The Complexity of Exactly Learning Algebraic Concepts

(Extended Abstract)

V. Arvind and N. V. Vinodchandran

Institute of Mathematical Sciences, Madras 600113, India.
email: {arvind,vinod}@imsc.ernet.in

Abstract. In this paper, we investigate the complexity of exact-learning classes of permutation groups and linear spaces over finite fields. These concept classes can be seen as special subclasses of the concept class of circuits. It turns out that in Angluin's exact-learning model these concept classes are learnable with equivalence queries but not learnable with membership queries, indeed not even exact learnable in a probabilistic sense.

More importantly, we raise the question whether the entire power of equivalence queries is needed to learn these concept classes. In order to answer this question we introduce the notion of a *teaching assistant*. Intuitively speaking, the exact learning model comprising of the teacher and learner, is enhanced to include an intermediate teaching assistant with which the learning algorithm communicates. Building on ideas from complexity theory, we quantify the complexity of teaching assistants for learning permutation groups and linear spaces. As our main theorems, we show:

- The representation class SYM of permutation groups are exactly learnable with an LWPP-assistant.
- The class of linear spaces over finite fields are exactly learnable with an SPP-assistant.

Since LWPP and SPP are low complexity classes, this means that these concept classes are of low complexity. As a negative result, we show that if 3-CNFs are exactly learnable with an LWPP-assistant (SPP-assistant), then NP $\subseteq$ LWPP (respectively NP $\subseteq$ SPP).

1 Introduction

One of the main objectives of computational learning theory is to classify the concept classes with respect to the difficulty of learning them in any reasonable learning model. In the Probably Approximately Correct (PAC) model of learning, introduced by Valiant [12], one can classify the complexity of a concept class by the number of samples for learning any concept in that class. One of the fundamental theorems about learning in this model is the complete characterization of classes that can be learned using polynomially many samples, using a combinatorial parameter called Vapnik-Chervonenkis dimension (VC-dimension) of the concept class. It is shown in [3] that a concept class is PAC-learnable with polynomially many samples if and only of it has polynomial VC-dimension.

Another such framework, which has been of general interest, is the exact learning model introduced by Angluin [1]. In this model a learner learns a concept by making use of a honest teacher, by querying her/him about the target concept. With reference to this model, the complexity of any concept class can be quantified using types of queries that a learner has to ask the teacher to learn any concept in the concept class. The types of queries that has been of main interest in the literature are membership queries and equivalence queries. In this direction many interesting results are known. Angluin [2] showed that only polynomially many equivalence queries are insufficient for learning concept classes which has a combinatorial property called approximate finger prints. Using this result she showed that deterministic finite automata, context free languages, CNF and DNF are not learnable using polynomially many equivalence queries. In [7], Gavalda proved that the nonexistence of approximate finger prints is a sufficient condition for learnability with polynomially many equivalence queries. In particular, he showed that the concept class of circuits are equivalence query learnable with the aid of an oracle in polynomial time hierarchy. This shows that circuits are, in a sense, 'easier' to learn than concept classes like DFAs, NFAs, CFGs, CNFs or DNFs. More recently, Hellerstein et al. [9] have shown that a concept class is polynomial query learnable with membership and equivalence queries, if and only if the concept class has a property called polynomial resilience property. It is also to be noted here that Watanabe [13] (see also [14]) used complexity-theoretic ideas to capture the complexity of various query types. He introduced the notion of machine types to quantify the complexity of exact learning via various query types.

The results mentioned above help us in classifying various concepts according to the difficulty of exactly learning them. However, the classification could be finer. For example, how do we compare two concept classes that are both exact learnable with polynomially many equivalence queries? It could be the case that one of these concepts is easier than the other because the full power of equivalence queries may not be needed to learn one although it is required to exactly learn the other.

The motivation of this paper is to develop a finer criteria for classifying the complexity of exact learning. For example, we show in this paper that the concept class SYM of permutation groups is polynomial-time exact learnable with equivalence queries. But using the refined learning model of this paper we show that SYM is an easier-to-learn concept than even 3-CNFs (which are also polynomial-time exact learnable with equivalence queries[1]) unless the complexity class NP is included in LWPP (which is believed to be unlikely).

The new ingredient in our model is the concept of *teaching assistants*. Intuitively, a teaching assistant for a particular concept class, helps a polynomial-time learner to learn that concept class. The learner actually communicates with the assistant who in turn makes membership queries to the teacher. By imposing resource bounds on the complexity of the assistant (more precisely, on the resources used by a Turing machine that accepts the assistant) we get different learning models of varying power.

In this paper, we discuss two algebraic concept classes in the context of our new learning model. These are the class of permutation groups (SYM) and the class of linear spaces over a fixed finite field F_p, (LS(p)). It is easy to see that neither concept class is exact learnable with polynomially many membership queries in any reasonable model of exact learning. On the other hand, as we show these classes are polynomial time exact learnable with equivalence queries. This will easily imply that these classes are polynomial-time exact learnable by an NP-assistant. It is natural to believe that linear spaces are in fact easier concepts than permutation groups. We prove results which indicate that this is in fact the case. More precisely, by exploiting the algebraic structure of these classes, we show that permutation groups are polynomial-time exact learnable with an LWPP-assistant and linear spaces over finite field F_p are polynomial-time exact-learnable with an SPP-assistant. Here the assistant classes LWPP($\mathcal{P}$) and SPP($\mathcal{P}$) with respect to representation class $\mathcal{P}$ are defined analogous to the complexity classes SPP and LWPP[5].

