Generating all Maximal Independent Sets of Bounded-degree Hypergraphs

Nina Mishra
Computer Science Department
University of Illinois at Urbana-Champaign
Urbana, IL 61801
nmishra@uiuc.edu

Leonard Pitt
Computer Science Department
University of Illinois at Urbana-Champaign
Urbana, IL 61801
pitt@cs.uiuc.edu

Abstract

We show that any monotone function with a read-k CNF representation can be learned in terms of its DNF representation with membership queries alone in time polynomial in the DNF size and n (the number of variables) assuming k is some fixed constant. The problem is motivated by the well-studied open problem of enumerating all maximal independent sets of a given hypergraph. Our algorithm gives a solution for the bounded degree case and works even if the hypergraph is not input, but rather only queries are available as to which sets are independent.

1 INTRODUCTION

A hypergraph H is a collection of subsets (edges) E of a finite set of vertices V. An independent set of a hypergraph is a subset of vertices, $V' \subseteq V$ such that no edge in E is contained in V'. An independent set I is maximal if no superset I' of I is also an independent set. Given a hypergraph H, the *hypergraph independent set problem* is that of enumerating all maximal independent sets of H. Note that while finding the maximum cardinality independent set is NP-hard [GJ79], finding a maximal independent set I is easy: iteratively add vertices to I while maintaining the property that I is an independent set. We consider here the problem of enumerating *all* maximal independent sets. It can be shown (see Section 2) that the hypergraph independent set problem is equivalent to the following: Given a monotone CNF formula f find a (reduced) monotone DNF formula g such that g is equivalent to f.

A more demanding problem is the setting in which the CNF description (equivalently, the hypergraph) is not explicitly given, but rather is hidden inside a "black box" to which only *membership queries* may be posed. A membership query for a Boolean function f is a vector $x \in \{0,1\}^n$, and is answered "yes" if $f(x) = 1$, and "no" otherwise. A membership query for a hypergraph $H = (V, E)$ is a subset $S \subseteq V$, and is answered "yes" if S is an independent set, and "no" otherwise.

Membership queries have been widely studied in the context of learning. In Angluin's seminal paper [Ang88], information theoretic barriers are given showing that there is no algorithm for learning monotone DNF (or CNF) formulas from membership queries alone, in time polynomial in the size of the target DNF (or CNF) formula. However, these lower bounds do not apply here, as the learning problem derived from the hypergraph independent set problem is that of finding a monotone DNF formula for a monotone function (available in CNF form, or via membership queries only) in time polynomial in the *sum* of the CNF and DNF sizes. Put another way, the problem to be solved is that of exhibiting a polynomial total time algorithm for finding a monotone DNF formula. (A *polynomial total time* algorithm is one that runs in time polynomial in the sum of the lengths of the input and output. There are other notions of polynomial time. See [JPY88] for a discussion.) We often omit the adjective "total" with the understanding that all of our algorithms run in polynomial total time. Applications of the hypergraph independent set problem abound, hence a general solution to the problem has been sought. After describing some of the applications and some recent results, we present a polynomial-time algorithm for the restricted case of bounded degree hypergraphs, using membership queries alone.

Motivation: In the context of data mining, an algorithm for the independent set problem could be used to find all the keys[1] in a relation. In addition to providing high-level information about a relation, the keys can be used for verifying that a collection of mined rules are in fact all the interesting rules in a relation [MT96]. Similarly, key enumeration is related to the problem of finding a small cover for the set of functional dependencies that hold in a database, a problem useful in database design or query processing [MR92b, MR92a, KM95].

Another example of the utility of the independent

[1] For a relation over attributes R, the *keys* are the minimal subsets X of R such that no two rows in the relation agree on all attributes in X.

COLT 97 Nashville, Tennesee, USA
Copyright 1997 ACM 0-89791-891-6/97/7..$3.50

set problem arises in the context of reasoning. Given a knowledge base that can be represented as a conjunction of propositional Horn clauses (with empty consequents), a solution to the independent set problem could be used to generate a collection of characteristic models [KKS93, KR94] to use in various reasoning tasks (for example, determining whether a query is entailed by the knowledge base) [Kha95, KMR95].

The independent set problem is also related to the problem of determining if a version space has converged. For a concept class C the version space [Mit82] induced by positives P and negatives N is the set of concepts in C consistent with P and N. A version space V has converged if $|V| = 1$. A solution to the CNF to DNF translation problem could be used to determine if a version space has converged for the class of monotone functions [HMP97] (the idea being to translate examples in P into terms in a DNF formula and examples in N into clauses in a CNF formula).

