Machine Learning, 37, 89–110 (1999)


# Efficient Read-Restricted Monotone CNF/DNF Dualization by Learning with Membership Queries

CARLOS DOMINGO carlos@is.titech.ac.jp
*Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, Meguro-ku, Ookayama, 152-8522 Tokyo, Japan*

NINA MISHRA nmishra@hpl.hp.com
*Hewlett-Packard Laboratories, 1501 Page Mill Rd, MS1U-4A, Palo Alto, CA 94304*

LEONARD PITT pitt@cs.uiuc.edu
*Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801*

**Editor:** Goldman

**Abstract.** We consider exact learning monotone CNF formulas in which each variable appears at most some constant $k$ times ("read-$k$" monotone CNF). Let $f : \{0,1\}^n \to \{0,1\}$ be expressible as a read-$k$ monotone CNF formula for some natural number $k$. We give an incremental output polynomial time algorithm for exact learning both the read-$k$ CNF and (not necessarily read restricted) DNF descriptions of $f$. The algorithm's only method of obtaining information about $f$ is through membership queries, i.e., by inquiring about the value $f(x)$ for points $x \in \{0,1\}^n$. The algorithm yields an incremental polynomial output time solution to the (read-$k$) monotone CNF/DNF dualization problem. The unrestricted versions remain open problems of importance.

**Keywords:** dualization, monotone CNF/DNF, read-restricted formulas, output polynomial-time algorithms, incremental algorithms, learning monotone formulas, membership queries

## 1. Introduction

We begin by defining the basic problems addressed and then follow with some motivation and application to other areas. We consider Boolean functions that map points in the $n$-dimensional Boolean hypercube $\{0,1\}^n$ onto $\{0,1\}$. More specifically, we are interested in *monotone* Boolean functions. We say that $y = \langle y_1 y_2, \ldots y_n \rangle$ is *above* $x = \langle x_1 x_2, \ldots x_n \rangle$ in the Boolean hypercube iff for each $i$, $y_i \geq x_i$. Then a monotone Boolean function is one which satisfies the following property: if $f(x) = 1$, then $f(y) = 1$ for each $y$ above $x$. Monotone Boolean functions are known to have unique reduced CNF (Conjunctive Normal Form) and DNF (Disjunctive Normal Form) expressions.

This paper investigates two related problems. To facilitate their descriptions, we introduce the notion of a membership query oracle for $f$ which is an oracle that on query "$x$" returns the value $f(x)$.

*Monotone CNF conversion:* Given a monotone CNF formula, find its equivalent reduced DNF formula.

*Monotone function identification:* Let $f$ be a monotone Boolean function. Given a membership query oracle for $f$ find both the CNF and DNF representations of $f$.

A natural dual problem to the first is the monotone DNF conversion problem, that is, given a monotone DNF, find its equivalent CNF. As we shall see in Section 2.1, these problems are of equivalent complexity. That is, an algorithm for one can be used to solve the other with relatively little increase in running time. Henceforth, our reference to the CNF/DNF conversion problem, or "the conversion problem" means either of the two problems.

An efficient algorithm for the monotone function identification problem can be used to efficiently solve the monotone CNF conversion problem, since the input CNF expression for the conversion problem can be used to answer the required membership queries for the identification problem. Furthermore, results of Bioch and Ibaraki (1995) show that an efficient algorithm for the conversion problem also may be used to obtain an efficient algorithm for the identification problem.

What do we mean by an efficient algorithm? We will be more careful about defining efficiency in Section 2. For now we note that since the output for these algorithms may be significantly larger than their input, we allow our algorithms to run in time polynomial in the size of their output.

Whether or not an efficient algorithm exists for either the conversion or identification problems is currently an open question. Recently, Fredman and Khachiyan (1996) gave an $O(m^{o(\log m)})$ time algorithm for the identification problem, where $m$ is the sum of the size of the desired CNF and DNF. This provides evidence that neither of the two problems are likely to be NP-hard. Whether or not there is an algorithm that is polynomial in $m$ remains open.

Since an efficient solution to the general problem is not known, some research has focused on determining which natural subcases of the general problem have efficient solutions. For example, in the event that each clause of the CNF has at most two variables, efficient solutions have been given under various definitions of efficiency (Tsukiyama et al., 1977; Lawler, Lenstra, & Kan, 1980; Karp & Wigderson, 1985; Johnson, Papadimitriou, & Yannakakis, 1988). Extending this work, Eiter and Gottlob (1995) give an efficient algorithm for the case in which the size of each clause is bounded by some constant. Finally, Makino and Ibaraki have also shown that an efficient solution exists for the class of monotone formulas with "constant maximum latency" (Makino & Ibaraki, 1997, 1998).

The restriction considered in this paper is based on limiting the number of "reads" (occurrences of each variable) in a formula, a restriction that has been well-investigated in the learning-theory literature (Angluin, Hellerstein, & Karpinski, 1993; Pillaipakkamnatt & Raghavan, 1995; Bshouty, Hancock, & Hellerstein, 1995a, 1995b; Pillaipakkamnatt & Raghavan, 1996; Aizenstein et al., 1998a, 1998b). Previous work has shown that it is possible to identify an arbitrary monotone read-once $(\wedge, \vee)$ formula under a stronger notion of polynomial time (Angluin, Hellerstein, & Karpinski, 1993; Gurvich & Khachiyan, 1995). We show here (for the conversion problem) that given a read-$k$ monotone CNF expression one can efficiently find its DNF expression. (Henceforth, we will refer to this problem as the **read-$k$-CNF conversion problem**.) For the identification problem, we show that given access to only membership queries of a function that can be represented as a read-$k$

monotone CNF, one can efficiently find both its CNF and DNF expressions. (Henceforth, we will refer to this problem as the **read-$k$-CNF/DNF identification problem**.)

### *1.1. Motivation*

We now consider relationships between naturally arising problems and the unrestricted cases of the monotone CNF conversion and monotone function identification problems. In some cases, the read-$k$ restriction that allows for our polynomial-time solution corresponds to a natural restriction (and solution thereof) for the related problem.

***Database applications.*** In the context of data-mining, an algorithm for the conversion problem can be used to find all of the keys 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 of the interesting rules in a relation (Mannila & Toivonen, 1996). 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 (Mannila & Räihä, 1992a, 1992b; Kivinen & Mannila, 1995).

Our read-$k$ algorithm can be used to enumerate all of the keys of a single relation provided that no attribute participates in more than $k$ (constant) minimal keys. The time required is polynomial in the size of the relation and the number of minimal keys.

Another recently studied problem in data mining is that of finding *association rules* that hold in a relation (table) $T$ (Agrawal et al., 1996). Each row of the table $T$ typically contains binary data. Let $\alpha$ be a subset of the attributes of $T$ and let $X$ be a single attribute not in $\alpha$. Let $A$ denote the number of rows of $T$ for which each attribute of $\alpha$ has value 1. Let $B$ denote the number of rows of $T$ for which each attribute of $\alpha \cup \{X\}$ has value 1. Let $|T|$ denote the number of rows in $T$. Then $\alpha \rightarrow X$ is an association rule with "support" $\sigma$ and "confidence" $\delta$ provided that (1) $B/|T| \geq \sigma$, and (2) $B/A \geq \delta$.

Efficiently enumerating association rules has become an important topic in data mining. Typically, a heuristic approach is taken wherein one first enumerates all of the "frequent sets" of $T$ (Agrawal, Imielinski, & Swami, 1993; Agrawal & Srikant, 1994; Srikant & Agrawal, 1995; Agrawal et al., 1996; Gunopulos, Mannila, & Saluja, 1997; Mannila & Toivonen, 1997). A frequent set is any set of attributes $S$ such that the fraction of rows that have all attributes of $S$ set to 1 is at least $\sigma$. Efficient algorithms for the conversion problem would be useful as a heuristic for enumerating all maximal frequent sets (Gunopulos, Mannila, & Saluja, 1997). Our read-$k$ restriction translates in this context to finding maximal frequent sets assuming that each variable appears in at most $k$ of the sets.