As a negative result we show that 3-CNFs are not polynomial-time exact learnable with LWPP-assistants unless NP $\subseteq$ LWPP. Similarly 3-CNFs are not polynomial-time exact learnable with SPP-assistants unless NP $\subseteq$ SPP. Both these inclusions would imply that NP is low for the probabilistic class PP which is believed to be unlikely. This shows that even though all these concepts are exact learnable with equivalence queries, 3-CNFs are harder to learn than class of permutation groups and class of linear spaces over a finite field.

In this extended abstract proofs of only main theorems are given[1].

2 Definitions and Notations

In this section we give complexity theoretic notations and define the framework for learning precisely. We fix the finite alphabet $\Sigma = \{0,1\}$. Let $\mathbf{Z}$ denotes the set of integers. A function $f : \Sigma^* \to \mathbf{Z}$ is *gap-definable* if there is a polynomial-time nondeterministic (in short, NP) machine M such that, for each $x \in \Sigma^*$, $f(x)$ is the difference between the number of accepting paths and the number of rejecting paths of M on input x (denoted by $M(x)$). Let GapP [5] denote the class of gap-definable functions. For each NP machine M let gap_M denote the GapP function defined by it. The corresponding language class PP is defined as follows: A language L is in PP if there is a GapP function f such that: $x \in L$ iff $f(x) > 0$. The counting classes of interest in this paper are SPP and LWPP defined using GapP functions [5]. A language L is in SPP if there is an $f \in$ GapP such that: $x \in L$ implies that $f(x) = 1$, and $x \notin L$ implies that $f(x) = 0$. A language L is in LWPP if there are functions $f \in$ GapP and $h \in$ FP such that: $x \in L$ implies that $f(x) = h(0^{|x|})$, and $x \notin L$ implies that $f(x) = 0$.

For formalizing the notion of learning via queries, we essentially adopt the scheme proposed by Watanabe [13]. We state the basic definitions.

[1] A full version of the paper is available on request

A *representation class* $\mathcal{P}$ is a tuple $\langle\{\langle R_n, \mu_n\rangle\}_{n\geq 1}, p, q\rangle$ where p and q are polynomials and for integer n, $R_n \subseteq \Sigma^{\leq q(n)}$ and $\mu_n : R_n \to 2^{\Sigma^{p(n)}}$ is a many-one mapping. Intuitively R_n consists of short representations of concepts in $\Sigma^{p(n)}$ and μ_n gives the concept represented by elements in R_n. Notice that a concept can have more than one representation. A representation class is said to be *honest* if it can be decided in time polynomial in n (1) whether a string $y \in \Sigma^{\leq q(n)}$ is a valid representation or not: that is $y \in R_n$; (2) whether, on input $\langle r, y\rangle$, y is in the concept $\mu_n(r)$ or not. In the sequel, we will be concerned only with honest representation classes that are honest.

Let $\mathcal{P}$ be a representation class. A *learner* for $\mathcal{P}$, in general, is an oracle Turing machine M which on input 0^n outputs a representation $r \in R_n$ of the *target concept* $\mu(r)$ set by the teacher. During the course of computation M can make queries to the teacher. In this paper we will be interested in *equivalence* and *membership* queries. A new type of query introduced in this paper is a query by the learner to a predefined assistant (defined formally in the next section).

We consider the following types of learners: deterministic and probabilistic. A deterministic learner for a representation class $\mathcal{P}$ is a deterministic oracle transducer which has to output a representation for any target concept set by the teacher. A probabilistic learner is a probabilistic oracle transducer which has to output some representation of the target concept on at least 1/2 the fraction of the random choices.

We consider polynomial-time exact learnability and polynomial-query exact learnability in this paper.

2.1 Teaching Assistants

Before formally defining assistants we give some motivation. In Angluin's model of exact learning the learner communicates with the teacher in order to learn a concept, where the teacher is of unbounded computational power. Two standard query types that are studied are equivalence queries and membership queries. It is well known that an equivalence query can be replaced by queries to an NP oracle which uses membership queries. Intuitively, we can think of the NP oracle as an intermediate agent between the learner and the teacher (which we term as a teaching assistant). The learner makes queries to the assistant and the assistant makes membership queries to the teacher. More precisely, a teaching assistant contains strings which encode information about the representation class. Assistants are of bounded computational power and help the learner by communicating with the teacher using only membership queries. Thus the power of communication of the assistant with the teacher is dependent on its resource bounds. Hence if a concept class is learnable using a teaching assistant with small resource bounds it implies that the concept class is relatively easy to learn.

We formally define assistants as sets dependent on the representation class. Let $\mathcal{P}$ be a representation class. For $\mathcal{P}$, *teaching assistants* are in general sets of the form: $\{\langle 0^n, x, r\rangle | n \in \mathbf{N}; x \in \Sigma^{\leq q(n)}; r \in R_n \text{ and } Q^r(x)\}$. Here q is any polynomial and Q is a predicate whose truth value depends on x and the concept represented by r.

