Active Learning of Group-Structured Environments

Gábor Bartók, Csaba Szepesvári*, Sandra Zilles

University of Alberta, Department of Computing Science,
Edmonton, Alberta, Canada
{bartok,szepesva,zilles}@cs.ualberta.ca

Abstract. The question investigated in this paper is to what extent an input representation influences the success of learning, in particular from the point of view of analyzing agents that can interact with their environment. We investigate learning environments that have a group structure. We introduce a learning model in different variants and study under which circumstances group structures can be learned efficiently from experimenting with group generators (actions). Negative results are presented, even without efficiency constraints, for rather general classes of groups showing that even with group structure, learning an environment from partial information is far from trivial. However, positive results for special subclasses of Abelian groups turn out to be a good starting point for the design of efficient learning algorithms based on structured representations.

1 Introduction

The question investigated in this paper is to what extent an input representation influences the success of learning, in particular from the point of view of analyzing agents that can interact with their environment. For simplicity, we assume that the agent has a finite number of actions, the execution of which results in a change in the state of the environment. The goal of the agent is to learn to predict the outcomes of its actions in terms of the changes of the environment's state. Of course, the agent has to have some inputs or sensations that reveal information about the state of the environment. In the simplest case the sensations reveal all the details of the state. Then the question is if the representation of the sensations influences the speed of learning. Is it necessary to assume a good representation, or can we design learning methods capable of efficiently learning in a broad class of environments irrespective of the input representation? This is a fundamental question in learning (and more broadly, in artificial intelligence).

In search problems for example, if the state is given with a factored (vectorial) representation then memory-based heuristics can be used to create good heuristics that lead to efficient planning methods (*e.g.,* [4, 9, 10]). Further, with a factored input representation predictive models can be learned efficiently in

* Csaba Szepesvári is on leave from MTA SZTAKI, Budapest, Hungary.

some non-trivial classes of environments, such as when the environment can be represented as a Bayes net described with bounded depth decision trees [16].

At first sight efficient learning given an arbitrary input representation seems impossible. By efficient learning we mean learning well ahead of the time before the agent has seen all the states. The conventional way to evaluate if an agent has learnt its environment is to look at if the agent is able to predict its future sensations given any (hypothetical) action sequence [13, 11]. This model leads to an easy negative result: If the sensations of the agent are some arbitrary codes (names) for the states then it is clearly impossible for the agent to disambiguate between state names not encountered before.

Luckily, predicting the names of states is not necessary for an agent to be successful. Take, *e.g.,* a reinforcement learning agent whose primary interest is to predict rewards and eventually gather them. By building an internal model of the environment, such agents might be able to create plans for gathering rewards in a successful way. One (admittedly contrived) example that shows why predicting names is not necessary is as follows: Assume the agent has two actions and a reward is achieved iff the two actions are taken in a certain combination. The agent that discovers this action sequence can succeed without ever memorizing any of the state names. (However, the observations now have an internal structure, *i.e.,* the reward is separated from the state names.) A different example is a nicely structured environment, *e.g.,* when the agent has d actions and the state can be represented with a vector over a finite set and the actions influence mutually disjoint sets of the components of this vector. In this case the agent that assumes "independence" of the actions unless some experience contradicts this, can learn a representation that can be used to predict outcomes after $O(d)$ interactions, while the size of the state space is exponential in d.

1.1 The approach of learning group-structured environments

Since learning without seeing all the states is impossible when the environment lacks structure, a natural goal is to require the time needed for learning to scale with how well-structured the environment is. We investigate this under the simplifying assumption that the environments can be represented as (mathematical) groups. This implies that these environments are deterministic and that for any sequence of actions the length of the orbit generated with the resulting composite action is independent of the starting state. Due to their strong structure, these environments look ideal for initiating the study. Moreover, there are many interesting environments which belong to this class, consider, *e.g.,* permutation games, such as Rubik's cube [9], the Topspin Puzzle or the Pancake Puzzle [7].

We introduce a model for efficient learning of group-structured environments by exploration, imposing a bound on the number of actions the agent can take before converging to a correct conjecture about the target group environment.

Learnability results turn out to strongly depend on the underlying presentation and thus the set of generators given as input to the learner, both if efficiency constraints are maintained and if they are dropped. However, the strength of the negative results actually will show exactly what motivates our research, namely

that the success of efficient learning algorithms is affected by the choice of the representation. Our results indicate that even group structure is not enough to guarantee success—the choice of generators presented to the learner is also essential. We study this for finitely generated and for finite groups, for Abelian groups, and for the special case of dihedral groups. In particular the Abelian groups have a structure that is more promising for efficient learning and we show possible steps towards exploiting their structure.

1.2 Related work

The closest to the present work is the work of Rivest and Schapire [13] who investigate the problem of inferring a finite automaton by interaction. (For extensions to stochastic environments see [8, 11].) They assume that the agent's sensations come from a finite set, which might be different from the set of states. (In [13] the observations are binary vectors, but this assumption can be lifted.) According to the definition in [13] an environment is learned when the agent can predict future sensations for an arbitrary sequence of actions given its current state. They give a randomized algorithm that is capable of learning permutation automata (with high probability) in time that is polynomial in the "diversity" associated with the environment. The diversity is the number of equivalence classes of tests in the environment. A test is a sequence of actions and sensations. The outcome of a test in a given state is true or false depending on whether the test's observations are sensed in the order specified in the test provided that the test's actions are executed, again, in the order specified in the test. Two tests are equivalent if they give the same outcomes independently of the start state.

The diversity d of an environment is (tightly) bounded by $\log_2(n) \leq d \leq 2^n$, where n the number of states of the environment [13]. The diversity depends to a large extent on how the sensations (*i.e.,* the inputs) are chosen, underlining again the importance of working with the right inputs.

Our framework corresponds to the case when the sensations have a one-to-one correspondence with the states. Thus the diversity of the resulting environment is n. As we are interested in learning with $o(n)$ interactions, the present work can be viewed as an attempt to improve upon the results of [13].

