On the Proper Learning of Axis Parallel Concepts

Nader H. Bshouty *  
Department of Computer Science  
Technion  
Haifa, Israel

Lynn Burroughs †  
Department of Computer Science  
Calgary University  
Calgary, Alberta, Canada

March 19, 2002

Abstract

We study the proper learnability of axis parallel concept classes in the PAC learning model and in the exact learning model with membership and equivalence queries. These classes include union of boxes, DNF, decision trees and multivariate polynomials.

For the constant dimensional axis parallel concepts \( C \) we show that the following problems have the same time complexity

1. \( C \) is \( \alpha \)-properly exactly learnable (with hypotheses of size at most \( \alpha \) times the target size) from membership and equivalence queries.
2. \( C \) is \( \alpha \)-properly PAC learnable (without membership queries) under any product distribution.
3. There is an \( \alpha \)-approximation algorithm for the \( \text{MIN\textsc{EQUI}} \) problem. (given a \( g \in C \) find a minimal size \( f \in C \) that is equivalent to \( g \)).

In particular, \( C \) is \( \alpha \)-properly learnable in polynomial time from membership and equivalence queries if and only if \( C \) is \( \alpha \)-properly PAC learnable in polynomial time under the product distribution if and only if \( \text{MIN\textsc{EQUI}} \) has a polynomial time \( \alpha \)-approximation algorithm. Using this result we give the first proper learning algorithm of decision trees over the constant dimensional domain and the first negative results in proper learning from membership and equivalence queries for many classes.

For the non-constant dimensional axis parallel concepts we show that with the equivalence oracle (1) \( \Rightarrow \) (3). We use this to show that (binary) decision trees are not properly learnable in polynomial time (assuming \( P \neq \text{NP} \)) and DNF is not \( s' \)-properly learnable (\( \epsilon < 1 \)) in polynomial time even with an NP-oracle (assuming \( \Sigma^P_2 \neq P^{\text{NP}} \)).

*This research was supported by the fund for promotion of research at the Technion. Research no. 120-025. Part of this research was done at the University of Calgary, Calgary, Alberta, Canada and supported by NSERC of Canada.

†This research was supported by an NSERC PGS-B Scholarship, an Izak Walton Killam Memorial Scholarship, and an Alberta Informatics Circle of Research Excellence (iCORE) Fellowship.
1 Introduction

We study the proper learnability of axis parallel concept classes in the PAC-learning model and in the exact learning model with membership and equivalence queries. A class $\mathcal{N}$-$\mathcal{P}$ of axis parallel concepts is a class of boolean formulas $\phi(T_1, T_2, \ldots, T_l)$ where $\phi$ is from a class of boolean formulas $\Phi$ (such as DNF, decision tree, etc.) and $\{T_i\}$ are boxes in $\mathcal{N}_m$, where $\mathcal{N}_m = \{0, \ldots, m-1\}$, that satisfy a certain property $\mathcal{P}$ (such as disjointness, squares, etc.). These classes include union of boxes, union of disjoint boxes, exclusive or of boxes, decision tree partition, and for the boolean case $\mathcal{N}_2^n$, they include DNF, decision trees, disjoint DNF and multivariate polynomials.

The term $\alpha$-proper learning refers to learning where the intermediate hypotheses used by the learner (in the equivalence queries) and the final hypothesis have size (number of boxes $T_i$) at most $\alpha$ times the size of the target formula. A class is properly learnable if it is 1-learnable. The following table summarizes the results for the $n$-dimensional boolean case.

<table>
<thead>
<tr>
<th>Upper for</th>
<th>Complexity</th>
<th>Source</th>
<th>Lower</th>
<th>Condition</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNF</td>
<td>Nonproper</td>
<td>$2^{O(n^{1/2})}$</td>
<td>[KS01]</td>
<td>Proper</td>
<td>$P \neq NP$</td>
</tr>
<tr>
<td>DNF</td>
<td>Nonproper</td>
<td>NP-oracle</td>
<td>[BCG96]</td>
<td>Proper</td>
<td>$P \neq NP$</td>
</tr>
<tr>
<td>DNF</td>
<td>Proper</td>
<td>$\Sigma^p_1$-oracle</td>
<td>OPEN</td>
<td>$\alpha$-Proper</td>
<td>$\Sigma^p_\alpha \neq P^{NP}$</td>
</tr>
<tr>
<td>CDNF</td>
<td>Nonproper</td>
<td>poly($n$)</td>
<td>[B95]</td>
<td>Proper</td>
<td>$P \neq NP$</td>
</tr>
<tr>
<td>CDNF</td>
<td>Proper</td>
<td>$\Sigma^p_1$-oracle</td>
<td>[HPRW96]</td>
<td>Proper</td>
<td>$P \neq NP$</td>
</tr>
<tr>
<td>Disj-DNF</td>
<td>Nonproper</td>
<td>poly($n$)</td>
<td>[BCV96]</td>
<td>Proper</td>
<td>$P \neq NP$</td>
</tr>
<tr>
<td>DT</td>
<td>Nonproper</td>
<td>poly($n$)</td>
<td>[B95]</td>
<td>Proper</td>
<td>$P \neq NP$</td>
</tr>
<tr>
<td>DT</td>
<td>Proper</td>
<td>$\Sigma^p_1$-oracle</td>
<td>OPEN</td>
<td>Proper</td>
<td>$P \neq NP$</td>
</tr>
<tr>
<td>MP</td>
<td>Nonproper</td>
<td>poly($n$)</td>
<td>[BCV96]</td>
<td>Proper</td>
<td>$P \neq NP$</td>
</tr>
<tr>
<td>MMP</td>
<td>Proper</td>
<td>poly($n$)</td>
<td>[SS93]</td>
<td>Proper</td>
<td>$P \neq NP$</td>
</tr>
</tbody>
</table>

Hellerstein et al. [HPRW96] show that proper learnability of a class $C$ from membership and equivalence queries is possible in a machine with unlimited computational power if and only if $C$ has polynomial certificates. They also show that if $C$ has a polynomial certificate then $C$ is properly learnable using an oracle for $\Sigma^p_4$. They then give a polynomial size certificate for CDNF (a polynomial size DNF that has a polynomial size CNF). This implies that CDNF is properly learnable using an oracle for $\Sigma^p_4$. For DNF, decision trees (DT), Disjoint DNF (DNF where the conjunction of every two terms is 0) and multivariate polynomials (with nonmonotone terms (MP)) it is not known whether they have polynomial certificates. Therefore it is not known if they are properly learnable. Pillaiapakkamnatt and Raghavan [PR96] show that if DNF is properly learnable then $P = NP$. On the other hand, Bshouty et al. [BCG96] show that any circuit is (nonproperly) learnable with equivalence queries only and the aid of NP-oracle. The best algorithm today for learning DNF runs in time $2^{O(n^{1/2})}$ [KS01].

CDNF, Decision trees, disjoint DNF and multivariate polynomials are (nonproperly) learnable in polynomial time from membership and equivalence queries [B95, BCG96, BBBKV00]. Multivariate polynomials with monotone terms (MMP) are properly learnable [SS93].

2
In this paper we use a new technique for finding negative results for learning from membership and equivalence queries (see Theorem 1). We use Theorem 1 and the result of Zantema and Bodlaender [ZB00] to show that if a decision tree is properly learnable from membership and equivalence queries then P=NP. We then use the result of Umans [U99] and show that if DNF is s-properly learnable with an NP-oracle, where s is the size of the DNF, then $\Sigma^p_2 = P^{NP}$. We show our results are still true even if the learner can use other oracles such as Subset, Superset, Disjointness, etc. Therefore, if P\neq NP then decision trees and DNF are not properly learnable from membership and equivalence queries (and all the other oracles defined in subsection 2.3).

The following table summarizes our results for axis parallel classes over $\mathcal{N}_m^n$ for a constant dimension n.

<table>
<thead>
<tr>
<th>Union of t Boxes</th>
<th>Positive results</th>
<th>Negative results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>poly(log m)</td>
<td>iff P=NP</td>
</tr>
<tr>
<td>Disjoint Union Boxes</td>
<td>dim=2 Proper</td>
<td>dim&gt;2 Proper</td>
</tr>
<tr>
<td>Decision tree</td>
<td>Proper</td>
<td></td>
</tr>
<tr>
<td>Xor of Boxes</td>
<td>$\alpha$-Proper</td>
<td>OPEN</td>
</tr>
</tbody>
</table>

For axis parallel classes over a constant dimension we show that these classes have polynomial certificates. Therefore by [HPRW96], they are properly learnable from membership and equivalence queries using the $\Sigma^p_4$ oracle. We further investigate the learnability of these classes and show that an NP-oracle is sufficient for proper learnability. We also show that the following problems have the same time complexity.

