

Models of active learning in group-structured state spaces^{*}

Gábor Bartók, Csaba Szepesvári¹, Sandra Zilles^{*}

*Department of Computing Science, University of Alberta
Edmonton, Alberta, Canada*

Abstract

We investigate the problem of learning the transition dynamics of deterministic, discrete-state environments. We assume that an agent exploring such an environment is able to perform actions (from a finite set of actions) in the environment and to sense the state changes. The question investigated is whether the agent can learn the dynamics without visiting all states. Such a goal is unrealistic in general, hence we assume that the environment has structural properties an agent might exploit. In particular, we assume that the set of all action sequences forms an algebraic group.

We introduce a learning model in different variants and study under which circumstances the corresponding “group structured environments” can be learned efficiently by experimenting with group generators (actions). It turns out that for some classes of such environments the choice of actions given to the agent determines if efficient learning is possible. 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.

Key words: learning theory, active learning, groups, algebraic structures

^{*} This paper is an extended version of [3].

^{*} Corresponding author.

Email addresses: bartok@cs.ualberta.ca (Gábor Bartók), szepesva@cs.ualberta.ca (Csaba Szepesvári), zilles@cs.ualberta.ca (Sandra Zilles).

¹ Csaba Szepesvári is on leave from MTA SZTAKI, Budapest, Hungary.

1 Introduction

Consider an agent that takes actions in its environment and receives sensations about the state of the environment in response. The goal of the agent is to learn to predict the effect of the actions on the environment. In particular, we are interested in scenarios in which the agent is able to learn this without visiting all states. This is an important problem since most environments have a very large (if not infinite) number of states and thus it would be unrealistic to assume that the agent can visit all of them.

It is known that when the sensations (*i.e.*, the representations of states) are vectors, predictive models can be learned efficiently even for stochastic environments provided that the environments' dynamics can be represented as a Bayes net in the form of a bounded depth decision tree, see Strehl, Diuk and Littman [15]. However, it is not always evident how to construct such a vector-based factored state representation given the “raw” sensory inputs. Factored representations often involve high-level concepts, such as objects, while the sensory inputs are low-level, such as pixels of an image.

Hence, a natural question to ask is how to deal with the case in which such a high-level representation is not given. For example, the agent might receive the state information in a completely unstructured form, such as in terms of names (IDs) of states that allow the agent only to distinguish two different states from each other, but not to distinguish any of their features from each other. Do there exist learning algorithms that can learn the dynamics of the environment efficiently, *i.e.*, by exploring only a small fraction of the state space? If efficient learning despite such a flat representation is possible then one does not need to worry about providing the learning agents with the “right” state representation, while in the opposite case one must be careful not to choose a representation that cannot be readily constructed given the “raw” sensations of the agent.

At first sight, efficient learning given an arbitrary state representation seems impossible. In fact, this is impossible in the conventional model when an agent is considered to have learned its environment if it is able to predict its future sensations given any (hypothetical) action sequence, cf. Rivest and Schapire [12] and Littman, Sutton and Singh [10]. Indeed, if the sensations of the agent are state names then the agent has no way of telling what state number will be sensed next when taking an action in a state that has not been visited beforehand.

Luckily, the agent being successful at predicting the “names” of states is just one way to characterize if an agent has learnt the dynamics of the environment. Another success criterion is that the agent is able *to tell, for any two*

action sequences, whether they lead to the same state given that they have been applied in the same, arbitrary initial state. If this is the case then the agent can relate the effect of any action sequence to that of other action sequences and thus can create a conceptual map of its environment. Indeed, disregarding computational issues, an agent successful according to this criterion can perform planning tasks such as navigating on the shortest path to a previously visited state or even to an unvisited state if this state is “named” by specifying a sequence of actions that lead to it.

In this paper we will use this new success criterion and assume that the sensations are unique identifiers assigned to the states. This assumption is equivalent to restricting the information an agent gets about to states in a way that no other relation between them than equality can be decided. Since efficient learning in the general case is still impossible, even with the new success criterion, we will make some additional strong assumptions about the environments.

In particular, we will assume that the possible environments obey a group structure. Such environments are characterized by two properties: *(i)* actions are reversible and *(ii)* if two action sequences lead to the same state from some initial state then they always do so, no matter what the initial state is (*i.e.*, we can think of these action sequences as being equivalent). This leads to a group-based description of the environments, where the elements of the group are the equivalence classes of the action sequences. Note that Assumption *(ii)* allows the agent to reason about the effect of actions in states that it has never visited and thus to learn the environment without visiting all the states.

Although limiting our attention to group-structured environments seems like a strong restriction, we will see that learning is still non-trivial. Moreover, in our opinion the group structure is a good starting point because some important environments can be well approximated by group-structured environments. For example, if some physical constraints are disregarded, multi-link robot manipulators can be thought of as working in a state space obeying a multi-dimensional toroid structure, a specific group structure.

Other examples are permutation games, such as Rubik’s cube (Korf [9]), the Topspin Puzzle or the Pancake Puzzle (Holte, Grajkowski and Tanner [7]), to name just a few.

Working in this framework we introduce notions of efficient learning, finite learning and learning in the limit. For efficient learning we require the agent to identify the environment by experimenting with a polynomial number of actions as a function of the logarithm of the size of the environment and the orders of the given “basic” actions.² We then prove the following results: If

² The order of an action is the minimal number of times it has to be applied in any

the class of environments is not restricted any further, then even learning in the limit is impossible (Theorem 17). Excluding infinite groups, finite learning becomes possible, though efficient learning is still impossible (Theorem 18). We show an example when efficient learning is possible in some class of environments, while for essentially the same class, just generated by a different set of basic actions, efficient learning is not possible (Corollary 19, Proposition 20). Here the change between the two classes of environments lies in the “complexity” (order) of the basic actions given to the agent.

Furthermore, we consider environments where the actions commute (such as in the case of an idealized multi-link robot in 2D when the actions actuate different joints) and the actions are the generators of the cyclic groups in the primary decomposition of the underlying group. We show that finite learning is impossible unless the environments are restricted to be finite in which case efficient learning becomes possible (cf. Theorems 22 and 23).

We also consider the problem of incremental learning of a semi-direct product construction (Section 5)—a way of modeling the case that an agent already knows its group environment, but then a new action is introduced, the effect of which needs to be learned by the agent. Finally, in Section 6 we show positive results for 3 specific one-player game environments, two of which are known as the *Hungarian rings* puzzle and the *Topspin* puzzle.

The closest to our results is the work of Rivest and Schapire [12], who investigate the problem of learning the dynamics of an environment represented by a finite automaton. (For extensions of their results to stochastic environments see Jaeger [8] and Littman *et al.* [10].) Rivest and Schapire assume that the agent’s sensations have the form of a binary vector. They assume that the environments can be represented by a permutation automaton (*i.e.*, all actions are reversible and the number of states is finite). According to their definition 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 learns every strongly connected, reduced permutation automaton (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 consists of an action-sequence, a , and an index, i , of a bit in the sensation vector. The outcome of test (a, i) in a given state s equals the value of the i th bit of the sensation vector observed after executing the sequence a from s . Two tests are equivalent if they have the same outcomes independent of the start state. An automaton is called reduced if any two states can be distinguished by some tests and

state s in order to get back to the state s . In our learning model, since the agent needs to learn at least the effect of the basic actions, the order of actions can be considered as a measure of “complexity” of actions.

it is strongly connected if any two states are accessible from each other by executing some sequence of actions.

The diversity d of a reduced environment is (tightly) bounded between $\log_2(n)$ and 2^n , where n is the number of states of the environment, see Rivest and Schapire [12]. The diversity depends to a large extent on how the sensations are chosen.

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 (strictly speaking, in order to define the diversity with state-valued sensations one needs to extend the definition of diversity to the case when the components of the sensation vector can take values in arbitrary sets). Hence, our targets can be viewed as being largely complementary: While Rivest and Schapire target learning efficiently when the representation is well-chosen in the sense that the diversity of the environment is small, our goal is to analyze when learning in logarithmic time is possible without this assumption.

The reader should note that our model of efficient learning of environments requires the agent to be *active*, *i.e.*, to carefully select which action to take in every step, depending on the outcome of previous actions. In case the actions were chosen arbitrarily, there would in general be no chance to learn the environment with a “small” number of experiments. There is a broad literature on active machine learning in general; here we would like to direct the reader to the specific literature on active learning of finite state machines. *Equivalence queries* and *membership queries* were used by Angluin [1] in her L^* algorithm for learning finite state automata. Equivalence queries allow a learner to propose an automaton A to an oracle, which in turn provides a counterexample for the target automaton A_T , *i.e.*, a word either accepted by A and not by A_T or accepted by A_T and not by A . Such queries have nothing in common with the experiments the agents in our model can perform. Membership queries though are similar to experiments in our model. In Angluin’s model the learner can select a word and receives information about whether or not the word is accepted by the target automaton. This is similar to performing a sequence of actions in our model and experiencing whether or not this action sequence brings the agent back to the state it started in.

Note that in Angluin’s model the learner can “reset” the automaton, *i.e.*, every membership query concerns a word processed from the start state, whereas in our model the agent is moving in the state space while taking actions and thus a “reset” is not allowed.

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 here, see, *e.g.*, models of learning alge-

braic structures from positive data as studied by Stephan and Ventsov [14]. Vinodchandran [16] and Babai and Szemerédi [2] analyze *black-box groups*—objects different from those in our group environment setting.