We are not aware of any research on our setting of group learning. Related work concerning groups and learning has a focus completely different from that of the framework we introduce, see, *e.g.,* models of learning algebraic structures from positive data [15]. Studies on *black-box groups* [17], [2] analyze objects different from those in our group environment setting. Related, but not focusing on learning is, *e.g.,* [5] that introduced a probabilistic algorithm to decide whether an algebraic structure is an Abelian group and [1] that investigated how to derive the irreducible decomposition of a given linear representation of a group.

2 Preliminaries

In this section we introduce the basic notions used throughout the paper. We assume the reader to be familiar with a few basic group theoretic notions; those used without any further explanation are taken from [14].

$\mathbb{N}$ denotes the set of all natural numbers, $\mathbb{Z}$ the set of all integers. Let $\mathcal{G} = (S_\mathcal{G}, \circ_\mathcal{G})$ be a group, where $S_\mathcal{G}$ is the domain of $\mathcal{G}$ and $\circ_\mathcal{G}$ the group operation. We always use $\lambda_\mathcal{G}$ to denote the neutral element in $\mathcal{G}$, but drop the subscript $\mathcal{G}$ in $\lambda_\mathcal{G}$, in $S_\mathcal{G}$, and in $\circ_\mathcal{G}$ if the underlying group is clear from the context. If $A \subseteq S_\mathcal{G}$ is any subset of $S_\mathcal{G}$ then by A^* we denote the set $\{(a_1, \ldots, a_m) \mid m \in \mathbb{N}\}$ of all finite sequences of elements in A, including the empty sequence. With every sequence $\alpha = (a_1, \ldots, a_m)$ we associate a group element $\mathcal{G}(\alpha) \in S_\mathcal{G}$, namely

$$\mathcal{G}(\alpha) = \begin{cases} \lambda_\mathcal{G}\,, & \text{if } m = 0\,, \\ a_1 \circ_\mathcal{G} (a_2 \circ_\mathcal{G} (\ldots \circ_\mathcal{G} (a_{m-1} \circ_\mathcal{G} a_m) \ldots))\,, & \text{if } m > 0\,. \end{cases}$$

From now on we will omit the symbol for group operations, wherever it is clear from the context, *e.g.,* for two elements $a, b \in S$ we write ab rather than $a \circ_\mathcal{G} b$.

The elements of a set $A \subseteq S_\mathcal{G}$ are called *generators* of $\mathcal{G}$ if every element in S can be written as a product of elements in A, *i.e.,* if $S_\mathcal{G} = \{\mathcal{G}(\alpha) \mid \alpha \in A^*\}$. A *relation* in $\mathcal{G}$ is a sequence $\alpha \in S_\mathcal{G}^*$ such that $\mathcal{G}(\alpha_1) = \lambda_\mathcal{G}$. A pair $\langle A \mid R\rangle$ is called a *presentation* of $\mathcal{G}$ iff A is a set of generators for $\mathcal{G}$ and R is a set of relations in $\mathcal{G}$ such that $\mathcal{G} = F_A/(R)$, *i.e.,* $\mathcal{G}$ is the factor group of the free group on A and its smallest normal subgroup that contains R. For ease of presentation, we usually omit set brackets when writing $\langle A \mid R\rangle$ explicitly. For instance, a presentation for the (so-called) Klein group is $\langle a, b \mid a^2, b^2, (ab)^2\rangle$ (rather than $\langle\{a, b\} \mid \{a^2, b^2, (ab)^2\}\rangle$). A presentation $\langle A \mid R\rangle$ is finite if both A and R are finite.

$\mathcal{G}$ is called (i) *finitely generated* if $\mathcal{G}$ has a finite set of generators; (ii) *finitely presented* if $\mathcal{G}$ has a finite presentation; (iii) *finite* if $S_\mathcal{G}$ is finite.

A *representation* of $\mathcal{G}$ over a vector space V is a homomorphism $\Phi : S_\mathcal{G} \mapsto \mathrm{GL}(V)$, where $\mathrm{GL}(V)$ is the *general linear group* of V.

For any $a \in S_\mathcal{G}$, $\langle a\rangle$ denotes the *cyclic subgroup* generated by a. C_k denotes the *cyclic group* of order k. A *p-group* is a finite group of order p^k, where p is a prime and k is a positive integer.

The *order* of an element $g \in S_\mathcal{G}$ is the lowest positive integer k such that $g^k = \lambda_\mathcal{G}$. We denote the order of g by $\sigma(g)$.

For any subset $\mathcal{H}$ we write $\mathcal{H} \leq \mathcal{G}$ if $\mathcal{H}$ is a subgroup of $\mathcal{G}$, and $\mathcal{H} \lhd \mathcal{G}$ if $\mathcal{H}$ is a normal subgroup of $\mathcal{G}$. If $\mathcal{H}_1 = (S_{\mathcal{H}_1}, \circ_\mathcal{G})$ and $\mathcal{H}_2 = (S_{\mathcal{H}_2}, \circ_\mathcal{G})$ are two subgroups of $\mathcal{G}$ then we define $S_{\mathcal{H}_1} S_{\mathcal{H}_2} = \{h_1 \circ_\mathcal{G} h_2 \mid h_1 \in S_{\mathcal{H}_1} \text{ and } h_2 \in S_{\mathcal{H}_2}\}$. The automorphism group of $\mathcal{G}$ is referred to by $\mathrm{Aut}(\mathcal{G})$.

If $\mathcal{G}_1$ and $\mathcal{G}_2$ are two groups, then $\mathcal{G}_1 \times \mathcal{G}_2$ denotes their direct product. Here note again that for ease of presentation we identify groups with their domains.

