Planning and Learning in Permutation Groups

Amos Fiat[1] Shahar Moses[2] Adi Shamir[3] Ilan Shimshoni[3] Gabor Tardos[4]

Abstract

Planning is the problem of synthesizing a desired behavior from given basic operations, and learning is the dual problem of analysing a given behavior to determine the unknown basic operations. In this paper we develop new algorithms for solving these problems in the context of invertible operations on finite state environments. In addition to their obvious AI applications, these algorithms can efficiently find the shortest way to solve Rubik's cube, test ping-pong protocols, and solve systems of equations over permutation groups.

1 Introduction

Let G be a permutation group over $S = \{1, 2 \ldots, q\}$ generated by $P = \{g_1, g_2, \ldots, g_k\}$. Each g in G can be represented (in infinitely many ways) by words in $(\Sigma \bigcup \Sigma^{-1})^*$ where Σ is the set of formal symbols $\{g_1, g_2, \ldots, g_k\}$ and Σ^{-1} is the set of their formal inverses $\{g_1^{-1}, g_2^{-1}, \ldots, g_k^{-1}\}$. The length of $g \in G$ with respect to P is defined as the length of the shortest word $w \in (\Sigma \bigcup \Sigma^{-1})^*$ that represents g, and the diameter of G is defined as the largest length of its members.

Sims [Sim70] described a simple method for deciding membership in G, which was shown to run in polynomial time by Furst Hopcroft and Luks [FHL80]. Unfortunately, the actual representations derived from this algorithm are usually exponentially long. The problem of determining the length of a given g was shown to be NP-hard by Even and Goldreich [EG81], and P-space complete by Jerrum [J85], and in fact it is not difficult to show that in some permutation groups there are permutations with super-polynomial length. However, most of the interesting groups have very small diameters and for them the Sims representations are very wasteful. Our goal in Section 2 is to develop a new planning algorithm which can find short representations whenever they exist. In Section 3 we describe some interesting applications of this algorithm in the context of Rubik's group (whose diameter is believed to be 20), and in particular describe the first practical algorithm for finding the shortest solution for any given state of Rubik's cube.

In Section 4 we consider the problem of Learning in permutation groups. In this problem an intelligent entity (baby, robot, etc.) is introduced into a new and unknown environment. He can apply several invertible operations which affect the environment, but he has no a-priori understanding of their nature. His goal is to infer a complete description of the basic operations from a large number of observations. We assume that the entity does not have complete control over the environment, cannot always isolate his operations, and cannot always observe the immediate effect of his actions, and thus we do not allow him to learn simply by bringing the environment to each one of its possible states and trying out each one of the basic operations. Our formalization of this learning problem consists of a set $S = \{1, 2 \ldots, q\}$ of states of the environment, a set $\Sigma = \{g_1, g_2, \ldots, g_k\}$ of unknown permutations, and a collection of observations of the form 'the cummulative effect of applying the sequence of operations specified in w to initial state i results in final state j". The main result of this section is a new algorithm which can infer the unknown g_i from these observations in almost linear time.

Remark: In this paper we denote permutations over $S = \{1, 2, \ldots, q\}$ by q-vectors whose i-th position indicates the value to which i is mapped. We enclose these vectors in angular brackets to avoid confusion with the cycle representation of permutations. We apply these permutations to arguments written to their left, and compose them from left to right. For example,
$< 3, 1, 2 >< 3, 2, 1 >= 1 < 3, 2, 1 >= 3$.

2 The Planning Algorithm

The basic problem considered in this section is to determine whether a given permutation $g \in G$ can be represented by some word $w \in (\Sigma \cup \Sigma^{-1})^*$ of length n, and to find such a representation if it exists. Simple adaptations of our algorithm can find the shortest representations of g, deal with a partially specified g, and yield a probabilistic upper bound on the diameter of G.

The total number of words of length n in $(\Sigma \cup \Sigma^{-1})^*$ is $R = (2k)^n$ (or $2k(2k-1)^{n-1}$ if we are interested only in reduced words which do not contain adjacent inverses), and thus the problem can be trivially solved by exhaustive search in $O(R)$ time and $O((k+n)q)$ space. Our main idea is to tradeoff time for space in an efficient way to get a $O(R^{1/2})$ time and $O(R^{1/4})$ space algorithm. With current technology, the new algorithm can solve representation problems with search spaces of up to $R = 2^{80}$ possible words, which is an improvement of many orders of magnitude over any previous algorithm.

To tradeoff time for space, we divide the (unknown) word w that represents g into t subwords $w = w_1 w_2 \ldots w_t$ (we assume that t divides n and thus all the w_i have the same length n/t). We replace the original generators by "super-

[1] Computer Science Dept., Tel-Aviv University, Israel
[2] Mathematics Dept., The Hebrew University, Israel
[3] Applied Mathematics Dept., The Weizmann Institute, Israel
[4] Algebra Dept., Eötvos University, Hungary

CH2806-8/89/0000/0274/$01.00 © 1989 IEEE

generators" consisting of all the permutations which can be obtained by composing n/t generators from $\Sigma \cup \Sigma^{-1}$, and search for a representation of g as a word of length t over the supergenerators. More generally, we define the following problem:

The t-list problem: Given a permutation g and t lists $L_1, L_2, \ldots L_t$ of m permutations each, determine whether $g \in L_1 L_2 \ldots L_t$.

The original representation problem can be viewed as the n-list problem in which each L_i contains the original generators and their inverses, whereas exhaustive search can be viewed as the 1-list problem in which L_1 contains all the permutations in G whose length is n. Our new algorithm is based on the 4-list version of the representation problem, and thus the size of each list (which determines the space complexity of the algorithm) is $m = (2k)^{n/4} = R^{1/4}$.