The rest of the paper is organized as follows: In Section 2 we present the basic notations and terms needed in the paper. In Section 3 our new model of learning is introduced. In Sections 4 and 5 we present negative and positive results in this model. In Section 6 we illustrate our results with simple one-player games.

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 Rothman’s textbook [13].

\mathbb{N} denotes the set of all natural numbers (including 0), \mathbb{Z} the set of all integers, and \mathbb{C} the set of all complex numbers.

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. We also identify $S_{\mathcal{G}}$ with \mathcal{G} in case it is unambiguous. If $A \subseteq S_{\mathcal{G}}$ is any subset of $S_{\mathcal{G}}$ then by A^* we denote the set $\{(a_1, \dots, a_m) : m \in \mathbb{N}\}$ of all finite sequences of elements in A , including the empty sequence. The elements of A^* are also referred to as *words*. With every sequence $w = (a_1, \dots, a_m)$ we associate a group element $\mathcal{G}(w) \in S_{\mathcal{G}}$, namely

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

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 $a \circ_{\mathcal{G}} b$.

For a set $A \subseteq S_{\mathcal{G}}$, let $\langle A \rangle_{\mathcal{G}} = \{\mathcal{G}(w) : w \in (A \cup \{a^{-1} : a \in A\})^*\}$ be the subset of $S_{\mathcal{G}}$ whose elements are obtained as a product of elements of A and their inverses.

(When it is clear over which group \mathcal{G} we are taking the products we will drop \mathcal{G} from $\langle A \rangle_{\mathcal{G}}$.) A set $A \subseteq S_{\mathcal{G}}$ is called a *generator system* of \mathcal{G} if $\langle A \rangle = S_{\mathcal{G}}$. Since it will be always clear from the context, which generator system an element $a \in S_{\mathcal{G}}$ is considered to belong to, we simply call the elements of a generator system *generators* and the system itself we simply call a *set of generators*.

For any subset \mathcal{H} we write $\mathcal{H} \leq \mathcal{G}$ if \mathcal{H} is a subgroup of \mathcal{G} , and $\mathcal{H} \triangleleft \mathcal{G}$ 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 $S_{\mathcal{H}_1} S_{\mathcal{H}_2} = \{h_1 \circ_{\mathcal{G}} h_2 : h_1 \in S_{\mathcal{H}_1} \text{ and } h_2 \in S_{\mathcal{H}_2}\}$.

A *relation* in \mathcal{G} is a sequence $w \in S_{\mathcal{G}}^*$ such that $\mathcal{G}(w) = \lambda_{\mathcal{G}}$. A pair $\langle A \mid R \rangle$ is called a *presentation* of \mathcal{G} , written $\langle A \mid R \rangle \cong \mathcal{G}$, 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 F_A on A and F_A 's 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 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 finitely presented group is always finitely generated, but the converse does not hold, see Rothman [13].

A *representation* of \mathcal{G} over a vector space V is a homomorphism $\Phi : S_{\mathcal{G}} \mapsto \text{GL}(V)$, where $\text{GL}(V)$ is the *general linear group* of V (i.e., the automorphism group of V). A representation is *faithful* if it is injective.

For any $a \in S_{\mathcal{G}}$, $\langle a \rangle$ denotes the *cyclic subgroup* generated by a . The symbol C_k denotes the *cyclic group* of order k . A finite *p-group* is a finite group of order p^k , where p is 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 $\sigma(g)$.

The group of all permutations of a finite set A , where $|A| = n$, is denoted by S_n .

The automorphism group of \mathcal{G} is referred to by $\text{Aut}(\mathcal{G})$.

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 $\mathcal{N} \triangleleft \mathcal{G}, \mathcal{H} \leq \mathcal{G}$ such that $S_{\mathcal{N}} S_{\mathcal{H}} = S_{\mathcal{G}}, S_{\mathcal{N}} \cap S_{\mathcal{H}} = \{\lambda_{\mathcal{G}}\}$. Let $\phi : S_{\mathcal{H}} \mapsto \text{Aut}(\mathcal{N})$ be a group homomorphism such that*

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

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

3 A model of 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 changes the state of the environment. Now assume that the agent can always observe the name of the state the environment is currently in (*i.e.*, the states could be numbered and the agent observes these numbers). Thus, the agent will be able to recognize when it gets back to some previously visited state, but since the names of the states do not have any further structure this is all the information that the agent can gain by observing these state names. However, this information already allows the agent to find out relations between actions. In the model defined in this paper, the goal of the agent is to be able to predict the effects of action sequences; we will specify this later.

3.1 Group environments

We assume the environment to be static and deterministic, *i.e.*, there is a function that determines for every state s and every action a the successor state after taking action a in state s . Formally, an environment is 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 $T(s, a)$.*

We call $\mathcal{E}' = (S', A, T')$ isomorphic to $\mathcal{E} = (S, A, T)$ (and vice versa) if the states in S can be mapped to states in S' via a bijective mapping $\phi : S \rightarrow S'$ such that $\phi(T(s, a)) = T'(\phi(s), a)$ for any state $s \in S$ and action $a \in A$. A reasonable success criterion for our learning model is that the agent should come up with an environment \mathcal{E}' that is isomorphic to \mathcal{E} .

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) monoid³ (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

³ A monoid is an algebraic structure corresponding to a semi-group with an identity element.

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 $s a_1 \dots a_m$ as a shorthand for $T(s, a_1 \dots a_m)$ when T is clear from the context.

Definition 3 (Equivalence of action sequences) *Let $\mathcal{E} = (S, A, T)$ be an environment and $w_1, w_2 \in A^*$ be action sequences. Then w_1 and w_2 are equivalent in \mathcal{E} (denoted by $w_1 \equiv_{\mathcal{E}} w_2$) iff $s w_1 = s w_2$ 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. Note that $(S_{\mathcal{E}}, \circ)$ is a monoid, where the identity element is the equivalence class of λ , the empty string.

We focus on settings 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 the following two properties are satisfied:*

Reversibility of actions: *For any $s \in S$, $a \in A$ there exists $w \in A^*$ such that $s a w = s$;*

Uniform effects: *If for some action sequences $w, w' \in A^*$ and some state $s \in S$, $s w = s w'$ then $w \equiv w'$.*

The first property says that actions are reversible at least when action sequences are considered, while according to the second property in a group environment one may conclude the equivalence of two action sequences by checking if they give rise to the same outcome when executed from an arbitrarily chosen initial state. In particular, the uniform effects assumption means that in a group environment “all states look the same” from the point of view of looking at the effects of action sequences: For any w_1, w_2, s_1, s_2 , if $s_1 w_1 = s_1 w_2$ then also $s_2 w_1 = s_2 w_2$. Thus, s_1 and s_2 are indistinguishable by looking at whether two (or more) action sequences give rise to the same state.

The rationale of calling an environment with the above two properties a group environment is given by the following proposition.

Proposition 5 *If \mathcal{E} is a group environment then $(S_{\mathcal{E}}, \circ)$ is a group.*

The statement follows easily from the definitions and thus its proof is omitted. Note that the reverse of this statement does not hold.

Proposition 6 *There is an environment with*

- (1) $(S_{\mathcal{E}}, \circ)$ is a group.
- (2) \mathcal{E} is not a group environment.

Proof Let $\mathcal{E} = (S, A, T)$, where $S = \{1, 2, 3, 4, 5, 6\}$, $A = \{a, b\}$, and where T is given by $1a = 2, 2a = 3, 3a = 1, 4a = 5, 5a = 6, 6a = 4$ and $1b = 4, 4b = 1, 2b = 3, 3b = 2, 6b = 5, 5b = 6$. Then $2a = 3 = 2b$, but $1a = 2 \neq 4 = 1b$, hence this environment is not a group environment, although $(S_{\mathcal{E}}, \circ)$ is a group. See Figure 1 for illustration. \square

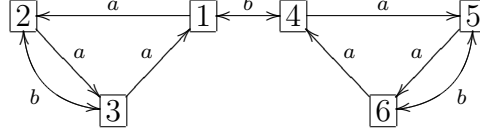


Fig. 1. Graph representation of an environment \mathcal{E} that is not a group environment despite $(S_{\mathcal{E}}, \circ)$ being a group.

However, in what follows we will show that if $(S_{\mathcal{E}}, \circ)$ is a group, then there is a group environment \mathcal{E}' such that the group $(S_{\mathcal{E}'}, \circ)$ is isomorphic to $(S_{\mathcal{E}}, \circ)$.

This is essentially a consequence of the following proposition, which states that there is a normal form for group environments over a given set A of generators.

Proposition 7 *Let $\mathcal{E} = (S, A, T)$ be a group environment. For an action sequence $w \in A^*$ let \bar{w} denote the equivalence class of w in \mathcal{E} . Let $S' = \{\bar{w} : w \in A^*\}$ and let $T' : S' \times A \rightarrow S'$ be defined by $T'(\bar{w}, a) = \overline{wa}$. Then $\mathcal{E}' = (S', A, T')$ is isomorphic to \mathcal{E} .*

Proof For simplicity assume that there exists a state s such that every state of S can be reached from s by following at least one action sequence. (Note that in this case every state is reachable from every other state. If this was not the case then the construction below could be repeated for all sets in the partition of S induced by the reachability relation of states.)