For quantifying the complexity of assistants, we define the following classes of teaching assistants with respect to a representation class. These are analogous to the well studied classes in complexity theory. Let $\mathcal{P}$ be a representation class and $L(\mathcal{P})$ be a teaching assistant.

- A teaching assistant for representation class $\mathcal{P}$ is said to be in the class $\mathrm{NP}(\mathcal{P})$ if there exists a polynomial time nondeterministic oracle Turing machine M which on input $\langle 0^n, x\rangle$ uses $\mu_n(r)$ as oracle and accepts $\langle 0^n, x\rangle$ if and only if $\langle 0^n, x, r\rangle \in L(\mathcal{P})$.
- A teaching assistant for representation class $\mathcal{P}$ is said to be in the class $\mathrm{SPP}(\mathcal{P})$ if there exists a polynomial time nondeterministic oracle Turing machine M which on input $\langle 0^n, x\rangle$ uses $\mu_n(r)$ as oracle and produces a *gap*=1 if $\langle 0^n, x, r\rangle \in L(\mathcal{P})$ and *gap*=0 if $\langle 0^n, x, r\rangle \notin L(\mathcal{P})$.
- A teaching assistant for representation class $\mathcal{P}$ is said to be in the class $\mathrm{LWPP}(\mathcal{P})$ if there exists a polynomial time computable function f and a polynomial time nondeterministic oracle Turing machine M which on input $\langle 0^n, x\rangle$ uses $\mu_n(r)$ as oracle and produces a *gap*=$f(n)$ if $\langle 0^n, x, r\rangle \in L(\mathcal{P})$ and *gap*=0 if $\langle 0^n, x, r\rangle \notin L(\mathcal{P})$.

Note that the representation r figuring in a string $\langle 0^n, x, r\rangle$ is just an *index* to the concept of interest, to which the machine accepting the assistant can make membership queries. It follows from definitions that $\mathrm{SPP}(\mathcal{P}) \subseteq \mathrm{LWPP}(\mathcal{P})$ for any representation class $\mathcal{P}$.

Definition 2.1 Let $\mathcal{C}$ denotes any of the above-defined assistant classes. We say a representation class $\mathcal{P}$ is polynomial-time exact learnable with a $\mathcal{C}$-assistant if there exists a teaching assistant $L(\mathcal{P}) \in \mathcal{C}(\mathcal{P})$ such that $\mathcal{P}$ is polynomial-time exact learnable by making queries to $L(\mathcal{P})$.

Theorem 2.2 *If a representation class $\mathcal{P}$ is polynomial-time exact learnable with equivalence queries, then $\mathcal{P}$ is polynomial-time exact learnable with an* NP-*assistant.*

Now we show that the representation class 3-CNF consisting of boolean formula in conjunctive normal form where each clause contains at most three literals are not polynomial-time exact learnable with an LWPP-assistant (SPP-assistant) unless NP $\subseteq$ LWPP (respectively NP $\subseteq$ SPP).

Theorem 2.3 *If the representation class 3-CNF is polynomial-time exact learnable with an LWPP-assistant then $NP \subseteq LWPP$. Similarly if the representation class 3-CNF is polynomial-time exact learnable with an SPP-assistant then* NP $\subseteq$ SPP.

Proof. We will prove that if 3-CNF is polynomial-time exact learnable with an SPP-assistant, then NP $\subseteq$ SPP. The other case can be similarly proved. For proving this, we show that if 3-CNF has a learning algorithm which uses an

SPP-assistant, then the co-NPcomplete language TAUT= $\{\langle 0^n, f\rangle | f$ encodes 3-CNF on n variables and f is a tautology$\}$ is in $\mathrm{P}^{\mathrm{SPP}}$. Since $\mathrm{P}^{\mathrm{SPP}}$ = SPP[5], the theorem follows.

Let M be a machine which on input 0^n learns an n-variable 3-CNF with an SPP-assistant $L($ 3-CNF$) = \{\langle 0^n, y, r\rangle | y \in \Sigma^{q(n)}; Q^r(y)\}$ accepted by a machine N. Define a language $A = \{\langle 0^n, y, f\rangle | y \in \Sigma^{q(n)}; f$ encodes a 3-CNF formula and $Q'(y, f)\}$ where the predicate Q' is such that if $Q'(y, f)$ is true then N on input $\langle 0^n, y\rangle$ with f as target formula will have *gap*=1 and if $Q'(y, f)$ is false then N will have a *gap*=0. It is easy to see that $A \in SPP$. Consider the deterministic machine M', for accepting TAUT, which on input $\langle 0^n, f\rangle$ does the following. M' simulates M on input n two times, sequentially. In the first simulation whenever M makes a query $\langle 0^n, y, r\rangle$ to $L($ 3-CNF$)$, M' makes a query $\langle 0^n, y, f\rangle$ to A. In the next simulation whenever M makes a query $\langle 0^n, y, r\rangle$ to $L($ 3-CNF$)$, M' makes a query $\langle 0^n, y, T\rangle$ to A (T is the trivial formula *true*). M' accepts $\langle 0^n, f\rangle$ if the output of M' in both the cases are the same.

3 Complexity of learning permutation groups

In this section we investigate the complexity of learning permutation groups. We first show that SYM is not polynomial-time exact learnable with membership queries even if the learning algorithm is probabilistic. On the other hand it turns out that SYM is polynomial-time exact learnable with equivalence queries. We refine this result to prove our first main theorem that SYM is polynomial-time exact learnable with an LWPP-assistant.

