\chi_A(x) = (-1)^{\sum_{i \in A} x_i}.$$

The function $\chi_A(x)$ is 1 if the parity of the bits in x indexed by A is even, and -1 if the parity is odd. For Boolean functions f and g . ($f, g : \{0, 1\}^n \rightarrow \{-1, +1\}$), we define the inner product of f and g by $\langle f, g \rangle = E[fg]$, where the expectation is over all vectors $\{0, 1\}^n$. The norm of a function f is defined by $\|f\| = \sqrt{E[f^2]}$. With the inner product and norm so defined, the set of functions $\{\chi_A | A \subseteq \{1, \dots, n\}\}$ is an orthonormal basis for the vector space of real-valued functions on the Boolean cube Z_2^n . Thus, every function $f : \{0, 1\}^n \rightarrow \mathbf{R}$ can be uniquely expressed as a linear combination of parity functions, by $f = \sum_A \hat{f}(A) \chi_A$, where $\hat{f}(A) = E[f \chi_A]$. The vector of coefficients \hat{f} is called the *Fourier transform* of f . For Boolean f , \hat{f} represents the correlation of f and χ_A with respect to the uniform distribution.

For the purpose of this thesis, we also define the *positive Fourier coefficient* (PFC), which we denote by $\hat{f}^p(A)$, as $\hat{f}^p(A) = (-1)^{|A|} E_{D^+}[\chi_A f]$. Note that since $f = 1$ on all positive examples, this reduces to $\hat{f}^p(A) = (-1)^{|A|} E_{D^+}[\chi_A]$. We use $(-1)^{|A|} \hat{E}_{D^+}[\chi_A]$ to denote the estimate of $\hat{f}^p(A)$.

7.4 Motivation for Spectral Analysis of MDNF

The results of this thesis are based on the properties of Fourier coefficients of sub-classes of monotone DNF formulas. For all of the sub-classes we consider, we show how to construct terms of the target formula using spectral analysis. In particular, in Chapter 9, we show that all terms of a read-once factorable MDNF formula that are significant in the approximation of the target formula have large positive Fourier coefficients. In contrast, all cross-term coefficients are negative. Using these

properties. we show how to construct the terms of a sparse formula by recursively constructing sub-terms with the same properties.

Chapter 8

Approximation Results for Monotone DNF

Before giving results for restricted classes of MDNF formulas, we need some preliminary results on the approximation of formulas. The first fact we give states that every MDNF formula can be approximated with error bounded by $\frac{\epsilon}{2}$ by an MDNF formula with terms of size $\log \frac{2s}{\epsilon}$. In the statement of the results of this chapter, we will use superscript f to denote a term from formula f , and similarly for g .

Fact 8.1 *Let $f = t_1^f + \dots + t_s^f$ be an MDNF formula. There exists an MDNF formula $g = t_1^g + \dots + t_s^g$ for which $|t_i^g| \leq \lg \frac{2s}{\epsilon}$, $t_i^g \subseteq t_i^f$, and $e^-(t_i^g) \leq \frac{\epsilon}{2s}$.*

The proof of this fact is given in [Ver 90]. By Fact 8.1, for every MDNF formula f , there exists a formula g with log-sized terms that approximates f well. In the results of this thesis, we show how to find an approximation to the formula g , which by Fact 8.1 is an approximation to f . We give two results in this chapter on approximation of the formula g , one version specifically for read-once MDNF

formulas, and a more general version with weaker bounds for the class of read-once factorable MDNF.

Lemma 8.2 *For a read-once MDNF formula $g = t_1^g + \dots + t_s^g$ with terms of size at most $\lg \frac{2s}{\epsilon}$ and for the uniform distribution D on examples of g , if $\Pr_D[\mathcal{DS}(t_i^g)] \leq \frac{\epsilon^2}{4s}$ for any term t_i^g , then $\Pr_D[g\Delta 1] \leq \frac{\epsilon}{2}$.*

Proof: We show that the size of the set $\mathcal{DS}(t_i^g)$ of vectors that satisfy term t_i^g but no other term is proportional to the size of the set of negative examples. Take any vector x in $\mathcal{DS}(t_i^g)$. This vector satisfies t_i^g but no other term. Now, changing any attribute in t_i^g from 1 to 0 gives a negative example, due to monotonicity. There are $2^{|t_i^g|}$ vectors in $\{0,1\}^{|t_i^g|}$, so at most $2^{|t_i^g|}$ negative examples can be generated in this way for each vector $x \in \mathcal{DS}(t_i^g)$. Since every term of g has at most $\lg \frac{2s}{\epsilon}$ attributes, the number of negative examples is at most $2^{\lg \frac{2s}{\epsilon}}$ times the number of examples in $\mathcal{DS}(t_i^g)$. Since $\Pr[\mathcal{DS}(t_i^g)] \leq \frac{\epsilon^2}{4s}$, the probability weight of the negative examples is at most $2^{\lg \frac{2s}{\epsilon}} \cdot \Pr[\mathcal{DS}(t_i^g)] \leq \frac{2s}{\epsilon} \cdot \frac{\epsilon^2}{4s} = \frac{\epsilon}{2}$. Thus, the probability of error in approximating formula g by the constant function $h = 1$ is at most $\frac{\epsilon}{2}$, and it follows that $\Pr[g\Delta 1] \leq \frac{\epsilon}{2}$. ■

By Fact 8.1 and Lemma 8.2, we have that for any read-once MDNF formula, if for any term in the formula the probability weight of the set of vectors that satisfy exactly one term is smaller than $\frac{\epsilon^2}{4s}$, then the formula can be approximated by the constant function $h = 1$.

Hancock gives a result for learning k - μ MDNF formulas (also known as read- k MDNF formulas, where each attribute occurs in at most k terms) on the uniform distribution [H 92]. His results are based on the following lemma:

Lemma 8.3 (Hancock [H 92]) *Let f be a $k\mu$ MDNF formula, and let p be the probability that $f(x) = 0$, where x is drawn uniformly at random. For any term T of f with ℓ variables, the probability on the uniform distribution that T is true and all other terms are false is at least $p2^{-k\ell}$.*

This lemma can be used to prove our Lemma 8.2, with $k = 1$. Hancock uses Lemma 8.3 to bound the influence of variables in read- k MDNF formulas, and uses this influence measure in a polynomial time algorithm for read- k MDNF on the uniform distribution. Since read- s MDNF, where s is the size of the formula, is equivalent to MDNF (each attribute can occur at most once in each term), we can derive from Lemma 8.3 that for any term T , the probability weight of the examples that satisfy only T is at least $2^{-s\ell}$ times the probability weight of the negative examples. This does not give a polynomial time learning algorithm for MDNF, however, because of the exponential dependence on s .

In the following, we give a lemma of a similar flavour to that of Lemma 8.3, in that we bound the size of the negative example space in terms of the probability weight of the examples that satisfy exactly one term. There are, however, several essential differences in our lemma. First, we give a lemma that applies to the class of read-once factorable MDNF. Secondly, the lemma we give provides bounds on the probability weight of negative examples in a particular *subspace* of the example space. In order to describe this subspace further, we introduce the following terminology.

Recall that for an MDNF formula $g = t_1^g + \dots + t_s^g$, we define the *greatest common factor* of g (denoted $\text{GCF}(g)$) to be the largest term t that is contained in every term t_i^g for $1 \leq i \leq s$. We refer to the set of examples satisfying t as the subspace defined by t . Note that the subspace defined by t may contain both

positive and negative examples. In particular, it contains all examples satisfying g_t , the restriction of g to t , and may contain negative examples as well.

The technique we will develop in the proofs of the following lemmas is to project a set of positive examples, or a formula, onto a larger subspace. For a term v , let $PS(v)$ be the set of vectors that are zero on all $x_i \notin v$, and that range over all possible combinations of assignments to the attributes in v . We call the set $PS(v)$ the projection set for v . We can then define the projection of a set of examples. For a set X of examples and a term v , let the projection function $P_v : 2^{\{0,1\}^n} \rightarrow 2^{\{0,1\}^n}$ be defined as: $P_v(X) = \{x \oplus y | x \in X, y \in PS(v)\}$, where \oplus denotes bitwise exclusive or. Thus, the projection function $P_v(X)$ takes each example in X and maps it onto each of the $2^{|v|}$ possible combinations of assignments to the attributes in v , and leaves all other attributes unchanged. We will refer to such a projection of examples as the projection of X over v .

We will also use the projection function $P_v(g)$ with a formula g as an argument to mean the projection of the satisfying set of g onto all possible combinations of assignments to the attributes of v . Note that this is equivalent to deleting all attributes from g that occur in v . For example, $P_{x_1}(x_1x_2 + x_1x_3) = x_2 + x_3$.

Let g_i be the Boolean function consisting of all terms of g except t_i^g . Thus, $g_i = t_1^g + \dots + t_{i-1}^g + t_{i+1}^g + \dots + t_s^g$. As an example, consider the formula

$$g = x_1x_2x_3 + x_1x_2x_4 + x_1x_5x_6 \tag{8.1}$$

We then have $g_1 = x_1x_2x_4 + x_1x_5x_6$, $g_2 = x_1x_2x_3 + x_1x_5x_6$, and $g_3 = x_1x_2x_3 + x_1x_2x_4$.

In the following proofs, we will consider various properties of the product $\prod_{1 \leq i \leq s} g_i$. First, note that by the above definition,

$$\prod_{1 \leq i \leq s} g_i = \prod_{1 \leq i \leq s} \sum_{j \neq i} t_j^g = (t_2^g + t_3^g + \dots + t_s^g) \cdot (t_1^g + t_3^g + \dots + t_s^g) \cdot \dots \tag{8.2}$$

We prove the following properties showing the relationship between the functions g_i and projections over factors of g .

Claim 8.4 *For an MDNF formula $g = t_1^g + \dots + t_s^g$, let $g_i = \sum_{j \neq i} t_j^g$. Then $\prod_{1 \leq i \leq s} g_i = \sum_{1 \leq i < j \leq s} t_i^g \cdot t_j^g$.*

Proof: First, we show that each term in $\sum_{1 \leq i < j \leq s} t_i^g \cdot t_j^g$ is generated by $\prod_{1 \leq i \leq s} g_i$.

Suppose that term t_i^g is satisfied. All formulas g_j , $j \neq i$ contain t_i^g : hence $\prod_{1 \leq i \leq s} g_i$ is satisfied if g_i is. Arguing in this manner for each i , we get the formula $t_1^g \cdot g_1 + t_2^g \cdot g_2 + \dots + t_s^g \cdot g_s$. Expanding each g_i gives $t_1^g \cdot (t_2^g + t_3^g + \dots + t_s^g) + t_2^g \cdot (t_1^g + t_3^g + \dots + t_s^g) + \dots + t_s^g \cdot (t_1^g + t_2^g + \dots + t_{s-1}^g)$, which is $\sum_{1 \leq i < j \leq s} t_i^g \cdot t_j^g$. We have thus shown that every term in $\sum_{1 \leq i < j \leq s} t_i^g \cdot t_j^g$ is generated by $\prod_{1 \leq i \leq s} g_i$ and it follows that

$$\prod_{1 \leq i \leq s} g_i = t_1^g \cdot g_1 + t_2^g \cdot g_2 + \dots + t_s^g \cdot g_s = \sum_{1 \leq i < j \leq s} t_i^g \cdot t_j^g. \quad (8.3)$$

■

As an illustration of Claim 8.4, we continue with the example from (8.1). We then get

$$\begin{aligned} \prod_{1 \leq i \leq 3} g_i &= g_1 \cdot g_2 \cdot g_3 \\ &= (x_1 x_2 x_4 + x_1 x_5 x_6) \cdot (x_1 x_2 x_3 + x_1 x_5 x_6) \cdot (x_1 x_2 x_3 + x_1 x_2 x_4) \\ &= x_1 x_2 x_4 \cdot x_1 x_2 x_3 + x_1 x_5 x_6 \cdot x_1 x_2 x_3 + x_1 x_5 x_6 \cdot x_1 x_2 x_4. \end{aligned} \quad (8.4)$$

Claim 8.5 *For an MDNF formula $g = t_1^g + \dots + t_s^g$ let $g_i = \sum_{j \neq i} t_j^g$. Then for any term t , $\prod_{1 \leq i \leq s} P_t(g_i) = P_t(\prod_{1 \leq i \leq s} g_i)$.*

The proof of Claim 8.5 follows from Claim 8.4 and the definition of the projection function.

We continue our example above to illustrate Claim 8.5. Consider the example from (8.1). and let $t = x_1$. In (8.4). we have $\prod_{1 \leq i \leq s} g_i = x_1 x_2 x_4 \cdot x_1 x_2 x_3 + x_1 x_5 x_6 \cdot x_1 x_2 x_3 + x_1 x_5 x_6 \cdot x_1 x_2 x_4 = x_1 x_2 x_3 x_4 + x_1 x_2 x_3 x_5 x_6 + x_1 x_2 x_4 x_5 x_6$. Now, taking the projection over t . we get $P_t \left(\prod_{1 \leq i \leq s} g_i \right) = x_2 x_3 x_4 + x_2 x_3 x_5 x_6 + x_2 x_4 x_5 x_6$. Taking the projection over each g_i before taking their product. we get $P_t(g_1) = x_2 x_4 + x_5 x_6$. $P_t(g_2) = x_2 x_3 + x_5 x_6$. and $P_t(g_3) = x_2 x_3 + x_2 x_4$, and taking their product gives $\prod_{1 \leq i \leq s} P_t(g_i) = (x_2 x_4 + x_5 x_6) \cdot (x_2 x_3 + x_5 x_6) \cdot (x_2 x_3 + x_2 x_4) = x_2 x_3 x_4 + x_2 x_3 x_5 x_6 + x_2 x_4 x_5 x_6$. Thus, we have $\prod_{1 \leq i \leq s} P_t(g_i) = P_t(\prod_{1 \leq i \leq s} g_i)$ for our example. as in Claim 8.5.

Lemma 8.6 (Projection Lemma) *For a read-once factorable MDNF formula $g = t_1^g + \dots + t_s^g$ with maximal factor set F . let $g_i = \sum_{j \neq i} t_j^g$. Then for any maximal factor $t \in F$. $\prod_{\{t_i^g | t \subseteq t_i^g\}} P_{t_i^g}(g_i) = \prod_{\{t_i^g | t \subseteq t_i^g\}} P_t(g_i)$.*

For the proof of this lemma. we require the following additional terminology.

Let $R_i = \{j | 1 \leq j \leq s \text{ and } j \neq i\}$ denote the set of all integers from 1 to s except i . We can then denote (8.2) by

$$\prod_{1 \leq i \leq s} g_i = \sum_{(j_1, j_2, \dots, j_s) \in R_1 \times R_2 \times \dots \times R_s} \prod_{1 \leq i \leq s} t_{j_i}^g. \quad (8.5)$$

Proof: [of Lemma 8.6] We will first show that every term in $\prod_{\{t_i^g | t \subseteq t_i^g\}} P_t(g_i)$ is also contained in $\prod_{\{t_i^g | t \subseteq t_i^g\}} P_{t_i^g}(g_i)$. Taking the projection of (8.3) over t . and applying Claim 8.5. $\prod_{\{t_i^g | t \subseteq t_i^g\}} P_t(g_i)$ can be expressed as:

$$\prod_{\{t_i^g | t \subseteq t_i^g\}} P_t(g_i) = \sum_{\{t_i^g | t \subseteq t_i^g\}} P_t(t_i^g) \cdot P_t(g_i).$$

We will show that each $P_t(t_i^g) \cdot P_t(g_i)$ is generated by $\prod_{\{t_i^g | t \subseteq t_i^g\}} P_{t_i^g}(g_i)$.

Applying (8.5) to the terms containing t , and projecting over t_i^g , we get

$$\prod_{\{t_i^g | t \subseteq t_i^g\}} P_{t_i^g}(g_i) = \sum_{(j_1, j_2, \dots, j_s) \in R_1 \times R_2 \times \dots \times R_s, \{t_i^g | t \subseteq t_i^g\}} \prod P_{t_i^g}(t_{j_i}^g). \quad (8.6)$$

For any k such that factor $t \subseteq t_k^g$, consider $P_{t_k^g}(g_k) \cdot \prod_{\{j | t \subseteq t_j^g, j \neq k\}} P_{t_j^g}(t_k^g)$. This is the term of (8.6) in which j_k varies over R_k , and all other j_i are equal.