A final database-related application is to the problem of enumerating all *minimal failing subqueries* of a given conjunctive query to a database. Consider a query to a student record database asking for the student IDs of all female senior students who took the course CS 372 in the Spring of 1993. If there are no such students, then the query is a failing query. The user then might broaden the search, successively dropping the query conditions that the student be female, be a senior, or that the course be in the Spring of 1993. However, it could be the case that the query fails because CS 372 is not even a valid course. It would be preferable for the user to determine this right away. That is, if the query returns the empty set, the user often would find it useful to know if the reason for the unsatisfiability of the

query is some smaller subset of the constraints in the query. A minimal failing subquery of a conjunctive query is just a minimal subset of the conjunctive constraints of a failing query to a database that also results in failure. It has recently been shown that the general problem of enumerating all such minimal failing subqueries is NP-hard (even when there are only polynomially many minimal failing subqueries) (Godfrey, 1997) when the database consists of more than a constant number of relations.[1] It is also shown how to enumerate failing subqueries in order of increasing size with time increasing exponentially in the size $k$ of the subqueries. In comparison, the result in this paper implies that if every attribute appears in at most $k$ minimal failing subqueries then there is an output polynomial time algorithm for enumeration.

***Graph-theory applications.*** The conversion problem is well motivated in graph theory as it is exactly the hypergraph independent set problem. 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 (Garey & Johnson, 1979), 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.

Another equivalent graph theoretic formulation is: Given a hypergraph $H$, enumerate all minimal vertex covers of $H$. A vertex cover (or hitting set) is a subset of vertices $V' \subseteq V$ that intersects each edge of the hypergraph. The minimal vertex covers are precisely the complements of the maximal independent sets. In the literature, generating all minimal vertex covers of a hypergraph is also referred to as the hypergraph transversal problem (Eiter & Gottlob, 1995). The read restriction we consider here in the CNF/DNF setting is equivalent to the natural restriction of limiting the degree of each vertex in the hypergraph in both the hypergraph transversal and independent set problems. Our result complements output polynomial time algorithms for versions of these hypergraph problems restricted by constant edge-size (Eiter & Gottlob, 1995).

***Reasoning and knowledge representation.*** Another example of the utility of the conversion 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, an efficient solution to the conversion problem could be used to efficiently generate a collection of characteristic models (Kautz, Kearns, & Selman, 1993; Khardon & Roth, 1994) to use in various reasoning tasks (for example, determining whether a query is entailed by a knowledge base) (Khardon, 1995; Khardon, Mannila, & Roth, 1999).

The conversion problem is also related to the problem of determining if a version space has converged. For a concept class $C$ the version space (Mitchell, 1982) induced by positive example set $P$ and negative example set $N$ is the set of concepts in $C$ consistent with $P$ and $N$. A version space $V$ has converged if $|V| = 1$. An efficient solution to the CNF/DNF conversion problem could be used to efficiently determine if a version space has converged for the class of monotone functions (Hirsh, Mishra, & Pitt, 1997).

The term *knowledge compilation* is used to denote methods by which one type of representation of knowledge is translated into another, so as to make it easier to use that knowledge. One example discussed in (Selman & Kautz, 1991) considers compiling an arbitrary (non-monotone) CNF formula $C$ into a pair of Horn formulas—one more general (say, $g$), and one more specific (say, $f$), than the formula being "compiled". While the problem of determining whether a CNF $C$ entails a clause $\alpha$ is NP-hard, the problem of determining whether a Horn formula $f$ or $g$ entails $\alpha$ has a polynomial-time solution. If we are lucky enough that $g$ entails $\alpha$, or if $f$ fails to entail $\alpha$, then $C$ must entail (respectively, fail to entail) $\alpha$. The CNF/DNF conversion problem is also a form of compilation since we are given a CNF (or DNF) and wish to compile that information into a DNF (respectively, CNF).

A desirable property of our polynomial time solution to the read-$k$ identification problem is it does not depend on the representation of the function provided to us (e.g., it could be an arbitrary Boolean formula). As long as that representation is polynomially evaluable and corresponds to a monotone read-$k$ CNF we can efficiently compile the representation into both its DNF and CNF form.

***Computational learning theory.*** The identification problem is also closely related to the concept learning problem studied in computational learning theory. The general concept learning problem is to learn to discriminate between objects that satisfy some unknown rule, or "concept", and those that do not. Let $X$ be the space of possible examples (e.g., $X = \{0, 1, \}^n$). An unknown concept $f$ classifies each point $x \in X$ either "+" or "−" indicating that $x$ is a positive or negative (respectively) example of $f$. The unknown concept $f$ is referred to as the target concept, and is often assumed to come from some known class of candidate concepts.

Typically, a learning algorithm obtains examples of $f$ either randomly from nature, or from a teacher, and is told which examples are positive and which are negative. Sometimes the learning algorithm is allowed to pose a membership query which is an example $x$ of its own choice, in response to which a teacher classifies $x$ as either a positive or negative example. Sometimes the algorithm is allowed to both obtain examples and pose membership queries.

The learnability of monotone Boolean functions has been widely studied under a variety of learning models. For example, an efficient algorithm exists for learning the class of monotone DNF formulas when both examples and membership queries are available (Angluin, 1988) (sketched in the appendix). Our result implies an efficient algorithm for learning monotone read-$k$ CNF formulas (and their corresponding DNF representations) using membership queries alone.

A more thorough review of applications of both the monotone CNF/DNF conversion problem and the monotone function identification problem can be found in (Eiter & Gottlob, 1995).

### *1.2. Overview*

The remainder of this paper is organized as follows. Section 2 reviews standard definitions, special terminology, complexity issues, and some related results on learning with

membership queries. In Section 3, we give an output polynomial time algorithm for the read-$k$-CNF conversion problem (finding a DNF expression equivalent to a given monotone read-$k$ CNF expression).

In Section 4, we solve in incremental polynomial time the apparently more difficult read-$k$-CNF/DNF identification problem (finding *both* the read-$k$-CNF expression, and the DNF expression, of a monotone Boolean function $f$ given only membership queries of $f$). While this problem appears to be more difficult than the read-$k$-CNF conversion problem, the results of Bioch and Ibaraki (1995) imply that the problems are in fact equivalent.

## 2. Preliminaries

### *2.1. Boolean formulas, etc.*

***Monotone functions.*** Let $V = \{v_1, \ldots, v_n\}$ be a collection of Boolean variables. A vector $v$ is any assignment of the variables in $V$ to 0 or 1, i.e., $v \in \{0,1\}^n$. We use the terms vector and assignment interchangeably. A Boolean function $f(v_1, \ldots, v_n)$ is a function $f : \{0,1\}^n \to \{0,1\}$. A *monotone Boolean function* $f$ has the property that if $f(x) = 1$ then for all $y \geq x$, $f(y) = 1$, where "$\geq$" is defined by the partial order induced by the $n$-dimensional Boolean hypercube $\{0,1\}^n$. Equivalently, $y \geq x$ means that each bit of the vector $y$ is at least as large as the corresponding bit of $x$. The class of Boolean formula $F$ over variable set $V = \{v_1, \ldots, v_n\}$ is defined inductively as follows: (1) each of the symbols $\{0, 1, v_1, \ldots, v_n\}$ is a Boolean formula over $V$; (2) if $F_1$ and $F_2$ are Boolean formulas over $V$ then so are $(F_1 \vee F_2)$, $(F_1 \wedge F_2)$; and (3) if $F_1$ is a Boolean formula over $V$ then so is $\overline{F_1}$. The class of *monotone Boolean formulas* $F$ is defined inductively in the same way, but using only rules (1) and (2). Thus, a monotone Boolean formula is one which contains no negation symbols. We use the terms formula and expression interchangeably. Each Boolean formula over $V$ describes a Boolean function of $n$ Boolean variables in the usual way, with the standard interpretation of the logical connectives $\vee$ (OR), $\wedge$ (AND) and $\neg$ (NOT).