We first recall basic definitions and formalize notation. We refer the reader to standard textbooks in group theory [8]

Let G be a finite group. $|G|$ denotes the order of G. e denotes the identity element of G. $H < G$ denotes that H is a subgroup of G. For $S \subseteq G$, $\langle S\rangle$ denotes the group generated by S. Let $H < G$. For $g \in G$ the set $\{hg \mid h \in H\}$ denoted by Hg is a *right coset* of H in G. Similarly, the set $gH = \{gh \mid h \in H\}$ is a *left coset* of H in G. The group of permutations on a set of n elements is denoted by S_n.

We now formally define the representation class SYM of permutation groups. By Lagrange's theorem[8], it follows that any subgroup $G < S_n$ has a generator set consisting less than or equal to $p(n)$ elements ($p(n) = O(n^2)$). SYM $= \langle\{\langle R_n, \mu_n\rangle\}_{n\geq 1}, p, q\rangle$, where each $r \in R_n$ encodes a generator set for the corresponding subgroup $\mu_n(r)$ of in S_n. Thus R_n contains some representation for each subgroup of S_n. It is known that testing membership of an elements g in a group presented by a generator set is $S \subseteq S_n$ can be done in time polynomial in n[6]. Hence it follows that SYM is honest.

We next state theorems about non-learnability of permutation groups with membership queries and learnability with equivalence queries. The proofs are omitted.

Theorem 3.1 *1. The representation class* SYM *is not deterministic polynomial-query learnable with membership queries.*

2. *The representation class* SYM *is not polynomial-time exact learnable with membership queries even if the learner is probabilistic.*[2]
3. *The representation class* SYM *is polynomial-time exact learnable with equivalence queries. Hence* SYM *is polynomial-time exact learnable with an NP-assistant*

We now show that permutation groups are polynomial-time exact learnable with an LWPP-assistant. We first state results from the theory of finite permutation groups which are needed for the proof of the main theorem.

Let $G < S_n$ and let $G^{(i)} = \{g \in G \mid g \text{ fixes } 1 \text{ to } i\}$. Then $\{e\} = G^{(n)} < G^{(n-1)} < \ldots < G^{(0)} = G$. Consider the right cosets of $G^{(i)}$ in $G^{(i-1)}$. Two elements $g_1, g_2 \in G^{(i-1)}$ are in the same right coset iff $g_1(i) = g_2(i)$. Consider $T_i = \{g_{i1}, g_{i2}, \ldots, g_{id_i}\}$ of distinct right coset representatives of $G^{(i)}$ in $G^{(i-1)}$. Notice that $G^{(i-1)} = \bigcup_{j=1}^{d_j} G^{(i)} g_{ij}$, where $d_i = |T_i|$. The set $\bigcup_{i=1}^{n} T_i$ is a generator set for G, and we refer to it as a *canonical generator set* for G.

Let $G < S_n$. Let σ be a permutation on $X \subseteq \{1, \ldots, n\}$. Let $G_{[X]}$ denotes the set $\{g \in G | g(i) = i \text{ for } i \in X\}$ and $G_{[\sigma]}$ denotes the set $\{g \in G | g(i) = \sigma(i) \text{ for } i \in X\}$. We state a useful group-theoretic lemma.

Lemma 3.2 *Let $G < S_n$ be a permutation group.*

1. $|G^{(i)}| = \prod_{j>i} d_j$ *for each* i, $1 \leq i \leq n$.
2. *The number K of canonical sets of generators for G is* $K = \prod_{i=1}^{n} |G^{(i)}|^{d_i}$.
3. $G_{[X]} < G$ *and* $|G_{[\sigma]}| = |G_{[X]}|$

Theorem 3.3 *The representation class* SYM *is polynomial-time exact learnable with an LWPP-assistant.*

Proof. Let $\mu_n(r) = G$ be the target group. Define the assistant $L(\text{SYM})$ as: $L(\text{SYM}) = \{\langle 0^n, k, i, j_1, \ldots, j_k, r\rangle | \exists g \in \mu_n(r) \text{ such that } g \text{ fixes } 1 \text{ to } i;\ g(i+l) = j_l \text{ for } 1 \leq l \leq k\}$. First we show that this set defines an LWPP-assistant by constructing a nondeterministic machine which produces zero gap if a string is not in the set and produces a $gap = (n!)^{2n^2+1}$ if it is in the set. Then we give a polynomial-time exact learning algorithm using $L(\text{SYM})$ as assistant for the class SYM.

Claim 3.3.1 *The set $L(\text{SYM})$ is accepted by a nondeterministic machine M which on input $\langle 0^n, k, i, j_1, \ldots, j_k\rangle$ produces a $gap = (n!)^{2n^2+1}$ using $\mu_n(r)$ as oracle if $\langle 0^n, k, i, j_1, \ldots, j_k, r\rangle \in L(\text{SYM})$ and produces zero gap if $\langle 0^n, k, i, j_1, \ldots, j_k, r\rangle \notin L(\text{SYM})$.*

Proof We first give a brief description of the machine M. Machine M simulates another machine N on all the paths where a "correct" guess is made. Intuitively, simulation of machine N by machine M is to make sure that the gap produced by M is as stated in the claim. Notice that N simulates another machine N' and all these machines uses $G = \mu_n(r)$ as oracle.