Definition 1 (Semi-direct product: inner definition). *Let $\mathcal{G} = (S_\mathcal{G}, \circ_\mathcal{G})$ be a group and $\mathcal{N} \lhd \mathcal{G}, \mathcal{H} \leq \mathcal{G}$ such that $S_\mathcal{N} S_\mathcal{H} = S_\mathcal{G}, S_\mathcal{N} \cap S_\mathcal{H} = \{\lambda_\mathcal{G}\}$. Let $\phi : S_\mathcal{H} \mapsto Aut(\mathcal{N})$ be the group homomorphism such that*

$$\forall n \in S_\mathcal{N}\ \forall h \in S_\mathcal{H}\ [\phi(h)(n) = hnh^{-1}]\,.$$

Then $\mathcal{G}$ is the semi-direct product of $\mathcal{N}$ and $\mathcal{H}$ with respect to ϕ, written $\mathcal{G} = \mathcal{N} \rtimes_\phi \mathcal{H}$.

3 A model for learning group-structured environments

In this section we introduce our basic model of learning group-structured environments, the underlying scenario of which is as follows:

An agent is exploring a (finite or infinite) state environment. There is a finite set of actions that the agent can take in every state of the environment; taking an action usually causes the state of the environment to change. Now assume the agent can always observe the name of the state the environment is currently in and thus can always recognize when it gets back to some previously visited state. This allows the agent to find out relations between actions.

We assume the environment to be static and deterministic, *i.e.,* there is one function that determines for every state s and every action a the successor state after taking action a in state s. Formally this can be defined as follows:

Definition 2 (Environment). *An environment is a triple $\mathcal{E} = (S, A, T)$, where S is a countable set, A is a finite set, and $T : S \times A \mapsto S$ is a mapping.*

The elements of S are called states; the elements of A are called actions. For every $s \in S$ and every $a \in A$ we denote by $a(s)$ the state $T(s, a)$.

Fix an environment (S, A, T). We now extend T to action sequences. For this we identify the set of action sequences with the free monoid $(A^*, \circ)$ (where $\circ$ is the concatenation over A^*). The empty action sequence, denoted by λ, is the identity element of this monoid. We extend the definition of T by $T(s, a_1 \dots a_m) = a_m(\dots(a_2(a_1(s)))\dots)$ for all $(a_1 \dots a_m) \in A^*$. We use $(a_1 \dots a_m)(s)$ as a shorthand for $T(s, a_1 \dots a_m)$.

Definition 3 (Equivalence of action sequences). *Let $\mathcal{E} = (S, A, T)$ be an environment and $\alpha_1, \alpha_2 \in A^*$ be action sequences. Then α_1 and α_2 are equivalent in $\mathcal{E}$ (denoted by $\alpha_1 \equiv_{\mathcal{E}} \alpha_2$) iff $\alpha_1(s) = \alpha_2(s)$ for all $s \in S$. Let $S_{\mathcal{E}}$ denote the corresponding set of equivalence classes over A^*.*

The concatenation on A^* induces an operator $\circ$ on $S_{\mathcal{E}}$, which we will call concatenation, too. The subscript $\mathcal{E}$ in $\equiv_{\mathcal{E}}$ and/or $S_{\mathcal{E}}$ may be omitted when unambiguous.

We focus on scenarios in which the environment obeys a group structure.

Definition 4 (Group environment). *Let $\mathcal{E} = (S, A, T)$ be an environment. $\mathcal{E}$ is called a group environment if $(S_{\mathcal{E}}, \circ)$ is a group. With a slight abuse of notation we refer to $(S_{\mathcal{E}}, \circ)$ also by $\mathcal{E}$.*

If $\mathcal{E} = (S, A, T)$ is a group environment then A is a set of generators for the corresponding group $\mathcal{E}$, *i.e.,* we are always considering finitely generated groups.

We will analyze the learning problem of determining the structure of an unknown group environment $\mathcal{E} = (S, A, T)$ by exploration. In particular, we assume that the agent can take any action $a \in A$ in any state $s \in S$. After taking action a, the agent will observe the name of the state $a(s)$. In every step the agent outputs a hypothesis about the environment. The basic question we pose is: for certain classes of group environments, can the agent—in a "reasonable" number of steps—learn to solve the *word problem* for $\mathcal{E} = (S_{\mathcal{E}}, \circ)$?

Definition 5 (Word problem). *Let $\mathcal{G}$ be a group and A a set of generators for $\mathcal{G}$. The word problem for $\mathcal{G}$ over A is solvable if there is a recursive decision procedure $d : A^* \to \{0, 1\}$ such that for all $w \in A^*$*

$$d(w) = \begin{cases} 1\,, & \text{if } w \equiv \lambda_{\mathcal{G}}\,, \\ 0\,, & \text{if } w \not\equiv \lambda_{\mathcal{G}}\,. \end{cases}$$

Clearly, if the agent possesses d as defined above then he knows what action sequences are equivalent. This is enough for the agent to construct a consistent representation of the environment.

Consequently, we model the agent as a learning algorithm as follows:

Definition 6 (Learning algorithm). *A learning algorithm is an algorithm L that fulfills the following properties.*

1. *L takes as its initial input a finite set A of actions and the name of a state $s_0 \in S$, where $\mathcal{E} = (S, A, T)$ is an unknown group environment.*
2. *L operates in steps, starting in Step 0, where in Step n for $n \in \mathbb{N}$ either (i) or (ii) holds.*
   (i) L sends some $a \in A$ to an oracle and receives $a(s_n)$ as a reply. Then L returns a recursive decision procedure D_n[1] and goes to Step $n + 1$ with $s_{n+1} = a(s_n)$.
   (ii) L stops and never goes to Step $n + 1$.

Note that the group corresponding to a group environment could be any finitely generated group; so when defining learning models and making formal statements about learning group environments, we assume a one-one correspondence between finitely generated groups and group environments. Thus our learning criterion is defined as follows, based on Gold's [6] model of learning languages.