A monotone Boolean function $f$ can be described by its *minimally positive assignments*. A minimally positive assignment of $f$ is a positive assignment with only negative assignments below it, i.e., a vector $v \in \{0,1\}^n$ such that $f(v) = 1$ and for all $u < v$ $f(u) = 0$. A monotone Boolean function can also be described dually by its *maximally negative assignments*, i.e., the vectors $u$ such that $f(u) = 0$ and for all $v > u$, $f(v) = 1$.

A *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 literals $v_{i_j}$, where a *literal* is either a variable $x_i$ or its negation $\overline{x_i}$. A term is monotone if all literals are un-negated variables. Henceforth, we consider only monotone terms. The (monotone) 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 $V$, $(t(\vec{v}) = 1) \rightarrow (f(\vec{v}) = 1)$. 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 $t$. A clause $c$ is implied by a function $f$ iff for any Boolean assignment $\vec{v}$ to the variables of $V$, $(f(\vec{v}) = 1) \rightarrow (c(\vec{v}) = 1)$. Any such clause is called an implicand. A prime implicand, or *maxterm*, of $f$, is an implicand $c$ such that no implicand of $f$ can be formed by removing one or more variables from the disjunction $c$.

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*. Likewise, every monotone Boolean function has a unique reduced monotone CNF expression, formed by the conjunction of all of its maxterms.

For a monotone function $f$, let cnf($f$) and dnf($f$) be its corresponding (unique reduced) CNF and DNF expressions, respectively. Let |cnf($f$)| denote the number of maxterms of cnf($f$) (similarly for the DNF representation). The length of the representation of cnf($f$) is at most $n \cdot |\text{cnf}(f)|$. For a CNF formula $C$, we denote by $|C|$ the number of maxterms of $C$ (similarly for a DNF formula $D$).

It turns out that each minimally positive assignment corresponds naturally to a minterm in the DNF expression. Likewise, each maximally negative point corresponds naturally to a maxterm in the CNF expression. For example, consider the following monotone Boolean function.

| $x_1$ | $x_2$ | $x_3$ | $f(x)$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

The minimally positive assignments of $f$ are 110 and 001, and the DNF expression for $f$ is $(x_1 \wedge x_2) \vee x_3$. Note that a minterm contains a variable $x_i$ whenever its corresponding minimally positive assignment contains a 1 in bit position $i$. The maximally negative assignments of $f$ are 010 and 100, and the CNF expression for $f$ is $(x_1 \vee x_3) \wedge (x_2 \vee x_3)$. Note that a maxterm contains a variable $x_i$ wherever its corresponding maximally negative assignment contains a 0 in bit position $i$.

***Duality.*** Recall that if $f$ is any Boolean formula, then the dual of $f$ (denoted dual(f)) is obtained by replacing each "$\vee$" with "$\wedge$", and vice-versa. (And, if $f$ contains the constants "0" or "1", each "0" is replaced with "1", and vice-versa.) Thus, the dual of a CNF is a DNF, and the dual of a DNF is a CNF. Moreover, the dual of a read-$k$-CNF is a read-$k$-DNF, and vice-versa. By the duality law (e.g., Tremblay & Manohar, 1961), any

identity or theorem about formulas remains true when the formulas are replaced by their duals.

Because of this duality, it is easily seen that an algorithm that solves the monotone CNF conversion problem can be modified in a straightforward way to solve the monotone DNF conversion problem. In particular, suppose $A$ is an algorithm for the monotone CNF conversion problem, and let $A(C)$ denote the DNF formula output by $A$ which is equivalent to an input CNF formula $C$. Suppose $D$ is a DNF formula we wish to "convert" to CNF. Then dual($D$) is a CNF, and $A$(dual($D$)) is the equivalent DNF for dual($D$). By the duality law, dual(dual($D$)) is equivalent to dual($A$(dual($D$))). But dual(dual($D$)) is just $D$. Consequently, to find a CNF equivalent to $D$ using algorithm $A$, we simply compute dual($A$(dual($D$))). The running time increases only by an additive linear term.

***Read-k.*** A Boolean formula is *read-k* if each variable appears at most $k$ times. Note that the property of being read-$k$ is a property of a formula, and not of the underlying function. We'll say that a function is read-$k$ if it can be represented by some read-$k$ formula. Similarly, a function is a read-$k$ CNF (respectively, DNF) if it can be represented as a read-$k$ CNF (respectively, DNF) formula. For a fixed $k$ not all Boolean functions are representable as a read-$k$ CNF formula. Note that every read-$k$ CNF formula has total length at most $k \cdot n$.

***Projections.*** Let $f$ be a Boolean function of $n$ variables $\{x_1, \ldots, x_n\}$. We define $f_{x_i \leftarrow b}$ as the projection of the Boolean function $f$ when variable $x_i$ is set to value $b \in \{0, 1\}$. That is, if $v = \langle v_1, v_2, \ldots, v_n \rangle$, then let $v_{x_i \leftarrow b} = \langle v_1, \ldots, v_{i-1}, b, v_{i+1}, \ldots, v_n \rangle$, and define $f_{x_i \leftarrow b}$ as the function such that for any $v$, $f_{x_i \leftarrow b}(v) = f(v_{x_i \leftarrow b})$. (Alternative definitions of projection consider $f_{x_i \leftarrow b}$ as a function of $n-1$ arguments $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$, whose values are defined on the subspace $\{0, 1\}^{i-1} \times \{b\} \times \{0, 1\}^{n-i}$.) Similarly, for a set of variables $A \subseteq \{x_1, \ldots, x_n\}$, we define $f_{A \leftarrow b}$ to be the projection of function $f$ when all the variables from set $A$ are set to value $b \in \{0, 1\}$. We will make use of such projections in the technical sections to follow.

### *2.2. Complexity issues*

The standard definition of a polynomial-time algorithm is one that runs in time polynomial in the size of its input. No such algorithm exists for the conversion problem since the output may be exponentially larger than the input. The following CNF formula of size $2n$ over the set $\{x_1, \ldots, x_n, y_1, \ldots, y_n\}$ of $2n$ variables exemplifies this behavior.

$$(x_1 \vee y_1) \wedge (x_2 \vee y_2) \wedge \cdots \wedge (x_n \vee y_n) \tag{1}$$

It is possible to show that the corresponding reduced DNF has size $2^n$:

$$\bigvee_{b_i \in \{x_i, y_i\}} (b_1 \wedge \cdots \wedge b_n)$$

Clearly there is no algorithm that can output a formula of size $2^n$ in time polynomial in $n$. This statement also applies to the class of read restricted CNF expressions since the formula in (1) is read-once.

Given that the output may be significantly larger than the input, a more reasonable measure of complexity allows an algorithm to run in time polynomial in both its input and output sizes. Such an algorithm is called a *total* (*or output*) polynomial time algorithm (Johnson, Papadimitriou, & Yannakakis, 1988; Eiter & Gottlob, 1995).

A stronger definition of a total polynomial time algorithm is an *incremental* polynomial time algorithm (Johnson, Papadimitriou, & Yannakakis, 1988; Bioch & Ibaraki, 1995; Eiter & Gottlob, 1995). In between all consecutive outputs an incremental polynomial time algorithm spends time polynomial in the input size $n$ and what it has output so far. Thus, if the algorithm outputs $o_1, o_2, \ldots, o_j$, where $o_j$ is the final output, then the time it takes to output $o_i$ for any $i \leq j$ is polynomial in $n$, the input size, and the sum of the sizes of $o_1, o_2, \ldots, o_{i-1}$. Clearly an algorithm that runs in incremental polynomial time also runs in total polynomial time. The converse does not necessarily hold.