[2] We mean in the PEC model defined in[4]

Description of $M(\langle 0^n, k, i, j_1, \ldots, j_k \rangle)$

1 **Guess** $g \in S_n$
2 **if** $g \in \mu_n(r)$ and $g(j) = j;\ \forall j; 1 \leq j \leq i$ and $g(i+l) = j_l;\ \forall l; 1 \leq l \leq k$
3 **then** Simulate Machine N on input $\langle 0^n, i+k \rangle$
4 **else** Branch into two paths
5 **Accept** on one path and **Reject** on the other path

We now show that M has an accepting behavior as stated in the above claim. Suppose $\langle 0^n, k, i, j_1, \ldots, j_k, r \rangle \notin L(\textsc{Sym})$. This means that there exists no $g \in G$ such that g fixes 1 to i and maps $g(i+l) = j_l$; for all $1 \leq l \leq k$. In this case the **if** test in *line 2* of M will be failed for all the guess and hence the machine M will produce a zero *gap*. Now let us consider the case when $\langle 0^n, k, i, j_1, \ldots, j_k, r \rangle \in L(\textsc{Sym})$. In this case, it follows from part 3 of Lemma 3.2 that number of such $g \in G$ is exactly equal to $|G^{(i+k)}|$, the number of $g \in G$ which fixes 1 to $i+k$. So number of $g \in G$ which passes the **if** test is equal to $|G^{(i+k)}|$. Now, suppose the machine N on input $\langle 0^n, i+k \rangle$ produces a *gap* $= (n!)^{2n^2+1}/|G^{(i+k)}|$, then since N is being simulated on exactly $|G^{(i+k)}|$ paths, the total gap produced by M will be $((n!)^{2n^2+1}/|G^{(i+k)}|) \times |G^{(i+k)}| = (n!)^{2n^2+1}$. We define the machine N such that it produces exactly this gap.

Before defining N we define the machine N' which will be used by the machine N. Machine N' makes membership queries to the target concept $G = \mu_n(r)$. It is easy to see that N' has the following behavior. If there exists $g \in G$ which fixes 1 to $(i-1)$ and maps i to j, then there will be $|G^{(i)}|$ accepting paths. Otherwise N will have zero accepting paths.

Description of $N'(0^n, i, j)$

Guess $g \in S_n$
if g fixes 1 to $(i-1)$ and maps i to j
then if $g \in \mu_n(r)$
then Accept
else Reject

An intuitive idea how the machine N produces the required gap is as follows. The machine N on input $\langle 0^n, i \rangle$ tries to compute $|G^{(i)}|$ in a nondeterministic manner using the target concept G as oracle. For that it first guesses an encoding of a canonical generator set $X_i = \langle T_{i+1}, \ldots, T_n \rangle$ of $G^{(i)}$ and verifies that it actually encodes a canonical generator set of some subgroup of S_n. It then verifies that the elements that X_i encode are indeed in $G^{(i)}$ by querying the target group G and then verifying that all these elements fixes 1 to i. There is a possibility that the guessed set of elements actually generate only a *proper* subgroup of $G^{(i)}$. On the paths along which a generator set for only a proper subgroup of $G^{(i)}$ is guessed, the machine produces a zero gap. The details are as follows.

Let $X_i = \langle T_{i+1}, \ldots, T_n \rangle$ where T_j, for $(i+1) \leq j \leq n$, corresponds to the encoding of a complete transversal of $G^{(i)}$ in $G^{(i-1)}$ in lexicographic order. (When there is no confusion, we use T_i to denote both complete transversal of $G^{(i)}$ in

$G^{(i-1)}$ and its lexicographic encoding.) Define a set S_i as $S_i = \{\langle j,k\rangle | (i+1) \leq j \leq k \leq n$ and there does not exists $g \in X$ which fixes 1 to $(j-1)$ and maps j to $k\}$. (By '$g \in X$' we mean g is encoded in X.)

Description of $N(0^n, i)$

```
1   Guess an encoding X_i = ⟨T_{i+1}, ..., T_n⟩ of canonical generator set for G^(i)
2   If X_i does not encode a canonical generator set produce a gap = 0
3   For all g ∈ X_i if g ∉ G^(i) produce a gap = 0 and stop
4   Using X_i compute α = ∏_{(i+1)≤k≤n} |T_k|
5   Using X_i compute β = ∏_{n≥j≥i+1} (∏_{j<k≤n} |T_k|)^{|T_j|}
6   Branch into (n!)^{n^2+1} / α.β and proceed
7   Using X_i construct the set S_i
8   Compute γ = ∏_{n≥j≥i} (∏_{n≥k>j} |T_k|)^{n-j-T_j+1}
9   Branch into (n!)^{n^2} / γ paths
10  Produce a gap of ∏_{⟨j,k⟩∈S_i} ((∏_{n≥l≥i} |T_l|) − acc_{N'}⟨0^n, j, k⟩)
```

Let us analyze the behavior of the machine M. It is clear from the description of M that after *line* 3 only those paths which correctly encode a subgroup of G only survive. All other paths contributes a zero *gap* to the overall *gap* produced by M. The rest of the computation is to make sure that all those paths which corresponds to a proper subgroup of $G^{(i)}$ also contributes a zero *gap*.