We claim that $\prod_{\{j | t \subseteq t_j^g, j \neq k\}} P_{t_j^g}(t_k^g) = P_t(t_k^g)$. This follows from the maximality of t , since t is a factor of t_j^g , and for every attribute x in $t_j^g - t$, there exists some term not containing x . Thus, if $x \in t$, x does not occur in $\prod_{\{j | t \subseteq t_j^g, j \neq k\}} P_{t_j^g}(t_k^g)$, but if x occurs in t_k^g but not in t_j^g for some $j \neq k$, then x will occur in $\prod_{\{j | t \subseteq t_j^g, j \neq k\}} P_{t_j^g}(t_k^g)$. Thus, $\prod_{\{j | t \subseteq t_j^g, j \neq k\}} P_{t_j^g}(t_k^g) = P_t(t_k^g)$.

Now, we have $P_{t_k^g}(g_k) \cdot \prod_{\{t_i^g | t \subseteq t_i^g\}} P_{t_j^g}(t_k^g) = P_{t_k^g}(g_k) \cdot P_t(t_k^g)$. But this is equivalent to $P_t(g_k) \cdot P_t(t_k^g)$, and we have completed the proof that every term of $P_t(t_i^g) \cdot P_t(g_i)$ is generated by $\prod_{\{t_i^g | t \subseteq t_i^g\}} P_{t_i^g}(g_i)$.

We now show that $\prod_{\{t_i^g | t \subseteq t_i^g\}} P_{t_i^g}(g_i)$ does not generate any term not in $P_t(t_i^g) \cdot P_t(g_i)$. In the above, we considered only the terms generated by $P_{t_k^g}(g_k) \cdot \prod_{\{j | t \subseteq t_j^g, j \neq k\}} P_{t_j^g}(t_k^g)$, which are terms in (8.6) where j_k varies over R_k , and all other j_i are equal: but $\prod_{\{t_i^g | t \subseteq t_i^g\}} P_{t_i^g}(g_i)$ also contains terms for which the j_i are not all equal. Consider $\prod_{\{j | t \subseteq t_j^g, j \neq k\}} P_{t_j^g}(t_{\ell_j}^g)$, where the ℓ_j may be any index. We claim that this product must contain $P_t(t_{\ell_p}^g t_{\ell_q}^g)$ for some indices ℓ_p and ℓ_q . (i.e., the projection over t of the product of two terms). Take the two terms with the smallest factors. One of these must have factor t , and no larger factor. Call this ℓ_1 , and the other ℓ_2 . Now, $P_{\ell_1}(t_{\ell_p}^g) P_{\ell_2}(t_{\ell_q}^g) = P_t(t_{\ell_p}^g t_{\ell_q}^g)$, for any ℓ_p and ℓ_q . But this term is redundant in $\prod_{\{t_i^g | t \subseteq t_i^g\}} P_{t_i^g}(g_i)$, hence there are no terms generated that are not contained in $\prod_{\{t_i^g | t \subseteq t_i^g\}} P_{t_i^g}(g_i)$. This completes

the proof that the terms of $P_t(t_i^g) \cdot P_t(g_i)$ are the only terms generated by $\prod_{\{t_i^g | t \subseteq t_i^g\}} P_{t_i^g}(g_i)$. ■

We finish with our example from (8.1) by demonstrating Lemma 8.6. Note that the formula of (8.1) is a read-once factorable MDNF, with factorization $g = x_1(x_2(x_3 + x_4) + x_5x_6)$. Thus, x_1 is a factor of g . Taking the projection of each g_i over t_i^g , we get $P_{t_1^g}(g_1) = x_4 + x_5x_6$, $P_{t_2^g}(g_2) = x_3 + x_5x_6$ and $P_{t_3^g}(g_3) = x_2x_3 + x_2x_4$. Now, taking the product of these projections over all i gives us $\prod_{1 \leq i \leq s} P_{t_i^g}(g_i) = (x_4 + x_5x_6) \cdot (x_3 + x_5x_6) \cdot (x_2x_3 + x_2x_4) = x_2x_3x_4 + x_2x_3x_5x_6 + x_2x_4x_5x_6$. We thus have that $\prod_{1 \leq i \leq s} P_{t_i^g}(g_i) = \prod_{1 \leq i \leq s} P_t(g_i)$.

We now state the main result of this section, which will be instrumental in proving the learnability of read-once factorable MDNF.

Lemma 8.7 (Diffraction Lemma) *For a read-once factorable MDNF formula $g = t_1^g + \dots + t_s^g$ with terms of size at most $\lg \frac{2s}{\epsilon}$ and with maximal factor set F , and for the uniform distribution D on examples of g , let $F' = \{t \in F \mid \text{every } t \subseteq t_i^g \text{ has } \Pr_D[\mathcal{DS}(t_i^g)] < \frac{\epsilon^2}{4s^2}, \text{ and } \Pr_D[\mathcal{DS}(t)] \geq \frac{\epsilon^2}{4s^2}\}$ be the set of maximal factors such that every term t_i^g containing a factor in F' has $\Pr_D[\mathcal{DS}(t_i^g)] \leq \frac{\epsilon^2}{4s^2}$. Then $\Pr_D[(\sum_{t \in F'} t) \Delta (\sum_{t \in F'} (\sum_{\{t_i^g | t \subseteq t_i^g\}} t_i^g))] \leq \frac{\epsilon}{2}$.*

Lemma 8.7 states that the terms in formula g that all have small disjoint probability weight can be approximated by a factor from the maximal factor set F , with error bounded by $\frac{\epsilon}{2}$. We call this lemma the diffraction lemma because it shows that if the “power” of the formula is concentrated in the subspaces of the common factors, and then it “diffracts” over orthogonal sub-spaces, then these subspaces cannot capture much of the negative example space.

Proof: Recall that $\mathcal{DS}(t_i^g)$ is the set of vectors that satisfy term t_i^g but no other term. The idea of the proof is to take each vector that satisfies exactly one term (i.e., is in $\mathcal{DS}(t_i^g)$) and to project it onto a polynomial number of negative examples. We show that if we restrict the space to vectors satisfying one or more of the common factors of formula g , then the projection covers all negative examples in the restricted space. More specifically, we show that all negative examples in the subspace restricted to satisfying one of more of the common factors of g are in $\cup_{1 \leq i \leq s} P_{t_i^g}(\mathcal{DS}(t_i^g))$.

The set of vectors $\mathcal{DS}(t_i^g)$ (i.e., the set satisfying t_i^g , but not satisfying any other term) is the satisfying set of the formula $t_i^g \cdot \bar{g}_i$. The projection $P_{t_i^g}(\mathcal{DS}(t_i^g))$ is the mapping of vectors in $\mathcal{DS}(t_i^g)$ onto all possible combinations of values over the attributes in t_i^g . Let g'_i be the formula obtained from g_i by deleting all attributes that occur in t_i^g . It is easy to verify that the projection $P_{t_i^g}(\mathcal{DS}(t_i^g))$ is the set of examples satisfying formula \bar{g}'_i . Thus, $\cup_{1 \leq i \leq s} P_{t_i^g}(\mathcal{DS}(t_i^g))$ is the set of examples satisfying formula $\bar{g}'_1 + \bar{g}'_2 + \dots + \bar{g}'_s$. Now, any negative example of g does not satisfy g , by definition. We can thus construct a formula that is satisfied by the set of negative examples not in the set $\cup_{1 \leq i \leq s} P_{t_i^g}(\mathcal{DS}(t_i^g))$ as follows:

$$\overline{\bar{g}'_1 + \bar{g}'_2 + \dots + \bar{g}'_s + g} = g'_1 \cdot g'_2 \cdot \dots \cdot g'_s \cdot \bar{g} = g'_1 \cdot g'_2 \cdot \dots \cdot g'_s \cdot \bar{t}_1^g \cdot \dots \cdot \bar{t}_s^g \quad (8.7)$$

Since $g'_i = P_{t_i^g}(g_i)$, by Lemma 8.6, we have $\prod_{1 \leq i \leq s} g'_i = \prod_{1 \leq i \leq s} P_{t_i^g}(g_i) = \prod_{t \in F} \prod_{\{t_i^g | t \subseteq t_i^g\}} P_{t_i^g}(g_i) = \prod_{t \in F} \prod_{\{t_i^g | t \subseteq t_i^g\}} P_t(g_i)$. Restricting this product to the space $\sum_{t \in F} t$, we get $(\sum_{t \in F} t) \cdot \prod_{t \in F} \prod_{\{t_i^g | t \subseteq t_i^g\}} P_t(g_i)$. But then $P_t(g_i)$ is restricted to t , giving $\prod_{\{t_i^g | t \subseteq t_i^g\}} g_i$. From (8.7), this would require that $\prod_{\{t_i^g | t \subseteq t_i^g\}} g_i \cdot \prod_{1 \leq i \leq s} \bar{t}_i^g$ be satisfiable, which is not possible. Since this formula represents all negative examples in the subspace defined by $\sum_{t \in F} t$ that are not

in the set $\cup_{1 \leq i \leq s} P_{t_i^g}(\mathcal{DS}(t_i^g))$, this implies that there is no negative example on this subspace that is not covered by the set $\cup_{1 \leq i \leq s} P_{t_i^g}(\mathcal{DS}(t_i^g))$.

We have now established that every negative example that satisfies $\sum_{t \in F} t$ must belong to the set $\cup_{1 \leq i \leq s} P_{t_i^g}(\mathcal{DS}(t_i^g))$. Now, we take the subset F' of F such that $F' = \{t \in F \mid \Pr_D[\mathcal{DS}(t_i^g)] \leq \frac{\epsilon^2}{4s^2} \text{ for all } t \subseteq t_i^g\}$. The probability weight of each set $P_{t_i^g}(\mathcal{DS}(t_i^g))$ is bounded by $\frac{\epsilon^2}{4s^2} \cdot 2^{\log \frac{2s}{\epsilon}} = \frac{\epsilon}{2s}$, since by assumption $\Pr_D[\mathcal{DS}(t_i^g)] \leq \frac{\epsilon^2}{4s^2}$, and the size of the projection set $\text{PS}(t_i^g)$ is at most $2^{\log \frac{2s}{\epsilon}}$. Thus, the probability weight of the set $\cup_{1 \leq i \leq s} P_{t_i^g}(\mathcal{DS}(t_i^g))$ is at most $\frac{\epsilon}{2}$. Since all negative examples satisfying $\sum_{t \in F'} t$ are contained in $\cup_{1 \leq i \leq s} P_{t_i^g}(\mathcal{DS}(t_i^g))$, we have that $\Pr_D[(\sum_{t \in F'} t) \Delta (\sum_{t \in F'} (\sum_{\{t_i^g \mid t \subseteq t_i^g\}} t_i^g))] \leq \frac{\epsilon}{2}$. ■

The bounds derived in Lemma 8.7 are weaker by a factor of $\frac{1}{s}$ than for the specialized case of read-once MDNF given in Lemma 8.2. Lemma 8.7 is stronger than Lemma 8.3 in that it applies to the larger class of read-once factorable MDNF.

In the following chapter, we give an algorithm for learning poly-disjoint one-read-once MDNF. We will apply Lemma 8.7 in the analysis of read-once factorable MDNF to obtain a learning algorithm for that class. We also apply Lemma 8.2 to obtain a tighter analysis for the special case of read-once MDNF. For this result, we show that for one-read-once MDNF, the positive Fourier coefficient, $\hat{f}^p(t)$ (defined in Section 7.3.3), of a term t is related to the probability of the disjoint set of examples, $\mathcal{DS}(t)$: thus by finding all terms with $\hat{f}^p(t) \geq \frac{\epsilon^2}{4s^2}$, we find the terms with $\mathcal{DS}(t) \geq \frac{\epsilon^2}{4s^2}$.

Chapter 9

Learning Sub-Classes of MDNF Formulas

In this chapter we use the lemmas proven in Chapter 8 to develop an algorithm for learning read-once factorable MDNF formulas on the uniform distribution. We will begin, however, by proving the result for read-once MDNF formulas using Lemma 8.2, for which tighter bounds on sample complexity can be shown than for the more general class of read-once factorable MDNF. We then give an algorithm for learning the class of poly-disjoint one-read-once MDNF formulas, and then show that this algorithm is also a learning algorithm for read-once factorable MDNF. These results appear in [Ver 98].

9.1 Learning Read-Once MDNF

There are several algorithms in the literature for learning read-once MDNF formulas on the uniform distribution. The first such result was due to Kearns, Li, Pitt and

Valiant [KLPV 87, KLV 94, Ke 93]. They showed that the terms of a read-once formula can be constructed by drawing a polynomial sized set of *negative* examples, and estimating the pairwise correlation of 0's between every pair of attributes x_i and x_j . The complexity of their algorithm is $O\left(\left(\frac{n}{\epsilon}\right)^8 r^2\right)$, where r is the number of "significant" attributes in the formula.

In a more recent work, Hancock [H 92] gave a result for learning read- k MDNF (also known as $k\mu$ -DNF) by constructing a set of "blocking sets", and using these to predict the value of the formula. The worst case running time of Hancock's algorithm is $O\left((2n)^{k+1} \frac{k}{\epsilon^2} \left(\frac{kn}{\epsilon}\right)^{2k} \log \frac{n}{\epsilon}\right)$. Applying this algorithm for the case where $k = 1$, we get a time complexity of $O\left(\left(\frac{n}{\epsilon}\right)^4 \log \frac{n}{\epsilon}\right)$.

The algorithms we give in this chapter for read-once MDNF, poly-disjoint one-read-once MDNF, and read-once factorable MDNF will begin by drawing examples of the formula from the uniform distribution, which we will refer to as D_0 . After having learned a term of the target formula on D_0 , the algorithm will then filter examples on distribution D_0 to produce a new distribution, D_1 . In general, the algorithm will learn the i th term of the target formula on distribution D_{i-1} .

Let D_0 be the initial (uniform) distribution on the examples. After each term of the formula is found, it is added to the hypothesis h . Let h_i denote the hypothesis h after the i^{th} term has been added. Let D_i be the uniform distribution on examples not satisfying any term of h_i .

In this section, we give an algorithm for learning read-once MDNF using Lemma 8.2 of the previous chapter. We include this result here, because the complexity of our algorithm is a lower order polynomial than for the algorithm of [KLV 94], and is of a similar order to that of [H 92]. The complexity of our algorithm is in a sense incomparable to these, however, because our bound is polynomial in the

size s of the formula (where s is the number of terms) rather than the domain size n . Our second motivation for presenting the special case of read-once MDNF first is that the results for learning poly-disjoint one-read-once MDNF and read-once factorable MDNF are generalizations of this result (but the complexity will be slightly greater). Thus, we can achieve a more efficient algorithm for the class of read-once MDNF than for the more general classes.

Algorithm Read-once. for learning a term of a read-once MDNF formula on distribution D_i , is given in Figure 9.1. The idea of the algorithm is as follows. We begin with an empty set m . For each attribute x_i such that index i is not already in m , we estimate the positive Fourier coefficient on $\{i\} \cup m$ (see the proof of correctness for the sample size required to estimate the PFC statistic). We add the attribute with the minimal positive Fourier coefficient that is of magnitude at least $\frac{\epsilon^2}{8s}$. We continue choosing attributes according to the minimal positive Fourier coefficient statistic until either the error of the term is small enough, or no attribute x_i has a positive Fourier coefficient statistic larger than $\frac{\epsilon^2}{8s}$. We use Algorithm Read-Once($D_{i-1}, \epsilon, \delta, s$) as a subroutine to find each term of the target formula in Algorithm Learn-Read-Once(ϵ, δ, s), given in figure 9.1.

Assume without loss of generality that Algorithm Learn-Read-Once(ϵ, δ, s) learns term t_i on the i^{th} iteration. If this is not the case, we can simply re-number the terms so that it holds: thus this assumption is just for notational convenience.