In this paper, we first give a total polynomial time algorithm for the read-$k$-CNF conversion problem (Section 3) and then show (in Section 4) how this may be used to obtain an incremental polynomial time algorithm for the read-$k$-CNF/DNF identification problem. In Section 4 we also provide a second method for solving the read-$k$-CNF/DNF identification problem that is motivated by an analysis of some structural properties of read-$k$ CNF formulas.

### *2.3. Membership queries*

In the read-$k$-CNF/DNF identification problem we consider the problem of finding both the CNF and DNF expressions with membership queries. One may well wonder if it is possible to efficiently learn just one of the expressions, say the DNF, with membership queries.

In fact, results of Angluin (1988) show that no algorithm exists for learning the class of CNF formulas with membership queries in time polynomial in the size of the DNF formula. Her result implies also that there are information theoretic barriers to learning the class of read-$k$ DNF formulas with membership queries (in time polynomial in the size of the read-$k$ DNF). The result also holds for CNF.

Furthermore, it is known that any algorithm that learns the class of monotone functions with membership queries must pose $\Omega(\max\{|\text{cnf}(f)|, |\text{dnf}(f)|\})$ queries (where $f$ is the monotone function to be compiled) (Korobkov, 1965; Bshouty et al., 1996). The result also applies to the class of monotone functions representable as read-$k$ CNF formulas when $k \geq 2$.

Finally, we note that while there is currently no known *time-efficient* algorithm for learning the class of monotone functions with membership queries, the information-theoretic complexity of the problem is understood better. Gainanov (1984) has given a time-inefficient (but query-efficient) algorithm that identifies every monotone function $f$ with $O(n \cdot \max\{|\text{cnf}(f)|, |\text{dnf}(f)|\})$ membership queries.

## 3. The conversion problem

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

We give an algorithm for finding a DNF expression equivalent to a given monotone read-$k$ CNF expression. Note that the fact that the CNF expression is monotone implies that its corresponding DNF expression is also monotone. However, the fact that the CNF expression is read restricted does not necessarily imply that the DNF expression is read-restricted. The formula (1) is a read-once CNF with no equivalent read-$k$ DNF for $k < 2^{n-1}$.

### *3.1. A first stab*

We first give a simple but inefficient algorithm for the general conversion problem that motivates the efficient solution to the read-$k$ case. The techniques we use are based on an inductive characterization of the problem: When can the DNF corresponding to a *subset* $C'$ of the clauses of a monotone CNF formula $C$ be used to compute the DNF of $C$? We demonstrate that a possible source of inefficiency of the algorithm is that the size of the DNF for $C'$ may be significantly (exponentially) larger than the size of the DNF for $C$. However, if we impose an order on the clauses of $C$ by considering those induced by larger and larger subsets of the variables, we can (in Section 3.2) show that the method yields an efficient algorithm for finding a monotone DNF representation of a given monotone read-$k$ CNF formula.

If $C = c_1 \wedge \cdots \wedge c_m$ is a monotone CNF formula 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:

$$\text{term-and-clause}(t, c) = \begin{cases} t & \text{if } t \text{ and } c \text{ share a variable} \\ (t \wedge y_1) \vee \cdots \vee (t \wedge y_m) & \text{otherwise.} \end{cases}$$

It is easy to see that the function term-and-clause returns the conjunction of its arguments. Independent of whether $t$ and $c$ share a variable, their conjunction is $(t \wedge y_1) \vee \cdots \vee (t \wedge y_m)$ by the distributive property. If $t$ and $c$ share a variable then $t \rightarrow c$ and the conjunction $t \wedge c$ is equivalent to $t$.

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

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

Note that term-and-clause$(t, c)$ runs in time $O(|t| \cdot |c|)$. Since $|D|$ calls are made to term-and-clause on a term and clause with at most $n$ variables, dnf-and-clause requires time $O(|D| \cdot n^2)$

We also define the function "reduce" that given a DNF formula, $D$, (not necessarily reduced) and its corresponding CNF formula, $C$, reduces $D$ in time linear in $|C|$ and $|D|$. For a term $t$ in $D$, note that $t$ is an implicant of $D$. Also $t$ is an implicant of $C$ since $C$ is equivalent to $D$. So by definition for any Boolean assignment $x$, $(t(x) = 1) \rightarrow (C(x) = 1)$. We wish to determine if $t$ is a prime implicant of $D$ (or $C$). By definition, $t$ is a prime implicant iff $C(u) \neq 1$, where $u$ is the minimally positive assignment corresponding to $(t - \{v\})$. In summary, reduce does the following.

reduce$(D, C)$
    $D' \leftarrow \emptyset$.
    for all $t$ in $D$
      if for all $v$ in $t$
        the minimally positive assignment corresponding to $t - \{v\}$ does not satisfy $C$
        then $D' \leftarrow D' \cup t$
    return $D'$

Since $O(n|C|)$ time is needed per term of the DNF formula $D$, the running time of reduce$(D, C)$ is $O(|D| \cdot n|C|)$. Since the CNF formulas considered in this paper are read-$k$, the running time of reduce is $O(|D| \cdot n^2)$.

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 $(t_1 \vee \cdots \vee t_s, c_{i+1})$ and reduce the resulting formula. Doing the above iteratively for each $i$ yields our "first stab" algorithm for 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$(D, c_i)$
            reduce$(D, c_1 \wedge \cdots \wedge c_i)$
        output $D$

A simple induction on $i \leq m$ shows that after the $i$-th iteration of the for loop, the formula $D$ is a (reduced) DNF that represents the CNF formula $c_1 \wedge \cdots \wedge c_i$. Consequently, when $i = m$, first-stab$(C)$ correctly outputs a reduced DNF formula $D$ that represents the same function as $C$. Note that the inductive proof (hence correctness of the algorithm) is independent of the ordering of clauses $\{c_i\}$.

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:

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

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 it is not guaranteed to do so in total polynomial time because the DNFs for intermediate CNFs constructed using a subset of the clauses of $C$ can be large.

### *3.2. 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 transversal (and independent set) problem.

Let $C$ be a monotone CNF formula[2] over variables $\{x_1, \ldots, x_n\}$. 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 3.1 demonstrated that if the clauses of $C$ are processed by dnf-and-clause in a "bad" order an intermediate CNF, $C'$, 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 $k$th clause passed to dnf-and-clause, the intermediate formula $C'$ corresponds to some projection $C_i$ of $C$, i.e., is a safe stage. In between safe stages the intermediate formula does not have a chance to grow by a factor of more than $O(n^k)$.

Algorithm read-$k$-cnf-to-dnf 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".

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

*Input*: Read-$k$ CNF monotone formula $C$
*Output*: DNF formula $D$ equivalent to $C$
$G \leftarrow$ True
for $i = 1$ to $n$
for each clause $c$ of $C_i$ that is not in $C_{i-1}$
$G \leftarrow$ dnf-and-clause$(G, c)$
$G \leftarrow$ reduce$(G, C_1 \wedge \cdots \wedge C_i)$
return $G$

We show that Algorithm read-$k$-cnf-to-dnf works correctly in total polynomial 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 $O(|D| \cdot n^{k+3})$.*

**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. (Note that reducing the DNF $G$ affects the representation of the formula but not the function itself.) 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 3.1 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 $k$, $1 \leq k \leq n$, 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-$k$ there are at most $k$ clauses in $P_i$ (those that contain $x_i$). So *during* any execution of the for loop in step 3 dnf-and-clause is called at most $k$ times. Note that after each call of dnf-and-clause in step 4, the growth of the intermediate DNF $G$ is bounded by a factor of $n$. The reason is that $G$ is multiplied 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 the formula $D$ is reduced. Note that there are $n$ iterations of the loop in step 2. For each of these $n$ iterations, at most $k$ calls are made to dnf-and-clause, and to reduce, on a DNF formula of size at most $|D| \cdot n^k$. Each of dnf-and-clause require $O(|DNF| \cdot n^2)$ time, so the overall running time is $O(n \cdot k \cdot |D| \cdot n^k \cdot n^2) = O(|D|n^{k+3})$. □

Observe that a dual statement can be made for read-$k$ DNF formulas: There is a total polynomial time algorithm that converts read-$k$ DNF formulas into their corresponding CNF formulas. The statement follows from the duality law given in Section 2.1.