Define $\phi : S \rightarrow S'$ by $\phi(sw) = \bar{w}$. If $sw_1 = sw_2$ then $\bar{w}_1 = \bar{w}_2$ by the uniform effects property of group environments, hence ϕ is well-defined. Notice that ϕ is surjective by its construction. We claim that it is also injective. Indeed, pick $s_1, s_2 \in S$ such that $\phi(s_1) = \phi(s_2)$. Assume that w_1, w_2 are action sequences such that $s_1 = sw_1$ and $s_2 = sw_2$. Then $\bar{w}_1 = \bar{w}_2$, i.e., w_1 and w_2 are equivalent. The definition of equivalence of action sequences implies $sw_1 = sw_2$, i.e., $s_1 = s_2$.

In order to show that ϕ is a homomorphism, let $s_1 \in S$, $a \in A$. Suppose $s_1 = sw_1$. Then $\phi(T(s', a)) = \phi(sw_1a) = \overline{w_1a} = T'(\phi(s_1), a)$.

Hence, ϕ is an isomorphism between \mathcal{E} and \mathcal{E}' . \square

The desired property, namely that group environments and groups of action sequences (as in Definition 3) are basically interchangeable, is an immediate corollary of Proposition 7. This property will be exploited when defining

concepts of learnability and allows us to talk about the group environment associated with a finitely generated group.

Note that, if $\mathcal{E} = (S, A, T)$ is a group environment, then A is a set of generators for the group $(S_{\mathcal{E}}, \circ)$. In particular, we consider finitely generated groups only.

Corollary 8 *Up to isomorphism, there is a one-to-one correspondence between group environments and finitely generated groups.*

Proof Let ϕ map group environments to finitely generated groups and ψ map finitely generated groups to environments as follows. For any group environment $\mathcal{E} = (S, A, T)$, let $\phi(\mathcal{E}) = ((S_{\mathcal{E}}, \circ), A)$. For any finitely generated group \mathcal{G} and any finite set of generators A with $\langle A \rangle = S_{\mathcal{G}}$, let $\psi(\mathcal{G}, A) = (S_{\mathcal{G}}, A, T_{\mathcal{G}})$, where $T_{\mathcal{G}}(s, a) = s \circ a$ for every $s \in S_{\mathcal{G}}$ and every $a \in A$. We claim that for any given group environment \mathcal{E} , \mathcal{E} is isomorphic to $\psi(\phi(\mathcal{E}))$. This follows from Proposition 7, since $\psi(\phi(\mathcal{E})) = (S', T', A)$ for the environment $\mathcal{E}' = (S', A, T')$ as defined in Proposition 7. \square

In particular, if, for a group environment \mathcal{E} , $\mathcal{G}_{\mathcal{E}}$ denotes the associated group over the equivalence class of the action sequences of \mathcal{E} , then for every two environments $\mathcal{E}_1, \mathcal{E}_2$,

$$\mathcal{G}_{\mathcal{E}_1} \cong \mathcal{G}_{\mathcal{E}_2} \text{ if and only if } \mathcal{E}_1 \cong \mathcal{E}_2 .$$

3.2 Agents

An agent is thought of as a computable device moving within a group-structured environment, starting in an arbitrary state. It is given A (the set of actions) and the name of the start state initially. At each time step the agent chooses an action, observes the name of the successor state, and returns a hypothesis “describing” an environment \mathcal{E}' . We consider such a hypothesis to be correct if \mathcal{E}' is isomorphic to the actual environment \mathcal{E} .

What does “describing” an environment mean? Since the environment has a group structure it suffices for the agent to learn the equivalence of action sequences (cf. Proposition 7). If the agent has a way to decide which action sequences are equivalent then it can “construct” an environment that is isomorphic to \mathcal{E} . Thus, the agent can (in theory) solve specific tasks in the isomorphic true environment, like navigating back to some previously visited state with the smallest possible number of actions or navigating to the first unvisited state.

In order to know which action sequences are equivalent the agent must be able to solve the *word problem* for $(S_{\mathcal{E}}, \circ)$, cf. Rothman [13].

Definition 9 (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 the problem, given any $w \in A^*$, to decide whether or not $w \equiv \lambda_{\mathcal{G}}$. The word problem for \mathcal{G} over A is solvable, if there is a recursive decision procedure $d : A^* \rightarrow \{0, 1\}$ such that for all $w \in A^*$,

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

The actual definition of the word problem over groups concerns not only $w \in A^*$ but $w \in (A \cup \{a^{-1} : a \in A\})^*$. However, as long as one is only concerned with the question of whether or not the word problem is solvable, both definitions are equivalent.

In fact, being able to decide the word problem is equivalent to being able to decide the equivalence of action sequences, the latter being our goal in learning.

Proposition 10 Let $\mathcal{E} = (S, A, T)$ be a group environment. For any $w, w' \in A^*$, let d and d' be defined as follows.

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

Then d is a recursive function if and only if d' is a recursive function.

Proof If d' is recursive, then d is recursive via $d(w) = d'(w, \lambda_{\mathcal{E}})$ for all $w \in A^*$.

If d is recursive then $d'(w, w')$, for two action sequences w, w' , can be computed as follows. Let $(w_i)_{i \in \mathbb{N}}$ be a recursive enumeration of A^* . Then $d'(w, w') = d(w w_{i^*})$, where i^* is the minimal i such that $d(w' w_i) = 1$. (Note that such an index i^* of an action sequence inverse to w' must exist and can thus be found effectively.) \square

However, if the word problem is solvable efficiently then this does not necessarily mean that deciding the equivalence of action sequences can be done efficiently (since finding the index i^* in the proof of Proposition 10 can be “slow”).

In what follows we will assume that the agent is either given low-complexity inverses of all the actions in A or that the order of all actions in A are finite, so that the inverses can be found in time $\sum_{a \in A} \sigma(a)$. Clearly, if the agent knows the inverses of all actions in A then being able to solve the word problem *efficiently* is equivalent to being able to decide *efficiently* for any two action sequences if they are equivalent.

In what follows we disregard the issue that the agent needs to know the inverses

of actions in A to solve for the equivalence of arbitrary action sequences given the solution of the word problem.

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

Definition 11 (Learning algorithm) *A learning algorithm is an algorithm L that fulfills the following properties.*

- (1) L takes as its 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 $a_n \in A$ to the environment \mathcal{E} and receives $s_{n+1} = s_n a_n$ as a reply. Then L returns a recursive decision procedure $D_n : A^* \rightarrow \{0, 1\}$ and goes to Step $n + 1$.⁴
 - (ii) L stops and never goes to Step $n + 1$.

Since by Corollary 8, finitely generated groups and group environments, up to isomorphism, uniquely determine each other, when defining learning models and making formal statements about learning group environments, we can specify the learning problem in terms of the underlying groups and their generators. This has the advantage that we can talk about learning familiar classes of groups – as long as we specify the generators. We thus define our learning criterion as follows.

Definition 12 (Learning group environments) *Let L be a learning algorithm, \mathcal{G} a finitely generated group, and A a finite set of generators of \mathcal{G} . Let $\mathcal{E} = (S, A, T)$ be the group environment corresponding to \mathcal{G} and A .*

L learns \mathcal{G} from A , if there is some $n_0 \in \mathbb{N}$ such that for all $s \in S$ and for all $n \geq n_0$ the following holds.

If L is given A and s as its input then L runs for at least $n_0 + 1$ steps and, if L has not stopped in at most n steps, the output of L in Step n is some decision procedure that solves the word problem for \mathcal{G} over A .

n_0 is then called the sample-complexity of L on (\mathcal{G}, A) .⁵

To consider learning a class of groups, each possibly from different sets of generators, we take a *group learning problem* to be a set \mathcal{C} of finitely generated groups specified as pairs (\mathcal{G}, A) , where $A \subset \mathcal{G}$ is a finite generator set for \mathcal{G} . For each such group learning problem, let

⁴ More precisely, L will return a program (in some suitable language) that represents the recursive decision procedure.

⁵ Since \mathcal{E} is a group environment, we can without loss of generality assume that there is *one* n_0 for all initial states s rather than making n_0 dependent on s .

- $\mathcal{G}_C = \{\mathcal{G} \mid \text{there is some } A \text{ with } (\mathcal{G}, A) \in C\}$.
- $A_C = \{A \mid \text{there is some } \mathcal{G} \text{ with } (\mathcal{G}, A) \in C\}$.

Definition 13 (Learning classes of group environments) *Let C be a group learning problem.*

The class \mathcal{G}_C of groups is learnable finitely with respect to the class A_C of generator sets, if there is a learning algorithm L such that, for every $(\mathcal{G}, A) \in C$ the following two conditions hold.

- (1) L learns \mathcal{G} from A .
- (2) There is some $n \in \mathbb{N}$ such that L learning \mathcal{G} from A stops after n steps.

The class \mathcal{G}_C of groups is learnable efficiently with respect to the class A_C of generator sets, if there is a learning algorithm L and a polynomial q such that, for every $(\mathcal{G}, A) \in C$ the following two conditions hold.

- (1) L learns \mathcal{G} finitely from A .
- (2) The sample-complexity n_0 of L on (\mathcal{G}, A) fulfills $n_0 \leq q(\log |S_{\mathcal{G}}| + \sum_{a \in A} \sigma(a))$.

In the definition of efficient learning, $\sum_{a \in A} \sigma(a)$ 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 (thus, any other subpolynomial function could also be used in the definition).

Obviously, in the case of finite learning, it does not make a difference whether or not a learning algorithm outputs decision procedures in every step. It suffices if a correct decision procedure is returned in the final step before the algorithm stops. We will use this implicitly in our proofs below.