Lemma 9.1 *If $g = t_1 + \dots + t_s$ is a read-once MDNF formula, then for every $1 \leq i \leq s$ and every $m \subset m(t_i)$, $E_{D_0^+}[(-1)^{|m|}\chi_m] = \Pr_{D_0^+}[\mathcal{DS}(t_i)]$.*

Proof: Note that $E_{D_0^+}[(-1)^{|m|}\chi_m] = \sum_{x \Rightarrow g} (-1)^{|m|}\chi_m(x)D_0^+(x)$, where $D_0^+(x)$ is the probability that distribution D_0^+ assigns to x . Let g_i be the formula

Algorithm **Read-Once**($D_{i-1}, \epsilon, \delta, s$)

1. Set $m = \emptyset$
2. While $e^-(m) \geq \frac{\epsilon}{2s}$
3. Choose the $\ell \notin m$ with the minimal
 $\hat{E}_{D_{i-1}^+} [(-1)^{|m \cup \{\ell\}|} \chi_{m \cup \{\ell\}}] \geq \frac{\epsilon^2}{8s}$
4. Set $m = \{\ell\} \cup m$
5. return $t(m)$

Algorithm **Learn-Read-Once**(ϵ, δ, s)

1. If $e^-(1) \leq \frac{\epsilon}{2}$ then
2. Set $h = 1$
3. Else
4. Set $h = 0$
5. While $e^+(h) \geq \frac{\epsilon}{2}$
6. $h = h + \text{Read-Once}(D_{i-1}, \epsilon, \delta, s)$
7. return h

Figure 9.1: Algorithm for Learning a Read-once MDNF Formula

consisting of all terms in g except t_i . We show that for any i . and for any set m containing j_i . $\sum_{x \Rightarrow g_i} \chi_m(x) = 0$. This holds since attribute x_{j_i} does not occur in g_i . hence any vector satisfying g_i with $x_{j_i} = 0$ also satisfies g_i if $x_{j_i} = 1$. Thus. $\sum_{x \Rightarrow g_i} (-1)^{|m|} \chi_m(x) D_0^+(x) = 0$. Note that $\mathcal{DS}(t_i)$ is the set of positive examples x such that $x \not\Rightarrow g_i$. Since by definition. $E_{D_0^+}[(-1)^{|m|} \chi_m] = \sum_{x \Rightarrow g} (-1)^{|m|} \chi_m(x) D_0^+(x)$. the Positive Fourier Coefficient of m is given by $E_{D_0^+}[(-1)^{|m|} \chi_m g] = \sum_{x \in \mathcal{DS}(t_i)} (-1)^{|m|} \chi_m(x) D_0^+(x)$. Now. $(-1)^{|m|} \chi_m(x) = 1$ for every vector in $\mathcal{DS}(t_i)$. and it follows that $E_{D_0^+}[(-1)^{|m|} \chi_m g]$. the Positive Fourier Coefficient of m . is equal to $\Pr_{D_0^+}[\mathcal{DS}(t_i)]$. ■

By the above lemma. we have shown that for all subterms of a term of the target formula. the positive Fourier coefficient statistic is equal to the size of the disjoint satisfying set of the term. We now show that for all cross-terms. the positive Fourier coefficient is negative.

Lemma 9.2 *If $g = t_1 + \dots + t_s$ is a read-once MDNF formula. then for every $1 \leq i \leq s$. every $m \subset m(t_i)$. and any $\ell \not\subseteq m(t_i)$. $E_{D_0^+}[(-1)^{|m \cup \ell|} \chi_{m \cup \ell}] \leq 0$.*

Proof: Let $D_{\mathcal{DS}(t_i)}$ be the uniform distribution on the set of vectors in $\mathcal{DS}(t_i)$.

Since the vectors in $\mathcal{DS}(t_i)$ do not satisfy any term t_j for $j \neq i$. $\Pr_{D_{\mathcal{DS}(t_i)}}[x_\ell = 0] \geq \frac{1}{2}$. It follows that $\sum_{x \in \mathcal{DS}(t_i)} (-1)^{|(m \cup \ell)|} \chi_{(m \cup \ell)}(x) D_0(x) \leq 0$. ■

We have proved Lemmas 9.1 and 9.2 above for the uniform distribution. In our algorithm for learning read-once MDNF. we will learn only the first term on the uniform distribution. and then skew the distribution by filtering in order to find subsequent terms. Recall that distribution D_i is formed by filtering examples out that satisfy the hypothesis h_{i-1} .

In the following lemma, we show that the bounds shown in the above lemmas apply to all distributions D_i . In fact, they improve on each D_i by magnifying the positive Fourier coefficient statistic. We will prove this lemma for one-read-once MDNF formulas, so that it can be used later to show learnability of poly-disjoint one-read-once MDNF.

Lemma 9.3 *Let $\{x_{i_1}, \dots, x_{i_s}\}$ be the read-once attributes in terms t_1, \dots, t_s , respectively. For $j \geq i$, and any m containing some attribute in $\{x_{i_1}, \dots, x_{i_s}\}$, we have $E_{D_{i-1}^+}[(-1)^{|m|}\chi_m] \geq cE_{D_0^+}[(-1)^{|m|}\chi_m]$ for some $c \geq 1$.*

Proof: Distribution D_i is formed by filtering out all examples that satisfy the hypothesis h_{i-1} . Since no term in h_{i-1} contains any attribute in $\{x_{i_1}, \dots, x_{i_s}\}$, $\sum_{x \Rightarrow h_{i-1}} \chi_m(x) = 0$ for any m containing an attribute in $\{x_{i_1}, \dots, x_{i_s}\}$, and it follows that $\sum_{x \Rightarrow h_{i-1}} (-1)^{|m|}\chi_m(x) = 0$. Thus, $\sum_{x \Rightarrow g, x \not\Rightarrow h_{i-1}} (-1)^{|m|}\chi_m(x) = \sum_{x \Rightarrow g} (-1)^{|m|}\chi_m(x)$. Now, $E_{D_0^+}[(-1)^{|m|}\chi_m] = \frac{1}{|S(g)|} \sum_{x \Rightarrow g} (-1)^{|m|}\chi_m(x)$, where $|S(g)|$ is the number of examples satisfying g . The expectation on distribution D_i is then $E_{D_{i-1}^+}[(-1)^{|m|}\chi_m] = \frac{1}{|S(g)| - |S(h_{i-1})|} \sum_{x \Rightarrow g} (-1)^{|m|}\chi_m(x)$, where $S(h_{i-1})$ is the set of examples satisfying h_{i-1} . We then have $E_{D_{i-1}^+}[(-1)^{|m|}\chi_m] = cE_{D_0^+}[(-1)^{|m|}\chi_m]$ for $c = \frac{|S(g)|}{|S(g)| - |S(h_{i-1})|}$. ■

We can now state the following theorem.

Theorem 9.4 *Algorithm Learn-Read-Once is a learning algorithm for the class of read-once MDNF formulas, with time complexity $O(n^{\frac{3}{\epsilon^4}} \log \frac{2}{\epsilon} (\log \frac{2n}{\delta} + \log \log \frac{2}{\epsilon}))$.*

Proof: By Fact 8.1, any MDNF formula f can be approximated by a formula g with terms of size at most $\lg \frac{2s}{\epsilon}$, with error bounded by $\frac{\epsilon}{2}$. By Lemma

8.2. if any term t_i of g has $\Pr[\mathcal{DS}(t_i)] < \frac{\epsilon^2}{4s}$, then g can be approximated by the constant function $h = 1$, with error bounded by $\frac{\epsilon}{2}$, and thus $e^-(1) \leq \epsilon$. If this is the case, Algorithm Learn-Read-Once will with high probability terminate and return the hypothesis $h = 1$. Otherwise, every term t_i must have $\Pr[\mathcal{DS}(t_i)] \geq \frac{\epsilon^2}{4s}$, which implies that $\Pr_{D^+}[\mathcal{DS}(t_i)] \geq \frac{\epsilon^2}{4s}$.

By Lemmas 9.1 and 9.3, it follows that the positive Fourier coefficient, defined as $E_{D_{i-1}^+}[(-1)^{|m \cup \{\ell\}|} \chi_{m \cup \{\ell\}}]$ will have a value of at least $\frac{\epsilon^2}{4s}$ for all subterms of the target formula, and by Lemma 9.2, a value less than zero for all cross-terms. We estimate the value of the coefficient for each attribute, and choose the attribute whose positive Fourier coefficient is estimated to be at least $\frac{\epsilon^2}{8s}$. Using a standard Chernoff bound argument [C 52, AV 79], by drawing a sample of size $O(\frac{s^2}{\epsilon^4}(\log \frac{sn}{\delta} + \log \log \frac{s}{\epsilon}))$ the probability that any attribute x_ℓ with $E_{D_{i-1}^+}[(-1)^{|m \cup \{\ell\}|} \chi_{m \cup \{\ell\}}] \geq \frac{\epsilon^2}{4s}$ has $\hat{E}_{D_{i-1}^+}[(-1)^{|m \cup \{\ell\}|} \chi_{m \cup \{\ell\}}] < \frac{\epsilon^2}{8s}$, or that any attribute with $E_{D_{i-1}^+}[(-1)^{|m \cup \{\ell\}|} \chi_{m \cup \{\ell\}}] < 0$ has $\hat{E}_{D_{i-1}^+}[(-1)^{|m \cup \{\ell\}|} \chi_{m \cup \{\ell\}}] > \frac{\epsilon^2}{8s}$ is bounded by $O(\frac{\delta}{sn \log \frac{s}{\epsilon}})$.

Thus, each execution of step 3 in Algorithm Read-once of Figure 9.1 will with high probability select an attribute x_ℓ in the same term as m . The i^{th} call to Algorithm Read-Once(D_i, ϵ, δ, s) will therefore output term t_i of g , which by Fact 8.1 is a subterm of f with $e^-(t_i) \leq \frac{\epsilon}{2s}$.

Each time we choose an attribute to add to the hypothesis, we need to test the positive Fourier coefficient statistic on each of the n attributes of the learning domain. Since each term has at most $\log \frac{2s}{\epsilon}$ attributes, we draw $s \log \frac{2s}{\epsilon}$ such samples, giving a total complexity of $O(n \frac{s^3}{\epsilon^4} \log \frac{s}{\epsilon} (\log \frac{sn}{\delta} + \log \log \frac{s}{\epsilon}))$.

■

9.2 Learning Poly-disjoint One-read-once MDNF

In the previous section, we showed that read-once MDNF is learnable on the uniform distribution. In this section, we will adapt the algorithm for read-once MDNF slightly, and prove that it is a learning algorithm for poly-disjoint one-read-once MDNF, where only one attribute in each term is required to be read-once.

The primary difference in Algorithm One-Read-Once is that the threshold on the positive Fourier coefficient is $\frac{\epsilon^2}{8s^2}$, compared to $\frac{\epsilon^2}{8s}$ in Algorithm Read-Once. Also, here the minimality of choice in step three (see Figure 9.2) is essential, because it ensures that the read-once attribute in each term is the first one found. In Algorithm Read-Once of Figure 9.1, minimality was not essential: we could as well have chosen any index for which the positive Fourier coefficient exceeded the threshold.

For this result, we show that for one-read-once MDNF formulas, the positive Fourier coefficient, $\hat{f}^p(t)$ (defined in Section 7.3.3), of a term t is related to the probability of the disjoint set of examples, $\mathcal{DS}(t)$: thus by finding all terms with $\hat{f}^p(t) \geq \frac{\epsilon^2}{4s^2}$, we find the terms with $\Pr_D[\mathcal{DS}(t)] \geq \frac{\epsilon^2}{4s^2}$. By doing so, we obtain a learning algorithm for poly-disjoint one-read-once MDNF, for polynomial $p = \frac{\epsilon^2}{4s^2}$.

The algorithm we give in this section for learning poly-disjoint one-read-once MDNF begins by drawing examples of the formula from the uniform distribution, which we will refer to as D_0 . After having learned a term of the target formula on D_0 , the algorithm will then filter examples on distribution D_0 to produce a new distribution, D_1 . In general, the algorithm will learn the i th term of the target formula on distribution D_{i-1} .

Let D_0 be the initial (uniform) distribution on the examples. After each term of the formula is found, it is added to the hypothesis h . Let h_i denote the hypothesis h after the i^{th} term has been added. Let D_i be the uniform distribution on examples not satisfying any term of h_i .

The algorithm for learning a term of a poly-disjoint one-read-once MDNF formula on distribution D_{i-1} is given in Figure 9.2. The idea of the algorithm is as follows. We begin with an empty set m . For each attribute x_i such that index i is not already in m , we estimate the Positive Fourier Coefficient on $\{i\} \cup m$ ¹. We add the attribute with the minimal Positive Fourier Coefficient that is of magnitude at least $\frac{\epsilon^2}{8s^2}$. We continue choosing attributes according to this statistic until either the error of the term is small enough, or no attribute x_i has a Positive Fourier Coefficient statistic larger than $\frac{\epsilon^2}{8s^2}$. We use Algorithm 1-Read-1($D_{i-1}, \epsilon, \delta, s$) as a subroutine to find each term of the target formula in Algorithm Learn-1-Read-1(ϵ, δ, s), given in Figure 9.2.

In the following lemma, we show that every term of a one-read-once MDNF formula has a positive PFC statistic. Let $\{x_{j_1} \dots x_{j_s}\}$ be the read-once attributes in terms t_1, \dots, t_s respectively.

Lemma 9.5 *If $g = t_1 + \dots + t_s$ is a one-read-once MDNF formula with read-once attributes $\{x_{j_1} \dots x_{j_s}\}$ in terms t_1, \dots, t_s respectively, then for every $1 \leq i \leq s$ and every $m \subset m(t_i)$ such that $j_i \in m$, $E_{D_0^+}[(-1)^{|m|} \chi_m] = \Pr_{D^+}[\mathcal{DS}(t_i)]$.*

Proof: Note that in the proof Lemma 9.1, we let j be the index of any attribute in term t_i of a read-once MDNF formula, and used the fact that x_j occurs in no other term to obtain the resulting lemma. Now, let j be the index, j_i , of the read-once attribute in term t_i , and the proof follows as for Lemma 9.1. ■

¹See the proof of Theorem 9.7 for the sample complexity required to estimate this statistic.

Algorithm 1-Read-1($D_{i-1}, \epsilon, \delta, s$)

1. Set $m = \emptyset$
2. While $e^-(m) \geq \frac{\epsilon}{2s}$
3. Choose the $\ell \notin m$ with the minimal
 $\hat{E}_{D_{i-1}}^+ [(-1)^{|m \cup \{\ell\}|} \chi_{m \cup \{\ell\}}] \geq \frac{\epsilon^2}{8s^2}$
4. Set $m = \{\ell\} \cup m$
5. return $t(m)$

Algorithm Learn-1-Read-1(ϵ, δ, s)

1. If $e^-(1) \leq \frac{\epsilon}{2}$ then
2. Set $h = 1$
3. Else
4. Set $h = 0$
5. While $e^+(h) \geq \frac{\epsilon}{2}$
6. $h = h + 1$ -Read-1($D_{i-1}, \epsilon, \delta, s$)
7. return h

Figure 9.2: Algorithm for Learning a Poly-disjoint One-read-once MDNF Formula

We can also generalize the proof of Lemma 9.2 in a similar manner, to obtain the following lemma for one-read-once MDNF.

Lemma 9.6 *If $g = t_1 + \dots + t_s$ is a one-read-once MDNF formula with read-once attributes $\{x_{j_1} \dots x_{j_s}\}$ in terms t_1, \dots, t_s , respectively, then for every $1 \leq i \leq s$, every $m \subset m(t_i)$ such that $j_i \in m$, and any $\ell \notin m(t_i)$, $E_{D_0^+}[(-1)^{|m \cup \{\ell\}|} \chi_{m \cup \{\ell\}}] \leq 0$.*

We can now state the following theorem.

Theorem 9.7 *The class of poly-disjoint one-read-once MDNF formulas is learnable on the uniform distribution. Algorithm Learn-1-Read-1 is a learning algorithm for this class, with time complexity $O(n^{\frac{s}{\epsilon^4}} \log_{\frac{s}{\epsilon}}(\log \frac{m}{\delta} + \log \log \frac{s}{\epsilon}))$.*

Proof: By Fact 8.1, any MDNF formula f can be approximated by a formula g with terms of size at most $\lg \frac{2s}{\epsilon}$, with error bounded by $\frac{\epsilon}{2}$. Thus, we assume hereafter without loss of generality that the target formula is such a g , and show in the remainder of the proof that g can be approximated with error $\frac{\epsilon}{2}$. Consider the index ℓ chosen at the first execution of step 3 of Algorithm 1-Read-1 (see Figure 9.2). ℓ is chosen to be the index of the attribute with the minimal estimated PFC, $\hat{E}_{D_{i-1}^+}[(-1)^{|m \cup \{\ell\}|} \chi_{m \cup \{\ell\}}]$, that is at least $\frac{\epsilon^2}{8s^2}$ in magnitude.