## 4. The identification problem

In this section, we construct an algorithm that improves the result given in the previous section in three ways: (1) The algorithm finds both the CNF and DNF of a function instead of finding the DNF given the CNF (2) The algorithm uses membership queries only rather than relying on representational information inherent in the CNF and (3) The algorithm runs in incremental polynomial time rather than total polynomial time.

### *4.1. The equivalence problem*

We first introduce a third problem which is simply that of determining if a given monotone CNF is equivalent to a given monotone DNF expression.

*Monotone CNF and DNF equivalence:* Let $C$ be a monotone CNF formula and let $D$ be a monotone DNF formula. Given $C$ and $D$, determine whether or not they are equivalent.

For our purposes we will be interested in the read-restricted version of this problem which we introduce below. The following has been proven for the general problem (restated in our terminology):

**Theorem 3 (Bioch and Ibaraki).** *There is a polynomial-time algorithm for the monotone CNF and DNF equivalence problem if and only if there is an incremental polynomial time algorithm for the monotone function identification problem.*

The proof appears in the appendix because an understanding of why the following corollary holds requires understanding Bioch and Ibaraki's proof.

The restriction of the equivalence problem to the read-$k$ case is as follows:

*Read-k Monotone CNF and DNF equivalence:* Let $C$ be a monotone read-$k$ CNF formula, and let $D$ be a monotone (not necessarily read-$k$) DNF formula. Given $C$ and $D$, determine whether or not they are equivalent.

While in general the proof of Theorem 3 does not apply to the read-restricted case, we show in the appendix that the proof can be appropriately modified.

**Corollary 4.** *There is a polynomial-time algorithm for the read-k monotone CNF and DNF equivalence problem if and only if there is an incremental polynomial time algorithm for the read-k-CNF/DNF identification problem.*

By Corollary 4, to obtain an incremental output polynomial time algorithm for the read-$k$-CNF/DNF identification problem we need only give a polynomial time algorithm for the read-$k$ monotone CNF and DNF equivalence problem.

### *4.2. Two algorithms for equivalence testing*

For several reasons, we include *two* polynomial-time algorithms for the read-$k$ monotone CNF and DNF equivalence problem. The first method follows easily via a reduction from the read-$k$-CNF conversion problem (Section 3). It demonstrates that the conversion problem is equivalent to the equivalence testing problem. This method will perhaps be most appreciated by those interested in the conversion problem or any of the equivalent formulations or related problems mentioned in the introduction. The second algorithm directly attacks the read-$k$ monotone CNF and DNF equivalence problem, and gives insight into some structural properties of read-$k$ CNF (and DNF) formulas. It should further be noted that as the unrestricted versions of the conversion, identification, and equivalence problems remain open, we find it useful to provide a variety of possible methods for further attack.

***4.2.1. By reduction.*** The proof of Theorem 2 gives a total polynomial time algorithm for the read-$k$-CNF conversion problem. We now show that this implies a polynomial time algorithm for the read-$k$ monotone CNF and DNF equivalence problem. Consequently, by Corollary 4, this gives an incremental polynomial time algorithm for the read-$k$-CNF/DNF identification problem.

**Theorem 5.** *There is a polynomial time algorithm for the read-k monotone CNF and DNF equivalence problem if and only if there is a total polynomial time algorithm for the read-k-CNF conversion problem.*

**Proof:** $\boxed{\leftarrow}$ Let $T$ be an algorithm for the read-$k$-CNF conversion problem that runs in time $p(|\text{cnf}(f)|, |\text{dnf}(f)|, n)$ where $p$ is some polynomial. To determine if the CNF formula $C$ and the DNF formula $D$ input to the equivalence problem are in fact equivalent, we run $T$ with input $C$ for $p(|C|, |D|, n)$ time steps.

Suppose $D$ was in fact the DNF representation of $C$. Then, since $T$ is assumed to produce the correct (minimized) output, it will halt with output $D$ in $p(|C|, |D|, n)$ steps. The reason is that since $f$ is the function represented by $C$, $|C| = |\text{cnf}(f)|$ and in fact $|D| = |\text{dnf}(f)|$. If $D$ is not the DNF representation of $C$ then one of two things may happen. $T$ may halt with incorrect output or $T$ may not finish running in $p(|C|, |D|, n)$ steps. Thus $D$ is equivalent

to $C$ iff $T$ halts in $p(|C|, |D|, n)$ steps and its output is equivalent to $D$. If $T$ halts with a DNF formula $D'$ then output "yes" or "no" depending on whether $D$ and $D'$ are equivalent. If $T$ does not halt then output "no" since obviously $C$ and $D$ are not equivalent.

$\boxed{\rightarrow}$ By Corollary 4, an algorithm for the equivalence problem implies an incremental polynomial time algorithm for the read-$k$-CNF/DNF identification problem. Clearly an incremental polynomial time algorithm for the read-$k$-CNF/DNF identification problem implies a total polynomial time algorithm for the read-$k$-CNF conversion problem. □

***4.2.2. Directly.*** Here we give a more direct method for testing the equivalence of a read-$k$ monotone CNF formula $C$ and an arbitrary monotone DNF $D$. The algorithm runs in time polynomial in the sum of the sizes of $C$ and $D$.

Before giving the main theorem of this section, we need the following definition and results adapted from the ones in (Hancock & Mansour, 1991).

*Definition 1.* A function $f : \{0,1\}^n \rightarrow \{0,1\}$ *depends* on the variable $x_i$ if there exists an assignment $y \in \{0,1\}^n$ such that $f_{x_i \leftarrow 0}(y) \neq f_{x_i \leftarrow 1}(y)$.

*Definition 2.* Let $f$ be a monotone function over variable set $\{x_1, \ldots, x_n\}$. We say that $A \subseteq \{x_1, \ldots, x_n\} - \{x_i\}$ is a *blocking set* for $x_i$ in $f$ if $f_{A \leftarrow 1}$ does not depend on variable $x_i$. A blocking set $A$ for $x_i$ is *minimal* if no proper subset of $A$ is also a blocking set for $x_i$. When the variable $x_i$ is clear from context, we simply refer to $A$ as a blocking set.

We explain intuitively what it means for a set of variables to block a formula $D$ (instead of a function). A set of variables $A$ blocks $x_i$ in $D$ if projecting the variables of $A$ in $D$ to 1 renders $x_i$ irrelevant. For example, let $D = (x_1 \wedge x_3) \vee (x_2 \wedge x_3)$. Note that projecting $x_2$ to 1 causes $D$ to be equivalent to $x_3$. In other words, $x_1$ is not needed when $x_2$ is set to 1 in $D$. Thus we say that $A = \{x_2\}$ blocks $x_1$ in $D$.

The idea behind our algorithm is that if we verify that the blocking sets for each variable $x_i$ of a CNF formula $C$ are also blocking sets for $x_i$ in $D$, then we will have checked enough projections to verify that $C = D$. Further, since $C$ is a read-$k$ CNF formula, the number of such projections is not too large.

To test whether a given (monotone) read-$k$ CNF formula $C$ is equivalent to a given DNF formula $D$, it is sufficient to test whether $D \rightarrow C$, and, assuming $D \rightarrow C$, test whether $C \rightarrow D$. Checking whether $D \rightarrow C$ is immediate: it is sufficient to test for each minterm of $D$ whether the corresponding minimal positive assignment satisfies $C$. To test whether $C \rightarrow D$, we will use the following lemma:

**Lemma 6.** *Let $C$ and $D$ be nonconstant monotone CNF and DNF formulas respectively. Further suppose that $D \rightarrow C$. Then the following two statements are equivalent*:
1. $C \rightarrow D$.
2. *For every variable $x_i$, every blocking set for $x_i$ in $C$ is a blocking set for $x_i$ in $D$.*

**Proof:** Suppose that the hypothesis of the lemma holds and that $C \rightarrow D$. But then $C$ is equivalent to $D$, and clearly each blocking set for each variable $x_i$ in $C$ is also a blocking

set for $x_i$ in $D$, since the definition of blocking set depends only on the function, and not the representation.