It does not make sense to consider efficient learning of infinite groups in terms of the definition proposed here, because then the bound on the number of steps is infinite (and would, in case some generators in A have infinite order, even remain so when removing $\log |S_{\mathcal{G}}|$ from the bound). The model of finite learning can still be studied for infinite groups. However, since even that model will turn out to be too restrictive in some tough cases, we introduce a third learning model, based on Gold's [6] model of learning languages.

Definition 14 (Learning in the limit) *Let C be a group learning problem.*

The class \mathcal{G}_C of groups is learnable in the limit with respect to the class A_C of generator sets, if there is a learning algorithm L such that L learns \mathcal{G} with respect to A , for every $(\mathcal{G}, A) \in C$.

Informally speaking, learning in the limit differs from finite learning in that the learning algorithm is not required to stop. It differs from efficient learning in that the learning algorithm is not given any sample-complexity bounds. Hence learning in the limit is a generalization of both our previous models: clearly, if \mathcal{C} is learnable finitely or efficiently then it is also learnable in the limit.

Note that none of our definitions of learning is concerned with computational complexity issues in terms of run-time; only sample-complexity is addressed.

We simplify our terminology for the case when a class of groups is learnable with respect to the class of all possible generator sets.

Notation 15 *Let \mathcal{C} be a group learning problem such that*

$$\forall \mathcal{G} \in \mathcal{G}_{\mathcal{C}} \forall A [A \text{ finite and } \langle A \rangle = S_{\mathcal{G}} \Rightarrow (\mathcal{G}, A) \in \mathcal{C}].$$

If $\mathcal{G}_{\mathcal{C}}$ is learnable finitely (efficiently, in the limit) with respect to $A_{\mathcal{C}}$ then we say that $\mathcal{G}_{\mathcal{C}}$ is learnable finitely (efficiently, in the limit) with respect to arbitrary generators.

We close this section with our 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 11, the only difference is that upon a query a sent to the oracle in Step n and state s_n , instead of $s_n a$ it receives the reply 1 if $s_n a = s_0$ and 0 if $s_n a \neq s_0$. The notion of 0/1-learnability (in the limit, finite, or efficient) can then immediately be derived from Definition 13 by replacing “learning algorithm” by “0/1-learning algorithm” in Definition 13. We call this model *learning with binary observations*.

This restriction of the observations the learner makes does not restrict its learning capabilities when learning with respect to *generators of finite order*, not even when efficiency issues are considered.

Lemma 16 *Let \mathcal{C} be a group learning problem such that $\sigma(a)$ is finite for all $A \in A_{\mathcal{C}}$ and all $a \in A$. $\mathcal{G}_{\mathcal{C}}$ is learnable in the limit (finitely learnable, efficiently learnable) with respect to $A_{\mathcal{C}}$ iff $\mathcal{G}_{\mathcal{C}}$ is 0/1-learnable in the limit (finitely 0/1-learnable, efficiently 0/1-learnable) with respect to $A_{\mathcal{C}}$.*

Proof We only show that a 0/1-learning algorithm can be constructed from a learning algorithm according to Definition 11; the other direction is trivial.

The idea is that all the information gathered by the original learner up to some stage is whether a subsequence of the action sequence posed leaves a state (and,

thus, all states) 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 polynomially only.

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.⁶

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 $w = (x_0, \dots, x_t)$ be the action sequence

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

Query the actions in this sequence one by one.

- Let M be the set of all subsequences x_j, \dots, x_{j+z} of w such that
 - $j = 0$ or the observation after querying x_{j-1} was 1; and
 - the observation after querying x_{j+z} was 1.
- If there exists an $i \leq n$ such that $(a_i, \dots, a_n) \in M$ then feed the state name returned in Step $i - 1$ to L for the minimal such i (or return s_0 if $i = 0$). If there is no such i then return a new state name and feed it to L .
- 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 the execution time. \square

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, when \mathcal{C} is the class of all finitely generated groups, one easily obtains a negative result, independent of how the generators

⁶ This straightforward method for obtaining the order of a group element is frequently used throughout the paper. From now on we will refer to it as “determining the order”.

are chosen. The class of finitely presented groups (and thus also the class of finitely generated groups) is not learnable in the limit, independent of the generators given to the learning algorithm. Formally, this can be stated as follows.

Theorem 17 *The following statements hold:*

- (1) *Let \mathcal{C} be a group learning problem for which $\mathcal{G}_{\mathcal{C}}$ equals the set of all finitely generated groups. Then $\mathcal{G}_{\mathcal{C}}$ is not learnable in the limit with respect to $A_{\mathcal{C}}$.*
- (2) *Let \mathcal{C} be a group learning problem for which $\mathcal{G}_{\mathcal{C}}$ equals the set of all finitely presented groups. Then $\mathcal{G}_{\mathcal{C}}$ is not learnable in the limit with respect to $A_{\mathcal{C}}$.*

Proof It is known (cf. Boone [4] and Novikov [11]) that there exists 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 for \mathcal{G} over A is unsolvable. Hence, for any finite generator set A for \mathcal{G} , no learning algorithm can output a decision procedure solving the word problem for \mathcal{G} over A . \square

Restricting our focus to classes of finite groups (for which the word problem is always solvable) we at least get finite learnability, yet efficient learning is still out of reach. Informally speaking, the class of all finite groups is learnable finitely independent of the set of generators given, yet not learnable efficiently without restrictions on the given generators.

Theorem 18 (1) *The class of all finite groups is not learnable efficiently with respect to arbitrary generators.*

- (2) *The class of all finite groups is learnable finitely with respect to arbitrary generators.*

Proof

ad 2. First, we need to introduce the concept of transformations of words with respect to a set of relations:

Given a set R of relations and a word w , a transformation of w with respect to R is a finite sequence of word manipulations of the following kinds:

- *Insert a relation $r \in R$ at some position $i \geq 0$; i.e., replace $w = w_1w_2$ by w_1rw_2 where $|w_1| = i$.*
- *Delete a relation $r \in R$ at some position $i \geq 0$; i.e., replace $w = w_1rw_2$ by w_1w_2 where $|w_1| = i$.*

The idea for a learning algorithm on input A is the following:

Step 1: Obtain a finite presentation $\langle A \mid R \rangle$ of the target group.

Step 2: Construct a decision procedure that solves the word problem for the target group \mathcal{G} over A .

Step 2 is trivial because the word problem for finite groups is *uniformly* solvable, *i.e.*, there is *one* recursive function that, given *any* finite presentation $\langle A \mid R \rangle$ of a finite group \mathcal{G} and any word $w \in A^*$, decides whether or not $w \equiv \lambda_{\mathcal{G}}$. It remains to give a procedure for step 1.

Our procedure uses a dovetailing technique. On input A the procedure interleaves two threads of computation, while maintaining a set R of currently known relations of the target group. Initially, $R = \emptyset$.

- Thread α experiments with all possible action sequences in an arbitrary systematic order to find sequences that are equivalent to λ . Whenever such a sequence is found, it is added to R .
- Thread β searches for a positive integer k such that every word of length k is equivalent to a shorter word. For every $k \geq 1$, thread β starts a new sub-thread β_k at time step k . The sub-thread β_k lists all words of length k and all their transformations with respect to the current set of relations R . It stops if it found a transformation for every listed word for which the resulting word is shorter than the original.

As soon as some thread β_k stops, the procedure suspends thread β and notifies thread α to stop after experimenting with all action sequences up to length $k - 1$.

Since the target group is finite, there is a k such that every word of length k is equivalent to a shorter word. On the other hand, the number of relations sufficient for describing the group is finite, hence thread α will find them in finite time. These facts imply that thread β will find a k with the desired property. The final set of relations will be all the words of length less than k which are equivalent to λ . It is easy to see that with this set R , $\langle A \mid R \rangle$ is a presentation of the target group, which concludes the proof. \square *ad 2.*

ad 1. Assume the class of all finite groups is efficiently learnable with respect to arbitrary generators. 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 set A of generators of \mathcal{G} , using no more than $q(\log(|S_{\mathcal{G}}|) + \sum_{a \in A} \sigma(a))$ many steps. By Lemma 16 we can assume without loss of generality that L is a 0/1-learning algorithm.

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

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

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

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 $w \in \{a, b\}^*$ with $|w| < 2q(m_*)$:

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

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 taking at most $q(\log(|S_{\mathcal{G}_2}|) + \sigma(a) + \sigma(b)) = q(\log(2q(m_*) + 2) + 4) < 2q(m_*)$ steps.⁷ This is a contradiction, so the class \mathcal{G}_C of all finite groups is not learnable efficiently with respect to the class A_C of all possible corresponding generator sets.

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

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

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

So assume $0 < |w'| < 2q(m_*)$. Now, w' is obtained from w 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 $w = (a_1, \dots, a_n) \in \{a, b\}^*$ let $w_{\text{odd}} = (a_1, a_3, \dots, a_{n_1})$ and $w_{\text{even}} = (a_2, a_4, \dots, a_{n_2})$, where $n_1 = n_2 + 1 = n$ for odd n and $n_1 + 1 = n_2 = n$ for even n . Let $\mu(w) = (\# \text{ of 'a's in } w_{\text{odd}}) - (\# \text{ of 'a's in } w_{\text{even}}) + (\# \text{ of 'b's in } w_{\text{even}}) - (\# \text{ of 'b's in } w_{\text{odd}})$. For example, if $w = aaababbabab$, then $w_{\text{odd}} = aaabbb$ and $w_{\text{even}} = abbaa$, so $\mu(w) = -1$.

When modifying w to w' , 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 (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 $w_1 \equiv_{\mathcal{G}_1} w_2$ then $\mu(w_1) \equiv \mu(w_2) \pmod{2q(m_*)}$.