We first argue that the first attribute chosen will, with high probability, be a read-once attribute. By definition of poly-disjoint, each term t_i^g in the formula g has $\Pr[\mathcal{DS}(t_i^g)] \geq \frac{\epsilon^2}{4s^2}$. In particular, the read-once attribute x_{j_i} must have $E_{D_{i-1}^+}[(-1)^{|j_i|} \chi_{\{j_i\}}] \geq \frac{\epsilon^2}{4s^2}$. Any attribute x_ℓ that is not a read-once attribute must either occur in no term of the formula, or else occur in two or more terms. If x_ℓ is not in any term of the formula, then $E_{D_{i-1}^+}[(-1)^{|\{\ell\}|} \chi_{\{\ell\}}] = 0$.

Furthermore, if ℓ occurs in two or more terms, then since all terms are poly disjoint, its probability must be at least $\frac{\epsilon^2}{4s^2}$ greater than that of the read-once attributes of those terms. Using Chernoff bounds [C 52, AV 79], a sample of size $O(\frac{s^4}{\epsilon^4}(\log \frac{sm}{\delta} + \log \log \frac{s}{\epsilon}))$ is sufficient to ensure with probability at least $1 - \frac{\delta}{sn \log \frac{s}{\epsilon}}$ that x_ℓ occurs in exactly one term.

By Lemma 9.5, once we have a read-once attribute from term t_i^g , on each subsequent execution of step 3, $E_{D_{i-1}^+} [(-1)^{|m \cup \{\ell\}|} \chi_{m \cup \{\ell\}}] \geq \frac{\epsilon^2}{4s^2}$ if attribute ℓ is in the same term as m . If ℓ is not in the same term as m , by Lemma 9.6, $E_{D_{i-1}^+} [(-1)^{|m \cup \{\ell\}|} \chi_{m \cup \{\ell\}}] < 0$. By the Chernoff bound argument above, a sample of size $O(\frac{s^4}{\epsilon^4}(\log \frac{sm}{\delta} + \log \log \frac{s}{\epsilon}))$ is sufficient to ensure with probability at least $1 - \frac{\delta}{sn \log \frac{s}{\epsilon}}$, that ℓ is in the same term as m .

Estimating the PFC requires $O(\frac{s^4}{\epsilon^4}(\log \frac{sm}{\delta} + \log \log \frac{s}{\epsilon}))$ examples for each attribute x_ℓ . Since there are s terms in the target formula each with $\log \frac{s}{\epsilon}$ attributes, and we measure this statistic on all attributes each time we choose an attribute, the overall time and sample complexities are $O(n \frac{s^5}{\epsilon^4} \log \frac{s}{\epsilon} (\log \frac{sm}{\delta} + \log \log \frac{s}{\epsilon}))$. The probability that any attribute is chosen incorrectly is bounded by δ . ■

9.3 Learning Read-once Factorable MDNF

We now apply the Diffraction Lemma of Chapter 8 to show that the learning algorithm of the previous section (see Fig. 9.2) is also a learning algorithm for the class of read-once factorable MDNF.

Theorem 9.8 *The class of read-once factorable MDNF formulas is learnable on the uniform distribution. Algorithm Learn-1-Read-1 is a learning algorithm for this*

class, with time complexity $O(n \frac{s^5}{\epsilon^4} \log \frac{s}{\epsilon} (\log \frac{m}{\delta} + \log \log \frac{s}{\epsilon}))$.

Proof: As in the proof of the previous section, by Fact 8.1, the MDNF formula f can be approximated by a formula g with terms of size at most $\lg \frac{2s}{\epsilon}$, with error bounded by $\frac{\epsilon}{2}$, so we assume hereafter without loss of generality that the target formula is such a g , and show in the remainder of the proof that g can be approximated with error $\frac{\epsilon}{2}$.

We use Algorithm 1-Read-1 of Figure 9.2 to learn a term of the read-once factorable MDNF formula g . Consider the index ℓ chosen at step 3 of the algorithm. Either x_ℓ is a read-once attribute, or else it is an attribute from the factor of two or more terms that all have $\Pr_{D^+}[\mathcal{DS}(t_i)] < \frac{\epsilon^2}{4s^2}$.

If x_ℓ is a read-once attribute, then by the same argument as in the proof of the previous theorem, Algorithm 1-Read-1 will return term t_i . If x_ℓ is not a read-once attribute in term t_i , then it is an attribute from the factor of two or more terms that all have $\Pr_{D^+}[\mathcal{DS}(t_i)] < \frac{\epsilon^2}{4s^2}$. By Lemma 8.7, all terms t_i with factor t can be approximated by t . Since the factored form is read-once, attribute x_ℓ does not occur in any other term: thus, x_ℓ is a read-once attribute in term t . Applying Lemmas 9.5 and 9.6 as in the proof of the previous theorem, Algorithm 1-Read-1 will return the greatest common factor t . The total error incurred by all such approximations by common factors is bounded by $\frac{\epsilon}{2}$, by Lemma 8.7.

The time and sample complexities are as shown in the proof of Theorem 9.7: $O(n \frac{s^5}{\epsilon^4} \log \frac{s}{\epsilon} (\log \frac{m}{\delta} + \log \log \frac{s}{\epsilon}))$. ■

Chapter 10

Towards Learnability of Monotone DNF

In the previous chapter, we have shown that the class of one-read-once MDNF is learnable under the uniform distribution using only an example oracle. We showed that for this class of formulas, the positive Fourier coefficient statistic is positive for all subterms containing the one-read-once attribute, whereas this statistic is negative for all such cross-terms. This property does not extend to the larger class of MDNF. In Section 10.2, we will show an example of cross-term coefficients of order three that are positive. We can show some partial results, however: we show in Section 10.1 that if a second order coefficient is positive, then the attributes indexed by that coefficient occur in the same term. Furthermore, if a second order coefficient is negative, then the two attributes indexed by that coefficient must occur in different terms (although, they may occur in the same term as well.)

10.1 Second order coefficients of MDNF

We begin by showing that if a second order coefficient of an MDNF formula is positive, the attributes indexed by that coefficient must occur in the same term.

Lemma 10.1 *If $E_{D^+}[\chi_{\{i,j\}}] > 0$, then there exists some term m of f such that x_i and x_j occur in m .*

Proof: Suppose that $E_{D^+}[\chi_{\{i,j\}}] > 0$, but that x_i and x_j are not in the same term for any term. Let f' be the formula consisting of all terms of f that contain x_i . By assumption, no term in f' contains x_j . Thus, $\sum_{x \Rightarrow f'} \chi_{\{x_i, x_j\}} = 0$. Now, we claim that $\sum_{x \not\Rightarrow f'} \chi_{\{x_i, x_j\}} < 0$. Thus, we have that $\sum_{x \in \{0,1\}^n} \chi_{\{x_i, x_j\}} < 0$, hence $E_{D^+}[\chi_{\{i,j\}}] < 0$, leading to a contradiction. ■

Lemma 10.2 *If $E_{D^+}[\chi_{\{i,j\}}] < 0$, then there exist terms m_k and m_ℓ of f such that x_i is in m_k , and x_j is in m_ℓ .*

Proof: Suppose that $E_{D^+}[\chi_{\{i,j\}}] < 0$, but that x_i and x_j do not occur in different terms: then either they occur only in the same term, or else at least one of x_i and x_j does not occur in any term. If the latter is the case, $E_{D^+}[\chi_{\{i,j\}}] = 0$. In the former case, we show that $E_{D^+}[\chi_{\{i,j\}}] > 0$. Let f' be the formula consisting of terms of f that do not contain both x_i and x_j . Since by assumption x_i and x_j do not occur in different terms, neither x_i nor x_j occurs in any term of f' , thus $\sum_{x \Rightarrow f'} \chi_{\{x_i, x_j\}} = 0$. Now, all examples x that satisfy f but not f' have $x_i = x_j = 1$, so $\sum_{\substack{x \Rightarrow f \\ x \not\Rightarrow f'}} \chi_{\{x_i, x_j\}} > 0$, and it follows that $E_{D^+}[\chi_{\{i,j\}}] > 0$. All cases lead to a contradiction, completing the proof. ■

Note that Lemma 10.2 does not imply that attributes x_i and x_j *cannot* be in the same term in f . it just implies that there exist two terms of f for which x_i is in one term, and x_j is in the other.

While Lemma 10.1 shows that for second order coefficients, a positive value indicates that the attributes indexed by the coefficient are in the same term, we show in the following that this is not the case for third order coefficients. We give an example of a third order coefficient that is not a subset of a term, and show that it has a positive value.

10.2 Third order coefficients of MDNF

In the previous section we showed that if the second order coefficients of an MDNF formula are positive, then those attributes must occur in the same term. We show here that this property does not necessarily extend to third order coefficients.

Lemma 10.3 *There exist MDNF formulas, and indices i, j, k , such that the expectation $E_{D^+}[(-1)^{|\{i,j,k\}|} \chi_{\{i,j,k\}}]$ is positive, but x_i, x_j and x_k do not all occur in any term.*

Proof: Consider the formula $f(x_1, \dots, x_7) = x_1x_2 + x_3x_4 + x_5x_6$. It is easily verified that $E_{D^+}[(-1)^3 \chi_{\{1,3,5\}}] = 1$, but $x_1x_3x_5$ is not a term of f . ■

By Lemma 10.3, it may be the case that terms of size three or greater may have PFC (positive Fourier coefficient) statistics greater than zero. Thus, cross-terms may have positive PFC statistics, so we cannot apply the algorithm for one-read-once terms to such formulas. In the following section, we show that the converse may hold as well, that is, that terms of an MDNF target formula may have negative PFCs.

10.3 Degenerate coefficients on Dense Terms

In this and later sections, we will distinguish terms of a monotone DNF formula for which our one-read-once MDNF algorithm works from those for which it does not work. Without defining these notions rigorously, we will use the term *sparse* to refer to terms with positive PFC's, and *dense* to refer to terms with negative PFCs. The reason for this choice of terminology will be apparent from the following example, which shows that when many terms of the target formula contain the same attributes (hence the name dense), the PFC may be negative.

For one-read-once MDNF, we showed that the positive Fourier coefficients of all subterms of the target formula containing the read-once attribute are positive. In the following, we give an example of an MDNF formula for which subterms have negative PFCs. Thus, Algorithm One-read-once of Figure 9.2 will not find such terms.

$$\begin{aligned}
 f(x_1, \dots, x_7) = & x_1 x_2 x_3 x_4 x_5 x_6 + \\
 & x_1 x_2 x_3 x_4 x_5 x_7 + \\
 & x_1 x_2 x_3 x_4 x_6 x_7 + \\
 & x_1 x_2 x_3 x_5 x_6 x_7 + \\
 & x_1 x_2 x_4 x_5 x_6 x_7 + \\
 & x_1 x_3 x_4 x_5 x_6 x_7 + \\
 & x_2 x_3 x_4 x_5 x_6 x_7
 \end{aligned}$$

The satisfying set for the above formula is:

1111110

1111101

1111011

1110111

1101111

1011111

0111111

1111111

Note that the positive Fourier coefficient has value 2 over $x_1x_2x_3$, and value 0 over $x_1x_2x_3x_4$, so the coefficient is positive for the sub-term of length three, but is zero for the term of length four.

10.4 An Heuristic for Monotone Terms

We have shown in Lemma 10.1 that if a second order coefficient is positive, then the attributes indexed by that coefficient must be in the same term. The positive Fourier coefficient defined in Section 7.3.3 was designed with this property in mind: that it have a positive value for subterms of the target formula. We have seen an example in Section 10.3, however, where this does not hold, and the positive Fourier coefficient takes on a negative value for subterms of the target formula. Indeed, Lemma 10.2 above does not preclude this even for second order coefficients, since

two attributes may occur in the same term, and yet also be in different terms, resulting in a negative value of the positive Fourier coefficient. Thus, constructing the positive Fourier coefficient of a term will fail in these cases.

To address this issue, we consider another statistic, based on the positive Fourier coefficient, which we call the *Expected Positive Fourier Coefficient*, defined as:

$$\text{EPFC}(S) = E_{A \subseteq S}[E_{D^+}[(-1)^{|A|} \chi_A]] \tag{10.1}$$

While the positive Fourier coefficient may degenerate to take on a negative value for subterms of the target formula, we can show that the expected positive Fourier coefficient will always be positive for all subterms. The proof is a modification of a proof by Bshouty, given in [J 94]. We give the proof in the following for completeness.

Fact 10.4 *For every DNF f with s terms and for every distribution D on the instance space of f , for every term t of f such that $\Pr[t = 1] \geq \frac{\epsilon}{s}$, and for every subset $S \subseteq m(t)$, $E_{A \subseteq S}[E_{D^+}[(-1)^{|A|} \chi_A]] \geq \frac{\epsilon}{s}$.*

Proof: Consider term t as a Boolean function, such that $t(x) = 1$ if x satisfies t , and $t(x) = -1$ if x does not satisfy t . For all x ,

$$\frac{t(x) + 1}{2} = \prod_{x_i \in m(t)} \frac{1 - \chi_{\{x_i\}}(x)}{2} = E_{A \subseteq S}[(-1)^{|A|} \chi_A(x)]$$

The expectation here is uniform over all subsets $A \subseteq S$. Let $t'(x) = (t(x) + 1)/2$. Then $E_{D^+}[ft'] = E_{A \subseteq S}[(-1)^{|A|} E_{D^+}[f \chi_A]]$. Since t is a term of f , for any x such that $f(x) = -1$, $t'(x) = 0$. Thus, $E_{D^+}[ft'] = E_{D^+}[t'] = \Pr[t = 1] \geq \frac{\epsilon}{s}$. ■

While the EPFC is guaranteed to be positive for all subterms of the target formula, it may also be positive for cross-terms. We have not yet been able to characterize exactly when the EPFC is more positive for subterms of the formula than for cross-terms, but this appears to be the case for dense formulas. On the other hand, the PFC seems to work well on the sparse formulas. Thus, we propose a heuristic based on using the PFC for dense parts of the formula, and the EPFC for sparse parts of the target formula. In the following, we describe the heuristic algorithm based on the coefficient statistics discussed above.

As discussed above, the PFC works well in finding sparse terms of the target formula, and the EPFC works well in finding the dense terms. Since a target formula may contain both sparse and dense terms, we interleave calls to subroutines that search for terms of the target formula using the PFC and the EPFC statistic, respectively. The subroutines are given in Figures 10.2 and 10.3. Note that the algorithms take distribution D_{i-1} as a parameter. Distribution D_{i-1} is formed by filtering distribution D_{i-2} to damp the influence of terms that have already been discovered. The distributional filter used is an adaptation the algorithm proposed by Freund in [Fre 90]. A similar filter was used by Jackson [J 94] to learn DNF using membership queries. We discuss this filtering technique further in the next section.

10.5 Filtering Distributions

The filtering technique we use to create distributions D_0, D_1, \dots is based on hypothesis boosting algorithms introduced by Schapire [S 90], and further studied by Freund [Fre 90]. The idea behind the technique is to run a (weak) learning algorithm on distribution D_{i-1} , and then use the hypothesis produced to damp, or filter

out. examples already learned in producing distribution D_i . Distribution D_i will thus focus on an area of the probability space that has not yet been learned.

Schapire's algorithms have slightly better asymptotic performance than Freund's, but the technique developed by Freund has the advantage of simplicity, and it is this technique we employ in our algorithm. The main idea of Freund's algorithm is to run a weak learning algorithm on distribution D_{i-1} to produce a weak hypothesis h_{i-1} during stage $i-1$. Distribution D_i is constructed so that the examples are expected to satisfy roughly half of the hypotheses found in stages 0 to $i-1$.

To illustrate Freund's algorithm in more detail, we require the following definitions. Suppose that WL is a weak learning algorithm, that when run on distribution D_i produces a $(\frac{1}{2} - \gamma)$ -approximate hypothesis h_i . Let $r_i(x) = |\{0 \leq j < i \mid h_j(x) = f(x)\}|$ be the number of hypotheses from stages 0 through $i-1$ that agree with f on example x . Distribution D_i is produced by drawing an example x from the initial (uniform) distribution D_0 , and "accepting" x with probability $\alpha_{r_i(x)}^i / \max_r \{\alpha_{r_i(x)}^i\}$, where

$$\alpha_r^{k-1} = \begin{cases} 1 & \text{if } r = \lfloor \frac{k}{2} \rfloor \\ 0 & \text{otherwise} \end{cases}$$

and

$$\alpha_r^i = (\frac{1}{2} - \gamma)\alpha_r^{i+1} + (\frac{1}{2} + \gamma)\alpha_{r+1}^{i+1} \quad \text{for } 0 \leq i \leq k-2$$

Solving this recursion gives

$$\alpha_r^i = \begin{cases} \binom{k-i-1}{\lfloor \frac{k}{2} \rfloor - r} (\frac{1}{2} + \gamma)^{\lfloor \frac{k}{2} \rfloor - r} (\frac{1}{2} - \gamma)^{\lfloor \frac{k}{2} \rfloor - i - 1 + r} & \text{if } i - \lfloor \frac{k}{2} \rfloor \leq r \leq \lfloor \frac{k}{2} \rfloor \\ 0 & \text{otherwise} \end{cases}$$

Examples are repeatedly drawn in this fashion until one is accepted. Freund's algorithm consists of running the weak learning algorithm WL on each distribution

D_i to produce a hypothesis h_i . The hypothesis of the boosting algorithm at stage i is then the majority vote over the hypotheses h_j from previous stages, for $0 \leq j \leq i$. Freund proves the following lemma in [Fre 90].