Conversely, suppose that every blocking set for each $x_i$ in $C$ is a blocking set for $x_i$ in $D$, and suppose by way of contradiction that $C$ does not imply $D$, that is, there is an assignment $y$ such that $C(y) = 1$ and $D(y) = 0$. Let $Y_1$ be the set of variables assigned "1" by $y$, and let $Y_0$ be the set of variables assigned "0" by $y$. Notice that $Y_1$ is a blocking set for any $x_i \in Y_0$ in $C$, since setting each variable in $Y_1$ to 1 forces $C$ to 1. Starting at $y$, iteratively flip 0-bits to 1, until the first assignment $p$ is obtained such that $D(p) = 1$. (There must be such an assignment because $D$ is not the constant 0 function and therefore $D(\bar{1}) = 1$.) Then $p$ witnesses that $D_{Y_1 \leftarrow 1}$ depends on some variable $x_i$ in $Y_0$ (the last one flipped), hence $Y_1$ is not a blocking set for $x_i$ in $D$, contradicting the assumption that every blocking set for $C$ was also one for $D$. □

Thus, to test whether or not $C$ implies $D$, assuming[3] that $D$ implies $C$, we need only obtain for each variable $x_i$ the set of all blocking sets for $x_i$ in $C$, and test whether they are also blocking sets for $x_i$ in $D$. In fact, we need only check that every minimal blocking set for $x_i$ in $C$ is a blocking set for $x_i$ in $D$, since any superset of a blocking set for $x_i$ is also a blocking set for $x_i$. What remains then, is a method to (1) efficiently list all minimal blocking sets for $C$, and (2) efficiently test whether each is a blocking set for $D$.

(1) *Obtaining the blocking sets:* Suppose that $C = c_1 \wedge \cdots \wedge c_s$ is a nonempty read-$k$ monotone CNF. We will show how to obtain the set of all minimal blocking sets for $x_1$ in $C$. The blocking sets for the rest of the variables $x_i$ are obtained similarly. First, we rewrite $C$ as $C = (x_1 \vee C^1) \wedge C^2$ where $C^1$ is the conjunction of the clauses of $C$ that contain $x_1$ with variable $x_1$ factored out, and $C^2$ contains the rest of the clauses of $C$. Notice that if all the clauses in $C$ contain $x_1$, then $C^2$ is the identically 1 function. We claim that the minterms of $C^1$ are exactly the minimal blocking sets for $x_1$ in $C$. (That is, for each minterm $t$ of $C^1$, the set of variables appearing in $t$ is a minimal blocking set for $x_1$, and if $A$ is a minimal blocking set for $x_1$, then the term formed by the conjunction of the variables in $A$ is a minterm of $C^1$.)

To prove the claim, we'll show that every implicant of $C^1$ is a blocking set for $x_1$ in $C$, and that every blocking set for $x_1$ in $C$ is an implicant of $C^1$. That the minterms (minimal implicants) correspond to the minimal blocking sets follows trivially.

Suppose first that $t$ is an implicant of $C^1$. Let $p$ be any assignment satisfying $t$ (hence $C^1$) with $x_1$ set to 0. If $p$ satisfies $C^2$, then $C(p) = 1$ and $C(p_{x_1 \leftarrow 1}) = 1$. If $p$ doesn't satisfy $C^2$, then neither does $p_{x_1 \leftarrow 1}$, since no clause of $C^2$ contains $x_1$. Thus, both $C(p)$ and $C(p_{x_1 \leftarrow 1}) = 0$. In other words, in any assignment with each variable of $t$ set to 1, flipping $x_1$ does not change the value of $C$. Equivalently, $C_{t \leftarrow 1}$ does not depend on $x_1$, so $t$ is a blocking set for $x_1$ in $C$.

Conversely, suppose that $A$ is a blocking set for $x_1$ in $C$. Suppose by way of contradiction that (the term whose variables are those in) $A$ is not an implicant of $C^1$. Then there must be some clause $\alpha$ of $C^1$ whose variable set is disjoint from $A$, since otherwise each clause in $C^1$ would be satisfied when the variables of $A$ were set to 1, and $A$ would be an implicant. Now let assignment $p$ set each variable in $\alpha \cup \{x_1\}$ to 0, and all other variables to 1. Note

that each variable in $A$ is set to 1 by $p$, since $A$ contains neither $x_1$ nor any variable in $\alpha$. Now, $p$ must satisfy $C^2$, because otherwise, there would be a clause $\beta \in C^2$ such that $\beta \subseteq \alpha \cup \{x_1\}$, contradicting the choice of $C$ as reduced. Observe that $C(p) = 0$, since $p$ satisfies neither $C^1$ (because of $\alpha$) nor $x_1$. But $C(p_{x_1 \leftarrow 1}) = 1$, since $p$ satisfies $C^2$. This contradicts the assumption that $A$ was a blocking set for $x_1$ in $C$. We conclude that $A$ must in fact be an implicant of $C^1$, and that the set of minterms of $C^1$ are exactly the set of minimal blocking sets for $x_1$ in $C$.

To find the minterms (hence minimal blocking sets) of $C^1$, observe that by the read-$k$ property of $C$, $C^1$ is a $k$-clause CNF formula and each minterm therefore has size at most $k$. We can enumerate all $O(n^k)$ terms of size at most $k$, and for each, test in linear time whether it is a prime implicant of $C^1$. Note that our blocking set approach does not work for arbitrary (not read-restricted) $C$ since in general we cannot bound the number of minterms of $C^1$.

(2) *Testing if each blocking set is a blocking set for $D$:* To test whether a minimal blocking set $A$ for a variable $x_i$ in $C$ is also a blocking set for $x_i$ in $D$, we simply set each variable of $A$ to 1 in the formula $D$, and simplify to obtain $D_{A \leftarrow 1}$. We then reduce $D_{A \leftarrow 1}$ by removing any subsumed terms. Since $D_{A \leftarrow 1}$ is now reduced, it will depend on $x_i$ if and only if $x_i$ appears in at least one term.

We now can state the following theorem:

**Theorem 7.** *There is a polynomial time algorithm to determine given a monotone read-$k$ CNF $C$ and a monotone DNF $D$ whether or not $C$ and $D$ are equivalent.*

**Proof:** We observed above that testing whether $D \rightarrow C$ is trivial. By Lemma 6, to determine whether $C \rightarrow D$, it is sufficient to check if every minimal blocking set for each variable in $C$ is also a blocking set for the same variable in $D$. Above, we showed how to (1) obtain all minimal blocking sets for $C$, and (2) determine whether each was also a blocking set for $D$. The running time of all steps was polynomial in $|C|$, $|D|$, and $n$. □

## Appendix

**Proof of Theorem 3:** An incremental polynomial time algorithm for the monotone function identification problem can be used to test monotone CNF/DNF equivalence as follows. Given CNF $C$ and DNF $D$, to determine if $C$ is equivalent to $D$, run the algorithm for the identification problem. When it asks for the value $f(x)$, give it $C(x)$. $C$ and $D$ are equivalent if and only if both $C$ and $D$ have been output within $p(|C|, |D|, n)$ steps, where $p(|\text{cnf}(f)|, |\text{dnf}(f)|)$ is the *total* amount of time that the incremental algorithm spends on a given $f$.

The converse is more interesting. First, we describe the *exact learning with queries* framework due to Angluin (1988). In this learning framework, a learner tries to learn a function $f$ from a known class $\mathcal{C}$ by asking queries to a teacher that knows $f$. In this paper we restrict to the problem of learning Boolean formulas, and therefore, the class $\mathcal{C}$ will

be a subclass of the class of all Boolean formulas. The goal of the learner is to output a hypothesis $h$ that is logically equivalent to $f$ by getting information about $f$ via queries.