1. C is $\alpha$-properly exactly learnable from membership and equivalence queries.
2. C is $\alpha$-properly PAC learnable (without membership queries) under any product distribution.
3. There is an $\alpha$-approximation algorithm for the MIN_EQUI problem (given a $g \in C$ find a minimal size $f \in C$ that is equivalent to $g$).
4. C is exactly learnable with a learning algorithm that uses all the queries (membership and nonproper equivalence, subset, superset, etc.) and outputs a hypothesis that has size at most $\alpha$ times the target size.

We start with some surprising results that follow from this. The first is (1)$\Rightarrow$(2). It is known that (proper) learnability from equivalence and membership queries implies (proper) learnability in the PAC model with membership queries [A88]. Here we show that in the case of finite dimensional space and for the product distribution we can change a weak learner (that learns with membership queries) to a strong learner (that learns without membership queries). Another surprising result that we show from this is: a decision tree over any constant dimension is properly learnable from membership and equivalence queries. We also show that decision tree is proper PAC-learnable under any distribution.

We also show that union of disjoint DNF in dimension 2 has a polynomial time proper learning algorithm. On the other hand, union of boxes and disjoint union of boxes over dimensions greater than 2 are properly learnable if and only if P=NP. Union of boxes is
log \( t \)-properly learnable where \( t \) is the number of boxes and Xor of boxes is \( \alpha \)-properly learnable for some constant \( \alpha \).

All the results in the literature for the constant dimensional domain give nonproper learning of the above classes in the exact learning model and there were no negative results for proper learning of these classes from membership and equivalence queries.

In [CM94] Chen and Maass give a proper exact learning of one box from equivalence queries. Beimel and Kushilevitz [BK98] show that \( \mathcal{N}^n_m \)-Disjoint DNF is (nonproperly) learnable from membership and equivalence queries. This result is also true for the nonconstant dimensional domain. The output hypothesis is represented as a \( \mathcal{N}^n_m \)-Multiplicity Automaton.\(^1\) Since \( \mathcal{N}^n_m \)-Multiplicity Automaton contains the class of \( \mathcal{N}^n_m \)-Multivariate Polynomials [BBBK00], the class of \( \mathcal{N}^n_m \)-Multivariate Polynomials is (nonproperly) learnable in polynomial time from membership and equivalence queries. In [BGGM99] Bshouty et al. give an \( O(d \log t) \)-proper learning algorithm that learns a union of \( t \) boxes in \( d \)-dimensional space. The algorithm in this paper is \( \log \log t \)-proper.

There are many algorithms in the literature that learn the union of boxes in the constant dimensional space [CH96, MW98] (and even any combination of thresholds in the constant dimensional space from equivalence queries only [BBK97, B98]). All of these algorithms are nonproper and return hypotheses that have large size.

2 Preliminaries

2.1 Learning Models

The learning criteria we consider are exact learning and PAC-learning.

In the exact learning model there is a function \( f \) called the target function \( f : \mathcal{N}^n_m \rightarrow \{0, 1\} \) (where \( \mathcal{N}_m = \{0, 1, \ldots, m-1\} \)), which has a formula representation in a class \( C \) of formulas defined over the variable set \( V_n = \{x_1, \ldots, x_n\} \). The goal of the learning algorithm is to halt and output a formula \( h \in C \) that is equivalent to \( f \).

The learning algorithm performs a membership query by supplying an assignment \( a \) to the variables in \( V_n \) as input to a membership oracle and receives in return the value of \( f(a) \). For our algorithms we will regard this oracle as a procedure \( MQ_f(a) \). The procedure’s input is an assignment \( a \) and its output is \( MQ_f(a) = f(a) \).

The learning algorithm performs an equivalence query by supplying a formula \( h \in C \) as input to an equivalence oracle with the oracle returning either “YES”, signifying that \( h \) is equivalent to \( f \), or a counterexample, which is an assignment \( b \) such that \( h(b) \neq f(b) \). For our algorithms we will regard this oracle as a procedure \( EQ_f(h) \).

We say that a class \( C \) of boolean functions is \( \alpha \)-properly exactly learnable from membership and equivalence queries in polynomial time if there is an algorithm that for any \( f \in C \) over \( V_n \), the algorithm runs in polynomial time, asks a polynomial number (polynomial in \( n, \log m \) and in the size of the target function) of membership and equivalence queries with hypothesis \( h \in C \) of size at most \( \alpha \) times the size of the target, and outputs a hypothesis

\( ^1 \)One can also define \( \mathcal{N}^n_m \)-Multiplicity Automaton. Using the algorithm from that paper, \( \mathcal{N}^n_m \)-Multiplicity Automaton is properly learnable from membership and equivalence queries.
\( h \in C \) that is equivalent to \( f \). The size of \( h \) is at most \( \alpha \) times the size of the target. We say that \( C \) is \textit{properly exactly learnable} if it is 1-properly exactly learnable.

The PAC learning model is as follows. There is a distribution \( D \) defined over the domain \( \mathcal{N}_m^n \). The goal of the learning algorithm is to halt and output a formula \( h \) that is \( \epsilon \)-close to \( f \) with respect to the distribution \( D \), that is,

\[
\Pr_D[f(x) = h(x)] \geq 1 - \epsilon.
\]

We say that \( h \) is an \( \epsilon \)-approximation of \( f \) with respect to the distribution \( D \). In the PAC or \textit{example query} model, the learning algorithm asks for an example from the \textit{example oracle}, and receives an example \((a, f(a))\) where \( a \) is chosen from \( \mathcal{N}_m^n \) according to the distribution \( D \).

We say that a class of boolean functions \( C \) is \( \alpha \)-\textit{properly PAC learnable} under the distribution \( D \) in polynomial time if there is an algorithm \( A \), such that for any \( f \in C \) over \( V_n \) and any \( \epsilon \) and \( \delta \), algorithm \( A \) runs in polynomial time, asks a polynomial number of queries (polynomial in \( n \), \( \log m \), \( 1/\epsilon \), \( 1/\delta \) and the size of the target function) and with probability at least \( 1 - \delta \) outputs a hypothesis \( h \in C \) that is an \( \epsilon \)-approximation of \( f \) with respect to the distribution \( D \). The size of \( h \) is at most \( \alpha \) times the size of \( f \). It is known from [A88] that if a class \( C \) is \( \alpha \)-properly exactly learnable in polynomial time from equivalence queries (and membership queries) then it is \( \alpha \)-properly PAC learnable (with membership queries) in polynomial time under any distribution \( D \).

We say that a distribution \( D \) is a \textit{product} distribution over \( \mathcal{N}_m^n \) if

\[
D(x_1, \ldots, x_n) = D_1(x_1)D_2(x_2) \cdots D_n(x_n)
\]

where each \( D_i \) is a distribution over \( \mathcal{N}_m \).

### 2.2 Axis Parallel Concept Classes

A \textit{boolean function} over \( \mathcal{N}_m^n \) is a function \( f : \mathcal{N}_m^n \to \{0,1\} \). The elements of \( \mathcal{N}_m^n \) are called \textit{assignments}. We will consider the set of \textit{variables} \( V_n = \{x_1, \ldots, x_n\} \) where \( x_i \) will describe the value of the \( i \)-projection of the assignment in the domain \( \mathcal{N}_m^n \) of \( f \). For an assignment \( a \), the \( i \)-th entry of \( a \) will be denoted by \( a_i \).

An \( \mathcal{N}_m^n \)-\textit{literal} is a function \([x_i \geq a] \) or \([x_i < a] \) where \( i \leq n \) and \( a \in \mathcal{N}_m \cup \{m\} \). Here \([x_i \geq a] = 1 \) if \( x_i \geq a \) and \( 0 \) otherwise. An \( \mathcal{N}_m^n \)-\textit{monotone literal} is \([x_i \geq a] \). An \( \mathcal{N}_m^n \)-\textit{term} is a product (conjunction) of literals. An \( \mathcal{N}_m^n \)-\textit{monotone term} is a product (conjunction) of monotone literals. An \( \mathcal{N}_m^n \)-\textit{DNF} is a disjunction of terms. An \( \mathcal{N}_m^n \)-\textit{Monotone DNF} is a disjunction of monotone terms. An \( \mathcal{N}_m^n \)-\textit{multivariate polynomial} is a sum of terms (mod 2). An \( \mathcal{N}_m^n \)-\textit{disjoint DNF} is an \( \mathcal{N}_m^n \)-DNF where the conjunction of every two terms is 0.