Definition 7 (Learning group environments). *Let $\mathcal{C}$ be a class of groups and P a set of presentations such that $\mathcal{C} = \{\mathcal{G} \mid \mathcal{G} = \langle A \mid R\rangle$ for some $\langle A \mid R\rangle \in P\}$. For every $\mathcal{G} \in \mathcal{C}$, let $P_{\mathcal{G}} = \{\langle A \mid R\rangle \in P \mid \mathcal{G} = \langle A \mid R\rangle\}$.*

We say that $\mathcal{C}$ is learnable (*) in the limit, (**) finitely, (***) efficiently with respect to P if there is a polynomial q and a learning algorithm L that, for every group $\mathcal{G} \in \mathcal{C}$, every presentation $\langle A \mid R\rangle \in P_{\mathcal{G}}$, and every state s in the group environment corresponding to $\mathcal{G}$, the followings hold: If L is given (A, s) as an input then there is a decision procedure D that solves the word problem for $\mathcal{G}$ over A and there is an $n_0 \in \mathbb{N}$ such that the output of L in Step n equals D and

(*) for all $n \geq n_0$ the output of L in Step n equals D.
(**) L stops in Step $n + 1$.
(***) for all $n \geq n_0$ the output of L in Step n equals D and $n_0 \leq q(\log |S_{\mathcal{G}}| + \sum_{a \in A} \sigma(a))$.

[1] More precisely, L will return a program (on some suitable language) that represents the recursive decision procedure.

In the third point, $\sum_{a\in A}\sigma(a)$ appears in the upper bound because the learning algorithms should be allowed to determine the orders of the actions. One could think intuitively that $\sum_{a\in A}\sigma(a)$ and $\log(|S_{\mathcal{G}}|)$ correlate polynomially, but in the case of non-Abelian groups, $|S_{\mathcal{G}}|$ can be arbitrarily large, while $\sum_{a\in A}\sigma(a)$ is fixed. That instead of $|S_{\mathcal{G}}|$ the bound includes $\log|S_{\mathcal{G}}|$ is motivated by the desire to learn well before seeing all the states.

If $\mathcal{C}$ is a class of groups and P a set of presentations such that $\mathcal{C} = \{\mathcal{G} \mid \mathcal{G} = \langle A \mid R\rangle$ for some $\langle A \mid R\rangle \in P\}$ then we call $(\mathcal{C}, P)$ a *group learning problem*.

It does not make sense to consider efficient learning of infinite groups, because then the bound on the number of steps is infinite. The models of finite learning and learning in the limit of course can be studied also for infinite groups.

We close this section with a first result, namely that in certain cases learning algorithms can be normalized to operate with a restricted form of queries. For this purpose we introduce the notion of 0/1-*learning algorithms*.

A 0/1-learning algorithm is defined very similarly to a learning algorithm in Definition 6, the only difference is that upon a query a sent to the oracle in Step n and state s_n, instead of $a(s_n)$ it receives the reply 1 if $a(s_n) = s_0$ and 0 if $a(s_n) \neq s_0$. The notion of 0/1-learnability (in the limit, finite, or efficient) can then immediately be derived from Definition 7 by replacing "learning algorithm" by "0/1-learning algorithm". We call this concept learning with *binary observation*.

Lemma 1. *Let $(\mathcal{C}, P)$ be a group learning problem such that $\sigma(a)$ is finite for all $\langle A \mid R\rangle \in P$ and all $a \in A$. $\mathcal{C}$ is learnable in the limit (finitely learnable, efficiently learnable) with respect to P iff $\mathcal{C}$ is* 0/1*-learnable in the limit (finitely* 0/1*-learnable, efficiently* 0/1*-learnable) with respect to P.*

Proof. We only show that a 0/1-learning algorithm can be constructed from a learning algorithm according to Definition 6; the other direction is trivial.

The idea is that all the information gathered by the original learner up to some stage is wether a subsequence of the action sequence posed leaves a state unchanged. This information can be recovered with the binary observations by testing all subsequences from the initial state. This way the number of queries is blown up only polynomially.

Formally, the 0/1-learner works as follows: Initially, starting from s_0, the learner experiments with all the actions:

- For each $a \in A$ repeatedly query a until the answer 1 is received and then set $\sigma(a)$ equal to the number of times a was queried.

Now, assume a learning algorithm L as in the original definition poses the query a_n in Step n. As a response the 0/1-learner does the following:

- Let $\alpha = (x_0, \ldots, x_t)$ be the action sequence

$$\circ_{i=0}^{n}[(a_i, a_{i+1}, \ldots, a_n, a_n^{-1}, \ldots, a_{i+1}^{-1}, a_i^{-1})]\,.$$

  Query the actions in this sequence one by one.
- Let M be the set of all subsequences $x_j, \ldots, x_{j+z}$ of α such that

- $j = 0$ or the reply after querying x_{j-1} was 1; and
- the reply after querying x_{j+z} was 1.

– If there exists an $i \leq n$ such that $(a_i, \ldots, a_n) \in M$ then return the state name returned in Step $i-1$ for the minimal such i (or return s_0 if $i = 0$). If there is no such i then return a new state name.
– Return the hypothesis that L returns.

Obviously this 0/1-learner solves all the learning problems that L solves at the price of a polynomial increase of execution time. □

4 An analysis of the group learning model

In this section we analyze our learning models first for the classes of all finitely generated and all finite groups. The results and proofs will motivate the analysis of some special subgroups like the dihedral groups and Abelian groups.

4.1 General results on learning group environments

Starting with a quite general case, the class of all finitely generated groups, one easily obtains a negative result, independent of the set of presentations chosen.

Theorem 1 *1. The class of finitely generated groups is not learnable in the limit with respect to any set of presentations.*
2. The class of finitely presented groups is not learnable in the limit with respect to any set of presentations.

Proof. Both assertions follow immediately from the fact that there is a finitely presented (and thus a finitely generated group) $\mathcal{G}$ such that, for every presentation $\langle A \mid R \rangle$ of $\mathcal{G}$, the word problem over A^* is unsolvable [3, 12]. □

Restricting our focus to classes of finite groups (for which the word problem is always solvable) we at least get learnability in the limit, yet efficient learnability is not achievable.

Theorem 2 *Let $\mathcal{C}$ be the class of all finite groups and P the set of all presentations of finite groups.*

1. $\mathcal{C}$ is not learnable efficiently with respect to P.
2. $\mathcal{C}$ is learnable finitely with respect to P.
3. $\mathcal{C}$ is learnable in the limit with respect to P.