Lemma 10.5 *Freund's boosting algorithm. when given a $(\frac{1}{2} - \gamma)$ -approximate learning algorithm for concept class \mathbf{C} . constants $\epsilon, \delta > 0$. and any initial distribution D_0 . will produce an ϵ -approximation of f with respect to D_0 for any $f \in \mathbf{C}$. with probability at least $1 - \delta$.*

The proof of Freund's lemma uses the quantity β_r^i , where

$$\beta_r^k = \begin{cases} 1 & \text{if } r \leq \lceil \frac{k}{2} \rceil \\ 0 & \text{otherwise} \end{cases}$$

and

$$\beta_r^i = (\frac{1}{2} - \gamma)\beta_r^{i+1} + (\frac{1}{2} + \gamma)\beta_{r+1}^{i+1} \quad \text{for } 0 \leq i \leq k - 2$$

Solving this recursion gives the tail of the binomial distribution

$$\beta_r^i = \begin{cases} 0 & \text{if } r > \lfloor \frac{k}{2} \rfloor \\ \sum_{j=0}^{\lfloor \frac{k}{2} \rfloor} \binom{k-i}{j} (\frac{1}{2} + \gamma)^j (\frac{1}{2} - \gamma)^{k-i-j} & \text{if } i - \lceil \frac{k}{2} \rceil \leq r \leq \lfloor \frac{k}{2} \rfloor \\ 1 & \text{if } r < i - \lceil \frac{k}{2} \rceil \end{cases}$$

The quantity β_r^i is essentially the expectation at stage i on the measure of examples that satisfy less than r of the hypotheses. The essential relationship between the weight function α_r^i and the expected loss β_r^i is the following:

$$\alpha_r^i = \beta_r^{i+1} - \beta_{r+1}^{i+1}$$

The weight function used in Freund's boosting algorithm sets the weights so that the probability of examples that satisfy half of the previously learned hypotheses

is boosted, thus giving preference to such examples. This weighting scheme has the undesirable property of re-discovering previous hypotheses frequently. In the following, we give an alternative weight function α_r^i that has the same essential properties as Freund's weight function, but that gives higher preference to examples that satisfy fewer hypotheses, thus preventing re-discovery of hypotheses.

10.6 Adaptation of Freund's Boosting Algorithm

We use Freund's method of distributional filtering using a majority vote over weak hypotheses in order to shift the distribution D_i towards examples that are not yet covered by the hypotheses h_i . Freund shows that the probability weighting strategy he proposes (described in Section 10.5) is the optimal weighting strategy if the example space is Euclidean, and no point in the domain has non-zero probability. This is not the case for Boolean domains. In our implementation of Freund's algorithm, we found that for a Boolean domain, the algorithm in general will oscillate by repeatedly re-discovering the same two weak hypotheses. The explanation for this is that Freund's algorithm gives an advantage to examples that satisfy half of the previous hypotheses. Thus, after finding two weak hypotheses, the probability of examples that satisfy one hypothesis but not both is magnified, and this results in re-discovering the same two hypotheses. To get around this, we have skewed the probabilities further than is done in Freund's algorithm to give more of an advantage to examples that are not yet covered by the current hypothesis. Instead of the probability weighting function defined by Freund (equation 10.5 above), we use the following:

$$\alpha_r^{k-1} = \begin{cases} 1 & \text{if } r = 0 \\ 0 & \text{otherwise} \end{cases}$$

and

$$\alpha_r^i = \left(\frac{1}{2} - \gamma\right)\alpha_r^{i+1} + \left(\frac{1}{2} + \gamma\right)\alpha_{r+1}^{i+1} \quad \text{for } 0 \leq i \leq k-2$$

Solving this recursion gives

$$\alpha_r^i = \begin{cases} \binom{k-i-1}{\lfloor \frac{k-i}{2} \rfloor - r} \left(\frac{1}{2} + \gamma\right)^{\lfloor \frac{k-i}{2} \rfloor - r} \left(\frac{1}{2} - \gamma\right)^{\lceil \frac{k-i}{2} \rceil - i - 1 + r} & \text{if } i - \lceil \frac{k-i}{2} \rceil \leq r \leq \lfloor \frac{k-i}{2} \rfloor \\ 0 & \text{otherwise} \end{cases}$$

Note that at stage $k-1$, the above weight function assign weight 1 to examples that satisfy none of the previous hypotheses, and weight 0 to all others. We then define the quantity β_r^i .

$$\beta_r^i = \begin{cases} 0 & \text{if } r > \lfloor \frac{k-i}{2} \rfloor \\ \sum_{j=0}^{\lfloor \frac{k-i}{2} \rfloor} \binom{k-i}{j} \left(\frac{1}{2} + \gamma\right)^j \left(\frac{1}{2} - \gamma\right)^{k-i-j} & \text{if } i - \lceil \frac{k-i}{2} \rceil \leq r \leq \lfloor \frac{k-i}{2} \rfloor \\ 1 & \text{if } r < i - \lceil \frac{k-i}{2} \rceil \end{cases}$$

These definitions of α and β satisfy the same recurrences as for Freund's weighting function, but with different initial conditions. In particular, we have the property:

$$\alpha_r^i = \beta_r^{i+1} - \beta_{r+1}^{i+1}$$

The proof of Freund's lemma depends on the above relations between α and β . Since our weighting function satisfies the same formulas, Freund's proofs of correctness hold. We refer the reader interested in further detail to Freund's work [Fre 90] on filtering distributions for the proofs of the correctness of the filtering technique.

We are now ready to give algorithm MDNF, as shown in Figure 10.1.

The algorithm first tests whether the target function can be ϵ -approximated by the constant function $f = 1$. If so the algorithm sets the hypothesis to the constant

Algorithm **Learn-MDNF**(ϵ, δ, s)

1. If $e^-(1) \leq \frac{\epsilon}{2}$ then
2. Set $h = 1$
3. Else
4. Set $h = 0$, and $i = 0$
5. While $e^+(h) \geq \frac{\epsilon}{2}$
6. $i = i + 1$
7. if (i is even)
8. $h = h + \text{Max-PFC}(D_{i-1}, \epsilon, \delta, s)$
9. else
10. $h = h + \text{Max-EPFC}(D_{i-1}, \epsilon, \delta, s)$
11. return h

Figure 10.1: Heuristic Algorithm for Learning MDNF

function and terminates. Otherwise, it initializes the hypothesis to the constant function 0, which is satisfied by no example, and proceeds to learn terms of the target formula by interleaving subroutine Max-PFC (for sparse terms) or Max-EPFC (for dense terms). The algorithm terminates when h is an ϵ -approximation to the target formula.

If subroutines MaxPFC and MaxEPFC of Figures 10.2 and 10.3 were weak learning algorithms for monotone DNF, then the filtering algorithm we give in Section 10.6 would give a learning algorithm for monotone DNF. Note that we have only proven that MaxPFC is a (weak) learning algorithm for the class of one-read-once MDNF formulas. We have not proven that EPFC works in learning the

Algorithm **Max-PFC**($D_{i-1}, \epsilon, \delta, s$)

1. Set $m = \emptyset$
2. While $e^-(m) \geq \frac{\epsilon}{2s}$
3. Choose the $\ell \not\subseteq m$ with the maximal
 $E_{D_{i-1}^+} [(-1)^{|m \cup \ell|} \chi_{m \cup \ell}] \geq \frac{\epsilon^2}{4s}$
4. Set $m = \ell \cup m$
5. return $t(m)$

Figure 10.2: Algorithm for Learning a Sparse Term

Algorithm **Max-EPFC**($D_{i-1}, \epsilon, \delta, s$)

1. Set $m = \emptyset$
2. While $e^-(m) \geq \frac{\epsilon}{2s}$
3. Choose the $\ell \not\subseteq m$ with the maximal
 $E_{A \subseteq S} [E_{D_{i-1}^+} [(-1)^{|m \cup \ell|} \chi_{m \cup \ell}]] \geq \frac{\epsilon^2}{4s}$
4. Set $m = \ell \cup m$
5. return $t(m)$

Figure 10.3: Algorithm for Learning a Dense Term

dense terms, but our empirical studies show that it works well in practice. Thus, if we consider MaxPFC to be a weak learning algorithm for sparse formulas and MaxEPFC to be a weak learning algorithm for dense terms, Freund's lemma shows that the distributional filter will shift the distribution so that "unlearned" examples are more frequent. We can thus focus the search for terms on the unlearned part of the probability space until a sufficiently accurate hypothesis is learned.

Chapter 11

Empirical Results

In the previous chapter, we presented an heuristic algorithm for learning MDNF formulas on the uniform distribution. In this chapter, we give some empirical results to support the use of this algorithm for learning MNDF formulas.

11.1 Overview of Results

We have run the MDNF learning algorithm on about one hundred different test cases, and in all cases tested the algorithm produces an ϵ -approximation to the target formula f . Many of these cases have been designed to contain both sparse and dense terms in the same formula. In many cases, the hypothesis also contains non-terms (cross-terms) that result from the EPFC statistic being applied to learn a sparse term. Terms of the minimal formula may also be discovered multiple times if the error parameter is very low, because the boosting algorithm will continue finding terms (perhaps the same terms) of higher probability weight until it focuses in on the area of the probability space where the smaller weight terms lie.

In this chapter, we discuss only a few of the empirical results obtained, to illustrate that the algorithms perform well at the two extremes, for formulas that are highly dense, and as well for formulas that are highly sparse.

11.1.1 Dense Formulas

The first example we consider is the maximally dense hypothesis, where the Hamming distance between any two terms is two (i.e., every term differs in two attributes from every other term.) Note that this example is highly dense, in that every attribute occurs in $s - 1$ terms, where s is the total number of terms. An example of this formula on a domain of size 7 is given below:

$$\begin{aligned}
 d(x_1, \dots, x_7) = & x_1x_2x_3x_4x_5x_6 + \\
 & x_1x_2x_3x_4x_5x_7 + \\
 & x_1x_2x_3x_4x_6x_7 + \\
 & x_1x_2x_3x_5x_6x_7 + \\
 & x_1x_2x_4x_5x_6x_7 + \\
 & x_1x_3x_4x_5x_6x_7 + \\
 & x_2x_3x_4x_5x_6x_7
 \end{aligned}$$

Let d_n denote the formula consisting of the conjunction of all possible terms on $n - 1$ variables. The above example is then d_7 .

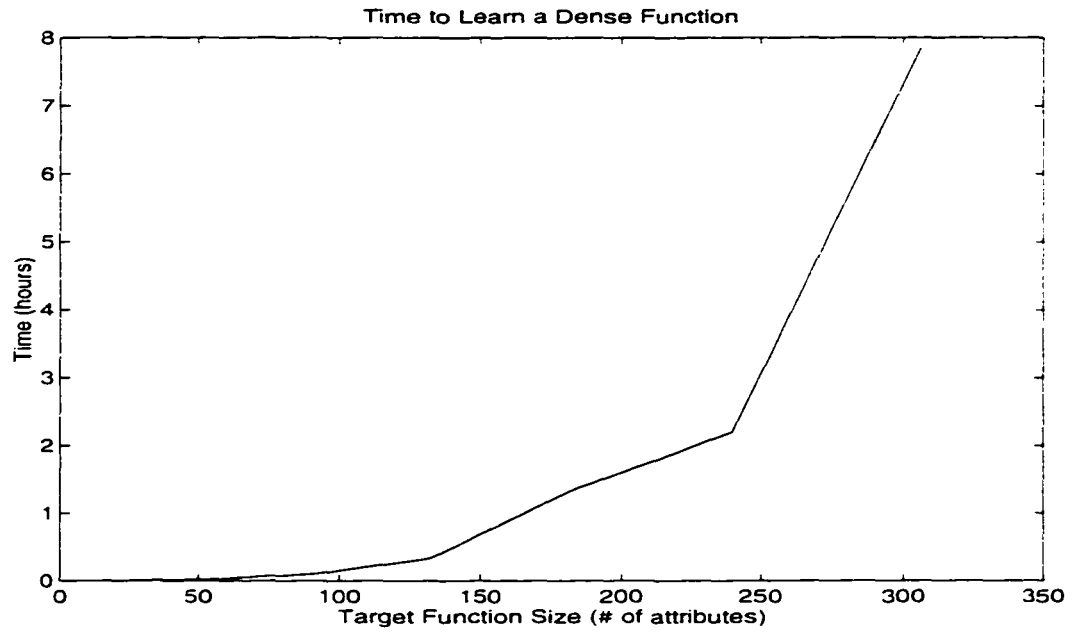


Figure 11.1: Running Time of Algorithm MDNF for a Dense Formula

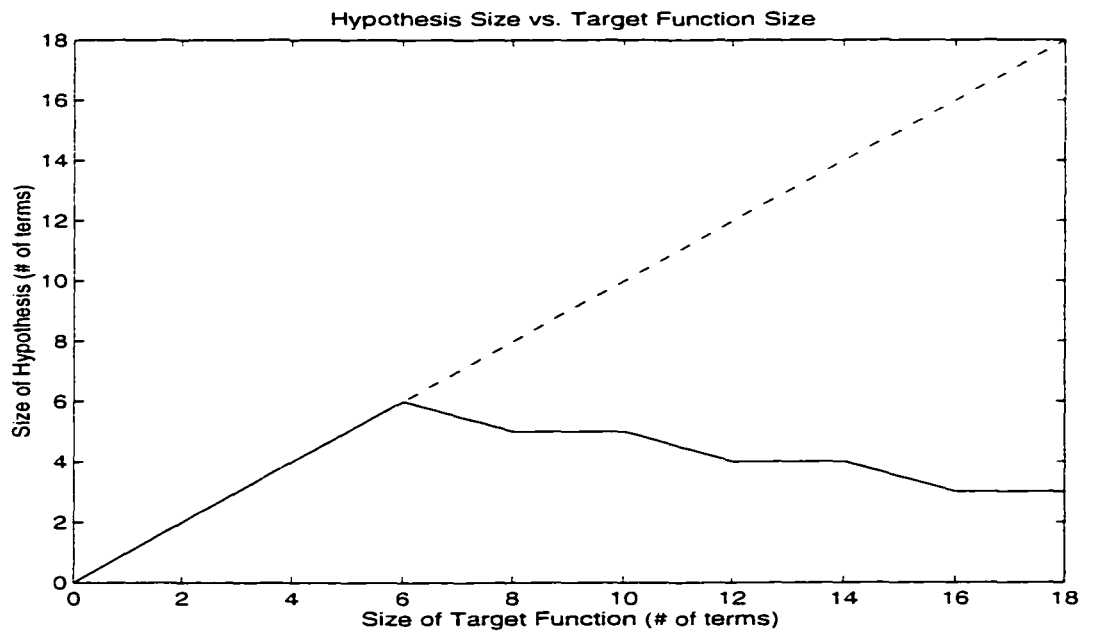


Figure 11.2: Hypothesis Size for a Dense Formula

In the graph of Figure 11.1, we show the running time of Algorithm MDNF as a function of the size of the formula (where the size here is measured by the total number of attributes occurring in the formula.) For this example, we have run Algorithm MDNF on d_4 , d_6 , d_8 , d_{10} , d_{12} , d_{14} , d_{16} and d_{18} , and with error and confidence parameters set at $\epsilon = 0.15$, and $\delta = 0.1$.

In Figure 11.2, we show the size of the hypothesis output by the algorithm, where the solid line indicates the size of the hypothesis, and the dashed line indicates the size of the target formula. In this example, the size of the hypothesis is actually less than the size of the target formula; this is because the error rate of $\epsilon = 0.15$ does not require the algorithm to learn all terms of the target formula.

The next example we consider is on a domain of 15 attributes. We randomly generate terms containing five attributes from the set $\{x_1, x_2, \dots, x_{10}\}$. There are $\binom{10}{5} = 252$ such terms possible. In Figure 11.3, we show the running time of Algorithm MDNF against the number of attributes in the target formula for this example. For this example, we ran Algorithm MDNF with $\epsilon = 0.15$, and $\delta = 0.1$.