For example, when \( n = 3 \), \( T = [x_1 \geq 2] \wedge [x_1 < 5] \wedge [x_2 \geq 9] \) is an \( \mathcal{N}_m^n \)-term and can be written as

\[
T = [2 \leq x_1 < 5] \wedge [9 \leq x_2 < 11] \wedge [0 \leq x_3 < 11].
\]

Therefore, every term \( \mathcal{N}_m^n \)-term can be written as

\[
T_i = \bigwedge_{k=1}^{n} [a_{i,k} \leq x_k < b_{i,k}],
\]
where \( a_{i,k}, b_{i,k} \in \mathcal{N}_m \cup \{m\} \).

Notice that if we take an \( \mathcal{N}_m^n \)-term \( T \) and replace all \( m \) with some \( m' > m \) then we get an \( \mathcal{N}_m^{n'} \)-term \( T' \). Therefore we will sometimes ignore \( m \) and talk about \( \mathcal{N}_m^n \)-terms where each \( m \) is replaced by \( \infty \). For an \( \mathcal{N}_m^n \)-term \( T \) we write \( T^\infty \) for the corresponding \( \mathcal{N}_m^n \)-term. For the above example the corresponding \( \mathcal{N}_m^n \)-term is

\[
T^\infty = [2 \leq x_1 < 5] \land [9 \leq x_2 < \infty] \land [0 \leq x_3 < \infty].
\]

An \( \mathcal{N}_m^n \)-decision tree (\( \mathcal{N}_m^n \)-DT) over \( V_n \) is a binary tree whose nodes are labeled with \( \mathcal{N}_m^n \)-literals and whose leaves are labeled with constants from \( \{0,1\} \). Each decision tree \( T \) represents a function \( f_T : \mathcal{N}_m^n \rightarrow \{0,1\} \). To compute \( f_T(a) \) we start from the root of the tree \( T \): if the root is labeled with the literal \( l \) and \( l(a) = 1 \), then \( f_T(a) = f_{T_R}(a) \) where \( T_R \) is the right subtree of the root (i.e., the subtree of the right child of the root with all its descendents). Otherwise, \( f_T(a) = f_{T_L}(a) \) where \( T_L \) is the left subtree of the root. If \( T \) is a leaf then \( f_T(a) \) is the label of this leaf.

Sometimes we write \( \mathcal{N}_m^n \)-DNF when we do not want to specify the dimension. A DNF (multivariate polynomial, decision tree) is an \( \mathcal{N}_2 \)-DNF (respectively, \( \mathcal{N}_2 \)-multivariate polynomial, \( \mathcal{N}_2 \)-decision tree).

In general, for every set of boolean functions \( \Phi \) (e.g., exclusive or, or, etc.) and property function \( P_t : (\mathcal{N}_m^n \text{-term})^t \rightarrow \{0,1\} \) that is computable in polynomial time, (e.g., \( P_t(T_1, \ldots, T_t) = 1 \) if \( T_1, \ldots, T_t \) are pairwise disjoint) we can build a concept over \( \mathcal{N}_m^n \) as follows. We define the concept class \( \mathcal{P}\Phi[\mathcal{N}_m^n] \) (or \( \mathcal{N}_m^n \)-\( \Phi \)) to be the set of all \( \phi(T_1, \ldots, T_t) \) where \( \phi \in \Phi \) and \( \{T_i\} \) are \( \mathcal{N}_m^n \)-terms with \( P_t(T_1^\infty, \ldots, T_t^\infty) = 1 \). The property \( P \) is always computable in polynomial time and independent of \( m \) and therefore can be defined for \( \mathcal{N}_m^n \)-terms for any \( m \). We will sometime ignore the superscript \( \infty \) and just write \( P_t(T_1, \ldots, T_t) \). These classes are called axis-parallel concept classes. When \( P_t \equiv 1 \) then we write \( \Phi[\mathcal{N}_m^n] \) (or \( \mathcal{N}_m^n \)-\( \Phi \)).

For example, let \( \Phi = \{x_1, x_1 \lor x_2, x_1 \lor x_2 \lor x_3, \ldots\} \). Let \( P_t(T_1, \ldots, T_t) = 1 \) if and only if \( T_i \land T_j = 0 \) for every \( 1 \leq i < j \leq t \). Then \( \Phi[\mathcal{N}_m^n] \) is the set of disjoint DNF and \( \Phi[\mathcal{N}_2^n] \) is the set of union of disjoint rectangles in the two dimensional space.

For an \( f \in \Phi[\mathcal{N}_m^n] \) we define size_{\Phi}(f) to be the minimal number of \( \mathcal{N}_m^n \)-terms \( T_1, \ldots, T_t \) with property \( P_t \) (that is, \( P_t(T_1^\infty, \ldots, T_t^\infty) = 1 \)) such that there is an \( h \in \Phi \) where \( f \equiv h(T_1, \ldots, T_t) \). For a decision tree \( f \) the size will be the minimal number of non-leaf nodes in an \( \mathcal{N}_m^n \)-decision tree that is equivalent to \( f \).

A monotone projection from \( \mathcal{N}_{m'} \) to \( \mathcal{N}_m \) is a function \( M : \mathcal{N}_{m'} \cup \{m'\} \rightarrow \mathcal{N}_m \cup \{m\} \) such that for every \( i, j \in \mathcal{N}_{m'} \) where \( i < j \) we have \( M(i) \leq M(j) \) and \( M(m') = m \). A monotone projection \( M : (\mathcal{N}_{m'} \cup \{m'\})^n \rightarrow (\mathcal{N}_m \cup \{m\})^n \) is \( \mathcal{M} = (M_1, \ldots, M_n) \) where each \( M_i : \mathcal{N}_{m'} \cup \{m'\} \rightarrow \mathcal{N}_m \cup \{m\} \) is a monotone projection. We say that \( \mathcal{C} = \Phi[\mathcal{N}_m^n] \) is closed under monotone projection if for any monotone projection \( \mathcal{M} \) whenever \( \mathcal{P}(T_1, \ldots, T_t) = 1 \) we also have \( \mathcal{P}(T_1 M_1, \ldots, T_t M_t) = 1 \). Notice that if \( f = \phi(T_1, \ldots, T_t) \in \Phi[\mathcal{N}_m^n] \) then \( f\mathcal{M} = \phi(T_1 M_1, \ldots, T_t M_t) \in \Phi[\mathcal{N}_m^n] \). When the class \( \mathcal{C} = \Phi[\mathcal{N}_m^n] \) is closed under monotone projection then \( f\mathcal{M} \in \Phi[\mathcal{N}_m^n] \).

For a monotone projection \( \mathcal{M} \) define the dual monotone projection \( M^* \) where \( M^*(y) \) is the minimal \( x \) such that \( M(x) \geq y \). Since \( M(m') = m \), the dual monotone projection is well
defined. For a monotone projection $\mathcal{M} = (M_1, \ldots, M_n)$ we define $\mathcal{M}^* = (M_1^*, \ldots, M_n^*)$. We now show

**Lemma 1** If

$$T = \bigwedge_{k=1}^{n} [a_{i,k} \leq x_k < b_{i,k}]$$

then

$$T\mathcal{M} = \bigwedge_{k=1}^{n} [a_{i,k} \leq M_k(x_k) < b_{i,k}] = \bigwedge_{k=1}^{n} [M_k^*(a_{i,k}) \leq x_k < M_k^*(b_{i,k})].$$

and therefore $T\mathcal{M}$ is again an $\mathcal{N}^n$-term.

**Proof.** We first show the following two properties of monotone projection.

1. If $M$ is monotone then $M^*$ is monotone.
2. $x \geq M^* M(x)$.

To prove (1), let $y_1 < y_2$ and $M^*(y_1) = x_1$ and $M^*(y_2) = x_2$. Then $x_2$ is the minimal integer such that $M(x_2) \geq y_2$. Since $M(x_2) \geq y_2 > y_1$, the minimal $x_1$ such that $M(x_1) \geq y_1$ must be less than or equal to $x_2$. Therefore,

$$M^*(y_1) = x_1 \leq x_2 = M^*(y_2).$$

To prove (2) let $M^*(M(x)) = z$. Then $z$ is the minimal integer such that $M(z) \geq M(x)$.

Then $M(z) = M(x)$ and since $z$ is minimal $M^*(M(x)) = z \leq x$.