First let us argue that if the guessed X_i encodes a canonical generator set for $G^{(i)}$ M produces a $gap = (n!)^{2n^2+1}/|G^{(i+k)}|$. Let ρ be such a path. In this case from part 1 of Lemma 3.2 it follows that the value of α computed in *line* 4 will be exactly $|G^{(i)}|$. Also from part 2 of Lemma 3.2 it follows that value of β computed in *line* 5 will be K_i, the number of canonical generators for $G^{(i)}$. So in *line* 6 ρ will contribute a $gap = (n!)^{n^2+1}/(|G^{(i)}|.K_i)$. (The number $(n!)^{n^2+1}$ is chosen in such a way that this division is possible.) At *line* 9, each such path again branch into $(n!)^{n^2}$ paths. So at this point the path ρ has contributed $(n!)^{2n^2+1}/|G^{(i)}|.K_i.\gamma$ paths. At *line* 10, since X_i is a canonical generator set for $G^{(i)}$ it follows that $\forall \langle j,k\rangle \in S_i \Rightarrow acc_{N'}(\langle 0^n, j, k\rangle) = 0$. So in this case $\prod_{\langle j,k\rangle \in S_i}(\prod_{n\geq l\geq i}|T_l|) = \prod_{n\geq j\geq i}(\prod_{n\geq k>j}|T_k|)^{(n-j-T_j+1)} = \gamma$. So after *line* 10 ρ will contribute a $gap = ((n!)^{2n^2+1}/|G^{(i)}|.K_i.\gamma) \times \gamma$. But, since there are K_i such guesses for a canonical generator set for $G^{(i)}$ the total contribution of all these generator sets will be equal to $(n!)^{2n^2+1}/|G^{(i)}|$. Next we show that those paths on which a generator set of some proper subgroup of $G^{(i)}$ is being guessed, will contribute a *gap*=0. This will prove the claim.

Let σ be a path on which canonical generator set for a proper subgroup of $G^{(i)}$ being guessed. Observe that, to show that this path contributes a *gap*=0, it suffices to show that the *gap* produced in *line* 10 is zero. Suppose X_i guessed on path σ is not a canonical generator set of G. Then there is a largest $j < i$, say j_0, such that T_{j_0} is not a complete right transversal of $G^{(j_0)}$ in $G^{(j_0-1)}$. Then $\bigcup_{n\geq k>j_0} T_k$ is a canonical generating set for $G^{(j_0)}$. It follows that there exists k and $g \in G$ such that $\langle i_0, k\rangle \in S_i$ and g fixes 1 to (j_0-1) and maps j_0 to k.

Consequently, machine N' on input $\langle 0^n, j_0, k\rangle$ will have $|G^{(j_0)}|$ accepting paths. Thus, since $\prod_{k\geq j_0}|T_k| = |G^{(i_0)}| = acc_{N'}(\langle 0^n, i_0, j\rangle)$, it follows that the product $\prod_{\langle j,k\rangle \in S_i}(acc_{N'}(\langle 0^n, j, k\rangle) - (\prod_{in \geq l \geq i}|T_l|)) = 0$. Hence the total *gap* produced by σ will be zero. ■

Claim 3.3.2 *There is a polynomial time deterministic machine* LEARN SYM *which computes a canonical generator set of the target group G for the class* SYM *using L(*SYM*) as assistant.*

Proof The learner LEARN SYM computes, using L(SYM) as oracle, those pairs $\langle i, j\rangle$ for which there is a $g \in G$ such that g fixes 1 to $(i-1)$ and maps i to j. For all such pairs it calls a subroutine CONSTRUCT on input $\langle 0^n, i, j\rangle$. Subroutine CONSTRUCT on input $\langle 0^n, i, j\rangle$ constructs lexicographically first $g \in G$ such that g fixes 1 to $(i-1)$ and maps i to j using a prefix search with the help of set L(SYM). We give descriptions of both machines below.

LEARN SYM(0^n)
for pairs $\langle i, j\rangle; 1 \leq i \leq j \leq n$ **do**
if $\langle 0^n, 1, i-1, j, r\rangle \in L$(SYM)
  then CONSTRUCT($0^n, i, j$)

CONSTRUCT($0^n, i, j$)
$k \leftarrow 1$
repeat
    Find first $l_k; i < l_k \leq n$ such that
    $\langle 0^n, k, i-1, j, x_1, \ldots, x_{k-1}, l_k, r\rangle \in$L(SYM)
    $x_k \leftarrow l_k$
    $k \leftarrow k+1$
  until $k = (n-i-1)$
return $(1, \ldots, i-1, j, x_1, \ldots, x_{n-i})$

It is easy to see that LEARN SYM runs in $O(n^4)$ time steps.

4 Complexity of learning linear spaces

In this section we investigate the complexity of learning linear subspaces over the finite field F_p of size p. First we give the necessary notations. For basic definitions and results, refer to [10].