In Figure 11.4, we show the size of the hypothesis output by Algorithm MDNF for the example of Figure 11.3. The thin solid line represents the size of the hypothesis output by Algorithm MDNF, and the dashed line indicates the size of the target function. (i.e. the size of the minimal hypothesis.)

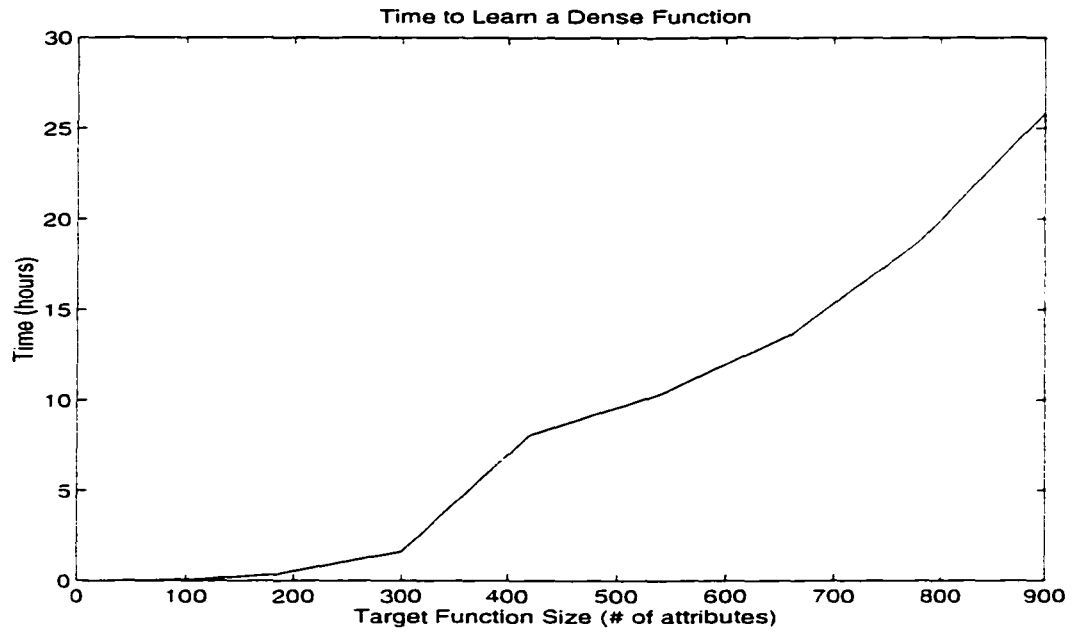


Figure 11.3: Running Time of Algorithm MDNF for a Random Dense Formula

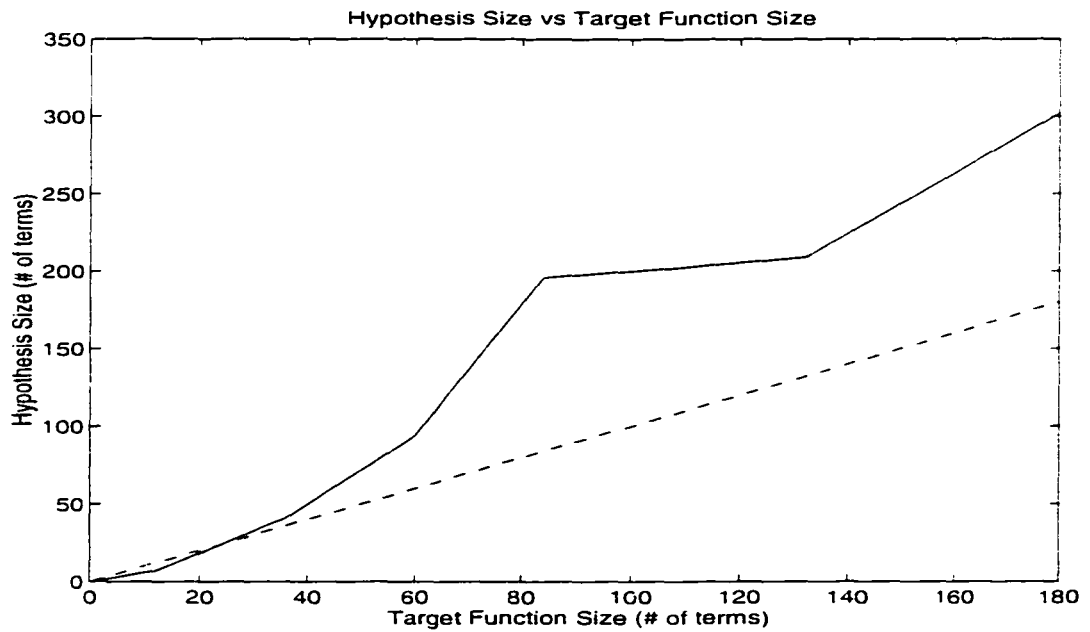


Figure 11.4: Hypothesis Size for a Random Dense Formula

11.1.2 Sparse Formulas

We now show similar empirical results for a class of formulas at the other extreme, where each attribute occurs only once in the formula. This is the well-known class of read-once MDNF. The graph of Figure 11.5 is for read-once MDNF formulas with terms of length five. We ran Algorithm MDNF with $\epsilon = 0.15$, and $\delta = 0.1$.

Note that in the graph of Figure 11.5, the running time increases faster as a function of the number of attributes in the target formula than was the case for the dense formulas in the previous section. The reason for this is that for read-once MDNF formulas, the weight of the set of vectors satisfying exactly one term, hence the coefficient of that term, decrease exponentially in the number of terms in the formula. The corresponding exponential increase in running time is reflected in Figure 11.5. As we showed in Section 9.2, however, we only need to consider terms with coefficients of magnitude at least $\frac{\epsilon^2}{4s^2}$, so this rate of growth would be bounded by a polynomial in ϵ and s .

In Figure 11.6, we show the size of the hypothesis as a function of the size of the target function. As in Figures 11.2 and 11.4, the thin solid line represents the size of the hypothesis output by Algorithm MDNF, and the dashed line indicates the size of the target function.

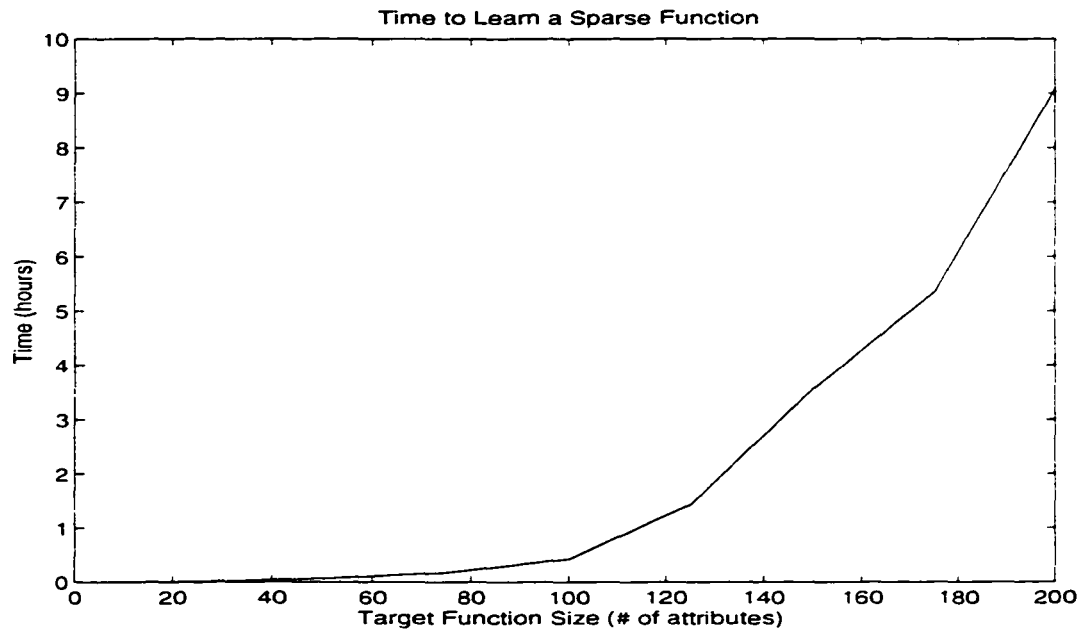


Figure 11.5: Running Time of Algorithm MDNF for a Sparse Formula

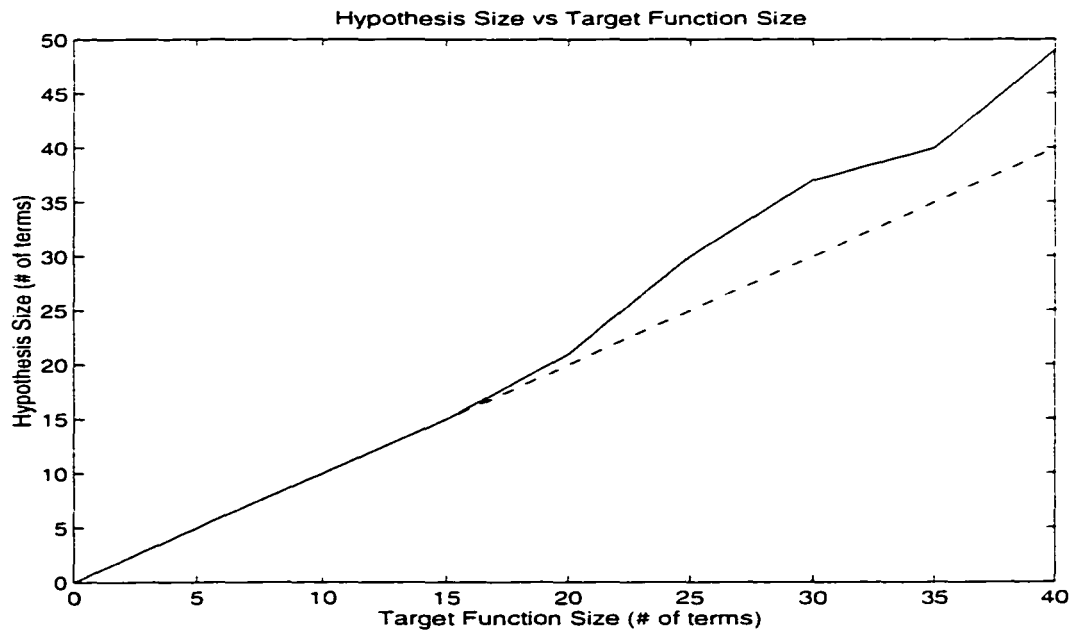


Figure 11.6: Hypothesis Size for a Sparse Formula

11.1.3 Sparse and Dense Formulas

The examples we have considered in the previous two sections contained only dense terms, or only sparse terms. As we have discussed previously, the PFC statistic works well empirically on the sparse formulas, and the EPFC statistic works well on dense formulas. In this section, we give empirical results for formulas containing both sparse and dense terms. Consider the formula

$$\begin{aligned}
 f = & \ x_1x_2x_3x_4 + & (11.1) \\
 & \ x_1x_2x_5x_6 + \\
 & \ x_7x_8x_9x_{10} + \\
 & \ x_7x_8x_{11}x_{12} + \\
 & \ x_1x_2x_7 + \\
 & \ x_1x_2x_8 + \\
 & \ x_1x_7x_8 + \\
 & \ x_2x_7x_8
 \end{aligned}$$

The first four terms of this formula form a one-read-once formula, which we showed in Chapter 9 are learnable in polynomial time using the PFC statistic. We add to that formula four terms that are dense on attributes $\{x_1, x_2, x_7, x_8\}$. We have shown in Section 10.3 that terms of this form are not learnable by the PFC statistic, since the PFC coefficient degenerates by becoming negative, making terms indistinguishable from cross-terms. We show in Section 11.1.1 of this chapter, however, that such dense terms can be learned using the EPFC heuristic. When Algorithm MDNF is run on the formula in (11.1) above, the size of the hypothesis produced

is roughly twice the size of the target formula.

The following is another example of a formula containing both sparse and dense terms, for which the algorithm also produces a hypothesis within twice the size of the minimal hypothesis.

$$\begin{aligned}
 f = & x_1x_2x_3x_4 + & (11.2) \\
 & x_1x_2x_5x_6 + \\
 & x_1x_2x_7x_8 + \\
 & x_1x_2x_9x_{10} + \\
 & x_{11}x_{12}x_{13}x_{14} + \\
 & x_{11}x_{12}x_{15}x_{16} + \\
 & x_{11}x_{12}x_{17}x_{18} + \\
 & x_{11}x_{12}x_{19}x_{20} + \\
 & x_{11}x_{12}x_{21}x_{22} + \\
 & x_1x_{11} + \\
 & x_1x_{12} + \\
 & x_2x_{11} + \\
 & x_2x_{12}
 \end{aligned}$$

11.1.4 Majority Functions

The last set of empirical results we discuss in this section is for the class of majority functions. The majority function on n attributes is defined as:

$$f(x_1, \dots, x_n) = \begin{cases} 1 & x_i = 1 \text{ for at least half of the } x_i\text{'s} \\ 0 & \text{otherwise} \end{cases}$$

Thus, for example,

$$\begin{aligned} f = & x_1x_2 + \\ & x_1x_3 + \\ & x_1x_4 + \\ & x_2x_3 + \\ & x_2x_4 + \\ & x_3x_4 \end{aligned}$$

is the majority function on four attributes. Let f_{maj_i} denote the majority function on i attributes. Note that the size of f_{maj_i} , in terms of the number of attributes occurring in the formula, is $\frac{n}{2} * \binom{n}{\frac{n}{2}}$. In the graph of Figure 11.7, we show the time to learn majority functions f_{maj_4} , f_{maj_6} , and f_{maj_8} as a function of the size of the target formula.

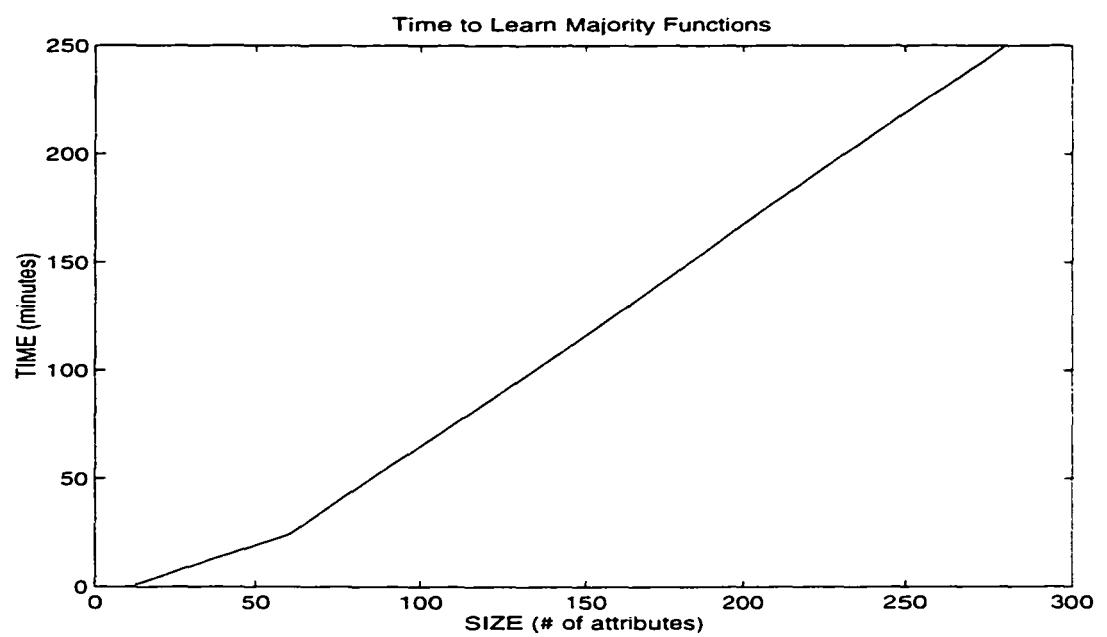


Figure 11.7: Running Time of Algorithm MDNF for Majority

11.2 Empirical Results on Real-World Databases

For the empirical results discussed in the above sections, we generated examples of target formulas uniformly at random, in order to test empirically the running time and hypothesis accuracy of the heuristic algorithms presented in Chapter 10 for learning monotone DNF formulas. In this section, we apply these algorithms to real-world databases from the UCI repository of machine learning databases [MM 98]. We tested the algorithms on two data sets from this site: the tic-tac-toe database, and the mushroom database.