Now we are ready to prove the result. It is enough to show that

$$M(x) < y \text{ if and only if } x < M^*(y).$$

Suppose $M(x) < y$. Let $z = M^*(y)$. Then $z$ is the minimal integer such that $M(z) \geq y$.

Now $M(z) \geq y > M(x)$ and therefore $M^*(y) = z > x$.

If $M(x) \geq y$ then by properties (1) and (2) we have $x \geq M^* M(x) \geq M^* (y)$. \qed

Now the proof of the following Lemma is straightforward.

**Lemma 2** If $C_m = \mathcal{P}\Phi[\mathcal{N}^n_m]$ is closed under monotone projection and $f \in C_m$, then $f\mathcal{M} \in C_{m'}$ for any monotone projection $\mathcal{M} : \mathcal{N}^n_{m'} \to \mathcal{N}^n_m$. Also,

$$\text{size}_{\mathcal{P}\Phi}(f\mathcal{M}) \leq \text{size}_{\mathcal{P}\Phi}(f).$$

In a similar way one can define the class $\mathcal{P}\Phi[\mathcal{N}_{m_1} \times \cdots \times \mathcal{N}_{m_n}]$. In that case a monotone projection is $\mathcal{M} = (M_1, \ldots, M_n)$ where $M_i : N_{m_i} \to N_{m_i}$, and for $f \in \mathcal{P}\Phi[\mathcal{N}_{m_1} \times \cdots \times \mathcal{N}_{m_n}]$.

$f\mathcal{M} \in \mathcal{P}\Phi[\mathcal{N}_{m_1'} \times \cdots \times \mathcal{N}_{m_n'}]$. All the results of this paper are also true for that class.

**Constructiveness Assumption:** We assume that there is an algorithm “Construct” such that for any function $\psi : \mathcal{N}_{m_1} \times \cdots \times \mathcal{N}_{m_n} \to \{0,1\}$ that is computable in polynomial time, Construct$(\psi)$ runs in time $\text{poly}(\prod_i m_i)$ and returns some formula $f \in \mathcal{P}\Phi[\mathcal{N}_{m_1} \times \cdots \times \mathcal{N}_{m_n}]$ that is equivalent to $\psi$, if there exist such a formula, and returns “error” otherwise. Such algorithms exist (and are in fact very trivial) for all the classes presented in this paper.
2.3 Oracles

The other oracles we will consider in this paper are the following, as defined in [A88].

- **Subset oracle.** $Sub_f(h)$ for $h \in C$. This oracle returns ‘YES’ if $h \Rightarrow f$ and returns a counterexample $a$ such that $h(a) = 1$ and $f(a) = 0$ otherwise.

- **Superset oracle.** $Sup_f(h)$ for $h \in C$. This oracle returns ‘YES’ if $h \Leftarrow f$ and returns a counterexample $a$ such that $h(a) = 0$ and $f(a) = 1$ otherwise.

- **Disjointness oracle.** $Disj_f(h)$ for $h \in C$. This oracle returns ‘YES’ if $h \land f = 0$ and returns a counterexample $a$ such that $h(a) = 1$ and $f(a) = 1$ otherwise.

- **Exhaustiveness oracle.** $Exh_f(h)$ for $h \in C$. This oracle returns ‘YES’ if $h \lor f = 1$ and returns a counterexample $a$ such that $h(a) = 0$ and $f(a) = 0$ otherwise.

Given a set of oracles $O$, we say that $O$ is easy (resp. NP-easy) for $C$ if every oracle in $O$ can be simulated in polynomial time for $C$ (resp. simulated using an NP-oracle).

**Lemma 3** Let $\Phi$ be a set of boolean functions. For a constant dimension $d$, membership and equivalence oracles and all the above oracles (subset, superset, etc.) are easy for $\Phi[\mathcal{N}^d_m]$.

**Proof.** Let $R_f(h)$ be an oracle asked for the target $f$. We take all the literals $[x_i Q a]$ for $Q \in \{\geq, <\}$ in the terms of the target $f$ and of the function $h$ in the oracle. Let $A$ be the set of all the $a$’s in those terms and the two constants 0 and $m$. Suppose $A = \{0 \leq a_1 < a_2 < \cdots < a_t \leq m\}$. Notice that $t \leq 2d(s_h + s_f)$ where $s_h$ and $s_f$ are the number of terms in $h$ and $f$, respectively. It is easy to see that the functions $h$ and $f$ are constant functions 0 or 1 in each subdomain $[a_{i1}, a_{i2}] \times \cdots \times [a_{id}, a_{id+1})$. We check the oracle $R_f$ for every such subdomain. The number of subdomains is $(t + 1)^d$ which is polynomial for any constant $d$. $\square$

2.4 Lattice Projection

In this section we will give the definition of the lattice projection of functions and prove some claims. This will be one of the main tools used in this paper. Since this technique is used for constant dimension we will use $d$ for the dimension $n$.

A lattice in $\mathcal{N}^d_m$ is $L = L_1 \times \cdots \times L_d$ where $L_i \subseteq \mathcal{N}_m$. Let $f = \phi(T_1, \ldots, T_l) \in \Phi[\mathcal{N}^d_m]$ where $\phi \in \Phi$ and

$$T_i = \bigwedge_{k=1}^d [a_{i,k} \leq x_k < b_{i,k}].$$

Let $L = L_1 \times \cdots \times L_d$ be a lattice in $\mathcal{N}^d_m$. For $a, b \in \mathcal{N}_m$ define

$$[a]_{L_i} = \max\{x \in L_i \mid x \leq a\} \cup \{0\}, \quad [b]_{L_i} = \min\{x \in L_i \mid x \geq b\} \cup \{m\}.$$

For an assignment $v \in \mathcal{N}^d_m$ we define $[v]_L = ([v_1]_{L_1}, \ldots, [v_d]_{L_d})$. Define
\[ T_i^L = \bigwedge_{k=1}^{d} [a_{i,k} \leq x_k < b_{i,k}] \]

and \( f^L = \phi(T_1^L, \ldots, T_t^L) \). We call \( f^L \) the lattice projection of \( f \) on \( L \).

**Lemma 4** For every \( u \) we have

\[ f^L(u) = f([u]_L) = f^L([u]_L). \]

**Proof.** Notice that the lattice projection is a monotone projection with \( M_i(x_i) = [x_i]_L \). It is also easy to see that \( M^*_i(x_i) = [x_i] \). Let \( M = (M_1, M_2, \ldots, M_d) \). Then \( M(x) = [x]_L \) is a monotone projection and \( M^*(x) = [x]_L \). We also have \( M \cdot M = M \). Therefore,

\[ f^L([u]_L) = f(M(u)) = f(MM(u)) = f^L(u) = f(Mu) = f([u]_L). \]

**Lemma 5** Let \( \mathcal{P} \Phi [\mathcal{N}^d_m] \) be closed under monotone projection. For any function \( f \in \mathcal{P} \Phi [\mathcal{N}^d_m] \) and lattice \( L \) we have \( f^L \in \mathcal{P} \Phi [\mathcal{N}^d_m] \).

**Proof.** Since \( M(x) = [x]_L \) is a monotone projection and \( \mathcal{P} \Phi [\mathcal{N}^d_m] \) is closed under monotone projection, the result follows. □

**Lemma 6** Let \( f = \phi(T_1, \ldots, T_t) \) where \( \phi \in \Phi \) and \( T_i = \bigwedge_{k=1}^{d} [a_{i,k} \leq x_k < b_{i,k}] \). If for every \( i \) and \( k \) we have \( a_{i,k}, b_{i,k} \in L \) then \( f^L \equiv f \).

**Proof.** If \( a_{i,k}, b_{i,k} \in L \) then \( [a_{i,k}]_L = a_{i,k}, [b_{i,k}]_L = b_{i,k} \), \( T_i^L = T_i \) and \( f^L = \phi(T_1^L, \ldots, T_t^L) = \phi(T_1, \ldots, T_t) = f \). □

### 2.5 Polynomial Certificates

Following the definition of [HPRW96], the class \( C = \mathcal{P} \Phi [\mathcal{N}^n_m] \) has polynomial certificates if for every \( f \in C \) of size \( t \) there are \( q = poly(t, n) \) assignments \( A = \{a_1, \ldots, a_q\} \) such that for every \( g \in C \) of size less than \( t \), \( g \) is not consistent with \( f \) on \( A \). That is, \( g(a) \neq f(a) \) for some \( a \in \Lambda \).

**Lemma 7** Let \( d \) be constant. If \( C = \mathcal{P} \Phi [\mathcal{N}^d_m] \) is closed under monotone projection then it has polynomial certificates.