⁷ Note here that $\sigma(a) = \sigma(b) = 2$ holds in both groups; technically we should in fact use subscripts with σ to relate it to the specific group where it is defined.

Since w' does not contain the substrings aa or bb , we have $\mu(w') = \pm|w'|$. Obviously, $\mu(\lambda) = 0$. Since $0 < |w'| < 2q(m_*)$, $\mu(w') \not\equiv \mu(\lambda) \pmod{2q(m_*)}$. Therefore $w' \notin_{\mathcal{G}_2} \lambda_{\mathcal{G}_2}$. \square *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. Clearly, it suffices to prove the statement only for \mathcal{G}_1 . Note that two action sequences w and β are equivalent in \mathcal{G}_1 , if β results from w 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, observe that $(ba)^k =_{\mathcal{G}_1} (ab)^{2q(m_*)-k}$ for $0 \leq k \leq 2q(m_*)$ (both forms obviously have the inverse $(ab)^k$). Hence also $a(ba)^k =_{\mathcal{G}_1} a(ab)^{2q(m_*)-k} =_{\mathcal{G}_1} (ba)^{2q(m_*)-k-1}b =_{\mathcal{G}_1} b(ab)^{2q(m_*)-k-1}$ for $0 \leq k < q(m_*)$.

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

This completes the proof. \square *ad 1.*

\square

The groups \mathcal{G}_1 and \mathcal{G}_2 used in the proof of the first assertion of Theorem 18 belong to the well-known class of finite dihedral groups (recall that a finite dihedral group is the group of symmetries of a regular polygon, including both reflections and rotations). Since the dihedral groups illustrate a few principled properties of our learning model, we study them in more detail.

4.2 Learning dihedral groups

For every $k \in \mathbb{N} \setminus \{0\}$, let D_k denote the finite dihedral group with $2k$ elements. Note that there is only one infinite dihedral group, namely $D_\infty \cong \langle a, b \mid a^2, b^2 \rangle$.

Theorem 18 and its proof immediately yield the following corollary, which states that the class of all finite dihedral groups cannot be learned efficiently if the generators given are not restricted in advance. The proof of Theorem 18.(1) actually shows this already for the case that the generators given are two reflections a and a' such that $D_k \cong \langle a, a' \mid aa, a'a', (aa')^k \rangle$.

- Corollary 19** (1) *The class of all finite dihedral groups is not learnable efficiently with respect to arbitrary generators.*
(2) *The class of finite dihedral groups is learnable finitely with respect to arbitrary generators.*

In fact, it is not very surprising that the finite dihedral groups cannot be learned efficiently with respect to unrestricted generators. The reason is that when two consecutive reflections are given as the input generators then those have order 2, while the size of the group can be arbitrarily high. In order to build a decision procedure that solves the word problem the size of the group must be known. This size can only be determined by taking as many reflection actions as there are elements in the group. Hence no algorithm can learn the group in time polynomial in the logarithm of the size of the group and the orders of the generators. This shows how much the choice of generator systems influences learnability: in fact the finite dihedral groups can be learned efficiently if the learner is always given a set of generators the order of one of which is proportional to the size of the group, thus allowing for enough experiments to identify the target group.

In particular, the class of finite dihedral groups is learnable efficiently if the given generators are a reflection a and a rotation b such that $D_k \cong \langle a, b \mid aa, b^k, abab \rangle$.

Proposition 20 *Let $\mathcal{C} = \{(D_k, \{a, b\}) : k \in \mathbb{N} \setminus \{0\}, D_k \cong \langle a, b \mid a^2, b^k, abab \rangle\}$. Then $\mathcal{G}_{\mathcal{C}}$ is learnable efficiently with respect to $A_{\mathcal{C}}$.*

Proof The learning algorithm L works as follows. Given the two generators a and b , L determines the order of a and b , which requires $\sigma(a) + \sigma(b)$ many steps. Knowing the order allows the learner to conclude which of the given generators is a rotation. This in turn allows the learner to construct a group \mathcal{G}' that is isomorphic to the unknown group \mathcal{G} . The knowledge of the orders allows L to find out the inverses of a and b over $\{a, b\}$. Knowing the inverses, one can construct an algorithm that decides, for any word $w \in \{a, b\}^*$, whether $w \equiv \lambda_{\mathcal{G}'}$ and thus whether $w \equiv \lambda_{\mathcal{G}}$. Thus, after taking $\sigma(c) + \sigma(d)$ many actions, L can stop and return this algorithm as its output. \square

Note that with $a' = ab$ we have the equivalence of the group presentations $\langle a, a' \mid aa, a'a', (aa')^k \rangle$ and $\langle a, b \mid aa, b^k, abab \rangle$. The corresponding environments differ only in the choice of the “primitive actions”. In fact, the primitive actions in one environment can be simulated by taking at most two actions of the other environment and vice versa. Yet, we see that one environment can be learned efficiently, while the other can not—at least in the current model of learning.⁸

⁸ This raises the question whether the definition of efficient learning should be changed. One idea is to allow the time of learning to depend on the length of the

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 with respect to arbitrary generator systems, but this is no longer true if we add just a single group to this class, namely the infinite dihedral group.⁹

Theorem 21 *Let $G = \{D_k : k \in (\mathbb{N} \setminus \{0\}) \cup \{+\infty\}\}$ denote the class of all dihedral groups.*

- (1) *G is not learnable finitely with respect to arbitrary generators.*
- (2) *G is learnable in the limit with respect to arbitrary generators.*

Proof ad 1. Assume G is learnable finitely with respect to arbitrary generators. Then there is a 0/1-learner L that learns every dihedral group finitely with respect to any set of two generators. Consider the following scenario of an oracle interacting with L on input $\{a, b\}$.

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

This scenario is valid if D_∞ is the target group. Thus eventually L will return a decision procedure solving the word problem for D_∞ over $\{a, b\}$ 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 G —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. □

words in a “short” presentation of the group. See Section 7 for a brief discussion of this issue.

⁹ 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. The difference here is though that learnability in the limit is not affected.

4.3 Learning Abelian groups

For finitely generated Abelian groups, a reasoning similar to the proof of Theorem 21 shows similar differences between learning in the limit and finite learning.

The following theorem states that the class of all finitely generated Abelian groups is learnable in the limit, but not learnable finitely, if the generators given generate cyclic subgroups of the target group. Note that every finitely generated Abelian group is isomorphic to the direct product of finitely many cyclic subgroups (see the Fundamental Theorem of Finitely Generated Abelian Groups).

Theorem 22 *Let $\mathcal{C} = \{(\mathcal{G}, A) : \mathcal{G} \cong \langle a_1 \rangle \times \dots \times \langle a_m \rangle, A = \{a_1, \dots, a_m\}\}$. Then the following holds:*

- (1) $\mathcal{G}_{\mathcal{C}}$ is learnable in the limit with respect to $A_{\mathcal{C}}$.
- (2) $\mathcal{G}_{\mathcal{C}}$ is not learnable finitely with respect to $A_{\mathcal{C}}$.

Proof ad 1. For a group \mathcal{G} , define \mathcal{G}^k recursively by $\mathcal{G}^1 = \mathcal{G}$ and $\mathcal{G}^k = \mathcal{G}^{k-1} \times \mathcal{G}$ ($k \geq 2$).

Every group \mathcal{G} in $\mathcal{G}_{\mathcal{C}}$ is isomorphic to $\mathbb{Z}^k \times C_{n_1}^{k_1} \times \dots \times C_{n_z}^{k_z}$ for some $k, z, k_i, n_i \in \mathbb{N}$, where the values $k, n_1, k_1, \dots, n_z, k_z$ are (up to rearranging indices) uniquely determined by \mathcal{C} . The number of generators here is $k + k_1 + \dots + k_z$.

A learning algorithm L witnessing Theorem 22 works as follows, given a set $A = \{a_1, \dots, a_m\}$ of m generators.

- (1) L initially always hypothesizes the target group to be \mathbb{Z}^m . This means that the decision procedure initially returned by L , given a word $w \in A^*$, decides as follows.
 - If w is empty then return 1.
 - If w is not empty then return 0.
- (2) Given a canonical enumeration of all pairs (a_l, t) for $t \geq 1$ and $l \in \{1, \dots, m\}$, L then memorizes the state names observed 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 hypothesis from $\mathbb{Z}^{k'} \times C_{n_1}^{k'_1} \times \dots \times C_{n_z}^{k'_z}$ to $\mathbb{Z}^{k'-1} \times C_{n_1}^{k'_1} \times \dots \times C_{n_j}^{k'_j+1} \times \dots \times C_{n_z}^{k'_z}$, where $n_j = t$. This means that the decision procedure returned by L , given a word $w \in A^*$, decides as follows.
 - For every generator $a_l \in A$ occurring in w , test whether the minimal t with $(a_l)^t \equiv \lambda$ has already been found and whether a_l occurs exactly γt times in w for some $\gamma \in \mathbb{N}$.
 - If all these tests are positive then return 1.

- If at least one of these tests is negative then return 0.

Moreover, from then on all pairs (a_l, t') in the enumeration will be skipped.