Finally, we note that the independent set problem is an example of knowledge compilation. The compilation process is often used to translate one representation of knowledge into another so as to make it easier to use that knowledge. One example, discussed in [SK91], considers compiling arbitrary (non-monotone) CNF formulas into Horn lower and upper bounds in order to make answering entailment questions easier. The independent set problem is also a form of compilation since we are given a CNF (hypergraph) and wish to compile that information into a DNF (all the maximal independent sets). Our membership query result strengthens this statement since it implies that regardless of what form the function is provided to us (e.g., it could be an arbitrary Boolean formula) as long as that representation is polynomially evaluable (and, of course, corresponds to a monotone read-k CNF), we can efficiently compile the function into its DNF form.

A more thorough review of applications of the maximal independent set problem can be found in [EG95].

Related Work: To date, however, there is no known polynomial-time algorithm for the general independent set problem. Some work has investigated relationships between the maximal independent set problem and other open problems [BI95, EG95]. Others have given superpolynomial time algorithms for the problem. For example, Bshouty et al. [BCG$^+$96] have shown that with an NP-oracle and membership queries, both the minterms and maxterms (that is, all the maximal independent sets and the hypergraph itself) can be efficiently enumerated. More recently, Fredman and Khachiyan [FK96] have given the fastest known algorithm for the problem: Their algorithm runs in time $O(m^{o(\log m)})$ (where m is the sum of the size of the hypergraph H (CNF), and the size of all the maximal independent sets of H (DNF)), hence providing evidence that the problem is unlikely to be NP-hard.

Since an efficient solution to the general problem is not known, some research has focussed on determining which natural subcases of the general problem have efficient solutions. For example, in the event that each edge consists of two vertices (i.e., when the hypergraph is a graph) efficent solutions have been given under various definitions of polynomial time [JPY88, LLK80, TIAS77, KW85]. Extending this work, when the cardinality of each edge of the hypergraph is bounded by some constant or when the hypergraph is acylic, there is a known efficient solution to the independent set problem [EG95].

The restriction considered in this paper is based on limiting the *degree* of each vertex in the hypergraph — the maximum number of edges in which any vertex is contained. In Boolean terminology, the restriction limits the number of "reads" (occurrences of a variable) in a formula, a restriction that has been well-investigated in the learning-theory literature [AHK93, PR94b, PR94a, BHH95a, BHH95b, ABK$^+$97, AHHPar]. Previous work has shown that it is possible to find the minterms of a read-once $(\wedge, \vee)$ formula under a stronger notion of polynomial time [AHK93, GK95]. In the hypergraph setting, this restriction trivializes to enumerating all maximal independent sets of degree-one (read-once) hypergraphs. We show here that if the degree of each vertex is bounded by some constant that there is an efficient solution to the independent set problem. In Boolean terminology, we show that if each variable appears in at most k clauses (i.e., the CNF formula is read-k), then there exists an efficient algorithm to generate the DNF formula, using membership queries alone.

Overview: The techniques we use are based on an inductive characterization of the problem — the question being, when can the minterms of a subset of the clauses of a monotone CNF formula be used to compute all of the minterms of the entire CNF formula? (Alternatively, when can the maximal independent sets of a subset, E', of the edges of a hypergraph be used to compute the maximal independent sets of a larger subset of edges, $E'' \supseteq E'$?) In Section 3 we begin with a simple yet inefficient algorithm for the general problem based on such an inductive characterization. The algorithm is not necessarily efficient since no order is imposed on how larger and larger subsets of the clauses of the CNF formula are considered. We demonstrate that a possible source of inefficiency is when minterms of a subset of the clauses of the CNF formula do not correspond to minterms of the entire CNF formula. When we do impose an order, by considering minterms over larger and larger subsets of the variables, we can (in Section 4) show that there is an efficient algorithm for finding a monotone DNF representation of a given monotone read-k CNF formula — in other words, that there is an efficient solution to the bounded degree hypergraph independent set problem. We generalize the result in Section 5 by giving an algorithm that uses membership queries alone to find the DNF formula.

Our results demonstrate a well-known phenomenon in learning theory: relaxing the hypothesis class often makes learning easier. An example of this fact is given in [PV88], where it is observed that k-term DNF formulas cannot be properly PAC learned, but can be PAC learned in terms of k-CNF expressions. Another example of this fact is in learning the class of DNF formulas.