**Proof.** Let \( f = \phi(T_1, \ldots, T_t) \) where \( \phi \in \Phi \) and \( T_i = \bigwedge_{k=1}^{d} [a_{i,k} \leq x_k < b_{i,k}] \). Let \( L = \{a_{i,k}, b_{i,k} | 1 \leq i \leq t \} \) and \( L = \bigtimes_{k=1}^{d} L_k \). Notice that \( |L| \leq (2t)^d = poly(t) \) for constant \( d \). We now show that if \( g \in \mathcal{P} \Phi [\mathcal{N}^d_m] \) is consistent with \( f \) on \( L \) then \( \text{size}_f(g) \geq t \).

Since \( g \) is consistent with \( f \) on \( L \), by Lemmas 4 and 6 we have \( g^L(x) = g([x]_L) = f([x]_L) = f^L(x) = f(x) \). By Lemma 5 we have \( g^L \in \mathcal{P} \Phi [\mathcal{N}^d_m] \). Therefore, by Lemma 2 \( \text{size}_f(g) \geq \text{size}_f(g^L) = \text{size}_f(f) \). □

It follows from [HPRW96] that if \( C = \mathcal{P} \Phi [\mathcal{N}^d_m] \) is closed under monotone projection then \( C \) is learnable from membership and equivalence queries using an oracle for \( \Sigma^P_4 \). In this paper we will show that an NP oracle is sufficient.
2.6 Approximation algorithms

We assume the reader is familiar with approximation algorithms, \( \alpha \)-approximation and some of the basic concepts in approximation theory and complexity theory. For a problem in which we seek to minimize the size of a formula equivalent to a function \( f \), an \( \alpha \)-approximation algorithm (for \( \alpha \geq 1 \)) returns a formula \( h \equiv f \) that has size at most \( \alpha \) times the size of the smallest formula equivalent to \( f \). Given a class \( C = \mathcal{P}_{\Phi}[\mathcal{N}_n^m] \), we define the optimization problem Minimal Equivalent Formula (MIN\EQ\text{UI}C) to be the following problem:

\[
\text{MIN\EQ\text{UI}C} \quad \text{Given a formula } f \in C = \mathcal{P}_{\Phi}[\mathcal{N}_n^m]. \\
\text{Find a minimal size } h \in C \text{ that is equivalent to } f.
\]

We expect the algorithm to run in time polynomial in \( n \) and the size of \( f \). In some cases the function is given as an \( n \)-dimensional \( m_1 \times m_2 \times \cdots \times m_n \) matrix and the time is expected to be polynomial in \( \prod m_i \). We call this the \textit{unary representation} of the input. The problem is defined as follows.

\[
\text{MIN\EQ\text{UI}C}_U \quad \text{Given an } m_1 \times \cdots \times m_n \text{ matrix } A \text{ representing a formula in } C = \mathcal{P}_{\Phi}[\mathcal{N}_m^1 \times \cdots \times \mathcal{N}_m^m]. \\
\text{Find a minimal size } h \in C \text{ that represents } A.
\]

A generalization of this problem is the MIN\EQ\text{UI}C\_U problem. In this problem the input is given in its unary representation. The matrix may contain \(*\) entries which denote unspecified values.

\[
\text{MIN\EQ\text{UI}C}_* \quad \text{Given an } m_1 \times \cdots \times m_n \text{ matrix } A \text{ with entries } 0, 1 \text{ and } *, \text{ representing a function in } C = \mathcal{P}_{\Phi}[\mathcal{N}_m^1 \times \cdots \times \mathcal{N}_m^m]. \\
\text{Find a minimal size } h \in C \text{ that is equivalent to } f \text{ on the specified values.}
\]

Another problem that is related to the latter problem is the minimal size consistency problem.

\[
\text{MIN\EQ\text{UI}C} \quad \text{Given a set } S = \{(a_1, f(a_1)), \cdots, (a_t, f(a_t))\} \text{ where } f \in C = \mathcal{P}_{\Phi}[\mathcal{N}_m^n]. \\
\text{Find a minimal size } h \in C \text{ that is consistent with } S. \text{ That is, } h(a_i) = f(a_i) \text{ for all } i.
\]

Some of these problems appear in the literature. For example, MIN\EQ\text{UI}C\_SDNF\_U is the problem of covering orthogonal polygons by rectangles. MIN\EQ\text{UI}C\_\text{Disj-DNF}\_U is the problem of partitioning orthogonal polygons into rectangles.

It turns out that MIN\EQ\text{UI}C and MIN\EQ\text{UI}C\_U are equivalent for the constant dimensional domain.

\[
\text{Lemma 8 Let } C = \mathcal{P}_{\Phi}[\mathcal{N}_m^d] \text{ for a constant } d, \text{ be closed under monotone projection. Then}
\]

1. MIN\EQ\text{UI}C has a polynomial time \( \alpha \)-approximation algorithm if and only if MIN\EQ\text{UI}C\_U has a polynomial time \( \alpha \)-approximation algorithm.

2. MIN\EQ\text{UI}C\_U has a polynomial time \( \alpha \)-approximation algorithm if and only if MIN\EQ\text{UI}C\_U has a polynomial time \( \alpha \)-approximation algorithm.

\[
\text{Proof.} \text{ We prove (1). The proof of (2) is similar. Let } \mathcal{A} \text{ be a polynomial time } \alpha \text{-approximation algorithm for MIN\EQ\text{UI}C\_U. Let } f \in C \text{ be represented as } \phi(T_1, \ldots, T_r)
\]

10
where \( \phi \in \Phi \) and \( \hat{T}_i = \bigwedge_{k=1}^d [a_{i,k} \leq x_k < b_{i,k}] \). We build an approximation algorithm \( \mathcal{B} \) for \textsc{Minequi} \( C \). Let \( L_k = \{ a_{i,k}, b_{i,k} \mid i = 1, \ldots, t', k = 1, \ldots, d \} \) and suppose \( L_k = \{ c_{1,k}, \ldots, c_{m_k,k} \} \) where \( c_1,k < c_2,k < \cdots < c_{m_k,k} \). Algorithm \( \mathcal{B} \) defines \( L = L_1 \times \cdots \times L_d \).

By Lemma 6 and 4 we have \( f^L(x) = f(x) = f([x]_L) \). Let \( A \) be \( m_1 \times \cdots \times m_d \) matrix where \( A[i_1,\ldots,i_d] = f(c_{i_1,1},\ldots,c_{i_d,d}) \). This matrix represents the function \( f \mathcal{M} \) where \( \mathcal{M}(i_1,\ldots,i_d) = (c_{i_1,1},\ldots,c_{i_d,d}) \) and let \( \mathcal{M}^-(x) = \mathcal{M}^-(\lfloor x \rfloor_L) \). Notice that the size of this matrix is at most \( \prod m_i \leq (2t')^d \) which is polynomial for a constant \( d \). Define the inverse function \( \mathcal{M}^{-1} : L \rightarrow \mathcal{N}_{m_1} \times \cdots \times \mathcal{N}_{m_d} \) where \( \mathcal{M}^{-1}(c_{1,1},\ldots,c_{i_d,d}) = (i_1,\ldots,i_d) \) and let \( \mathcal{M}^-(x) = \mathcal{M}^{-1}(\lfloor x \rfloor_L) \). Notice that \( \mathcal{M}^- \) is a monotone projection. Since \( C \) is closed under monotone projection we have \( f \mathcal{M} \in \mathcal{P} \Phi [\mathcal{N}_{m_1} \times \cdots \times \mathcal{N}_{m_d}] \). Algorithm \( \mathcal{B} \) runs algorithm \( \mathcal{A} \) on \( f \mathcal{M} \) to find \( \phi \in \Phi \) and \( T_1,\ldots,T_t \) such that

\[ \mathcal{P}_t(T_1,\ldots,T_t) = 1, \quad f \mathcal{M} = \phi(T_1,\ldots,T_t) \quad \text{and} \quad t \leq \alpha \cdot \text{size}_\phi(f \mathcal{M}). \]

Consider the function

\[ g = \phi(T_1,\mathcal{M}^-,\ldots,T_t,\mathcal{M}^-). \]

Since \( C \) is closed under monotone projection and \( \mathcal{M}^- \) is a monotone projection we have \( \mathcal{P}_t(T_1,\mathcal{M}^-,\ldots,T_t,\mathcal{M}^-) = 1 \). Therefore, \( g \in \mathcal{P} \Phi [\mathcal{N}^d_m] \). We also have by Lemma 2,

\[ \text{size}_\phi(g) \leq t \leq \alpha \cdot \text{size}_\phi(f \mathcal{M}) \leq \alpha \cdot \text{size}_\phi(f). \]

Finally, we have

\[ g(x) = \phi(T_1,\mathcal{M}^-,\ldots,T_t,\mathcal{M}^-) = (f \mathcal{M})(\mathcal{M}^-(x)) = (f \mathcal{M})(\mathcal{M}^-([x]_L)) = f([x]_L) = f(x). \]

The algorithm returns \( g = \phi(T_1,\mathcal{M}^-,\ldots,T_t,\mathcal{M}^-). \)

For the other direction, suppose that we have an algorithm for \textsc{Minequi} \( C \). Given an \( m_1 \times \cdots \times m_d \) matrix \( A \), by the constructiveness assumption we can build a formula \( f \in \mathcal{P} \Phi [\mathcal{N}_{m_1} \times \cdots \times \mathcal{N}_{m_d}] \) that represents \( A \) in time \( \text{poly}(\prod m_i) \) and then using the algorithm for \textsc{Minequi} \( C \) we get the desired representation.\( \square \)

### 3 Approximation Algorithms and Learning

In this section we show the connection between approximation and learning.