It is easy to see that L identifies every group in \mathcal{G}_C in the limit with respect to any corresponding set A with $(\mathcal{G}, A) \in \mathcal{C}$.

ad 2. Assume to the contrary that \mathcal{G}_C is finitely learnable with respect to A_C , 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 $C_n \in \mathcal{G}_C$ consistent with the scenario experienced by L ; L fails to identify them. \square

For the class considered in the previous result, finite learning was not possible because the target group could be infinite and indistinguishable from infinitely many finite potential target groups. The situation obviously changes when we consider only finite Abelian groups, due to Theorem 18. But in this case we even get a positive result for efficient learning. The following theorem states that every finite Abelian group is efficiently learnable if the generators given generate the finite cyclic subgroups of the target group. Note that every finite Abelian group is isomorphic to the direct product of finitely many finite cyclic subgroups.

Theorem 23 *Let $\mathcal{C} = \{(\mathcal{G}, A) : \mathcal{G} \cong \langle a_1 \rangle \times \dots \times \langle a_m \rangle, A = \{a_1, \dots, a_m\}, \forall i [\sigma(a_i) < +\infty]\}$. Then \mathcal{G}_C is efficiently learnable with respect to A_C .*

Proof Let the algorithm work on group \mathcal{G} with generators A from \mathcal{C} . Since the generators given to the learning algorithm commute, it suffices to find out the order of the generators in A . Since all actions have finite order, this can be done in $\sum_{a \in A} \sigma(a)$ steps. Knowing the orders allows the learning algorithm to identify the inverses of the generators. Then, as before, it suffices to specify a decision procedure that decides if a word $w \in A^*$ satisfies $\mathcal{G}(w) = 1$. A suitable procedure is the one that checks if the number of occurrences of all generators $a \in A$ in w is an integer multiple of the order $\sigma(a)$. Thus, the learner can output this procedure and stop. \square

Note that, in general, the sample-complexity of our learning algorithm on a finite Abelian group is “much smaller” than the size of the group.

We do not know if efficient learning of finite Abelian groups is possible when the given generators do not generate the cyclic subgroups of the target group.

5 Learning group extensions: a special case

Let us now discuss a result 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 faithful representation of a single extension of a group \mathcal{G} (here only for the special case that the extension of \mathcal{G} has 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_a :

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

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

- (1) $\sigma(\Phi'(a)) = \sigma$.
- (2) The group resulting from restricting Φ' to \mathcal{G} is isomorphic to \mathcal{G} .
- (3) $\Phi'^{-1}(a)\Phi'(g)\Phi'(a) = \Phi'(\phi(g))$ for all $g \in S_{\mathcal{G}}$.
- (4) $\Phi'(a)$ 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 $\Phi(\phi_a(g))$, in particular of representing $\phi_a(g)$ using only the given generator elements.

However, a special case of this construction can be used to prove the following result.

Theorem 24 *Let $\mathcal{C} = \{(\mathcal{G}, \{a_1, a_2\}) : \mathcal{G} = \langle a_1 \rangle \rtimes_{\phi} \langle a_2 \rangle, \sigma(\phi(a_2)) = \sigma(a_2), \sigma(a_1) < \infty, \sigma(a_2) < \infty\}$. Then $\mathcal{G}_{\mathcal{C}}$ is efficiently learnable with respect to $A_{\mathcal{C}}$.*

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

$$a_1 a_2 a_1^{-1} \underbrace{a_2 a_2 \dots a_2}_z a_1 a_2^{-1} a_1^{-1},$$

where z fulfills $a_2^z = (a_1 a_2 a_1^{-1})^{-1}$; 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 $\phi(a_2) = a_2^{-6}$.) Third, L constructs the linear representation as described above Theorem 24, knowing that $\Phi(a_2) = \cos(2\pi/\sigma(a_2)) + i \cdot \sin(2\pi/\sigma(a_2)) \in \mathbb{C}^{1 \times 1}$ is a primitive complex unit root. \square

The reader should note that the preliminary version [3] of this paper contained a further claim (Theorem 6) on learning Abelian groups using linear representations. Unfortunately, the proof sketch given there is wrong and cannot be fixed. The corresponding theorem is excluded here.

6 Efficiently learnable group-structured games

In this section we introduce some classes of groups which are related to known puzzle games.

6.1 *Topspin*

A first class of puzzle-related groups we consider is based on the single-player permutation game *Topspin*. This game consists of a ring of α numbered tiles in α possible positions. The section of the first x positions of this ring, containing x tiles, can be flipped.

To be more specific, the player has two actions. The first action is to rotate all α tiles in the ring by one position. The second action is to change the positions of the tiles a_1, \dots, a_x that are currently in positions $1, \dots, x$. If a_i is in position i for $1 \leq i \leq x$, then, after taking the second action, a_i is in position $x + 1 - i$.¹⁰

A schematic illustration of this game is shown in Figure 2.

The *Topspin* game can be represented as an algebraic group. In Definition 25, we use α and x as parameters for defining a class of groups.

¹⁰In the original version of the puzzle, the goal state has tile i in position i for all $i \in \{1, \dots, \alpha\}$.

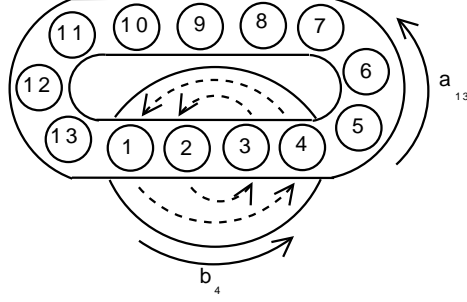


Fig. 2. The Topspin puzzle with $\alpha = 13$ and $x = 4$.

Definition 25 Let $\alpha, x \in \mathbb{N}$ and $2 \leq x < \alpha$. The group $TS(\alpha, x) = \langle a_\alpha, b_x \rangle$ is defined as the subgroup of S_α that is generated by the following two elements a_α and b_x .

- $a_\alpha = (1\ 2\ \dots\ \alpha)$, and
- $b_x = \begin{cases} (1\ x)(2\ x-1)\dots(x/2\ x/2+1), & \text{if } x \text{ is even;} \\ (1\ x)(2\ x-1)\dots((x-1)/2\ (x+3)/2), & \text{if } x \text{ is odd.} \end{cases}$

It is not hard to prove that an agent exploring the Topspin game for unknown parameters α and x can learn the game efficiently, according to the model introduced in Definition 13.

Theorem 26 The class $\{TS(\alpha, x) : 2 \leq x \leq \alpha/2\}$ is efficiently learnable with respect to $\{\{a_\alpha, b_x\} : 2 \leq x \leq \alpha/2\}$.

Proof On input of a set of two generators, an algorithm learning the target group $TS(\alpha, x)$ efficiently can be defined by the following instructions.

- (1) Determine the orders of the generators.
- (2) Let the generator whose order is 2 be named b , the other one be named a . Let $\alpha = \sigma(a)$.
- (3) Issue the queries

$$ba^i ba^{-i} ba^i ba^{-i}$$

for all values $i \geq 2$ in order of ascending i until a (minimal) value \hat{i} is found such that

$$ba^{\hat{i}} ba^{-\hat{i}} ba^{\hat{i}} ba^{-\hat{i}} \equiv \lambda.$$

- (4) Return a decision procedure solving the word problem for the group $TS(\alpha, \hat{i})$ where $a = a_\alpha$ and $b = b_{\hat{i}}$.

It is not hard to see that the decision procedure returned by this algorithm is consistent with the target group. \square

6.2 Hungarian Rings

The following definition formalizes a class of groups based on a variant of the “Hungarian rings” puzzle. Imagine a single-player tile game consisting of a set of numbered tiles arranged in two circles—one containing α_1 many tiles and one containing α_2 many tiles, see Figure 3.

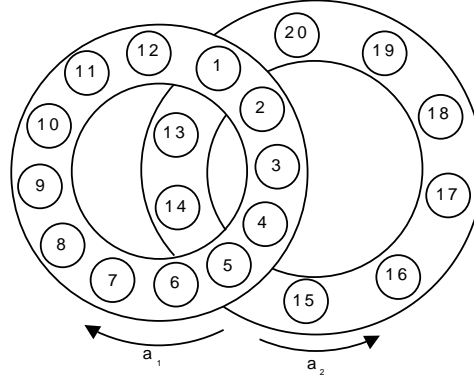


Fig. 3. The Hungarian Rings puzzle with $\alpha_1 = 12$, $\alpha_2 = 10$, $x_1 = 4$, $x_2 = 3$.

The two rings overlap, *i.e.*, they share exactly two tiles. There are $x_1 - 1$ many tiles in the first circle in between the two overlapping tiles and $x_2 - 1$ many tiles in the second circle in between the overlapping tiles.

There are exactly two actions the player can take: rotating all tiles in circle 1 by one position and rotating all tiles in circle 2 by one position. Since the circles overlap, these two actions do not commute.¹¹

It is not hard to see that the structure underlying this puzzle can be described as an algebraic group. With the four parameters α_1 , α_2 , x_1 , and x_2 , we obtain a class of groups defined as follows.

Definition 27 Let $\alpha_1, \alpha_2, x_1, x_2 \in \mathbb{N} \setminus \{0\}$ where $x_i \leq \alpha_i/2$ for $i = 1, 2$. The group $HR(\alpha_1, \alpha_2, x_1, x_2)$ is defined as the subgroup of $S_{\alpha_1 + \alpha_2 - 2}$ that is generated by the following two elements a_1^* and a_2^* .

- $a_1^* = (1 \ 2 \ \dots \ \alpha_1)$,
- $a_2^* = (1 \ \alpha_1 + 1 \ \alpha_1 + 2 \ \dots \ \alpha_1 + x_2 - 1 \ x_1 + 1 \ \alpha_1 + x_2 \ \dots \ \alpha_1 + \alpha_2 - 3 \ \alpha_1 + \alpha_2 - 2)$.