Proof. The third assertion is easy to prove; a learning algorithm can exhaustively explore the effects of the actions.

The proof of the second assertion requires a dove-tailing argument by interleaving two procedures: one exploring action sequences in order to determine group relations, the other trying to find a $k \in \mathbb{N}$ such that all action sequences of length k are equivalent to a shorter action sequence. Such a k must exist and can

be found in a finite number of steps after enough relations are known. Knowing such a k, exploration using all sequences of length up to k will yield all further relations necessary to uniquely identify the target group.

A detailed proof we only give for the first assertion.

Assume $\mathcal{C}$ is learnable efficiently with respect to P. Then there is a learning algorithm L and a polynomial q such that L learns every finite group $\mathcal{G}$ with domain $S_{\mathcal{G}}$ efficiently from any of its sets A of generators, using no more than $q(\log(|S_{\mathcal{G}}|) + \sum_{a \in A} \sigma(a))$ many steps. By Lemma 1 we can assume without loss of generality that L is a 0/1-learning algorithm.

Note that there is an $m_* \in \mathbb{N}$ such that $2q(m) > q(\log(2q(m) + 2) + 4)$ for all $m \geq m_*$.

We define two groups and show that they cannot be distinguished by L:

- $\mathcal{G}_1 = \langle a, b \mid a^2, b^2, (ab)^{q(m_*)} \rangle$
- $\mathcal{G}_2 = \langle a, b \mid a^2, b^2, (ab)^{q(m_*)+1} \rangle$

We will show below that (i) the size of the domain of $\mathcal{G}_1$ is $2q(m_*)$, (ii) the size of the domain of $\mathcal{G}_2$ is $2q(m_*)+2$, (iii) for all $\alpha \in \{a, b\}^*$ with $|\alpha| < 2q(m_*)$:

$$\alpha \equiv_{\mathcal{G}_1} \lambda_{\mathcal{G}_1} \iff \alpha \equiv_{\mathcal{G}_2} \lambda_{\mathcal{G}_2}$$

This implies that $\mathcal{G}_1$ and $\mathcal{G}_2$ are distinct finite groups but L cannot distinguish $\mathcal{G}_1$ from $\mathcal{G}_2$ by asking $q(\log(|S_{\mathcal{G}_2}|) + \sigma(a) + \sigma(b)) = q(\log(2q(m_*) + 2) + 4)$ $(< 2q(m_*))$ many queries.[2] This is a contradiction, so the class of all finite groups is not learnable efficiently with respect to all presentations of finite groups.

Claim 1. Let $\alpha \in \{a, b\}^*$ with $|\alpha| < 2q(m_*)$. Then $\alpha \equiv_{\mathcal{G}_1} \lambda_{\mathcal{G}_1} \iff \alpha \equiv_{\mathcal{G}_2} \lambda_{\mathcal{G}_2}$.

Proof of Claim 1. We show one direction only, the other one is similar.

Let α be as given and assume that $\alpha \equiv_{\mathcal{G}_2} \lambda_{\mathcal{G}_2}$. We know that α can be reduced to a $\mathcal{G}_1$-equivalent sequence α' with $0 \leq |\alpha'| \leq |\alpha|$, such that α' does not contain the substrings aa or bb. If $|\alpha'| = 0$ then $\alpha \equiv_{\mathcal{G}_1} \lambda_{\mathcal{G}_1}$. We show that if $0 < |\alpha'| < 2q(m_*)$ then $\alpha \not\equiv_{\mathcal{G}_2} \lambda_{\mathcal{G}_2}$. This is a contradiction and we are done.

So assume $0 < |\alpha'| < 2q(m_*)$. α' results from α by iteratively applying the following rules.

(I) insert or delete aa (II) insert or delete bb (III) insert or delete $(ab)^{2q(m_*)}$

We introduce a function $\mu : \{a, b\}^* \mapsto \mathbb{Z}$. For any $\alpha = (a_1, \ldots, a_n) \in \{a, b\}^*$ let $\alpha_{odd} = (a_1, a_3, \ldots, a_{n_1})$ and $\alpha_{even} = (a_2, a_4, \ldots, a_{n_2})$, where $n_1 = n_2 + 1 = n$ for odd n and $n_1 + 1 = n_2 = n$ for even n. Let $\mu(w) =$ (# of 'a's in α_{odd}) − (# of 'a's in α_{even}) + (# of 'b's in α_{even}) − (# of 'b's in α_{odd}). For example, if $\alpha = aaababbabab$, then $\alpha_{odd} = aaabbb$ and $\alpha_{even} = abbaa$, so $\mu(\alpha) = -1$.

When modifying α to α', the parities of the old/remaining letter positions do not change. Therefore, only the new/disappearing letters will affect the value of μ. Using rules (I) or (II), the value of μ will not change at all. The use of rule

[2] Note here that $\sigma(a) = \sigma(b) = 2$ holds in both groups; technically we should in fact have subscripts with σ in order to relate it to a specific group.

(III), depending on its type (insert or remove) and the position (odd or even) chosen, will either increase or decrease the value of μ by $2q(m_*)$.

This implies that if $\alpha_1 \equiv_{\mathcal{G}_1} \alpha_2$ then $\mu(\alpha_1) \equiv \mu(\alpha_2) \mod (2q(m_*))$.

Since α' does not contain the substrings aa or bb, we have $\mu(\alpha') = \pm|\alpha'|$. Obviously, $\mu(\lambda) = 0$. Since $0 < |\alpha'| < 2q(m_*)$, $\mu(\alpha') \not\equiv \mu(\lambda) \mod (2q(m_*))$. Therefore $\alpha' \not\equiv_{\mathcal{G}_2} \lambda_{\mathcal{G}_2}$. □ *Claim 1.*

Claim 2. The size of the domain of $\mathcal{G}_1$ is $2q(m_*)$; the size of the domain of $\mathcal{G}_2$ is $2q(m_*) + 2$.