**Theorem 9** If \( C \) is \( \alpha \)-properly exactly learnable from a set \( \mathcal{O} \) of oracles and \( \mathcal{O} \) is easy (resp., \( NP \)-easy) for \( C \) then \textsc{Minequi} \( C \) has an \( \alpha \)-approximation algorithm (resp., with the aid of an \( NP \)-oracle).

**Proof.** Let \( \mathcal{A} \) be a learning algorithm that uses the oracles in \( \mathcal{O} \) to learn a hypothesis of size less than \( \alpha \) times the size of the target. Since \( \mathcal{O} \) is easy for \( C \), all the oracles in \( \mathcal{O} \) can be simulated in polynomial time for \( C \). So we can run algorithm \( \mathcal{A} \) and simulate all the oracles in polynomial time. Since the learning is \( \alpha \)-proper, the output hypothesis has size less than \( \alpha \) times the size of the target.\( \square \)

The next Theorem follows from Lemma 8 and the Consistence Theorem [PV88].
Theorem 10 Let $C = \mathcal{P}[\mathcal{N}_m^d]$ for a constant $d$ be closed under monotone projection. The following three problems have the same time complexity.

1. $C$ is $\alpha$-properly PAC-learnable.

2. There is an $\alpha$-approximation algorithm for $\text{MINEQU}_C$.

3. There is an $\alpha$-approximation algorithm for $\text{MINEQU}_C^Y$.

Theorem 11 Let $C = \mathcal{P}[\mathcal{N}_m^d]$ for a constant $d$ be closed under monotone projection. The following problems have the same time complexity.

1. $C$ is $\alpha$-properly exactly learnable from membership and equivalence queries.

2. $C$ is $\alpha$-properly PAC learnable (without membership queries) under any product distribution.

3. There is an $\alpha$-approximation algorithm for the $\text{MINEQU}_C^Y$ problem.

Proof. We first show (1) $\equiv$ (3). By Theorem 9 and since by Lemma 3 the oracles are easy, we have (1) $\Rightarrow$ (3). Now we show (3) $\Rightarrow$ (1). Let $A$ be an $\alpha$-approximation algorithm for $\text{MINEQU}_C$. Let $f = \phi(T_1, \ldots, T_t)$ be the target function where $\phi \in \Phi$, $f \in \mathcal{P}[\mathcal{N}_m^d]$ and

$$T_i = \bigwedge_{k=1}^d [a_{i,k} \leq x_k < b_{i,k}].$$

The learning algorithm works as follows. At each stage it holds $d$ sets $L_1, \ldots, L_d$ where $L_k \subseteq \{a_{i,k}, b_{i,k} \mid i = 1, \ldots, t\} \cup \{0\}$. $L_k$ is initially $\{0\}$. The elements of $L_k$ are sorted in an increasing order:

$$L_k = \{c_{1,k} < c_{2,k} < \cdots < c_{i,k}\}.$$

The learning algorithm then builds the function $f^L$. It saves a table $A$ of size $\prod_{k=1}^d (i_k + 1)$ where $A[j_1, \ldots, j_d] = f(c_{j_1,1}, \ldots, c_{j_d,d})$. This can be done using the membership oracle. Now, $f^L$ can be defined using this table by $f^L(u) = f([u]_L) = A[j_1, \ldots, j_d]$ where $[u]_L[i] = c_{j_i}$ for $i = 1, \ldots, d$. Since $f^L \in \mathcal{P}[\mathcal{N}_m^d]$ (the class is closed under monotone projection), we can use algorithm $A$ and construct a formula $h$ that is equivalent to $f^L$ and has size at most $\alpha$ times the size of $f^L$. Since the size of $f^L$ is at most $\alpha$ times the size of $f$, the size of $h$ is at most $\alpha$ times the size of $f$.

After we construct $h \equiv f^L(x)$, we ask the equivalence query $\text{EQ}(h(x))$. Let $v$ be the counterexample. That is, $f^L(v) = h(v) \neq f(v)$. Then by Lemma 4 $f([v]_L) = f^L([v]_L) = f^L(v) \neq f(v)$. Intuitively, since $f([v]_L) \neq f(v)$, the straight line that connects the two points $v$ and $[v]_L$ hits the “boundary” of $f$. Now we give an algorithm to find a point close to this boundary and using this point we will add a new point to the lattice $L$ that is equal to one new $a_{i,j}$ or $b_{i,j}$. 

12
The algorithm works as follows. It first finds the smallest value \( k \) (or some \( k \) using binary search) such that

\[
f([v_1]_{L_1}, \ldots, [v_{k-1}]_{L_{k-1}}, v_k, v_{k+1}, \ldots, v_d) \neq f([v_1]_{L_1}, \ldots, [v_{k-1}]_{L_{k-1}}, [v_k]_{L_k}, v_{k+1}, \ldots, v_d).
\]

Such a \( k \) exists since \( f([v]_L) \neq f(v) \). Then (again with a binary search) the algorithm finds an integer \( c \) such that \([v_k]_{L_k} < c \leq v_k\) where

\[
f([v_1]_{L_1}, \ldots, [v_{k-1}]_{L_{k-1}}, c-1, v_{k+1}, \ldots, v_d) \neq f([v_1]_{L_1}, \ldots, [v_{k-1}]_{L_{k-1}}, c, v_{k+1}, \ldots, v_d).
\]

Now we prove the following.

**Claim 1.** We have \( c \in \{a_{i,k}, b_{i,k}\} \) for some \( i \), and \( c \not\in L_k \).

**Proof of Claim 1.** Let \( \theta(x_k) = f([v_1]_{L_1}, \ldots, [v_{k-1}]_{L_{k-1}}, x_k, v_{k+1}, \ldots, v_d) \). Here \( v_i \) are constant so the function \( \theta \) is a function on one variable \( x_k \). So \( \theta(x_k) = \phi(T_1, \ldots, T_m) \) where each \( T_i \) is either 0, 1 or \([a_{i,k} \leq x_k < b_{i,k}]\). Since \( \theta(c-1) \neq \theta(c) \) we must have some \( i \) where either \( a_{i,k} \leq c \) and \( a_{i,k} > c-1 \), or \( c-1 < b_{i,k} \) and \( c \geq b_{i,k} \). In the first case \( c = a_{i,k} \) and in the second case \( c = b_{i,k} \). Thus \( c \in \{a_{i,k}, b_{i,k}\} \).

Now notice that \([v_k]_{L_k} \neq v_k\) and \([v_k]_{L_k} < c \leq v_k\). Therefore, \( c \) was not in \( L_k \). \( \Box \)

We add \( c \) to \( L_k \) and update the table by adding all the missing values \( f(v) \) where \( v \in L_1 \times \cdots \times L_d \). We now show that this algorithm runs in polynomial time. Notice that the number of equivalence queries is at most

\[
\sum_{k=1}^{d} |\{a_{i,k}, b_{i,k} \mid i = 1, \ldots, t \} \cup \emptyset| \leq (2t + 1)d,
\]

and the number of membership queries is at most the size of the table which is \((2t + 1)^d\) plus the number of membership queries needed for the binary search. We do one binary search for each equivalence query. Therefore the algorithm uses at most \((2t + 1)^d + (2t + 1)^d \log m\) membership queries.

We now give the proof that (2) is equivalent to (3).