¹¹In the original version of the puzzle each tile has one of four possible colours. A goal state would have all tiles of the same colour lined up with each other, for all four colours. Our version here is different in that tiles have unique names (they are numbered) and thus they are pairwise distinguishable. Hence colours become irrelevant.

$A(\alpha_1, \alpha_2, x_1, x_2)$ then denotes the set $\{a_1^*, a_2^*\}$ of generators.¹²

The following theorem states that an agent exploring the Hungarian Rings game for unknown parameters $\alpha_1, \alpha_2, x_1, x_2$, can learn the game efficiently, according to the model introduced in Definition 13.

Theorem 28 *The class $\{HR(\alpha_1, \alpha_2, x_1, x_2) : \alpha_1, \alpha_2, x_1, x_2 \in \mathbb{N} \setminus \{0\}, x_1 \leq \alpha_1/2, x_2 \leq \alpha_2/2\}$ is efficiently learnable with respect to $\{A(\alpha_1, \alpha_2, x_1, x_2) : \alpha_1, \alpha_2, x_1, x_2 \in \mathbb{N} \setminus \{0\}, x_1 \leq \alpha_1/2, x_2 \leq \alpha_2/2\}$.*

Proof The learning algorithm will operate as follows, given the generators a_1 and a_2 as input.

- (1) Determine the orders of the generators. Let $\alpha_1 = \sigma(a_1)$, $\alpha_2 = \sigma(a_2)$.
- (2) For each $i \in \{1, \dots, \lfloor \alpha_1/2 \rfloor\}$, $j \in \{1, \dots, \lfloor \alpha_2/2 \rfloor\}$, determine the order of the element $(a_1^i a_2^j a_1^{-i} a_2^{-j})$ and let $T_{i,j} = \sigma(a_1^i a_2^j a_1^{-i} a_2^{-j})$.
- (3) Define

$$(x_1, x_2) = \begin{cases} (\lfloor \alpha_1/2 \rfloor, \lfloor \alpha_2/2 \rfloor), & \text{if } T_{\lfloor \alpha_1/2 \rfloor, \lfloor \alpha_2/2 \rfloor} = 1, \\ (\lfloor \alpha_1/2 \rfloor, j), & \text{if } T_{\lfloor \alpha_1/2 \rfloor, j} = 3 \text{ and } T_{\lfloor \alpha_1/2 \rfloor, z} = 2 \text{ for } z \neq j, \\ (i, \lfloor \alpha_2/2 \rfloor), & \text{if } T_{i, \lfloor \alpha_2/2 \rfloor} = 3 \text{ and } T_{z, \lfloor \alpha_2/2 \rfloor} = 2 \text{ for } z \neq i, \\ (i, j), & \text{if } T_{i,j} = 2 \text{ and } T_{z,z'} \neq 2 \text{ for all } (z, z') \neq (i, j). \end{cases}$$

- (4) Return a decision procedure solving the word problem for $HR(\alpha_1, \alpha_2, x_1, x_2)$ over the given set of generators.

To prove that this algorithm learns every possible target group in \mathcal{C} efficiently, the following assertions need to be shown.

- The values $T_{i,j}$ are upper bounded by a constant (in fact, by 5).¹³
- The cases distinguished in the definition of (x_1, x_2) are complete and pairwise disjoint.
- The cases distinguished in the definition of (x_1, x_2) uniquely determine the parameters x_1 and x_2 of the target group exactly in the way they are set in this definition.

All three assertions can be proven with the help of Figure 4, distinguishing the cases (a), (b), (b'), and (c).

Case (a). $\alpha_1/2 = x_1$ and $\alpha_2/2 = x_2$ (here both α_1 and α_2 are even).

In this case, $T_{x_1, x_2} = 1$ and $T_{i, x_2} = T_{x_1, j} = 2$ for all $i < x_1$ and all $j < x_2$.

¹² Note that the definition of a_1^* and a_2^* depends on the parameters $\alpha_1, \alpha_2, x_1, x_2$.

¹³ This property guarantees that when the algorithm executes step 2, it takes a polynomially bounded number of actions.

$T_{i,j}$	1	2	...	$\alpha_2/2$
1	3	3	3	2
2	3	3	3	2
\vdots	3	3	3	2
$\alpha_1/2$	2	2	2	1

(a) $\alpha_1/2 = x_1$ and $\alpha_2/2 = x_2$ (here α_1 and α_2 are even).

$T_{i,j}$	1	2	...	x_2	...	$\lfloor \alpha_2/2 \rfloor$
1	3	3	3	5	3	3
2	3	3	3	5	3	3
\vdots	3	3	3	5	3	3
$\alpha_1/2$	2	2	2	3	2	2

(b) $\alpha_1/2 = x_1$ (here α_1 is even) and $\alpha_2/2 \neq x_2$.

$T_{i,j}$	1	2	...	x_2	...	$\lfloor \alpha_2/2 \rfloor$
1	3	3	3	5	3	3
2	3	3	3	5	3	3
\vdots	3	3	3	5	3	3
x_1	5	5	5	2	5	5
\vdots	3	3	3	5	3	3
$\lfloor \alpha_1/2 \rfloor$	3	3	3	5	3	3

(c) $\alpha_1/2 \neq x_1$ and $\alpha_2/2 \neq x_2$.

Fig. 4. The matrix T for the three different cases (a), (b), and (c). Note that the case “ $\alpha_2/2 = x_2$ (α_2 is even) and $\alpha_1/2 \neq x_1$ ” is symmetric to the one illustrated in (b).

Moreover, $T_{i,j} = 3$ for $i < x_1$ and $j < x_2$.

Case (b). $\alpha_1/2 = x_1$ (here α_1 is even) and $\alpha_2/2 \neq x_2$.

In this case, $T_{x_1, x_2} = 3$ and $T_{x_1, j} = 2$ for $j \neq x_2$. Furthermore, $T_{i, x_2} = 5$ for $i < x_1$ and $T_{i, j} = 3$ for $i < x_1$ and $j \neq x_2$.

Case (b’). $\alpha_2/2 = x_2$ (α_2 is even) and $\alpha_1/2 \neq x_1$.

This case is symmetric to Case (b).

Case (c). $\alpha_1/2 \neq x_1$ and $\alpha_2/2 \neq x_2$.

In this case, $T_{x_1, x_2} = 2$ and $T_{i, x_2} = T_{x_1, j} = 5$ for $i \neq x_1$ and $j \neq x_2$. Moreover, $T_{i, j} = 3$ for $i \neq x_1$ and $j \neq x_2$.

It is easy to see that the verification of the cases (a), (b), (b'), and (c) proves the assertions claimed above. To verify the cases, consider the effect of the action sequences $a_1^i a_2^j a_1^{-i} a_2^{-j}$.

When executing an action sequence of the form $a_1^i a_2^j a_1^{-i} a_2^{-j}$, most of the tiles in the game remain unaffected. The affected tiles are shown in Figure 5. Let those elements be named b, c, d, e, f and g . If $i \neq x_1$ and $j \neq x_2$, then $a_1^i a_2^j a_1^{-i} a_2^{-j} = (bcd)(efg)$, thus the order of this action sequence is 3.

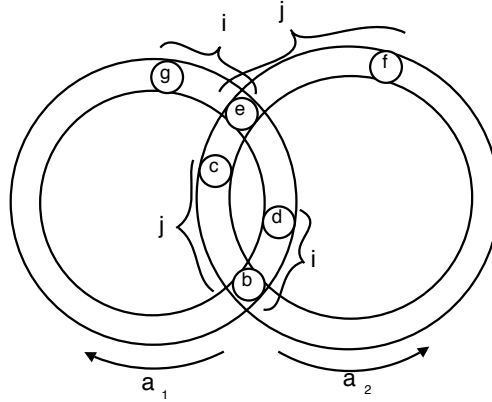


Fig. 5. The tiles affected by the action sequence $a_1^i a_2^j a_1^{-i} a_2^{-j}$. In certain cases, some of these tiles might coincide.

With certain values of the parameters, the following coincidences are possible:

- (1) $i = x_1 \rightarrow d = e$
- (2) $j = x_2 \rightarrow c = e$
- (3) $i = (a_1 - x_1) \rightarrow g = b$
- (4) $j = (a_2 - x_2) \rightarrow f = b$

Note that the third case is possible only if $x_1 = a_1/2$ and it implies that the first case holds as well. (The same is true with the fourth and second case.)

- If $i = x_1$ and none of the other coincidences hold then $\sigma(a_1^i a_2^j a_1^{-i} a_2^{-j}) = \sigma((bcfge)) = 5$.
- If $i = x_1$ and $j = x_2$ ($c = d = e$) then $\sigma(a_1^i a_2^j a_1^{-i} a_2^{-j}) = \sigma((be)(fg)) = 2$.
- If $i = x_1 = a_1 - x_1 = a_1/2$ then $\sigma(a_1^i a_2^j a_1^{-i} a_2^{-j}) = \sigma((be)) = 2$.
- If $i = x_1 = a_1 - x_1 = a_1/2$ and $j = x_2 \neq a_2/2$ ($d = c = e, g = b$) then $\sigma(a_1^i a_2^j a_1^{-i} a_2^{-j}) = \sigma((ebf)) = 3$.
- If $i = x_1 = a_1 - x_1 = a_1/2$ and $j = x_2 = a_2 - x_2 = a_2/2$ ($c = d = e, f = g = b$) then $\sigma(a_1^i a_2^j a_1^{-i} a_2^{-j}) = \sigma((id)) = 1$.