While there is currently no known polynomial-time algorithm for this concept class, Bshouty shows that any Boolean function can be learned with equivalence and membership queries, using the hypothesis class of disjunctions of unate CNF formulas, in time polynomial in the sum of the sizes of the DNF and CNF representations [Bsh95].

For the class of read-k CNF formulas, Angluin shows that finding a description of the read-k CNF formula with membership queries only requires at least $2^{n/2}-1$ membership queries (where n is the number of variables) [Ang88]. (Actually, she gives the equivalent dual result for DNF formulas.) We show here that when we expand the hypothesis space to arbitrary monotone DNF formulas, there is an efficient algorithm for learning the class of read-k CNF formulas with a number of membership queries polynomial in the DNF size and n, for k some fixed constant.

2 PRELIMINARIES

Let $V = \{v_1, \ldots, v_n\}$ be a collection of Boolean variables. A Boolean function $f(v_1, \ldots, v_n)$ is a function $f : \{0,1\}^n \to \{0,1\}$. A (monotone, since no negations are allowed) *term* t is the function represented by a conjunction (AND) $t = v_{i_1} \wedge v_{i_2} \wedge \cdots \wedge v_{i_s}$ of variables. The term t evaluates to 1 if and only if each of the variables $v_{i_1}, v_{i_2}, \ldots, v_{i_s}$ have value 1. Similarly, a (monotone) clause c is the function represented by a disjunction (OR) $c = v_{j_1} \vee v_{j_2} \vee \cdots \vee v_{j_m}$ of variables. The clause c evaluates to 1 if and only if at least one of the variables $v_{j_1}, v_{j_2}, \ldots, v_{j_m}$ has value 1.

A monotone DNF expression is a disjunction (OR) of monotone terms $t_1 \vee t_2 \vee \cdots \vee t_a$, and evaluates to 1 iff at least one of the terms has value 1. If $T = \{t_1, \ldots, t_a\}$ is a set of terms, then $\vee T$ is the DNF expression $t_1 \vee t_2 \vee \cdots \vee t_a$. Similarly, a monotone CNF expression is a conjunction of monotone clauses $c_1 \wedge c_2 \wedge \cdots \wedge c_b$, and evaluates to 1 iff each of the clauses has value 1. If $C = \{c_1, \ldots, c_b\}$ is a set of clauses then $\wedge C$ is the CNF expression $c_1 \wedge c_2 \wedge \cdots \wedge c_b$.

A term t implies a function f iff for any boolean assignment $\vec{v}$ to the variables of V, $(t(\vec{v}) = 1) \to (f(\vec{v}) = 1)$. Any such term is called an implicant. A prime implicant, or *minterm*, of f, is an implicant t such that no implicant of f can be formed by removing one or more variables from the conjunction t. It is easily shown and well-known that every monotone Boolean function has a unique monotone DNF expression, formed by the disjunction of all of its minterms. We call such a DNF expression *reduced.*

A hypergraph H is a pair $H = (E, V)$, where V is a finite set of vertices, and $E \subseteq 2^V$ is a set of subsets ("hyperedges") of V. A subset $I \subseteq V$ is *independent* in H if for every $e \in E$, $e - I \neq \emptyset$. An independent set I is maximal if no superset of I is also an independent set of H.

A *hitting set* (or *vertex cover*) of a hypergraph H is a subset of vertices $V' \subseteq V$ such that for each edge $e \in E$, $|e \cap V'| \geq 1$. A hitting set S is minimal if no proper subset of S is also a hitting set of H.

It is well known and easily observed that I is a maximal independent set of H if and only if $V - I$ is a minimal hitting set of H. Consequently, the problem of generating all maximal independent sets trivially reduces to that of generating all minimal hitting sets. (In the literature, generating all minimal hitting sets is also referred to as the hypergraph transversal problem [EG95].)

It is now easy to see that this problem is also equivalent to the following problem: Given a monotone CNF expression C, find an equivalent (reduced) monotone DNF D. To see this, let $H = (V, E)$ be a hypergraph. We form a CNF expression C whose variables are named by the vertices of V. For each edge $e = \{v_1, \ldots, v_s\} \in E$, define the clause $c_e = (v_1 \vee v_2 \vee \cdots \vee v_s)$. Let $C = \bigwedge\{c_e : e \in E\}$. It is now easily argued that $\{v_{j_1}, \ldots, v_{j_s}\}$ is a minimal hitting set of H if and only if $v_{j_1} \wedge \cdots \wedge v_{j_s}$ is a minterm in the DNF representation of C. Consequently, the problem of generating all minimal hitting sets of H (and, as above, the problem of generating all maximal independent sets of H) reduces to that of generating all minterms of the DNF for C. Since C is monotone, generating all the minterms of the DNF for C is in turn equivalent to finding the unique reduced DNF expression equivalent to C.