11.2.1 Tic-Tac-Toe Database

The tic-tac-toe database contains all possible tic-tac-toe endgames in which X had the first move. A board configuration is represented by nine variables, one for each square on the board. Each variable can take on the value "X", "O", or "?", corresponding to the square containing an X, an O, or being empty. Since our algorithms use binary valued variables, we replaced each variable by three binary variables, one for each possible value of the original variable. For example, the variable for square one is replaced by the binary variables: square one has X, square one has O, square one is empty. Thus, the resulting representation consists of 27 binary variables.

The database classifies game boards in which X wins. Thus, a positive example is one where X wins the game, and a negative example is either a board configuration where O wins, or where there is a draw. This database contains 625 positive examples, and 333 negative examples. We ran our algorithms on this data set, drawing both positive and negative examples uniformly at random from the database. Since the example space is not uniform over all possible assignments to

the variables. it was not clear a priori that our Fourier analysis algorithms would perform well on this data set.

The hypothesis obtained by the algorithm characterized exactly the concept of "Win for X". In addition, the hypothesis obtained was minimal. The resulting hypothesis was of the form: (X on square 1 and X on Square 2 and X on Square 3) or (X on square 1 and X on Square 5 and X on Square 9) or That is, it contained eight conjuncts, three corresponding to X's on horizontal rows, three corresponding to X's on vertical rows, and two corresponding to X's on the diagonals.

11.2.2 Mushroom Database

The other database we used to test our algorithms is the mushroom database of Schlimmer [Sch 87, MM 98]. This database classifies mushrooms as poisonous or edible based on 22 physical attributes, each of which has between two and twelve possible values. We mapped each value of each variable onto a Boolean attribute, such that if that attribute has value 1, then the original variable takes on the corresponding nominal value. Applying this mapping to all variables produced a domain of 126 Boolean variables. See Appendix A for a list of the attributes and the mapping onto Boolean variables.

The database contains approximately 8000 examples of mushrooms. In our first set of experiments, we chose 4000 examples at random for the training set, and used the remaining 4000 as the test set. We ran the algorithm for several configurations of the accuracy and confidence parameters. The results of these experiments are presented in Table 11.1.

On this data set, the algorithm constructed a set of 14 conjunctive terms, each with an average of ten attributes. The actual hypothesis output can be found

Accuracy	Hypothesis Size	Error of Hypothesis	
		Pos	Neg
0.3	2	0.0546	0.0179
0.25	5	0.0730	0.0083
0.2	6	0.0281	0.0189
0.15	7	0.0304	0.0095
0.1	9	0.0145	0.0027
0.05	11	0.0049	0.0015
0.01	14	0.0000	0.0000

Table 11.1: Output Hypothesis Size and Accuracy on 50% of Data

in Appendix A. The classification accuracy of this set of rules was 100% for both edible and poisonous mushrooms, on *both* the training data *and* the test data. Thus, the set of rules formed describe the data completely. To the author's knowledge, these are the first results that obtain 100% accuracy on both positive and negative examples for this data set. In addition, the hypothesis obtained is a monotone formula.

The fraction of the data partitioned into the data set and test set for our first example was somewhat arbitrarily set to 50%. We also tested the algorithm with smaller training sets, comprising 20%, 10% and 1% of the data, drawn at random, in order to determine how well the hypothesis generalizes with small training sets. Tables 11.2, 11.3 and 11.4 present these results, respectively.

As expected, the error on the test data is slightly higher when the training set is smaller. When trained on 20% of the data, the testing errors were not significantly different from training on 50% of the data. When trained on 10% of the data,

Accuracy	Hypothesis Size	Error of Hypothesis	
		Pos	Neg
0.3	4	0.0551	0.0189
0.25	6	0.0225	0.0154
0.2	6	0.0240	0.0161
0.15	8	0.0133	0.0011
0.1	7	0.0145	0.0026
0.05	12	0.0059	0.0000
0.01	15	0.0000	0.0000

Table 11.2: Output Hypothesis Size and Accuracy on 20% of Data

Accuracy	Hypothesis Size	Error of Hypothesis	
		Pos	Neg
0.3	2	0.0620	0.0163
0.25	2	0.0609	0.0163
0.2	7	0.0256	0.0167
0.15	6	0.0251	0.0195
0.1	10	0.0058	0.0156
0.05	10	0.0058	0.0048
0.01	15	0.0010	0.0011

Table 11.3: Output Hypothesis Size and Accuracy on 10% of Data

Accuracy	Hypothesis Size	Error of Hypothesis	
		Pos	Neg
0.3	2	0.0599	0.0158
0.25	2	0.0626	0.0209
0.2	3	0.0588	0.0231
0.15	2	0.0635	0.0193
0.1	4	0.0465	0.0178
0.05	5	0.0145	0.0177
0.01	5	0.0148	0.0178

Table 11.4: Output Hypothesis Size and Accuracy on 1% of Data

the error rates were slightly higher. We also ran the algorithm with a test set comprising only 1% of the data. While the errors on this set of tests were still quite small, the measured error was no longer within the bounds of the error parameter ϵ . The error rate seemed to asymptote at around 1.45% on the positive examples and 1.78% on the negative examples, regardless of the value of the accuracy parameter.

11.3 Summary of Empirical Results

In this chapter, we have presented empirical results for Algorithm MDNF on several types of formulas, ranging from highly dense formulas to highly sparse formulas. In all cases tested, the algorithm outputs a nearly minimal hypothesis. Due to the interleaving of the PFC and EPFC statistics in the main loop of the algorithm, the “wrong” statistic may be used half the time, thus resulting in the discovery of terms that are not terms of the minimal hypothesis. Since this does not occur more

than half the time, the size of the output hypothesis is in practice within twice the size of the minimal hypothesis.

As discussed in Chapter 8, we only need to consider terms for which the probability weight of the vectors that satisfy only that term is at least $\frac{\epsilon^2}{4s^2}$. Using this fact, in the proof of Theorem 9.7, we show that the worst case running time of our algorithm to learn each attribute of a poly-disjoint one-read-once MDNF formula is $\tilde{O}(\frac{s^4}{\epsilon^4})$. Note that the graphs of the running times for some of the examples appear to be a low order polynomial, and others appear to have a higher rate of growth. For example, the graphs of Figures 11.3 and 11.7 appear to be quadratic, yet the graphs of Figures 11.1 and 11.5 appear to be exponential. As noted previously, the graph of the running time in Figure 11.5, for a highly sparse formula, comes much closer to the worst case than for the other examples. The examples with lower order running times are in turn much better than the theoretical worst-case possibility. From our computational experience, quadratic running time is more often the case. An explanation of why this may be the case is due to the fact that the worst-case scenario is overly pessimistic: it assumes that for each term, the probability weight of the vectors that satisfy only that term can be $\frac{\epsilon^2}{4s^2}$. In general, this will not be the case for all terms simultaneously. Since there are s terms in the target formula, a more typical case would be when each term has probability weight of $\frac{1}{s}$ on the examples that satisfy only that term, and this would lead to complexity quadratic in s . This bound appears to be closer to what is seen empirically for most examples we have studied. An average case analysis of the algorithm we present for one-read-once MDNF is an interesting avenue of future research.

Chapter 12

Conclusions and Open Problems

In the latter part of this thesis, we have given an algorithm for learning poly-disjoint one-read-once MDNF formulas and read-once factorable MDNF formulas on the uniform distribution. In Chapter 10, we gave an heuristic for extending this result to the larger class of MDNF formulas. In Chapter 11, we demonstrate that the algorithm is efficient in practice. While we have not been able to prove that the algorithm works for all of monotone DNF, we have not found any formulas for which it does not work. Thus, the algorithm warrants further study and analysis. It would also be interesting to conduct an average-case analysis of the algorithm we give in Chapter 9 for read-once factorable MDNF. The worst-case complexity bound is a fifth order polynomial in the size of the target formula. In the average case, the complexity seems to be cubic: this intuition is supported by the empirical results given in Chapter 11.

In the heuristic we give in Chapter 10 for learning monotone DNF formulas, we adapted Freund's filtering algorithm [Fre 90], which was developed for boosting weak learning algorithms to strong learning algorithms. The reason for the

adaptation was that Freund's weighting scheme is designed so that the examples drawn on the filtered distribution satisfy approximately half of the weak hypotheses constructed in previous stages of the algorithm. This weighting scheme gives an advantage to previously discovered hypotheses, resulting in frequent re-discovery and a larger final hypothesis. With the modified weighting scheme we develop, all of Freund's proofs still apply, so that we still have a valid hypothesis boosting algorithm. Our weighting scheme gives an advantage to un-discovered hypotheses, though, resulting in less re-discovery, and generally simpler hypotheses. The worst-case bounds on the number of boosting stages required, hence also the complexity of the resulting hypothesis, is the same as for Freund's weighting scheme. For formulas on discrete domains, and for MDNF formulas in particular, better bounds may be attainable for our weighting scheme.

Finally, the EPFC statistic on which we base our heuristic for constructing terms of an MDNF formula is the expectation on the Fourier coefficient over all subsets of some sub-space. Each Fourier coefficient is in a way a measure of the non-randomness of the target formula over that sub-space. Thus, the expectation over all sub-spaces of the coefficient gives some measure of the non-randomness of the formula over the entire space. It would be interesting to consider this measure of randomness further.

Bibliography

- [Ali 92] F. Alizadeh. "Combinatorial Optimization with Semi-Definite Matrices". In *Proceedings of the 2nd Conference on Integer Programming and Combinatorial Optimization*, Carnegie Mellon University, pp. 385-405. 1992.
- [AHK 93] D. Angluin, L. Hellerstein, M. Karpinski. "Learning Read-Once Formulas with Queries". *Journal of the ACM* Vol. 40, pp. 185-210. 1993.
- [AP 95] H. Aizenstein and L. Pitt. "On the Learnability of Disjunctive Normal Form Formulas". *Machine Learning*, Vol. 19, pp. 183-208. 1995.
- [AV 79] D. Angluin, L. Valiant. "Fast Probabilistic Algorithms for Hamiltonian Circuits and Matchings". *Journal of Computer and Systems Sciences*, Vol. 18, pp. 155-193. 1979.
- [ALMS 92] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy. "Proof Verification and Hardness of Approximation Problems". In *Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science*, pp. 14-23. 1992.

- [BFLS 91] L. Babai, L. Fortnow, L.A. Levin, M. Szegedy. "Checking Computations in Polylogarithmic Time". In *Proceedings of the 23rd Annual ACM Symposium on the Theory of Computing*. pp. 21-31. 1991.
- [B 86] E.B. Baum. "Intractable Computations without Local Minima". *Physical Review Letters*. Vol. 57, No. 21. pp. 2764-2767. 1986.
- [BGS 95] M. Bellare, O. Goldreich and M. Sudan. "Free Bits, PCPs and Non-Approximability - Towards Tight Results". In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*. 1995.
- [B 89] A. Blum. "An $\tilde{O}(n^{0.4})$ -Approximation Algorithm for 3-Coloring". In *Proceedings of the 21st Annual ACM Symposium on the Theory of Computing*. pp. 535-542. 1989.
- [BFJK 94] A. Blum, M. Furst, J. Jackson, M. Kearns, Y. Mansour and S. Rudich. "Weakly Learning DNF and Characterizing Statistical Query Learning Using Fourier Analysis". In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing*. pp. 253-262. 1994.
- [C 52] H. Chernoff. "A measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the Sum of Observations". *Annals of Mathematical Statistics*. Vol. 23. pp. 493-509. 1952.
- [CK 95] P. Crescenzi and V. Kann. "A Compendium of NP Optimization Problems". unpublished manuscript. available on the world-wide-web at <http://www.nada.kth.se/~viggo/problemlist/compendium.html>.
- [D 90] R. Dechter. "Enhancement Schemes for Constraint Processing: Back-jumping, Learning, and Cutset Decomposition". *Artificial Intelligence*. Vol. 41. pp. 273-312. 1990.

- [DP 88] R. Dechter and J. Pearl. "Network-Based Heuristics for Constraint-Satisfaction Problems". *Artificial Intelligence*, Vol. 34, pp. 1-38, 1988.
- [FG 95] U. Feige and M. Goemans. "Approximating the Value of Two Prover Proof Systems, With Applications to MAX 2-SAT and MAX DICUT". preprint, 1995.
- [FW 92] E.C. Freuder and R.J. Wallace. "Partial Constraint Satisfaction". *Artificial Intelligence*, Vol. 58, pp. 21-70, 1992.
- [Fre 90] Y. Freund. "Boosting a Weak Learning Algorithm by Majority". In *Proceedings of the 1990 Workshop on Computational Learning Theory*, pp. 202-216, 1990.
- [GJ 79] M.R. Garey and D.S. Johnson. *Computers and Intractability*, W.H. Freeman & Co., New York, 1979.
- [GVY 93] N. Garag, V. Vazirani and M. Yannakakis. "Approximate max-flow min-(multi)cut theorems and their applications". In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing*, pp. 693-707, 1993.
- [G 89] D. Gentner. "The Mechanisms of Analogical Learning". In *Similarity and Analogical Learning*, Cambridge University Press, Cambridge, pp. 199-241, 1989.
- [GW 94] M.X. Goemans and D.P. Williamson. ".878-Approximation Algorithms for MAX CUT and MAX 2SAT". In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing*, pp. 422-431, 1994.

- [GKS 90] S. Goldman, M. Kearns and R. Schapire. "Exact Identification of Circuits Using Fixed Points of Amplification Functions". In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*. pp. 193-202. 1990.
- [H 92] T.R. Hancock. "The Complexity of Learning Formulas and Decision Trees that have Restricted Reads". PhD Thesis. Department of Computer Science. Harvard University. Technical Report TR-15-92. 1992.
- [HKLW 88] D. Haussler, M. Kearns, N. Littlestone, M.K. Warmuth. "Equivalence of Models for Polynomial Learnability". In *Proceedings of the 1988 Workshop on Computational Learning Theory*. 1988.
- [HT 89] K.J. Holyoak and P. Thagard. "Analogical mapping by constraint satisfaction". *Cognitive Science*, Vol. 13, pp. 295-355. 1989.
- [HT 95] K.J. Holyoak and P. Thagard. *Mental leaps: Analogy in creative thought*. MIT Press/Bradford Books, Cambridge, MA. 1995.
- [H 82] J.J. Hopfield. "Neural Networks and Physical Systems with Emergent Collective Computational Properties". *Proceedings of the National Academy of Sciences of the United States of America*, Vol. 79, pp. 2554-58. 1982.
- [HMM 85] S.L. Hurst, D.M. Miller, J.C. Muzio. *Spectral Techniques in Digital Logic*. Academic Press, London. 1985.
- [J 94] J. Jackson. "An Efficient Membership-Query Algorithm for Learning DNF with Respect to the Uniform Distribution". In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*. pp. 42-53. 1994.

- [JJ 94] J.R. Josephson and S.G. Josephson. (Eds.). *Abductive inference: Computation, philosophy, technology*. Cambridge University Press, Cambridge, 1994.
- [KMS 94] D. Karger, R. Motwani, M. Sudan. "Approximate Graph Coloring by Semidefinite Programming". In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*. pp. 2-13, 1994.
- [Ka 75] R.M. Karp. "On the Computational Complexity of Combinatorial Problems". *Networks*. Vol. 5. pp. 45-68. 1975.
- [Ke 89] M. Kearns. "The Computational Complexity of Machine Learning". PhD Thesis, Department of Computer Science, Harvard University, 1989.
- [Ke 93] M. Kearns. "Efficient Noise-Tolerant Learning from Statistical Queries". In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*. pp. 285-295. 1993.
- [KLPV 87] M. Kearns, M. Li, L. Pitt, L. Valiant. "On the Learnability of Boolean Formulae". In *Proceedings of the 19th Annual ACM Symposium on the Theory of Computing*. pp. 285-295. 1987.
- [KLV 94] M. Kearns, M. Li and L. Valiant. "Learning Boolean Formulas". *Journal of the ACM*. Vol. 41. No. 6. pp. 1298-1328. 1994.
- [KV 88] M. Kearns, L.G. Valiant. "Learning Boolean Formulae or Finite Automata is As Hard as Factoring". Tech Report TR- 14-88. Aiken Computer Laboratory, Harvard University, 1988.