The above cases explain all the entries in table T .

This completes the proof. \square

Remark 29 *Note that the group $HR(\alpha_1, \alpha_2, x_1, x_2)$ equals the alternating group $A_{\alpha_1 + \alpha_2 - 2}$ if both α_1 and α_2 are odd, and equals $S_{\alpha_1 + \alpha_2 - 2}$ itself, if at least one of the parameters α_1 and α_2 is even. Moreover, the group $TS(\alpha, x)$ in some cases ($x \equiv 2 \pmod{4}$) equals S_α , cf. Chen and Skiena [5].*

This shows that if the generators are well chosen then such complex groups as the symmetric groups are efficiently learnable.

6.3 A tile game

The tile game presented in this section can be seen as loosely related to sliding-tile puzzles. However, the version designed here is not close to any actual game, we defined it for illustration purposes.

The game consists of an $n \times n$ table of n^2 numbered tiles in which the single player can move tiles according to $3n - 1$ actions. These actions can be described as follows:

- (\hat{a}_{ni}) for $1 \leq i \leq n$. The action \hat{a}_{ni} horizontally rotates all tiles in the i^{th} row of the table by one position.
- (\hat{b}_{ni}) for $1 \leq i \leq n$. The action \hat{b}_{ni} vertically rotates all tiles in the i^{th} column of the table by one position.
- (\hat{c}_{ni}) for $2 \leq i \leq n$. The action \hat{c}_{ni} diagonally rotates all tiles in the diagonal corresponding to the positions $(n, i) \dots (i, n)$ by one position.

Figure 6 shows the table and the actions. Note that while the first two groups of actions are of order n , the order of \hat{c}_i is $n - i + 1$.

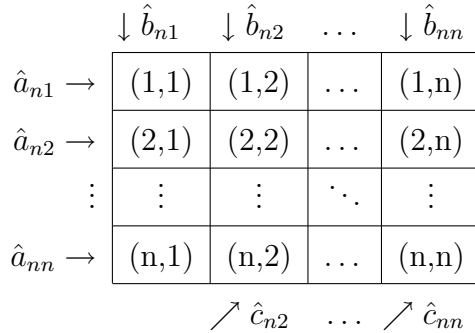


Fig. 6. The “tile game”.

This game can be represented as a group in the parameter n , formally defined as follows.

Definition 30 Let $n \in \mathbb{N} \setminus \{0\}$. The group $TG(n)$ is the subgroup of S_{n^2} generated by the union of the following three sets of generators.

- $\{\hat{a}_{ni} : 1 \leq i \leq n \text{ and } \hat{a}_{ni} = ((i, 1) \dots (i, n))\}$
- $\{\hat{b}_{ni} : 1 \leq i \leq n \text{ and } \hat{b}_{ni} = ((1, i) \dots (n, i))\}$
- $\{\hat{c}_{ni} : 1 \leq i \leq n \text{ and } \hat{c}_{ni} = ((n, i) (n-1, i+1) \dots (i, n))\}$

The main difference between this class of groups and the ones presented in the two previous subsections is that here the number of actions is not constant but depends on the parameter n .

From the point of view of learning, it is therefore not enough to determine the value of the parameter n . It is additionally required to determine the concrete role of each action given as an input to the learning algorithm. Nevertheless, as the following theorem states, the class $\{TG(n) : n \in \mathbb{N} \setminus \{0\}\}$ is efficiently learnable with respect to the natural choice of its generators.

Theorem 31 The class $\{TG(n) : n \in \mathbb{N} \setminus \{0\}\}$ is efficiently learnable with respect to $\{\{\hat{a}_{n1}, \dots, \hat{a}_{nn}, \hat{b}_{n1}, \dots, \hat{b}_{nn}, \hat{c}_{n2}, \dots, \hat{c}_{nn}\} : n \in \mathbb{N} \setminus \{0\}\}$.

Proof On input of a set $\{d_1, \dots, d_k\}$ of k different generators, an algorithm learning the target group $TG(n)$ efficiently can be defined by the following instructions.

- (1) Let $n = k + 1/3$.
- (2) Obtain the orders of the generators.
- (3) For all $i \in \{2, \dots, n\}$, let c_i be the unique given generator d_j with $\sigma(d_j) = n - i + 1$.
- (4) Pick an arbitrary given generator $d \in \{d_1, \dots, d_k\} \setminus \{c_2, \dots, c_n\}$. Let \hat{B} be the set of all given generators in $\{d_1, \dots, d_k\} \setminus \{d, c_2, \dots, c_n\}$ that do not commute with d . Let $\hat{A} = \{d_1, \dots, d_k\} \setminus (\{c_2, \dots, c_n\} \cup \hat{B})$.
 (* Note that \hat{A} contains d and both \hat{A} and \hat{B} are of cardinality n . Due to the symmetry of the tile game, the roles of \hat{A} and \hat{B} can be swapped without changing the underlying group. *)
- (5) Iteratively, for $i \in \{1, \dots, n-1\}$ in order of ascending i , let a_i be the unique generator in $\hat{A} \setminus \{a_1, \dots, a_{i-1}\}$ that commutes with c_{i+1} and let b_i be the unique generator in $\hat{B} \setminus \{b_1, \dots, b_{i-1}\}$ that commutes with c_{i+1} .
- (6) Let a_n be the unique element in $\hat{A} \setminus \{a_1, \dots, a_{n-1}\}$. Let b_n be the unique element in $\hat{B} \setminus \{b_1, \dots, b_{n-1}\}$.
- (7) Return a decision procedure solving the word problem for $TG(n)$ where
 - for each $i \in \{1, \dots, n\}$, a_i takes the role of \hat{a}_{ni} and b_i takes the role of \hat{b}_{ni} , and
 - for each $i \in \{2, \dots, n\}$, c_i takes the role of \hat{c}_{ni} .

It is easy to see that the algorithm outputs a decision procedure consistent with the target group. It is also obvious that the number of queries the algorithm

issues is not higher than the efficiency constraints allow. \square

7 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 set of generators given as input to the learner, not only under efficiency constraints, but even when no requirements in terms of efficiency are imposed on the learner. Our negative results suggest that it is in general too strong a requirement to learn with respect to all possible generators of a group—which is in fact not surprising and gives answers to some of the questions that motivated our research.

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 motivated by the contrasting results we obtained concerning dihedral groups. The *positive* results on efficient learning of finite dihedral groups assume that the learner is given generators the order of which is proportional to the size of the target group. Since our model of efficient learning allows the learner to take a number of actions that is polynomial in the order of the generators, the learner can here take a number of actions polynomial in the size of the group. Thus every state can be visited and learning becomes trivial. The *negative* results on efficient learning of finite dihedral groups show that generators of lower order, which *disallow* the learner to take as many actions as there are states in the environment, are not suitable for learning environments with the structure of finite dihedral groups. This example motivates the design and analysis of models that capture the case of learning when the “size of the representation” given to the learner is not linear but only logarithmic in the size of the group.

However, it is worth noting that our positive result on learning finite Abelian groups (Theorem 23) illustrates cases in which efficient learning under our current model is possible even if the sum of the orders of the generators given may be logarithmic in the size of the target group. Note though that we have positive results on efficient learning of finite Abelian groups only in case the given generators generate the cyclic subgroups of the target group. Whether or not other types of generators are in general suitable for efficient learning of finite Abelian groups remains an open question.

Acknowledgements

We thank Barnabás Póczos, András Antos, Marcus Hutter, and Robert Holte for helpful discussions.

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

References

- [1] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.
- [2] L. Babai and E. Szemerédi. On the complexity of matrix group problems I. In *IEEE Symposium on Foundations of Computer Science*, pages 229–240, 1984.
- [3] G. Bartók, C. Szepesvári, and S. Zilles. Active learning of group-structured environments. In *Proceedings of the 19th International Conference on Algorithmic Learning Theory*, Lecture Notes in Computer Science 5254, pages 329–343. Springer-Verlag, 2008.
- [4] W.W. Boone. The word problem. *Annals of Mathematics*, 70:207–265, 1959.
- [5] T. Chen and S. Skiena. Sorting with fixed-length reversals. *Discrete Applied Mathematics*, 71:269–295, 1996.
- [6] E.M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.
- [7] R. Holte, J. Grajkowski, and B. Tanner. Hierarchical heuristic search revisited. In *Symposium on Abstraction, Reformulation and Approximation*, pages 121–133, 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] M.L. Littman, R. Sutton, and S. Singh. Predictive representations of state. In *Advances in Neural Information Processing Systems 14 (NIPS)*, pages 1555–1561, 2002.
- [11] P.S. Novikov. On the algorithmic undecidability of the word problem in group theory. In *Proceedings of the Steklov Institute of Mathematics*, volume 44, pages 1–143, 1955. In Russian.
- [12] R.L. Rivest and R.E. Schapire. Diversity-based inference of finite automata. *Journal of the ACM*, pages 555–589, 1994.

- [13] J.J. Rothman. *An Introduction to the Theory of Groups*. Springer, 1995.
- [14] F. Stephan and Y. Ventsov. Learning algebraic structures from text. *Theoretical Computer Science*, 268(2):221–273, 2001.
- [15] A.L. Strehl, C. Diuk, and M.L. Littman. Efficient structure learning in factored-state MDPs. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI-07)*, pages 645–650, 2007.
- [16] N.V. Vinodchandran. *Counting Complexity and Computational Group Theory*. PhD thesis, Institute of Mathematical Sciences, Chennai, India, 1999.