3 A FIRST STAB

Throughout the rest of the paper, we will be manipulating DNF/CNF expressions. We assume that any such expression is monotone, and that it has been put into reduced (minimal) form.

We first give a simple inefficient algorithm for the general case which motivates the efficient solution to the bounded degree (read-k) case. If $C = c_1 \wedge \cdots \wedge c_m$ is a monotone CNF formula (for which we wish to find an equivalent DNF representation), we construct the DNF for C inductively by constructing the DNF expressions for $c_1 \wedge \cdots \wedge c_i$ for each $i \leq m$. Assume inductively that $g = t_1 \vee \cdots \vee t_s$ is the (unique, since C is monotone) DNF for $c_1 \wedge \cdots \wedge c_i$. Then the DNF formula for $c_1 \wedge \cdots \wedge c_i \wedge c_{i+1}$ is equivalent to

$$\begin{aligned} g \wedge c_{i+1} &= (t_1 \vee \cdots \vee t_s) \wedge c_{i+1} \\ &= (t_1 \wedge c_{i+1}) \vee \cdots \vee (t_s \wedge c_{i+1}) \end{aligned}$$

The above is not quite in DNF form since each disjunct is not necessarily a term. Each disjunct can be translated to a collection of terms by a simple application of the distributive property. We define the function "term-and-clause" that takes a term t and a clause $c = (y_1 \vee \cdots \vee y_m)$ as input and returns their conjunction as follows: term-and-clause(t, c) =

$$\begin{cases} \quad t & \text{if } t \text{ and } c \text{ share a variable} \\ (t \wedge y_1) \vee \cdots \vee (t \wedge y_m) & \text{otherwise} \end{cases}$$

It is easy to see that the function term-and-clause returns the disjunction of its arguments: Independent of whether t and c share a variable, their conjunction, by the distributive property, is $(t \wedge y_1) \vee \cdots \vee (t \wedge y_m)$. In the special case that t and c do share a variable, say y_1, the conjunction $t \wedge y_1 = t$. Further, in this case t

would subsume the remaining terms, and consequently the disjunction would be equivalent to t.

We also find useful the function "dnf-and-clause" that simply takes a (reduced) DNF formula and clause as input, and returns the (reduced) result of calling term-and-clause with each term of the DNF. Thus,

$$\text{dnf-and-clause}(D, c) = \bigvee_{t \in D} \text{term-and-clause}(t, c).$$

We now have an algorithm to construct a DNF formula equivalent to the CNF formula $c_1 \wedge \cdots \wedge c_{i+1}$, given a DNF formula $t_1 \vee \cdots \vee t_s$ for $c_1 \wedge \cdots \wedge c_i$. We simply call dnf-and-clause on the input $(t_1 \vee \cdots \vee t_s, c_{i+1})$. Doing the above for each i yields our "first stab" algorithm at translating a CNF formula to a DNF formula:

```
first-stab($C = c_1 \wedge \cdots \wedge c_m$)
        $D \leftarrow$ True
        for $i$:= 1 to $m$
            $D \leftarrow$ dnf-and-clause($D, c_i$)
        output $D$
```

Observe that the order in which clauses are processed is irrelevant to first-stab's correctness, since after the last clause is processed, the DNF D produced is equivalent to the original CNF C. But, first-stab is not necessarily efficient: Let

$$C = \bigwedge_{i,j \in \{1,\ldots,n\}} (x_i \vee y_j).$$

Suppose the order in which the clauses are considered is $(x_i \vee y_i)$, $i = 1, \ldots, n$ followed by the remaining clauses (in any order). Then, after the first n clauses, the DNF formula D obtained by first-stab is exactly:

$$D = \bigvee_{b_i = x_i \text{ or } y_i} (b_1 \wedge \cdots \wedge b_n)$$

There are 2^n terms in D, yet the DNF formula equivalent to C has only 2 terms, namely,

$$(x_1 \wedge \cdots \wedge x_n) \vee (y_1 \wedge \cdots \wedge y_n).$$

In summary, first-stab works correctly regardless of the ordering of clauses, but is not guaranteed to do so in polynomial total time.