Proof of Claim 2. We prove the statement only for $\mathcal{G}_1$; for $\mathcal{G}_2$ the proof is similar. Note that two action sequences α and β are equivalent in $\mathcal{G}_1$, if β results from α by iteratively eliminating all substrings aa, bb, $(ab)^{q(m_*)}$. Hence every element in the domain of $\mathcal{G}_1$ can be written as a product in which none of these subsequences occur. These are

$$a(ba)^k,\ b(ab)^k,\ (ab)^k,\ (ba)^k$$

for $0 \leq k < q(m_*)$ (note that $(ab)^0 = (ba)^0 = \lambda$).

Now note that $(ba)^k \equiv_{\mathcal{G}_1} (ab)^{2q(m_*)-k}$ for $0 \leq k \leq 2q(m_*)$ (both forms obviously have the inverse $(ab)^k$). Hence also $a(ba)^k \equiv_{\mathcal{G}_1} a(ab)^{2q(m_*)-k} \equiv_{\mathcal{G}_1} (ba)^{2q(m_*)-k-1}b \equiv_{\mathcal{G}_1} b(ab)^{2q(m_*)-k-1}$ for $0 \leq k < q(m_*)$.

Hence the domain of $\mathcal{G}_1$ has exactly $2q(m_*)$ elements. □ *Claim 2.*

This completes the proof. □

$\mathcal{G}_1$ and $\mathcal{G}_2$ belong to the well-known class of finite dihedral groups. So not even those are learnable efficiently if all possible presentations are taken into account in the group learning problem. Since the dihedral groups illustrate a few principled properties of our learning model, we study them in more detail.

4.2 Learning dihedral groups

We will use the dihedral groups to illustrate two general phenomena in learning theory, namely the impact of the representation scheme for the information given to the learning algorithm and the impact that slight changes to the class of target concepts can have on learnability.

For every $k \geq 1$, let D_k denote the finite dihedral group with $2k$ elements. Note that there is only one infinite dihedral group, namely $D_\infty = \langle a, b \mid a^2, b^2 \rangle$.

Theorem 2 and its proof immediately yield the following corollary.

Corollary 1. *Let $\mathcal{C}$ be the class of finite dihedral groups and $P = \{\langle A \mid R \rangle \mid \langle A \mid R \rangle \in \mathcal{C}$ and $|A| = 2\}$.*

1. *$\mathcal{C}$ is not learnable efficiently with respect to P.*
2. *$\mathcal{C}$ is learnable finitely with respect to P.*
3. *$\mathcal{C}$ is learnable in the limit with respect to P.*

In fact it is not very surprising that the finite dihedral groups cannot be learned efficiently with respect to presentations with 2 generators. The reason is

simply that any learning algorithm would have to make a number of experiments that is linear in the size of the group (rather than logarithmic). This leads us to a nice observation showing how much the choice of generator systems, *i.e.,* the representation of the input to the learning algorithm, influences learnability. In fact the finite dihedral groups can be learned efficiently if we choose a set of presentations in which the size of the group is linear in the order of one of the generators, thus allowing for enough experiments to identify the target group.

Fact 1 *Let $\mathcal{C}$ be the class of finite dihedral groups and $P = \{\langle\{c, d\} \mid c^2, d^k, cdcd\rangle \mid k \geq 1\}$. Then $\mathcal{C}$ is learnable efficiently with respect to P.*

The proof is omitted; note that with $a = c$, $b = cd$, and $ab = d$ we have the equivalence of the group presentations $\langle a, b \mid a^2, b^2, (ab)^k\rangle$ and $\langle c, d \mid c^2, d^k, cdcd\rangle$.

The second phenomenon that can be easily illustrated using dihedral groups is how unstable learnability results can be with respect to slight changes to the target class. The class of all finite dihedral groups can be learned finitely from all binary generator systems, but this is no longer true if we add just a single group to this class, namely the infinite dihedral group.[3]

Theorem 3 *Let $\mathcal{C}^+$ be the class of all dihedral groups and $P^+ = \{\langle A \mid R\rangle \mid \langle A \mid R\rangle \in \mathcal{C}^+$ and $|A| = 2\}$.*

1. *$\mathcal{C}^+$ is not learnable finitely with respect to P^+.*
2. *$\mathcal{C}^+$ is learnable in the limit with respect to P^+.*

Proof. ad 1. Assume $\mathcal{C}^+$ is learnable finitely with respect to P^+. Then there is a 0/1-learner L that learns every dihedral group finitely from any set of two generators. Simulate L on input $\{a, b\}$ as follows (simulate an oracle in parallel).

Always reply 0 for the first action L takes. If the first two actions by L are equal, *i.e.,* they are aa or bb, then reply 1 after the second action. In any other case, for growing action sequences, reply 0.

This scenario is valid if D_∞ is the target group. Thus eventually L will return a decision procedure for D_∞ and stop the process. At this point of the learning process there are infinitely many finite dihedral groups consistent with the scenario. These are not identified by L although they are in $\mathcal{C}^+$—a contradiction.

ad 2. A learning algorithm on input $\{a, b\}$ initially tries to determine the order of both of the generators. In case one of these orders is $k \neq 2$, the algorithm will return a decision procedure for D_k forever. As long as one of the orders is not yet determined, the algorithm returns a decision procedure for D_∞.

In case both generators turn out to be of order 2, the algorithm tries to determine the minimal k such that $\lambda \equiv (ab)^k$, if such a k exists. As long as no such k is found, the algorithm returns a decision procedure for D_∞. As soon as such a k is found, the algorithm will start returning a decision procedure for D_k forever.

It is not hard to see that this algorithm witnesses Assertion 2. □

[3] This very much resembles results in Inductive Inference, where Gold [6] showed that no class of languages that contains all finite languages and at least one infinite language, can be learned in the limit.

4.3 Learning Abelian groups

For finitely generated Abelian groups, a reasoning similar to the proof of Theorem 3 shows similar differences between learning in the limit and finite learning.