Let U be a vector space over a field F. We will denote the zero vector of U by $\mathbf{0}$. U is called nontrivial if U contains a vector other than $\mathbf{0}$. $V < U$ denotes that V is a subspace of U. For $X \subseteq U$ let $\langle X\rangle$ denotes the vector space spanned by X. $U \oplus V$ denotes the direct sum of U and V. F^n denotes direct sum of F, n times. Let $\mathcal{E}$ denotes a system of homogeneous linear equations over n variables. Then $Sol(\mathcal{E})$ denotes the set $\{u \in F^n | u \text{ is a solution for } \mathcal{E}\}$. For a vector variable x,

x_i be the variable denoting the i^{th} component of x. Let X_i denotes the equation $x_i = 0$ and $\mathcal{X}_i$ denotes the system of equations $\{X_1, \ldots, X_i\}$.

Next we formally define the representation class $\mathrm{LS}(p)$ of linear spaces over F_p^n. $\mathrm{LS}(p) = \langle\{\langle R_n, \mu_n\rangle\}_{n\geq 1}, t, q\rangle$. Each $r \in R_n$ encodes a set of linearly independent vectors of F_p^n. The polynomial t is chosen such a way that a vector in F_p^n can be coded by strings in $\Sigma^{t(n)}$. For any $r \in R_n$, $\mu_n(r)$ represents the space spanned by the vectors encoded in r. Since it is known that number of independent vectors needed to span a vector space of dimension d is exactly d, q is chosen in such a manner that n vectors can be encoded in strings of length $q(n)$. Clearly $\mathrm{LS}(p)$ is an honest representation class.

We first state some easy-to-prove results (similar to those stated in Theorem 3.1) about non-learnability with membership queries and learnability with equivalence queries.

Theorem 4.1 *1. The representation class* $\mathrm{LS}(p)$ *is not deterministic polynomial query learnable with only membership queries.*

2. The representation class $\mathrm{LS}(p)$ *is not probabilistic polynomial time learnable with only membership queries.*

3. The representation class $\mathrm{LS}(p)$ *is deterministic polynomial time learnable using equivalence queries.*

We now prove the main theorem of this section. we design a polynomial-time exact learning algorithm for $\mathrm{LS}(p)$ with an SPP-assistant. We first state some useful results from linear algebra.

Proposition 4.2 *Let $W < V < U$ are finite dimensional vector spaces over a field F. Let $A = \{\alpha_1, \ldots, \alpha_n\}$ be a basis for U and $B = \{\beta_1, \ldots, \beta_m\}$ be a basis for W. Let $A' \subseteq A$ be such that $A' \cup B$ is a set of maximally independent vectors in U. Then $A' \cup B$ is a basis for U. Moreover, $V = W \oplus (\langle A'\rangle \cap V)$.*

Let F be a field, and U be a nontrivial subspace of F^n. Consider a series of vector spaces $U_0, \ldots, U_n$ where $U_i = U \cap Sol(\mathcal{X}_i)$. Then U_k, where k is the maximum j such that U_j is nontrivial is called *0-index* of U. Next proposition is obvious.

Proposition 4.3 *Let F be a field, and U be a nontrivial subspace of F^n. Let k be the 0-index of U. Then U_k is a one dimension subspace of F^n.*

Theorem 4.4 *The representation class* $\mathrm{LS}(p)$ *is polynomial-time learnable with an SPP-assistant.*

Proof Sketch.

For simplicity of presentation, in this extended abstract, we prove the theorem for the class $\mathrm{LS}(2)$. This proof can be adapted with minor modifications, for the general case. We prove this in two parts. First we define a teaching assistant $L(\mathrm{LS}(2))$ and show that $\mathrm{LS}(2)$ can be polynomial time learnable using a deterministic machine with $L(\mathrm{LS}(2))$ as oracle. Then we prove that $L(\mathrm{LS}(2)) \in \mathrm{SPP}(\mathrm{LS}(2))$.

On input 0^n, let $V < F_2^n$ be the target space $\mu_n(r)$. Define $L(\mathrm{LS}(2))$ as: $L(\mathrm{LS}(2)) \triangleq \{\langle 0^n, A, i, y, r\rangle | i$ is the 0-index of $(\langle A\rangle \cap \mu_n(r)); \exists z \in \Sigma^{n-|y|}$ with $yz \in \langle A\rangle \cap \mu_n(r); yz \neq \mathbf{0}\}$.

We first give an intuitive idea of how to learn the target concept V. Our learner uses the same technique of "constructing a basis from inside" to learn V, as in the case of a learner who uses equivalence queries. Suppose we have constructed an independent set $B \subseteq V$ of vectors. We can use $L(\mathrm{LS}(2))$ to prefix search for a vector $v \in V$ but not in $\langle B\rangle$ as follows. Let A be a basis for F_2^n. Construct a set A' such that $A' \cup B$ is maximally independent vectors. (This can be done using a series of rank computations, which is in polynomial time.) Next, find out the 0-index of the space $(\langle A'\rangle \cap V)$ by serially searching using $L(\mathrm{LS}(2))$ as oracle, by asking queries of the form $\langle 0^n, A', i, \epsilon, r\rangle$ (ϵ is the empty string) by varying i starting from 1. Once 0-index has been found out, using prefix searching, v can be constructed. Notice that, as far as $\langle B\rangle$ is not the full target space, Proposition 4.2 ensures that the space $(\langle A'\rangle \cap V)$ is nontrivial and every element in this space, except $\mathbf{0}$, is independent of vectors in $\langle B\rangle$. This ensures that by adding the vector v thus constructed to the set B, will 'grow' the space $\langle B\rangle$ 'towards' the target concept V. This process terminates in maximum of n steps. We now give a description of the learning algorithm.