4 THE BOUNDED DEGREE CASE

We now show how in some circumstances we can impose an order on the clauses so that the sizes of the intermediate DNFs remain small. The result implies a polynomial-time algorithm for the bounded degree hypergraph independent set problem.

Let C be a monotone CNF formula over variables $\{x_1, \ldots, x_n\}$, and for each i between 0 and n define

$$C_i(x_1, \ldots, x_n) = C(x_1, \ldots, x_i, 1, \ldots, 1).$$

Thus, C_i is just C with all variables indexed greater than i hardwired, or projected, to 1. Note that C_0 is the constant 1 function, represented by the empty set of clauses, and that $C_n = C$. It is readily apparent that the CNF C_i is obtained from C by removing any clause containing a variable in $\{x_{i+1}, \ldots, x_n\}$. (If no clauses remain then C_i is equivalent to the constant 1 function.) Analogously, if D_i is the DNF for C_i, then D_i is obtained from the DNF D for C by removing each of the variables $\{x_{i+1}, \ldots, x_n\}$ from any term in which it participates. (If an empty term results, the DNF becomes the constant 1 function.) We thus have the following

Observation 1 *If C, D, C_i, and D_i are defined as above, then for $0 \leq i \leq n$, $|C_i| \leq |C|$ and $|D_i| \leq |D|$.*

The example in Section 3 demonstrated that if the clauses of C are processed by dnf-and-clause in a "bad" order, an intermediate CNF C', containing a subset of clauses of C, might have a DNF, D' that is exponentially larger than the size of the actual DNF D for C. Observation 1 points out that if C' is in fact a projection of C, then the size of D' will be bounded by the size of D.

By a *safe* stage of first-stab, we mean a stage where the terms that have been passed to dnf-and-clause so far correspond to a projection of C. As long as we can ensure that the time spent by the algorithm in between these safe stages is small, we can guarantee that there is not enough opportunity for the algorithm to construct a DNF that is too much larger than the actual DNF D.

We can employ these observations to our advantage in the case of read-k CNF formulas. We order the clauses in such a way so that after at most every kth clause passed to dnf-and-clause, the intermediate formula C' corresponds to some projection C_i of C, hence is a safe stage. In between safe stages, the intermediate formula does not have a chance to grow by a factor of more than $O(n^k)$, where n is the number of variables, and k is constant.

Algorithm read-k-cnf-to-dnf (Figure 1) begins with the projection C_0 that sets all variables to 1. The algorithm then iteratively "unprojects" each variable x_i such that at most k clauses of C that had been projected away because of x_i, now "reappear". We show that Algorithm read-k-cnf-to-dnf works correctly in polynomial total time.

Theorem 2 *Let C be a read-k monotone CNF formula over n variables. Then read-k-cnf-to-dnf(C) outputs the DNF formula D equivalent to C in time $O(|D| \cdot n^{k+1})$.*

Proof: We first argue that the algorithm halts with the correct output and then discuss the algorithm's efficiency. Let P_i be the set of clauses of C_i that are not contained in C_{i-1}. Step 3 is executed by read-k-cnf-to-dnf for each value of i between 1 and n. During such a step, for each clause c in P_i, the statement $G \leftarrow$ dnf-and-clause(G, c) is executed. Since the clauses of C are exactly the disjoint union $P_1 \cup P_2 \cup \cdots \cup P_n$, it follows from the discussion in Section 3 and the correctness of first-stab that the algorithm processes all clauses, and outputs the DNF equivalent to C.

To see that the algorithm runs within the stated time bound, first note that for each k, $1 \leq k \leq n$,

```
read-$k$-cnf-to-dnf($f$)

Input: read-$k$ CNF monotone formula $C$
Output: DNF formula $D$ equivalent to $C$

  1. $G \leftarrow$ True
  2. for $i = 1$ to $n$
  3.     for each clause $c$ of $C_i$ that is not in $C_{i-1}$
  4.         $G \leftarrow$ dnf-and-clause($G, c$)
  5. return $G$
```