- [KM 92] J.D. Kececioglu and E.W. Meyers. "Combinatorial Algorithms for DNA Sequence Assembly". Technical Report TR 92-37. The University of Arizona. Tucson, Arizona. 1992.
- [KMSV 94] S. Khanna, R. Motwani, M. Sudan and U. Vazirani. "On Syntactic versus Computational Views of Approximability". In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing*, pp. 819-830. 1994.
- [Khar 94] R. Khardon. "On using the Fourier transform to learn Disjoint DNF". *Information Processing Letters*, Vol. 49, pp. 219-222. 1994.
- [Kh 92] M. Kharitonov. "Cryptographic Lower Bounds for Learnability of Boolean Functions on the Uniform Distribution". In *Proceedings of the 1992 Workshop on Computational Learning Theory*, 1992.
- [KARR 90] P. Klein, A. Agrawal, R. Ravi and S. Rao. "Approximation Through Multicommodity Flow". In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, pp. 726-737. 1990.
- [Ku 92] V. Kumar. "Algorithms for Constraint-Satisfaction Problems - A Survey". *AI Magazine*, Vol. 13, No. 1, pp. 32-44. 1992.
- [KM 93] E. Kushilevitz and Y. Mansour. "Learning Decision Trees Using The Fourier Spectrum". *SIAM J. Comput.*, Vol. 22, No. 6, pp. 1331-1348. 1993.
- [LMN 89] N. Linial, Y. Mansour, N. Nisan. "Constant Depth Circuits, Fourier Transform, and Learnability". In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, pp. 574-579. 1989.

- [LST 89] L. Lovász, M. Saks and W.T. Trotter. "An On-line Graph Coloring Algorithm with Sub-linear Performance Ratio". *Discrete Mathematics*. Vol. 75. pp. 319-325. 1989.
- [LY 93] C. Lund and Y. Yannakakis. "On the Hardness of Approximating Minimization Problems". In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing*. pp. 286-293. 1993.
- [Mac 77] A.K. Mackworth. "Consistency in Networks of Relations" *Artificial Intelligence*. Vol. 8. pp. 99-118. 1977.
- [MF 93] A.K. Mackworth and E.C. Freuder. "The Complexity of Constraint Satisfaction Revisited". *Artificial Intelligence*. Vol. 59. pp. 57-62. 1993.
- [Man 92] Y. Mansour. "An $O(n^{\log \log n})$ Learning Algorithm for DNF under the Uniform Distribution". In *Proceedings of the 1992 Workshop on Computational Learning Theory*. pp. 53-61. 1992.
- [MM 98] Merz, C.J. and Murphy, P.M.. UCI Repository of machine learning databases
[<http://www.ics.uci.edu/~mlearn/MLRepository.html>]. University of California Irvine. Department of Information and Computer Science. 1998.
- [MJPL 92] S. Minton, M.D. Johnston, A.B. Philips and P. Laird. "Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems". *Artificial Intelligence*. Vol. 58. No. 1-3. pp. 161-205. 1992.
- [OJ 94] P. O'Rorke and J.R. Josephson. (Eds.). *Automated abduction: Inference to the best explanation*. AAAI Press, Menlo Park, CA. 1994.

- [PY 91] C.H. Papadimitriou and M. Yannakakis. "Optimization, Approximation, and Complexity Classes". *Journal of Computer and System Sciences*. Vol. 43. pp. 425-440. 1991.
- [PV 86] L. Pitt. L. Valiant. "Computational Limits on Learning from Examples". Tech. Rep. TR-05-86. Aiken Computer Laboratory. Harvard University. 1986.
- [RM 86] D.E. Rumelhart and J.L. McClelland. (Eds.). *Parallel distributed processing: Explorations in the microstructure of cognition*. MIT Press/Bradford Books. Cambridge MA. 1986.
- [SG 76] S. Sahni and T. Gonzalez. "P-Complete Approximation Problems". *Journal of the Association for Computing Machinery*. Vol. 23. No. 3. pp. 555-565. July 1976.
- [S 90] R.S. Schapire. "The Strength of Weak Learnability". *Machine Learning*. Vol. 5. pp. 197-227. 1990.
- [S 92] R. Schapire. *The Design and Analysis of Efficient Learning Algorithms*. Doctoral Dissertation. MIT. The MIT Press. 1992.
- [Sch 87] J. C. Schlimmer. "Concept Acquisition Through Representational Adjustment". PhD Thesis. Department of Information and Computer Science. University of California, Irvine. 1987.
- [Th 89] P. Thagard. "Explanatory Coherence". *Behavioral and Brain Sciences*. Vol. 12. No. 3. pp. 435-467. 1989.
- [Th 92] P. Thagard. *Conceptual Revolutions*. Princeton University Press. Princeton. 1992.

- [Th 94] P. Thagard. "Probabilistic Networks and Explanatory Coherence". In *Automated Abduction: Inference to the Best Explanation*. unpublished manuscript. 1994.
- [THING 90] P. Thagard, K.J. Holyoak, G. Nelson and D. Gochfeld. "Analog Retrieval by Constraint Satisfaction". *Artificial Intelligence*. Vol. 46. pp. 259-310. 1990.
- [TM 95] P. Thagard and E. Millgram. "Inference to the best plan: A coherence theory of decision". In *A. Ram and D.B. Leake (Eds.), Goal-directed learning Cambridge*. pp. 439-454. MIT Press, Cambridge, MA. 1995.
- [TV 98] P. Thagard and K. Verbeurgt. "Coherence as Constraint Satisfaction". *Cognitive Science*. Vol. 22. No. 1. pp. 1-24. 1998.
- [TSSW 96] L. Trevisan, G. Sorkin, M. Sudan and D. Williamson. "Gadgets, Approximation, and Linear Programming". In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*. pp. 617-626. 1996.
- [Val 84] L. Valiant. "A Theory of the Learnable". *Communications of the ACM*. Vol. 27. no. 11. pp. 1134-1142. 1984.
- [Ver 90] K. Verbeurgt. "On the Learnability of DNF Formulae". Master's Thesis. University of Toronto, Toronto, Ontario. 1990.
- [Ver 90a] K. Verbeurgt. "Learning DNF Under the Uniform Distribution in Quasi-polynomial Time". In *Proceedings of 1990 Workshop on Computational Learning Theory*. pp. 314-326. 1990.

- [Ver 98] K. Verbeugt. "Learning Sub-Classes of Monotone DNF on the Uniform Distribution". In *Proceedings of the 1998 Conference on Algorithmic Learning Theory*. 1998.

Appendix A

In this appendix, we give the hypotheses output in the empirical tests on the mushroom database (see Chapter 11). Table A.1, given at the end of this appendix, shows the attributes used to characterize examples of mushrooms as positive examples, corresponding to poisonous mushrooms, or negative examples, corresponding to edible mushrooms.

In the following, we give the hypothesis output for select runs of our learning algorithm on the mushroom database. The set of hypotheses we show here are based on a training set containing 50% of the data, and a test set containing 50% of the data. We give here the most accurate hypothesis obtained, which classifies both the training set and the test set with 100% accuracy, and also the smallest hypothesis obtained. The most accurate hypothesis was obtained with input parameters $\epsilon = .01$ and $\delta = .01$, and the smallest hypothesis with parameters $\epsilon = .3$ and $\delta = .3$.

In the following, conjunctive terms are indicated by lists of pairs of attributes. Conjunctive terms in the same hypothesis are separated by a blank line.

Example 1: $\epsilon = .01$, $\delta = .01$ The classification accuracy of this hypothesis is 100% on both the training set and the test set.

bruises=no
gill-attachment=descending
gill-spacing=close
stalk-surface-below-ring=silky
veil-type=partial
veil-color=white
ring-number=one

bruises=no
gill-attachment=descending
gill-spacing=close
stalk-surface-above-ring=silky
veil-type=partial
veil-color=white
ring-number=one

gill-attachment=descending
gill-spacing=close
veil-type=partial
veil-color=white
ring-number=one
spore-print-color=white
population=several

odor=foul
gill-attachment=descending

gill-spacing=close
veil-type=partial
veil-color=white
ring-number=one

cap-surface=smooth
bruises=yes
gill-attachment=descending
gill-spacing=close
stalk-shape=enlarging
stalk-root=bulbous
stalk-surface-below-ring=smooth
stalk-color-above-ring=white
stalk-color-below-ring=white
veil-type=partial
veil-color=white
ring-number=one
ring-type=pendant

cap-surface=smooth
bruises=yes
gill-attachment=descending
gill-spacing=close
stalk-shape=tapering
stalk-color-above-ring=white
stalk-color-below-ring=white

veil-type=partial
veil-color=white
ring-number=one
ring-type=pendant

bruises=yes
gill-attachment=descending
gill-spacing=close
gill-size=narrow
stalk-shape=enlarging
stalk-surface-above-ring=smooth
stalk-surface-below-ring=smooth
stalk-color-above-ring=white
veil-type=partial
veil-color=white
ring-number=one

gill-attachment=descending
gill-size=narrow
stalk-shape=enlarging
stalk-root=bulbous
stalk-surface-above-ring=smooth
stalk-surface-below-ring=smooth
stalk-color-above-ring=white
stalk-color-below-ring=white
veil-type=partial

veil-color=white
ring-number=one
ring-type=pendant

gill-attachment=descending
gill-spacing=close
stalk-shape=enlarging
stalk-shape=tapering
stalk-surface-above-ring=smooth
stalk-surface-below-ring=smooth
stalk-color-above-ring=white
stalk-color-below-ring=white
veil-type=partial
veil-color=white
ring-number=one
population=several

gill-attachment=descending
stalk-shape=enlarging
stalk-root=bulbous
stalk-surface-above-ring=smooth
stalk-surface-below-ring=smooth
stalk-color-above-ring=white
stalk-color-below-ring=white
veil-type=partial
veil-color=white

ring-type=pendant

habitat=woods

bruises=yes

odor=none

gill-attachment=descending

gill-spacing=close

gill-size=broad

stalk-shape=enlarging

stalk-root=bulbous

stalk-surface-above-ring=smooth

stalk-surface-below-ring=smooth

stalk-color-above-ring=white

stalk-color-below-ring=white

veil-type=partial

veil-color=white

ring-number=two

ring-type=pendant

spore-print-color=green

population=several

bruises=yes

gill-attachment=descending

gill-spacing=close

gill-size=narrow

stalk-shape=enlarging

stalk-color-above-ring=white

veil-type=partial

veil-color=white

ring-number=one

ring-type=pendant

gill-attachment=descending

gill-spacing=close

stalk-shape=tapering

stalk-surface-above-ring=smooth

stalk-surface-below-ring=smooth

veil-type=partial

veil-color=white

ring-number=one

ring-type=evanescent

population=several

cap-surface=scaly

bruises=no

stalk-shape=enlarging

stalk-root=club

veil-type=partial

Example 2 $\epsilon = .3$, $\delta = .3$ The accuracy of this hypothesis on the test set is 94.54% on positive examples (poisonous mushrooms) and 98.21% on negative examples (edible mushrooms).

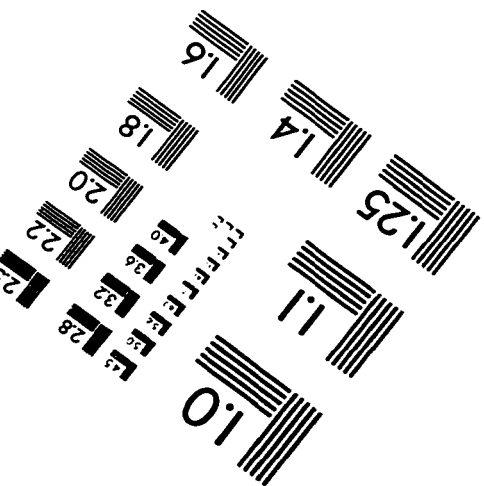
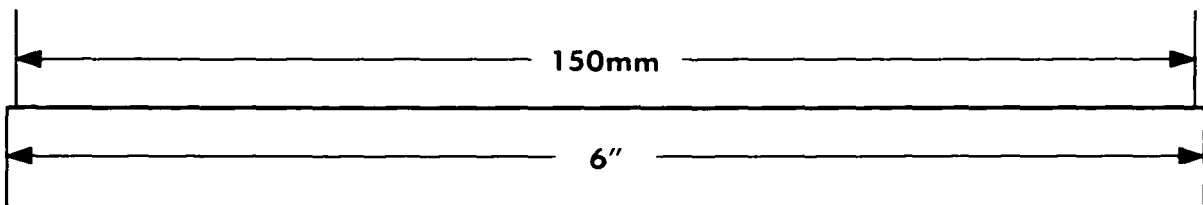
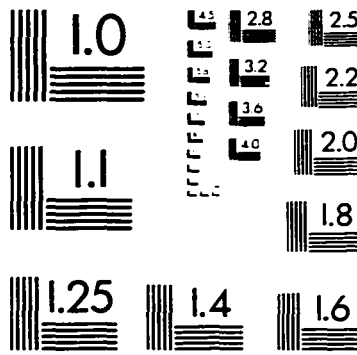
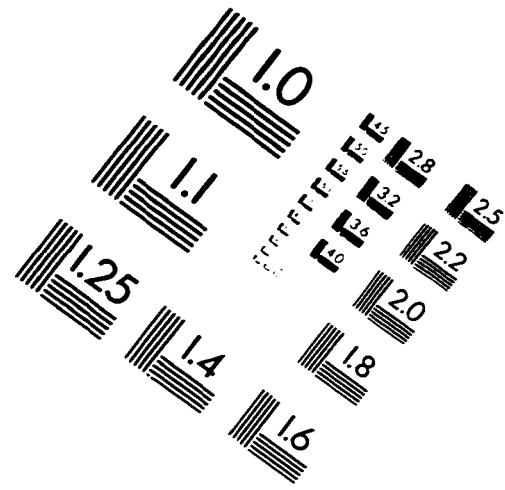
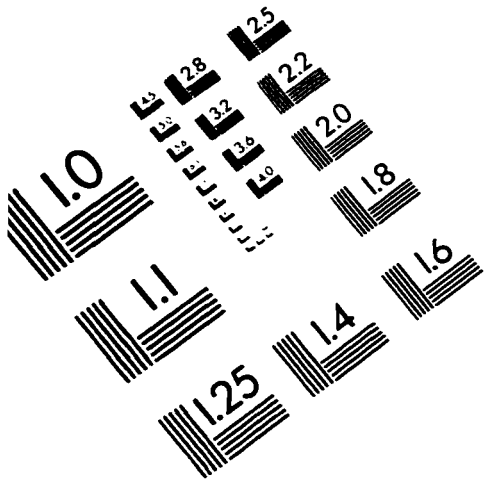
bruises=no
gill-attachment=descending
gill-spacing=close
veil-type=partial
veil-color=white
ring-number=one

odor=foul
gill-attachment=descending
gill-spacing=close
veil-type=partial
veil-color=white
ring-number=one

Attribute Number	Attribute Name	Attribute Values
1-6	cap-shape	bell, conical, convex, flat, knobbed, sunken
7-11	cap-surface	fibrous, grooves, scaly, smooth, brown
12-20	cap-color	buff, cinnamon, gray, green, pink, purple, red, white, yellow
21-22	bruises	yes, no
23-31	odor	almond, anise, creosote, fishy, foul, musty, none, pungent, spicy
32-35	gill-attachment	attached, descending, free, notched
36-38	gill-spacing	close, crowded, distant
39-40	gill-size	broad, narrow
41-52	gill-color	black, brown, buff, chocolate, gray, green, orange, pink, purple, red, white, yellow
53-54	stalk-shape	enlarging, tapering
55-61	stalk-root	bulbous, club, cup, equal, rhizomorphs, rooted, missing-value
62-65	stalk-surface-above-ring	fibrous, scaly, silky, smooth
66-69	stalk-surface-below-ring	fibrous, scaly, silky, smooth
70-78	stalk-color-above-ring	brown, buff, cinnamon, gray, orange, pink, red, white, yellow
79-87	stalk-color-below-ring	brown, buff, cinnamon, gray, orange, pink, red, white, yellow
88-89	veil-type	partial, universal
90-93	veil-color	brown, orange, white, yellow
94-96	ring-number	none, one, two
97-104	ring-type	cobwebby, evanescent, flaring, large, none, pendant, sheathing, zone
105-113	spore-print-color	black, brown, buff, chocolate, green, orange, purple, white, yellow
114-119	population	abundant, clustered, numerous, scattered, several, solitary
120-126	habitat	grasses, leaves, meadows, paths, urban, waste, woods

Table A.1: Attributes for the Mushroom Database

IMAGE EVALUATION TEST TARGET (QA-3)



APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989

© 1993, Applied Image, Inc., All Rights Reserved