Theorem 4 *Let $\mathcal{C}$ be the class of all finitely generated Abelian groups. Let $P = \{\langle a_1, \ldots, a_k \mid R\rangle \mid \langle a_1\rangle \times \ldots \times \langle a_k\rangle \in \mathcal{C}\}$.*

1. *$\mathcal{C}$ is learnable in the limit with respect to P.*
2. *$\mathcal{C}$ is not learnable finitely with respect to P.*

Proof. ad 1. According to the fundamental theory of finitely generated Abelian groups every group in $\mathcal{C}$ has the form $\mathbb{Z}^k \times C_{n_1}^{k_1} \times \cdots \times C_{n_z}^{k_z}$ for some $k, z, k_i, n_i \in \mathbb{N}$. The number of generators here is $k + k_1 + \cdots + k_z$.

A learning algorithm L witnessing Theorem 4 works as follows: Given a set $A = \{a_1, \ldots, a_k\}$ of generators, L initially always outputs a decision procedure for $\mathbb{Z}^{|A|}$. Given a canonical enumeration of all pairs (a_l, t) for $t \geq 1$ and $l \in \{1, \ldots, k\}$, L then queries the state names while taking action sequences $(a_l)^t$ for all (l, t) in canonical order. Whenever a sequence $(a_l)^t$ takes L back to the state it was in before this sequence, L changes its output from $\mathbb{Z}^{k'} \times C_{n_1}^{k'_1} \times \cdots \times C_{n_z}^{k'_z}$ to $\mathbb{Z}^{k'-1} \times C_{n_1}^{k'_1} \times \cdots \times C_{n_j}^{k'_j+1} \times \cdots \times C_{n_z}^{k'_z}$, where $n_j = t$. Moreover, from then on all pairs (a_l, t') in the enumeration will be skipped.

It is easy to prove that L learns all groups in $\mathcal{C}$ in the limit.

ad 2. Assume to the contrary that $\mathcal{C}$ is finitely learnable with respect to P, witnessed by a learning algorithm L.

Consider the behaviour of L for the target group $\mathbb{Z}$, generated by a single element. After finitely many experiments using this generator element, each of which leads L to a new state, L returns a decision procedure for $\mathbb{Z}$ and terminates.

At this point there are still infinitely many finite cyclic groups $C_n \in \mathcal{C}$ consistent with the scenario experienced by L; L fails to identify them. □

Concerning efficient learning, some properties of finite Abelian groups motivate the study of different kinds of learners, as introduced in the next section.

5 Learning injective representations

Every finite Abelian group can be written as a direct product of p-groups (for different primes p). This kind of group presentation turns out to be well suited for efficient learning, even for a specific kind of learning algorithms, namely some that return representations instead of decision procedures. Note that a decision procedure for a group can easily be obtained from a representation of the group.

A *representation-learning algorithm* is defined similarly to a learning algorithm (see Definition 6), except for that the outputs of the algorithm are always injective representations over $\mathbb{C}$ (of course with a correct representation in the end). Definitions of representation-learning are derived as usual.

Theorem 5 *Let $\mathcal{C}$ be the class of all finite Abelian groups. Let $P = \{\langle a_1, \ldots, a_k \mid R\rangle \mid \mathcal{G} = \langle a_1\rangle \times \ldots \times \langle a_k\rangle$ and $\sigma(a_i) < \infty$ for all $i \in \{1, \ldots, k\}\}$. $\mathcal{C}$ is efficiently representation-learnable from P.*

Proof. (Sketch.) We define a learning algorithm L on input A as follows:

For all $a \in A$, L determines $\sigma(a)$ by taking the action a until s_0 is reached.

Then L outputs a representation Φ, where $\Phi(a_i)$ is defined as a diagonal matrix where the i^{th} element of the diagonal is a primitive $\sigma\left(b_i\right)^{th}$ complex unit root; the other diagonal entries are 1. □

Here again we assume a specific underlying presentation. In the case of learning with respect to all possible presentations this result can still be used to show a reducibility between our learning problem and the problem of cutting down a set of generators for a p-group $\mathcal{G}$ to an independent set of generators for $\mathcal{G}$.

Formally we define the problem of finding independent generators as follows:

Given a prime p, given a generator system A for an unknown p-group $\mathcal{G}$, find a subset of A that is an independent generator system for $\mathcal{G}$.

Theorem 6 *Let $\mathcal{C}$ be the class of all finite Abelian groups. Let P be the set of all presentations of finite Abelian groups. The problem of learning $\mathcal{C}$ efficiently with respect to P by representation is polynomially reducible to the problem of efficiently finding independent generators.*

Proof. (Sketch.) The idea is—given a generator system A—to construct generators for p-groups (with different primes p) such that the target group can be written as a direct product of these p-groups. In order to apply Theorem 5 we need to make sure that the generators for the p-groups are independent from each other, using an efficient algorithm for finding independent generators. □

To close the section on representation-learning we discuss a result not related to Abelian groups but especially motivated by our original scenario of an agent exploring an unknown environment. Assume the agent has successfully learned a group environment but after that a new action is introduced to the environment. We model this as a problem of finding a representation of a single extension of a group $\mathcal{G}$ (for instance in the form of a semi-direct product) if a representation of $\mathcal{G}$ is known.

Suppose we have a group $\mathcal{G}$ with a generator system A and a d-dimensional linear representation Φ. Our aim is to extend $\mathcal{G}$ by a new generator element a, *i.e.,* we want to construct a representation for a single extension $\mathcal{G}' = \mathcal{G} \rtimes_\phi \langle a\rangle$. Let us assume in addition that $\sigma(a) = \sigma(\phi(a)) = \sigma$. The new representation Φ' can then be defined as follows, where we abbreviate $\phi(a)$ by ϕ:

$$\Phi'(g) = \begin{pmatrix} \Phi(g) & 0 & \cdots & 0 \\ 0 & \Phi(\phi(g)) & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & \Phi\left(\phi^{\sigma-1}(g)\right) \end{pmatrix} \text{ if } g \in S_\mathcal{G},\ \Phi'(a) = \begin{pmatrix} 0 & I_d & 0 & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & I_d \\ I_d & 0 & \cdots & 0 \end{pmatrix}$$

That this indeed yields the desired representation is implied by the following easy to verify properties.