Figure 1: An algorithm to obtain a DNF representation of a read-k CNF formula

after the iteration of the for loop of step 2 with $i = k$, the algorithm is at a safe stage, because the DNF returned is the projection D_i. We need to show that the formula does not grow too large in between safe stages. Since C (hence C_i) is read-k, *during* any execution of the for loop in step 3 dnf-and-clause is called at most k times, because there are at most k terms that are in P_i. Moreover, after each call of dnf-and-clause in step 4, the growth of the intermediate DNF G is bounded by a factor of n, since it is the result of multiplying G by a clause of size at most n. Consequently, after calling dnf-and-clause (at most) k times (i.e., until the next safe stage is reached), the size of each intermediate formula G between safe stages can grow to at most $|D| \cdot n^k$. Once the next safe stage is reached, the size of G is guaranteed again to be at most $|D|$. Since there are n iterations of the loop in step 2, and for each iteration of this loop, at most k calls are made to dnf-and-clause on a DNF formula of size at most $|D| \cdot n^k$, the total running time is bounded by $O(|D| \cdot n^{k+1})$. □

Observe that a dual statement can be made for read-k DNF formulas – namely, that there is a polynomial total time algorithm that converts read-k DNF formulas into their corresponding CNF formulas.

In addition, it is possible to show that if the CNF formula C has the property that each variable occurs at most k times or at least $|C| - j$ times, then we can efficiently find a DNF representation of C. For example, a formula where each variable appears in at most one clause or all but one clause is:

$$(x_1 \vee x_5)(x_2 \vee x_5 \vee x_6)(x_3 \vee x_5 \vee x_6)(x_4 \vee x_6)$$

(The variables x_1, x_2, x_3, x_4 appear in at most one clause and the variables x_5, x_6 appear in all but one clause.)

To see how to obtain the DNF for such a formula, note that the clauses of C can be partitioned into those that consist exclusively of variables that occur in at least $|C| - j$ clauses (call these T_{C-j}) and those that do not. If we unproject all the variables that occur in all but j clauses then we obtain a subset of the clauses in the CNF formula C, namely T_{C-j}. For any subset of the clauses of C, each variable appears in all but j clauses. Thus, for T_{C-j}, each variable appears in all but j clauses. Note that any minterm of a CNF formula where each variable occurs in all but j clauses has length at most $(j+1)$ — if x occurs in all but j clauses, including x in a term means that all but j clauses are hit, and there are at most j ways to hit these remaining clauses. Since T_{C-j} has a DNF representation where each term has length at most $(j+1)$, we can enumerate all such possible terms (there are $O(n^{j+2})$ such terms) and eliminate those that are not consistent. Using this DNF formula as a starting point for algorithm read-k-cnf-to-dnf in Figure 1, we can iteratively unproject the remaining read-k variables and call dnf-and-clause for each new term.

Claim 3 *Let C be a monotone CNF formula where each variable either appears in at most k clauses or all but j clauses. There exists an algorithm that finds a DNF representation of C in time $O(n^{j+2} + |g| \cdot n^{k+1})$.*

Finally, it is possible to show that the claim can be extended to the class of unate CNF formulas where each variable appears in at most k clauses or all but j clauses. (A formula is *unate* if each variable does not appear both negated and unnegated.)

5 MEMBERSHIP QUERIES

We now consider a more general version of the problem where the algorithm has access to a membership oracle only instead of the actual read-k CNF formula. Before giving an algorithm that uses membership queries exclusively, we present a few alternative approaches that assume we have C. These approaches shed light on how C can be replaced with a membership oracle.

The algorithm read-k-cnf-to-dnf constructs the DNF D_i from D_{i-1} by iteratively processing clauses in P_i (where, as in the proof of Theorem 2, P_i is the collection of clauses in C_i that are not in C_{i-1}). An alternative is to process the clauses in P_i *all at once* to first obtain a DNF for the conjunction of clauses of P_i, which we denote by DNF($\wedge P_i$), and then conjoin DNF($\wedge P_i$) to D_{i-1}. Stated more precisely, D_i is characterized by:

$$D_i = \text{DNF}[D_{i-1} \wedge \text{DNF}(\wedge P_i)]$$

The first two methods of computing D_i are, in fact, based on finding $DNF(\wedge P_i)$ in a different manner.

Exhaustive Method: The obvious way to find the formula DNF($\wedge P_i$) is to simply multiply out the clauses in P_i. The critical observation here is that $|P_i| \leq k$ since C is read-k. Consequently, repeatedly distributing clauses over clauses (each of length $\leq n$) requires time $O(n^k)$. Conjoining D_{i-1} (whose size is bounded by $|D|$) with a formula of length n^k requires time $O(|D| \cdot n^k)$. After eliminating redundant terms, we know that $|D_i| \leq |D|$, and we are once again at a safe stage, prepared to compute D_{i+1}.

k-DNF Method: Since $|P_i| \leq k$, it is easy to see that each term in DNF($\wedge P_i$) has length at most k. Thus, P_i has a k-DNF representation. DNF($\wedge P_i$) is easily computed by the following characterization:

$$\mathrm{DNF}(\wedge P_i) = \bigvee\{t : |t| \leq k \text{ and } t \rightarrow P_i\}$$

The remainder of the computation of D_i proceeds in a manner similar to the Exhaustive Method.