There are two kinds of queries that have been commonly used in the learning literature and that we consider here: membership and equivalence queries. The input of a membership query is a Boolean assignment $x \in \{0, 1\}^n$ and the answer is the value $f(x)$. An equivalence query takes as input a hypothesis $h$ and returns "yes" if $h$ is equivalent to $f$ and otherwise an arbitrary counterexample $y \in \{0, 1\}^n$ such that $h(y) \neq f(y)$ is returned.

One of the early positive results in this model is due to Angluin (1988) (see also Valiant, 1984) and states that the class of monotone DNF formulas is exactly learnable using membership and equivalence queries. The time and number of queries is polynomial in the number of terms of the DNF $D$ to be learned. It will be helpful to have a basic understanding of this learning algorithm.

Let $D$ be a monotone DNF over $n$ variables and $m$ terms. The algorithm works by iteratively and greedily collecting all the terms of $D$. It starts with the empty formula representing the identically 0 function. The algorithm maintains a hypothesis that implies $D$. In fact, each hypothesis $h$ will consist of a subset of the terms of $D$. (Hence, $h$ implies $D$.) The algorithm uses an equivalence query to generate a new counterexample to $h$, which by the above comment, must be an assignment $y$ such that $h(y) = 0$ and $D(y) = 1$. Because this positive example $y$ of $D$ is not above any minterm of $h$, an assignment $y' \leq y$ (in the Boolean hypercube) can be found using at most $n$ (the number of variables) membership queries, such that $h(y') = 0$ and $D(y') = 1$, and such that no $y'' < y$ has this property. It is easily seen that $y'$ corresponds to a minterm of $D$ that is not yet in $h$. Consequently, the algorithm adds that minterm to $h$ and iterates the above process with another equivalence query. After $m$ iterations of this process, $h = D$. We will denote this algorithm by $\mathcal{A}$.

It is easy to observe that algorithm $\mathcal{A}$ can be "inverted" to obtain a dual algorithm $\mathcal{A}^\delta$ that exactly learns any monotone CNF formula $C$. In this case, the algorithm starts with the set of empty clauses, representing the identically 1 function. It maintains a hypothesis that is always implied by $C$. Each equivalence query is used to obtain a negative example. With membership queries this negative example will allow $\mathcal{A}^\delta$ to obtain a maximum negative example from which a clause is constructed. This clause is added to the current hypothesis. The algorithm will use as many iterations as clauses in $C$.

Now suppose that we have an algorithm *Equiv* for testing the equivalence of a monotone DNF, $h$, and CNF, $h^\delta$, formula in time $q(|h|, |h^\delta|, n)$ where $q$ is some polynomial. We show how to use *Equiv* to solve the identification problem in incremental polynomial time. Let $f$ be the function to be "identified". We assume we have an oracle for $f$ that returns the value $f(x)$ on query $x$. To find both its DNF and CNF representations, we will run both $\mathcal{A}$ and $\mathcal{A}^\delta$ in parallel, using the oracle for $f$ to answer membership queries, until both of them stop and ask an equivalence query. Let $h$ be the hypothesis output by $\mathcal{A}$ and let $h^\delta$ be the hypothesis output by $\mathcal{A}^\delta$. By the properties of these algorithms just mentioned, $h$ implies $f$, $f$ implies $h^\delta$ and thus, $h$ implies $h^\delta$. Now we run *Equiv* with $h$ and $h^\delta$ as inputs. If the answer of *Equiv* is "yes", then $h$ and $h^\delta$ are logically equivalent, and since $h \Rightarrow f \Rightarrow h^\delta$ we conclude that we have obtained both the DNF and CNF expressions for $f$.

Otherwise, if *Equiv* answers "no", then we use an algorithm *Ctrx* (described below) to obtain a counterexample $x$ such that $h(x) \neq h^\delta(x)$. We use a membership query with input

$x$ to determine which hypothesis is misclassifying $x$ and pass $x$ as a counterexample to the algorithm that delivered that hypothesis. That algorithm will resume its computation and at some point it will output a new hypothesis. We then proceed in the same fashion. In other words, we can combine both algorithms to avoid asking equivalence queries by using algorithms *Equiv* and *Ctrx*. However, now the number of membership queries will be polynomial in the *sum* of the sizes of the monotone DNF and monotone CNF formulas for $f$.

How can we implement algorithm *Ctrx*? We describe here how algorithm *Ctrx* can be easily constructed using algorithm *Equiv*. First, we choose one variable, say $x_1$, and we run *Equiv* twice with inputs $h_{x_1 \leftarrow 0}$ and $h^{\delta}_{x_1 \leftarrow 0}$ in one case and inputs $h_{x_1 \leftarrow 1}$ and $h^{\delta}_{x_1 \leftarrow 1}$ in the other. Note that the formulas corresponding to $h_{x_1 \leftarrow 0}$ and $h_{x_1 \leftarrow 1}$ can be obtained by simply projecting $x_1$ to 1 and 0 (respectively) whenever the variable $x_1$ appears in the formula. Also note that after projecting $x_1$, the formulas $h_{x_1 \leftarrow 0}$ and $h_{x_1 \leftarrow 1}$ are monotone DNF formulas. Similar remarks apply to the CNF representations of $h^{\delta}_{x_1 \leftarrow 0}$ and $h^{\delta}_{x_1 \leftarrow 1}$.

Since we already know that $h$ and $h^{\delta}$ are not equivalent, at least one of the answers of the two runs is "no". Suppose that Equiv($h_{x_1 \leftarrow 1}$, $h^{\delta}_{x_1 \leftarrow 1}$) returns "no". Then we know that there is a counterexample where the first bit is set to 1. To obtain the next bit of a counterexample, we call Equiv twice with inputs $h_{x_1 \leftarrow 1, x_2 \leftarrow 1}$, $h^{\delta}_{x_1 \leftarrow 1, x_2 \leftarrow 1}$ and $h_{x_1 \leftarrow 1, x_2 \leftarrow 0}$, $h^{\delta}_{x_1 \leftarrow 1, x_2 \leftarrow 0}$. Again, at least one of these calls to Equiv must return "no". The run that fails provides information about the second bit of the counterexample. We can repeat the above procedure with further projected formulas so that after at most $2n$ calls we will have obtained a complete counterexample.

To analyze the time complexity, note that obtaining a counterexample $x$ requires time $O(n \cdot q(|h|, |h^{\delta}|, n))$ since $2n$ calls to Equiv are made on formulas of size at most $|h|$ and $|h^{\delta}|$. Determining which hypothesis misclassifies $x$ requires one membership query to obtain its true classification plus time linear in $|h|$ and $|h^{\delta}|$ to determine which hypothesis $h$ or $h^{\delta}$ needs to be modified. Thus the time needed to produce the next output (minterm or maxterm) is $2n \cdot q(|h|, |h^{\delta}|, n) + |h| + |h^{\delta}|$. Since $|h| + |h^{\delta}|$ is the size of what the algorithm has output so far, and the algorithm requires time polynomial in $|h| + |h^{\delta}|$ to produce the next output, the algorithm runs in incremental polynomial time. □

**Proof of Corollary 4:** In general Theorem 3 does not immediately apply to an arbitrary subset of CNF formulas. In the case of read-$k$-CNF formulas any hypotheses posed by the combined $\mathcal{A}$ and $\mathcal{A}^{\delta}$ algorithm will also be a read-$k$ monotone CNF. The reason is that any hypothesis of this algorithm is a subset of the clauses of the read-$k$ monotone CNF to be learned. The remainder of the proof is unchanged. □

**Acknowledgments**

We thank Dan Oblinger for entertaining our numerous, random musings, Tibor Hegedus for pointing out Theorem 5 and other comments, Heikki Mannila for his encouragement and for his comments on an earlier draft, and the referees for their insightful comments.