To prove (2) \( \Rightarrow \) (3), suppose \( C \) is \( \alpha \)-properly PAC-learnable under any product distribution. We show that (3) is true for \textsc{MinEquiC}_\( t \). Then by Lemma 8 the result follows. Let \( A \) be an \( m_1 \times \cdots \times m_d \) matrix, an instance for \textsc{MinEquiC}_\( \ell \). Define the product distribution \( D(i_1, \ldots, i_d) = 1/(\prod_{i=1}^{d} m_i) \) (uniform over \( \mathcal{N}_{m_1} \times \cdots \times \mathcal{N}_{m_d} \)). We now run algorithm \( \mathcal{A} \) with error \( \epsilon = 1/(2 \prod_{i=1}^{d} m_i) \). The hypothesis we get is consistent with \( A \) with probability at least \( 1 - \delta \) and has size at most \( \alpha \) times the target size.

To prove (3) \( \Rightarrow \) (2), let \( \mathcal{A} \) be an \( \alpha \)-approximation algorithm for \textsc{MinEquiC}. Let \( r(1/\epsilon, 1/\delta) \) be the number of examples needed to learn \( C \), assuming we have unlimited computational power. This \( r \) is polynomial and can be upper bounded by the VC-dimension Theorem [BEHW89]. Let \( \mathcal{B} \) be a polynomial time (nonproper) PAC-learning algorithm for \( C \) under any distribution \( D \). Such an algorithm exists. See for example [B98]. The idea of the proof is very simple. Since we cannot use membership queries, we learn a nonproper hypothesis \( \hat{h} \) that is close to the target function \( f \) and then use \( \hat{h} \) for membership queries.
We do that by first running \( \mathcal{B} \) to nonproperly learn the target function with a small error. Algorithm \( \mathcal{B} \) will output some hypothesis \( \hat{h} \). Then we use the hypothesis \( \hat{h} \) to simulate membership queries of \( f \). We show that when the distribution is the product distribution then with high probability \( \hat{h} \) simulates membership queries of \( f \).

We define the following algorithm to learn \( f \).

**Proper Learning**

1. Run \( \mathcal{B} \) with \( \epsilon = 1/(8r^d) \) where \( r = r(1/\epsilon, 8) \) and \( \delta = 1/8 \). Let \( \hat{h} \) be the output hypothesis.
2. Get \( r = r(1/\epsilon, 8) \) examples \( (x^{(1)}, f(x^{(1)})), \ldots, (x^{(r)}, f(x^{(r)})) \).
3. Define \( L_i = \{ x^{(j)} \} \mid j = 1, \ldots, r \} \) and \( L = L_1 \times \cdots \times L_d \).
4. Define the \( \hat{m}_1 \times \cdots \times \hat{m}_d \) matrix \( A \), where \( \hat{m}_i = |L_i| \), and \( A[i_1, \ldots, i_d] = \hat{h}(x^{(i_1)}_1, \ldots, x^{(i_d)}_d) \).
5. Run \( \mathcal{A} \) on \( A \) and let \( \hat{h} : \mathbb{N}_{\hat{m}_1} \times \cdots \times \mathbb{N}_{\hat{m}_d} \rightarrow \{0, 1\} \) be the output.
6. Define \( g = h \mathcal{M}^* \) where \( \mathcal{M}^*(x) = \mathcal{M}^{-1}(\lfloor x \rfloor L) \), and \( \mathcal{M}^{-1}(x^{(i_1)}_1, \ldots, x^{(i_d)}_d) = (i_1, \ldots, i_d) \).
7. Output \( g \)

In algorithm **Proper Learning**, step 1 learns some function \( \hat{h} \). Steps 2-3 take examples and build a lattice \( L \). Since we do not have membership queries to find the value of the target on this lattice we use instead \( \hat{h} \) to find the values. Steps 4-6 build a consistent hypothesis.

We now show that with probability \( 7/8 \) all of the membership queries that are simulated by \( \hat{h} \) give a correct answer. Notice that since the distribution is the product distribution, \( x^{(i_1)}_1, \ldots, x^{(i_d)}_d \) are chosen independently. Therefore,

\[
\Pr[\hat{h}(x^{(i_1)}_1, \ldots, x^{(i_d)}_d) \neq f(x^{(i_1)}_1, \ldots, x^{(i_d)}_d)] \leq \epsilon
\]

and

\[
\Pr[\exists x \in L : \hat{h}(x) \neq f(x)] \leq \epsilon |L| \leq \epsilon r^d = \frac{1}{8}.
\]

Since the learning algorithms \( \mathcal{B} \) and \( \mathcal{A} \) also have failure probability at most \( 1/8 \), algorithm **Proper Learning** succeeds with probability at least \( 1 - (3/8) > 1/2 \). We can run the above algorithm many times to achieve success \( 1 - \delta \). This completes the proof of Theorem 11. \( \square \)

**Corollary 12** Let \( C = \mathcal{P} \Phi[\mathbb{N}_{m}^{d}] \) for a constant \( d \) be closed under monotone projection. Then \( C \) is properly learnable from membership and equivalence queries using an oracle for NP.

**Proof.** Since \( \text{MINIMUM} \) can be solved in polynomial time using an NP-oracle, by Theorem 11, \( C \) is properly learnable from membership and equivalence queries using an oracle for NP. \( \square \)

Let \( C = \mathcal{P} \Phi[\mathbb{N}_{m}^{d}] \) for a constant \( d \) be closed under monotone projection. Consider these problems:

1. \( C \) is exactly learnable with a learning algorithm that uses oracles that are easy for \( C \), and outputs a hypothesis of size at most \( \alpha \) times the target size.

2. \( C \) is exactly learnable with a learning algorithm that uses equivalence queries only. The hypotheses may not be of small size but the output hypothesis has size at most \( \alpha \) times the target size.
Since all the oracles for $C = \mathcal{P}\Phi[\mathcal{N}_m^d]$ are easy, both problems give an $\alpha$-approximation algorithm for \textsc{Minequi}$C$. Therefore they are equivalent to the problems in Theorem 11.

This shows that a negative result for the $\alpha$-approximation of \textsc{Minequi}$C$ will give a negative result for the $\alpha$-proper learnability of $C$ from all of the oracles mentioned in section 2.3.

4 Positive and Negative results for Proper Learning

In this section we will prove positive and negative results for the $\alpha$-proper learning of different axis parallel classes.

4.1 Decision trees

In this section we give the results for Decision trees.

We first show

Lemma 13 The class of Decision trees is easy for all the oracles.

Proof. We show that for any $\phi : \{0, 1\}^2 \rightarrow \{0, 1\}$ there is a polynomial time algorithm $\mathcal{A}$ such that for any two decision trees $T_f$ for $f$ and $T_g$ for $g$, $\mathcal{A}$ can decide in polynomial time if $\phi(f, g) \equiv 0$ and if $\phi(f, g) \neq 0$ then $\mathcal{A}$ finds an assignment $x_0$ such that $\phi(f(x_0), g(x_0)) = 1$.

The algorithm takes the decision tree $T_f$ and replaces each leaf $v$ in $T_f$ by a decision tree $T_{f, v} \equiv T_g$. Then it takes each leaf $u$ in $T_{f, v}$ and label it with $\phi(l_v, l_u)$ where $l_v$ is the label of $v$ in $T_f$ and $l_u$ is the label of $u$ in $T_{f, v}$. It is easy to see that this new tree computes $\phi(f, g)$. We will call this tree $T'$.

Each path in the tree $T'$ from its root to a leaf labeled with 1 defines a term (see for example [B95]). If all such terms are identically 0 then there is no assignment that gives value 1 in $T'$ and therefore $\phi(f, g) \equiv 0$. Otherwise, there is a term that is 1 for some assignment $x_0$ and then the algorithm returns $x_0$.

Theorem 14 If there is a proper learning algorithm for $\mathcal{N}_2^d$-decision trees from membership and equivalence queries (and other oracles) then $P = NP$.

Proof. Decision tree is easy for all the oracles. Then, this result follows from Theorem 9 because \textsc{Minequi}$\mathcal{N}_2^d$-Decision Tree is NP-complete [ZB00].

Theorem 15 There is a proper learning algorithm from membership and equivalence queries for $\mathcal{N}_m^d$-decision tree for constant dimension $d$.