Projection Method: Since each term in D_i is a conjunction of a term in D_{i-1} and some term of length at most k, every implicant u of D_i (equivalently, of C_i) is in the set

$$U = \{s \wedge t : s \in D_{i-1}, |t| \leq k, t \subseteq \{x_1, \ldots, x_i\}\}.$$

Thus, we can enumerate every such $u \in U$ (there are only $O(|D| \cdot n^k)$ such terms) and include u in D_i if and only if u implies C_i. If C is available, we can determine C_i, and check whether or not u implies C_i immediately.

Up to this point we have assumed that the CNF C is provided to the algorithm. We turn now to the question of how C can be replaced with a membership oracle. Actually, we begin by showing how we can replace C with a membership oracle for C_i. Using the Projection Method, we can test whether or not u implies C_i by simply testing whether or not the characteristic vector χ_u of u is a positive example of C_i. (The *characteristic vector* of a term t contains a 1 in position i if x_i appears in t and a 0 in position i otherwise.) While a membership oracle for C_i is not available, we can simulate such an oracle with a membership oracle for C, since by definition

$$C_i(x) = C(x_1, \ldots, x_i, 1, \ldots, 1).$$

Summarizing, the algorithm proceeds as follows: For each projection C_i for i ranging from 0 to n, using the DNF we have computed from D_{i-1}, we test if $u \rightarrow C_i$ where u is the conjunction of some term from D_{i-1} and a term of length at most k over the variables $\{x_1, \ldots, x_i\}$. We determine if u is an implicant of C_i by posing a membership query on the example y, where the jth bit of y is $(\chi_u)_j$ if $j \leq i$ and 1 otherwise. We keep the term u in D_i if and only if the membership query returns true. The algorithm is given in Figure 2. Thus, we have:

Theorem 4 *The class of monotone functions f expressible as read-k CNF formulas is learnable with membership queries alone in time $O(|DNF(f)| \cdot n^{k+2})$.*

ACKNOWLEDGEMENTS

We would like to thank Dan Oblinger for entertaining our numerous, random musings, and Heikki Mannila for his encouragement and for his comments on an earlier draft.

References

[ABK+97] H. Aizenstein, A. Blum, R. Khardon, E. Kushilevitz, L. Pitt, and D. Roth. On learning read-k-satisfy-j dnf. *To appear,* SIAM Journal on Computing. *Preliminary version appears in Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory*, 1997.

[AHHPar] H. Aizenstein, T. Hegedus, L. Hellerstein, and L. Pitt. Complexity theoretic hardness results for query learning. *Computational Complexity*, To appear.

[AHK93] Dana Angluin, Lisa Hellerstein, and Marek Karpinski. Learning read-once formulas with queries. *Journal of the ACM*, 40(1):185–210, January 1993.

[Ang88] Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, April 1988.

[BCG+96] Nader H. Bshouty, Richard Cleve, Ricard Gavaldà, Sampath Kannan, and Christino Tamon. Oracles and queries that are sufficient for exact learning. *Journal of Computer and System Sciences*, 52(3):421–433, June 1996.

[BHH95a] Nader H. Bshouty, Thomas R. Hancock, and Lisa Hellerstein. Learning arithmetic read-once formulas. *SIAM Journal on Computing*, 24(4):706–735, August 1995.

[BHH95b] Nader H. Bshouty, Thomas R. Hancock, and Lisa Hellerstein. Learning Boolean read-once formulas over generalized bases. *Journal of Computer and System Sciences*, 50(3):521–542, June 1995.

[BI95] Jan C. Bioch and Toshihide Ibaraki. Complexity of identification and dualization of positive Boolean functions. *Information and Computation*, 123(1):50–63, 15 November 1995.

[Bsh95] Nader H. Bshouty. Exact learning Boolean functions via the monotone theory. *Information and Computation*, 123(1):146–153, 15 November 1995.

[EG95] Thomas Eiter and Georg Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278–1304, December 1995.