## Notes

1. In the case of a single relation the problem is easily shown to be equivalent to the conversion problem. Given the recent results of Fredman and Khachiyan (1996) the single relation case is unlikely to be NP-complete.
2. We assume that $C$ is not the constant 0 function for the remainder of this paper. The conversion problem is trivial when $C$ is the constant 0 function since $D$ is also the constant 0 function.
3. The assumption that $D \rightarrow C$ in the hypothesis of Lemma 6 is necessary. It is not difficult to construct an example without this assumption for which the first statement holds but not the second.

## References


Agrawal, R., Imielinski, T., & Swami, A. (1993). Mining associations between sets of items in massive databases. *Proc. of the ACM-SIGMOD 1993 International Conference on Management of Data* (pp. 207–216). Washington, DC.

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., & Verkamo, I. (1996). Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining* (pp. 307–328). Menlo Park, CA: AAAI Press.

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. *Proc. of the 20th International Conference on Very Large Databases*. Santiago, Chile.

Aizenstein, H., Blum, A., Khardon, R., Kushilevitz, E., Pitt, L., & Roth, D. (1998a). On learning read-*k*-satisfy-*j* DNF. *SIAM Journal on Computing*, *27*(6), 1515–1530.

Aizenstein, H., Hegedus, T., Hellerstein, L., & Pitt, L. (1998b). Complexity theoretic hardness results for query learning. *Computational Complexity*, *7*, 19–53.

Angluin, D. (1988). Queries and concept learning. *Machine Learning*, *2*(4), 319–342.

Angluin, D., Hellerstein, L., & Karpinski, M. (1993). Learning read-once formulas with queries. *Journal of the ACM*, *40*(1), 185–210.

Bioch, J., & Ibaraki, T. (1995). Complexity of identification and dualization of positive Boolean functions. *Information and Computation*, *123*(1), 50–63.

Bshouty, N., Cleve, R., Gavaldà, R., Kannan, S., & Tamon, C. (1996). Oracles and queries that are sufficient for exact learning. *Journal of Computer and System Sciences*, *52*(3), 421–433.

Bshouty, N., Hancock, T., & Hellerstein, L. (1995a) Learning arithmetic read-once formulas. *SIAM Journal on Computing*, *24*(4), 706–735.

Bshouty, N., Hancock, T., & Hellerstein, L. (1995b). Learning Boolean read-once formulas over generalized bases. *Journal of Computer and System Sciences*, *50*(3), 521–542.

Eiter, T., & Gottlob, G. (1995). Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, *24*(6), 1278–1304.

Fredman, M., & Khachiyan, L. (1996). On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, *21*(3), 618–628.

Gainanov, D.N. (1984). On one criterion of optimality of an algorithm for evaluating monotonic boolean functions. *USSR Computational Mathematics and Mathematical Physics*, *24*, 176–181.

Garey, M., & Johnson, D. (1979). Computers and intractability—A guide to the theory of NP-completeness. W.H. Freeman and Company.

Godfrey, P. (1997). Minimization in cooperative response to failing queries. *International Journal of Cooperative Information Systems World Scientific*, *6*(2), 95–149.

Gunopulos, D., Mannila, H., & Saluja, S. (1997). Discovering all most specific sentences by randomized algorithms. In F. Afrati & P. Kolaitis (Eds.), *Proceedings of International Conference on Database Theory*. Delphi, Greece.

Gurvich, V., & Khachiyan, L. (1995). Generating the irredundant conjunctive and disjunctive normal forms of monotone boolean functions. (Technical Report, LCSR-TR-251). Dept. of Computer Science, Rutgers University, Discrete Applied Math, to appear.

Hancock, T., & Mansour, Y. (1991). Learning monotone $k\mu$-DNF formulas on product distributions. *Proc. 4th Annu. Workshop on Comput. Learning Theory* (pp. 179–183). San Mateo, CA: Morgan Kaufmann.

Hirsh, H., Mishra, N., & Pitt, L. (1997). Version spaces without boundary sets. *Proceedings of the 14th National Conference on Artificial Intelligence* (pp. 491–496).

Johnson, D., Papadimitriou, C., & Yannakakis, M. (1988). On generating all maximal independent sets. *Information Processing Letters*, *27*(3), 119–123.

Karp, R., & Wigderson, A. (1985). A fast parallel algorithm for the maximal independent set problem. *Journal of the ACM*, *32*(4), 762–773.

Kautz, H., Kearns, M., & Selman, B. (1993). Reasoning with characteristic models. *Proceedings of the 11th National Conference on Artificial Intelligence* (pp. 34–39). Washington, DC: AAAI Press.

Khardon, R. (1995). Translating between horn representations and their characteristic models. *Journal of AI Research*, *3*, 349–372.

Khardon, R., Mannila, H., & Roth, D. (1999). Reasoning with examples: Propositional formulae and database dependencies. *Acta Informatica*, *36*(4), 267–286.

Khardon, R., & Roth, D. (1994). Reasoning with models. *Proceedings of the 12th National Conference on Artificial Intelligence*, (Vol. 2, pp. 1148–1153). Seattle, Washington: AAAI Press.

Kivinen, J., & Mannila, H. (1995). Approximate inference of functional dependencies from relations. *Theoretical Computer Science*, *149*(1), 129–149.

Korobkov, V. (1965). On monotone functions of the algebra of logic. *Problemy Kibernetiki*, *13*, 5–28.

Lawler, E., Lenstra, J., & Kan, A. Rinnooy (1980). Generating all maximal independent sets: NP-hardness and polynomial-time algorithms. *SIAM Journal on Computing*, *9*(3), 558–565.

Makino, K., & Ibaraki, T. (1997). The maximum latency and identification of positive Boolean functions. *SIAM Journal on Computing*, *26*, 1363–1383.

Makino, K., & Ibaraki, T. (1998). A fast and simple algorithm for identifying 2-monotonic positive Boolean functions. *Journal of Algorithms*, *26*(2), 291–305.

Mannila, H., & Räihä, K. (1992a). On the complexity of inferring functional dependencies. *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, *40*.

Mannila, H., & Räihä, K. (1992b). *The Design of Relational Databases*. Addison-Wesley.

Mannila, H., & Toivonen, H. (1996). On an algorithm for finding all interesting sentences. In R. Trappl (Ed.), *Cybernetics and Systems* (pp. 973–978).

Mannila, H., & Toivonen, H. (1997). *Levelwise Search and Borders of Theories in Knowledge Discovery*. (Report C-1997-8). University of Helsinki, Department of Computer Science.

Mitchell, T. (1982) Generalization as search. *Artificial Intelligence*, *18*, 203–226.

Pillaipakkamnatt, K., & Raghavan, V. (1995). Read-twice DNF formulas are properly learnable. *Information and Computation*, *122*(2), 236–267.

Pillaipakkamnatt, K., & Raghavan, V. (1996). On the limits of proper learnability of subclasses of DNF formulas. *Machine Learning*, *25*, 237–263.

Selman, B., & Kautz, H. (1991). Knowledge compilation using horn approximations. In Kathleen Dean, Thomas L., & McKeown (Eds.), *Proceedings of the 9th National Conference on Artificial Intelligence* (pp. 904–909). MIT Press.

Srikant, R., & Agrawal, R. (1995). Mining generalized association rules. *Proceedings of the 21st International Conference on Very Large Databases*. Zurich, Switzerland.

Tremblay, J.P., & Manohar, R. (1961). *Discrete Mathematical Structures with Applications to Computer Science*. McGraw-Hill.

Tsukiyama, S., Ide, M., Ariyoshi, H., & Shirakawa, I. (1977). A new algorithm for generating all the maximal independent sets. *SIAM Journal on Computing*, *6*(3), 505–517.

Valiant, L. (1984). A theory of the learnable. *Commun. ACM*, *27*(11), 1134–1142.

Received
Accepted
Final Manuscript