1. $\sigma\left(\Phi'\left(a\right)\right) = \sigma$.
2. The group resulting from restricting Φ' to $\mathcal{G}$ is isomorphic to $\mathcal{G}$.
3. $\Phi'^{-1}\left(a\right)\Phi'\left(g\right)\Phi'\left(a\right) = \Phi'\left(\phi\left(g\right)\right)$ for all $g \in S_{\mathcal{G}}$.
4. $\Phi'\left(a\right)$ is not contained in the image of $\mathcal{G}$ under Φ'.

The dimension of the constructed representation is $d\sigma$. This construction is in general not trivial because of the problem of calculating $\Phi(\phi(g))$. However, a special case of this construction can be used for proving the following result.

Theorem 7 *Let* $P = \{\langle a_1, a_2 \mid R\rangle \mid \langle a_1, a_2 \mid R\rangle = \langle a_1\rangle \rtimes_\phi \langle a_2\rangle, \sigma(\phi(a_2)) = \sigma(a_2), \sigma(a_1) < \infty, \sigma(a_2) < \infty\}$. *Let* $\mathcal{C} = \{\mathcal{G} \mid \mathcal{G} = \langle A \mid R\rangle$ *for some* $\langle A \mid R\rangle \in P\}$. $\mathcal{C}$ *is efficiently representation-learnable with respect to* P.

Proof. (Sketch.) A learning algorithm L on input $\{a_1, a_2\}$ works as follows:

First, L determines $\sigma(a_1)$ and $\sigma(a_2)$ in the usual way. Second, L experiments with

$$a_1 a_2 a_1^{-1} \underbrace{a_2 a_2 \ldots a_2}_{\sigma(a_2)} a_1 a_2^{-1} a_1^{-1};$$

similarly with a_1 and a_2 swapped. (For example, if $a_1 a_2 a_1^{-1} a_2^6 \equiv \lambda$, then $\mathcal{G} = \langle a_2\rangle \rtimes_\phi \langle a_1\rangle$ with $\phi\left(a_2\right) = a_2^{-6}$.) Third, L constructs the linear representation as described above Theorem 7, knowing that $\Phi\left(a_2\right) = \left(\sqrt[\sigma(a_2)]{1}\right) \in \mathbb{C}^{1\times 1}$ is a primitive complex unit root. □

6 Conclusions

We introduced and analyzed a model for (efficient) learning of group-structured environments by exploration. In order to capture the idea that an agent should learn its environment without visiting all the states, we imposed a bound on the number of actions the agent can take up to convergence to a correct conjecture about the target group environment.

Learnability results strongly depend on the underlying presentation and thus the set of generators given as input to the learner, both if efficiency constraints are maintained and if they are dropped. Our negative results suggest that it is in general too strong a requirement to learn with respect to all possible presentations of a group—which is in fact not surprising and gives answers to some of the questions we posed in the introduction.

A direction for future work clearly is to characterize cases in which the size of a minimal representation of a group (in a general coding scheme for finite or finitely generated groups) is logarithmic instead of linear in the size of the group. This is for instance the essence of some of the contrasting results concerning dihedral groups. Relaxations of the learning model would be a further natural and quite promising extension of our approach.

Acknowledgements

We thank Barnabás Póczos, András Antos and Marcus Hutter for helpful discussions. Thanks are also due to the anonymous referees for their insightful comments.

This research was funded in part by the National Science and Engineering Research Council (NSERC), iCore and the Alberta Ingenuity Fund.

References

1. L. Babai and K. Fried. Approximate representation theory of finite groups. In *Proc. 32nd Annual Symposium on Foundations of Computer Science*, pages 733–742, 1991.
2. L. Babai and E. Szemerédi. On the complexity of matrix group problems I. In *IEEE Symposium on Foundations of Computer Science*, 1984.
3. W.W. Boone. The Word Problem. *The Annals of Mathematics*, 70:207–265, 1959.
4. J.C. Culberson and J. Schaeffer. Pattern databases. *Computational Intelligence*, 14:318–334, 1998.
5. K. Friedl, G. Ivanyos, and M. Santha. Efficient testing of groups. In *Proc. 37th Annual ACM Symposium on Theory of Computing*, pages 157–166, 2005.
6. E.M. Gold. Language identification in the limit. *Inform. Control*, 10:447–474, 1967.
7. R. Holte, J. Grajkowski, and B. Tanner. Hierarchical heuristic search revisited. In *Symposium on Abstraction, Reformulation and Approximation*, 2005.
8. H. Jaeger. Observable operator models for discrete stochastic time series. *Neural Computation*, 12:1371–1398, 2000.
9. R.E. Korf. Finding optimal solutions to Rubik's cube using pattern databases. In *AAAI/IAAI*, pages 700–705, 1997.
10. R.E. Korf. Analyzing the performance of pattern database heuristics. In *Proc. 22nd AAAI Conference on Artificial Intelligence*, pages 1164–1170, 2007.
11. M.L. Littman, R. Sutton, and S. Singh. Predictive representations of state. In *Advances in Neural Information Processing Systems 14*, pages 1555–1561, 2002.
12. P.S. Novikov. On the algorithmic undecidability of the word problem in group theory. In *Proc. Steklov Institute of Mathematics*, volume 44, pages 1–143, 1955. In Russian.
13. R.L. Rivest and R.E. Schapire. Diversity-based inference of finite automata. *J. ACM*, pages 555–589, 1994.
14. J.J. Rothman. *An Introduction to the Theory of Groups*. Springer, 1995.
15. F. Stephan and Y. Ventsov. Learning algebraic structures from text. *Theoret. Comput. Sci.*, 268(2):221–273, 2001.
16. A.L. Strehl, C. Diuk, and M.L. Littman. Efficient structure learning in factored-state MDPs. In *Proc. 22nd AAAI Conference on Artificial Intelligence*, pages 645–650, 2007.
17. N.V. Vinodchandran. *Counting Complexity and Computational Group Theory*. PhD thesis, Institute of Mathematical Sciences, Chennai, India, 1999.