[FK96] Michael L. Fredman and Leonid Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21(3):618–628, November 1996.

[GJ79] M. R. Garey and D. S. Johnson. Computers and intractability — A guide to the theory of NP-completeness. *Freeman; Bell Lab, Murray Hill NJ*, 1979.

[GK95] Vladimir Gurvich and Leonid Khachiyan. Generating the irredundant conjunctive and disjunctive normal forms of monotone boolean functions. *Technical Report, LCSR-*

```
find-DNF($i$)

Input: $i$
Output: DNF formula $D_i$ equivalent to $C_i$

If $i = 0$ return True
$D_{i-1} \leftarrow$ find-DNF($i - 1$)
$D_i \leftarrow \emptyset$
for each clause $u = s \wedge t$ where $s \in D_{i-1}$, $|t| \leq k$, $t \subseteq \{x_1, \ldots, x_i\}$
    $(q_u)_j = \begin{cases} (\chi_u)_j & \text{if } j \leq i \\ 1 & \text{otherwise} \end{cases}$
    if MQ($q_u$)=1 then $D_i \leftarrow D_i \cup \{u\}$
return $D_i$
```

Figure 2: An algorithm to obtain a DNF representation of a read-k CNF formula using membership queries only

TR-251, Dept of Computer Science, Rutgers University, Discrete Applied Math, to appear, August 1995.

[HMP97] Haym Hirsh, Nina Mishra, and Leonard Pitt. Version spaces without boundary sets. *AAAI*, 1997. To appear.

[JPY88] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.

[Kha95] Roni Khardon. Translating between horn representations and their characteristic models. *Journal of AI Research*, 3:349–372, 1995.

[KKS93] H. A. Kautz, M. J. Kearns, and B. Selman. Reasoning with characteristic models. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 34–39, Washington, DC, July 1993. AAAI Press.

[KM95] Jyrki Kivinen and Heikki Mannila. Approximate inference of functional dependencies from relations. *Theoretical Computer Science*, 149(1):129–149, 18 September 1995.

[KMR95] Roni Khardon, Heikki Mannila, and Dan Roth. Reasoning with examples: Propositional formulae and database dependencies. *Technical Report, TR-15-95, Harvard University*, 1995.

[KR94] R. Khardon and D. Roth. Reasoning with models. In *Proceedings of the 12th National Conference on Artificial Intelligence*, volume 2, pages 1148–1153, Seattle, Washington, July–August 1994. AAAI Press.

[KW85] Richard M. Karp and Avi Wigderson. A fast parallel algorithm for the maximal independent set problem. *Journal of the ACM*, 32(4):762–773, October 1985.

[LLK80] E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Generating all maximal independent sets: NP-hardness and polynomial-time algorithms. *SIAM Journal on Computing*, 9(3):558–565, August 1980.

[Mit82] Tom Mitchell. Generalization as search. *Art. Int.*, 18:203–226, 1982.

[MR92a] Mannila and Raiha. On the complexity of inferring functional dependencies. *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, 40, 1992.

[MR92b] H. Mannila and K.-J. Räihä. *The Design of Relational Databases.* Addison-Wesley, 1992.

[MT96] Heikki Mannila and Hannu Toivonen. On an algorithm for finding all interesting sentences. *Cybernetics and Systems, R. Trappl, ed.*, pages 973–978, 1996.

[PR94a] K. Pillaipakkamnatt and V. Raghavan. On the limits of proper learnability of subclasses of DNF formulas. In *Proc. 7th Annu. ACM Workshop on Comput. Learning Theory*, pages 118–129. ACM Press, New York, NY, 1994.

[PR94b] K. Pillaipakkamnatt and V. Raghavan. Read-twice DNF formulas are properly learnable. In *Computational Learning Theory: Eurocolt '93*, volume New Series Number 53 of *The Institute of Mathematics and its Applications Conference Series*, pages 121–132, Oxford, 1994. Oxford University Press.

[PV88] L. Pitt and L. Valiant. Computational limitations on learning from examples. *J. ACM*, 35:965–984, 1988.

[SK91] Bart Selman and Henry Kautz. Knowledge compilation using horn approximations. In Kathleen Dean, Thomas L.; McKeown, editor, *Proceedings of the 9th National Conference on Artificial Intelligence*, pages 904–909. MIT Press, July 1991.

[TIAS77] Shuji Tsukiyama, Mikio Ide, Hiromu Ariyoshi, and Isao Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM Journal on Computing*, 6(3):505–517, September 1977.