LEARN-LINEAR(0^n)
1 $B \leftarrow \{\mathbf{0}\}$
2 $A \leftarrow \{e_1, \ldots, e_n\}$ (* Standard basis for F_2^n *)
3 Construct $A' \subseteq A$ such that $B \cup A'$ is maximally independent
4 $i \leftarrow 1$
5 **while** $\langle 0^n, A', i, \epsilon, r\rangle \notin L(\mathrm{LS}(2))$
6   **do** $i \leftarrow i + 1$
7 **if** $i = n + 1$
8   **then output** B and stop
9 $y \leftarrow \epsilon$
10 **for** $j \leftarrow 1$ **to** n
11   **do if** $\langle 0^n, A', i, y1, r\rangle \in L(\mathrm{LS}(2))$
12   **then** $y \leftarrow y1$
13   **else** $y \leftarrow y0$
14 $B \leftarrow B \cup \{y\}$
15 **goto** 3

We now show that $L(\mathrm{LS}(2)) \in \mathrm{SPP}(\mathrm{LS}(2))$. We construct a GapP machine M which on input $\langle 0^n, A', i, y\rangle$, uses $\mu_n(r)$ as oracle, and has the property that $\langle 0^n, A', i, y, r\rangle \in L(\mathrm{LS}(2))$ if $M(\langle 0^n, A', i, y\rangle)$ produces $gap = 1$, and M produces $gap = 0$ if the input is not in $L(\mathrm{LS}(2))$.

To this end we define another NP machine N (which uses $\mu_n(r)$ as oracle). On input $\langle 0^n, A', i, y, k\rangle$ the machine N guesses a non-zero vector $v \in A' \cap Sol(\mathcal{X}_k)$ such that v is an extension of y and yields an accepting path iff $v \in \mu_n(r)$. It is easy to see that if k is greater than the 0-index of $A' \cap \mu_n(r)$ then none of the vectors v give rise to an accepting path. Also if k is exactly the 0-index then there is exactly one vector v that yields an accepting path (assuming that y is the prefix of that vector, of course).

We now design the GapP machine M using N. Let $gap_N(\langle 0^n, A', i, y, k\rangle)$ denote the number of accepting paths produced by the machine N on input $\langle 0^n, A', i, y, k\rangle$.

Description of $M(\langle 0^n, A, i, y\rangle)$

1 Produce a *gap* of $gap_N(\langle 0^n, A', i, y, i\rangle) \cdot \prod_{k=i+1}^{n}(1 - gap_N(\langle 0^n, A', i, y, k\rangle))$.

By the closure properties of GapP functions, M is clearly a GapP machine. Notice that on input $\langle 0^n, A', i, y\rangle$, if i is not the 0-index then M produces $gap = 0$. Furthermore, if i is the 0-index then M produces $gap = 1$ if y is the prefix of the unique non-zero vector $v \in A' \cap \mu_n(r) \cap Sol(\mathcal{X}_i)$, and otherwise produces $gap = 0$. Hence $L(\mathrm{LS}(2))$ is in $\mathrm{SPP}(\mathrm{LS}(2))$.

■

Acknowledgments We thank the referees for helpful comments.

References

1. D. Angluin. Queries and concept learning. *Machine Learning*, 2:319-342, 1988.
2. D. Angluin. Negative results for equivalence queries. *Machine Learning*, 5:121-150, 1990.
3. A. Blummer, A. Ehrenfeucht D. Haussler and M Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM,* 36(4):929-965, 1989.
4. J. L. Balcázar, J. Díaz, R. Gavaldà and O. Watanabe. A note on the query complexity of learning DFA. *Proceedings of the Third Workshop on Algorithmic Learning Theory*, 53-62, 1992.
5. S. Fenner, L. Fortnow, S. Kurtz. Gap-definable counting classes. *Proc. 6th Structure in Complexity Theory Conference,* 30-42, 1991.
6. M. Furst, J. E. Hopcroft and E. Luks. Polynomial time algorithms for permutation groups. *Proc. 21st IEEE Symposium of Foundations of Computer Science*, 36-45, 1980.
7. R. Gavaldà. On the power of equivalence queries. *EUROCOLT*, 193-203, 1993.
8. M. Hall. *The Theory of Groups.* Macmillan, New York, 1959.
9. L. Hellerstein, K. Pillaipakkamnatt, V. Raghavan and D. Vilkins. How many queries are needed to learn? *Proc. of the* 27^{th} *Symposium on the Theory of Computing*, 190-199, 1995.
10. K. Hoffmann and R. Kunz. *Linear Algebra.* Prentice Hall Inc., 1971.
11. J. Köbler, U. Schöning, J. Torán. Graph isomorphism is low for PP. *Journal of Computational Complexity,* 2: 301-310, 1992.
12. L. Valiant. A theory of the learnable. *Communications of the ACM,* 27(11):1134-1142, 1984.
13. O. Watanabe. A formal study of learning via queries. 17^{th} *International Colloquium on Automata, Languages and Programming,* LNCS 443: 139-152, 1990.

14. O. Watanabe and R. Gavaldà. Structural analysis of polynomial time query learnability. *Mathematical Systems Theory,* 27:231-256, 1994.