Proof. We describe an algorithm for \textsc{Minequi}$C_U$ where $C$ is the class of $\mathcal{N}_m^d$-decision trees. The algorithm easily generalizes to other constant dimensions $d$. Let $A$ be an $m \times m$ binary matrix. Each node $v$ of the decision tree for $A$ partitions a submatrix of $A$ into two submatrices, which are passed to the left and right subtrees $v$. The partitioning continues until each submatrix contains only 0s or only 1s, and the corresponding path in the decision tree terminates with a leaf. Define $S(a, b, x, y)$ to be the size (number of non-leaf nodes) of
a minimum decision tree that partitions the submatrix with rows \( a \) through \( x \) and columns \( b \) through \( y \). Then

\[
S(a, b, x, y) = \begin{cases} 
0 & \text{if the submatrix is monochrome,} \\
1 + \min \left\{ \min_{i \leq x} (S(a, b, i - 1, y) + S(i, b, x, y)), \right. \\
\left. \min_{j \leq y} (S(a, b, x, j - 1) + S(a, j, x, y)) \right\} & \text{otherwise.}
\end{cases}
\]

There are at most \( O(m^4) \) submatrices, so the number of subproblems is polynomial, and \( S(0, m - 1, 0, m - 1) \) can be computed in time \( O(m^5) \). The subproblems also provide information to build the minimal tree. For general \( d \), the algorithm has time complexity \( O(dn^{2d+1}) \) which is polynomial for constant \( d \). \( \square \)

**Theorem 16** There is a proper PAC-learning algorithm for \( \mathcal{N}^d_m \)-decision trees for constant \( d \).

**Proof.** The same algorithm above will also solve MINEQU\( ^* \) \( C_U \). By Theorem 10 the result follows. \( \square \)

### 4.2 DNF and Union of Boxes

In this section we give the results for DNF, Union of Boxes and disjoint union of Boxes. We first prove

**Theorem 17** There is an \( \epsilon < 1 \) such that: If the class \( \mathcal{N}^n_m \)-DNF is \( s' \)-properly learnable with membership and equivalence oracles (and all the other oracles) where \( s \) is the size of the DNF, then \( \Sigma^p_2 = P^{NP} \).

**Proof.** The oracles are NP-easy for \( \mathcal{N}^2_2 \)-DNF and approximating MINEQU\( ^1 \mathcal{N}^2_2 \)-DNF within \( s' \) is \( \Sigma^p_2 \)-hard [U99]. Then the result follows from Theorem 9. \( \square \)

For union of boxes in a constant dimension we have the following.

**Theorem 18** For union of boxes over dimension 2 (\( \mathcal{N}^2_m \)-DNF) we have

1. There is a \( log \) \( t \)-proper learning algorithm for union of boxes over dimension 2 from membership and equivalence queries, where \( t \) is the optimal number of boxes required.

2. There is an \( \alpha \) such that there is an \( \alpha \)-proper learning algorithm for union of boxes over dimension 2 from any set of oracles if and only if \( P = NP \).

**Proof.** Part 1 uses the fact that MINEQU\( ^1 \mathcal{N}^2_m \)-DNF has a \( log \) \( t \)-approximation algorithm [Fr89]. Part 2 uses the result that MINEQU\( ^1 \mathcal{N}^2_m \)-DNF is NP-complete and does not admit an approximation scheme unless \( P = NP \) [BD92]. Then both parts follow from Theorem 11 and Lemma 8. \( \square \)

For union of disjoint boxes (\( \mathcal{N}^n_m \)-disjoint DNF) we have the next Theorem.

**Theorem 19** We have

16
1. There is a proper learning algorithm for union of disjoint boxes over dimension 2 from membership and equivalence queries.

2. There is a proper learning algorithm for union of disjoint boxes over dimension 3 from membership and equivalence queries if and only if $P = NP$.

**Proof.** This follows from Theorem 11 and Lemma 8, and the fact that problem $\text{MIN\textsc{equi}}_N V_m$-disjoint DNF$_U$ is the same as the problem of partitioning a (set of) orthogonal polygons into a minimum number of boxes. This problem is in P for dimension 2 [LLLMP79], and NP-complete for dimension 3 [DK91]. □

### 4.3 Multivariate Polynomials and Xor of Boxes

In this subsection we investigate the learnability of Multivariate Polynomials.

For Multivariate Polynomials with Monotone terms we have

**Theorem 20** There is a proper learning algorithm for $N^d_m$-Monotone Multivariate Polynomial from membership and equivalence queries.

**Proof.** We give an algorithm to optimally solve the $\text{MIN\textsc{equi}}_C V$ problem where $C$ is the class of $N^d_m$-Monotone Multivariate Polynomials (Xor of monotone rectangles). A monotone term $[x_1 \geq a][x_2 \geq b]$ in the polynomial is a rectangle with lower left corner $(a, b)$, and covering all the points $(c, d)$ with $(a, b) < (c, d)$ (i.e., $a < c$ and $b \leq d$ or $a \leq c$ and $b < d$). Any nonzero input matrix $A = (a_{ij})$ over $\{0, 1\}$ will have a point $p = (c, d)$ such that $A[p] = 1$ and $A[q] = 0$ for all $q < p$. Then an optimal cover must use the monotone rectangle $R = [x_1 \geq c][x_2 \geq d]$ to cover $p$. We update the matrix by setting $A[i, j] = 1 - A[i, j]$ for all points $(i, j)$ covered by $R$. Repeating the process until $A = 0$ will yield the minimum number of rectangles. This algorithm for the 2-dimensional case generalizes to $d$ dimensions for any constant $d$. □

In a moment we will give a result for Multivariate Polynomial with nonmonotone terms. But first we consider “Almost Monotone” Multivariate Polynomials. An Almost-Monotone Multivariate Polynomial is a sum of terms that contain literals of the form $[x_i \geq \alpha]$ for $i = 1, \ldots, d$ and $[x_1 < \beta]$ (only $x_1$ may be negated).

**Lemma 21** There is a proper learning algorithm for $N^d_m$-Almost-Monotone Multivariate Polynomial from membership and equivalence queries.

**Proof.** We consider the 2-dimensional case and note that it generalizes to $d$ dimensions. An Almost-Monotone Rectangle $[x_1 \geq a][x_1 < a'][x_2 \geq b]$ covers all points $(c, d)$ with $a \leq c < a'$ and $d \geq b$. Suppose that we have two points $(a, b), (a', b)$ such that $A[c, d] = 0$ for all $(c, d)$ with $d < b$, $A[c, b] = 0$ for $c = a'$ and all $c < a$, and $A[c, b] = 1$ for all $a \leq c < a'$. Then for $f = T_1 + T_2 + \cdots + T_l$ that represents $A$, we have $f = [x_2 \geq b]f$ and therefore without loss of generality, no term $T_i$ contains a literal $[x_2 \geq b]$ with $d < b$.

Now consider the row of $A$ that corresponds to $x_2 = b$. Let $k$ be the number of transition points $a$ for which $A[a, b] \neq A[a + 1, b]$. Then $f$ must have at least $\lceil k/2 \rceil$ rectangles $T_i$ covering this row (since a rectangle has two vertical edges, it can cover just two transitions).
So without loss of generality we may assume that consecutive 1s are covered by their own rectangle, and the points \((a, b) \leq (c, b) < (a', b)\) are covered by \(R = [x_1 \geq a][x_1 < a'][x_2 \geq b]\) in an optimal cover.

So the algorithm works as follows. It finds the two points \((a, b)\) and \((a', b)\) as described above. It adds rectangle \(R = [x_1 \geq a][x_1 < a'][x_2 \geq b]\) to the cover. It then toggles the matrix entries that are covered by \(R\), and recurses. □

**Theorem 22** There is a \(2^{d-1}\)-proper learning algorithm for \(N_m^d\)-Multivariate Polynomial from membership and equivalence queries.

**Proof.** To prove the Theorem we show that every multivariate polynomial \(f\) containing \(t\) terms can be written as a sum (Xor) of at most \(2^{d-1}t\) Almost-Monotone terms. Thus the algorithm to optimally solve the Almost Monotone Multivariate Polynomial problem, is a \(2^{d-1}\)-approximation algorithm for the Multivariate Polynomial problem. Let \(f = T_1 + \cdots + T_t\) be a \(N_m^d\)-multivariate polynomial. We change each term to a sum of almost-monotone terms as follows:

\[
T_i = \prod_{k=1}^{d} [x_k \geq a_{i,k}] [x_k < b_{i,k}]
\]

\[
= [x_1 \geq a_{i,1}] [x_1 < b_{i,1}] \prod_{k=2}^{d} ([x_k \geq a_{i,k}] + [x_k \geq b_{i,k}])
\]

So \(f\) is a sum of at most \(2^{d-1}t\) almost-monotone terms. □

**References**