By modifying the permutations stored in the L_i lists, we can show:

Lemma 1 *The problem of determining whether $g \in L_1 L_2 L_3 L_4$ can be reduced to the problem of determining whether $L_1' L_2' \bigcap L_3' L_4'$ is not empty, with $|L_j| = |L_{i_j}'|$.*

Proof: Let $L_1' = L_1$, $L_2' = L_2$, $L_3' = \{g\tau^{-1} | \tau \in L_4\}$, and $L_4' = \{\tau^{-1} | \tau \in L_3\}$. If $\tau_1 \tau_2 = g\tau_4^{-1}\tau_3^{-1}$, $\tau_i \in L_i$, then $g \in L_1 L_2 L_3 L_4$. □

To find a common permutation h in $L_1' L_2'$ and $L_3' L_4'$, we generate their entries in lexicographically increasing order, and then search the two sorted lists in time which is linear in their length.

The main problem left is how to generate the m^2 permutations in $L_1' L_2'$ (or $L_3' L_4'$) in lexicographically increasing order without actually storing or sorting so many permutations. This on-the-fly generation has been considered by Schroeppel and Shamir [SS81] in the context of the knapsack problem. When L_1' and L_2' are lists of numbers, they proposed to generate all their sums $L_1' + L_2'$ in increasing numeric order by using the following algorithm:

1. Sort L_1' into increasing numeric order, with x_1 being its smallest element.
2. Create a priority queue Q which initially contains the sums $x_1 + y$ for all $y \in L_2'$.
3. Repeat until Q becomes empty: Delete the pair $x + y$ with the smallest sum from Q, print its sum, and replace it with $x' + y$ where x' is the successor of x in L_1' (if such a successor does not exist, insert nothing).

This algorithm uses only linear space, and its running time (up to logarithmic factors) is proportional to the number of printed values. The sortedness of the printed outputs is an easy consequence of the monotonicity of addition (namely, that $x \leq x'$ implies $x + y \leq x' + y$). Unfortunately, the composition of permutations is an inherently non monotonic operation (not only under lexicographic ordering, but under any total order whatsoever), and thus the Schroeppel-Shamir algorithm cannot be directly used in permutation groups. □

Example 1 *Consider the permutations*

$$x = < 1,7,6,5,4,3,2 > \quad \text{and} \quad x' = < 2,1,3,4,5,6,7 >,$$

x' is the immediate successor to x under lexicographic ordering. Then for $y = < 7,1,2,3,4,5,6 >$ the composition xy is $< 7,6,5,4,3,2,1 >$ while the composition $x'y$ is $< 1,7,2,3,4,5,6 >$. Since $xy \gg x'y$, these permutations demonstrate the non-continuity as well as the non-monotonicity of the composition operation in permutation groups.

The main difficulty in using this algorithm is that we have to traverse the x in L_1' in a different order for each y in L_2', in order to get a continuously ascending sequence of xy compositions. To overcome this difficulty, we store the permutations in L_1' as a tree T rather than as a linear list. The tree has height q and each vertex has q sons, so that each root-to-leaf path has a q-vector $< i_1, i_2 \ldots, i_q >$ over $S = \{1, 2, \ldots q\}$ as its name. The permutations in L_1' correspond to a subset of T's leaves, and we prune all the unused leaves along with their unused ancestors to make sure that the size of T is proportional to the size of L_1'. The natural order $<$ on the remaining leaves is the lexicographic order on their path names. However, for any permutation y we can define a new lexicographic order $<_y$ on the leaves of T by reordering the sons $1, 2, \ldots, q$ of each internal vertex V so that $1y^{-1} < 2y^{-1} < \ldots < qy^{-1}$. By visiting the leaves of the tree T in the new $<_y$ order we can show:

Lemma 2 *Given a tree T (which can be exponentially large in q) and two permutations x and y, we can find the x' in T for which $x'y$ is the smallest permutation larger than xy (under standard lexicographic ordering) in $O(q^2)$ time.*

Consider now the point in the execution of the algorithm at which the pair of permutations xy has just been deleted from Q. We can now use the algorithm of Lemma 2 to efficiently find the new pair $x'y$ which has to be reinserted into Q. Since we associate each y in L_2' with all the possible x in L_1', and get a lexicographically non-decreasing sequence of outputs from Q, we have shown:

Theorem 3 *The representation problem in permutation groups can be solved by the 4-list algorithm in $O(R^{1/2})$ time and $O(R^{1/4})$ space.*

A simple generalization of this algorithm can solve the representation problem for any time and space complexities along the tradeoff curve $TS^2 = O(R)$ for $S \leq O(R^{1/4})$. A natural question is whether better tradeoffs can be obtained by solving the t-list problem for larger values of t. The surprising answer is that arbitrarily good tradeoffs are possible:

Theorem 4 *1. The 10-list problem over permutation groups can be solved in time and space complexities related by $TS^3 = O(R)$ for $S \leq O(R^{1/10})$.*

2. The 22-list problem over permutation groups can be solved in time and space complexities related by $TS^4 = O(R)$ for $S \leq O(R^{1/22})$.

3. *More generally, the t_j-list problem for $t_j = 2^j + 2^{j-1} - 2$ can be solved in time and space complexities related by $TS^j = O(R)$ for $S \leq O(R^{1/t_j})$.*

The proof of this theorem will be given in the full paper. However, it should be stressed that all these 'improved" tradeoffs are less important because the "real" time/space tradeoff does not favor time to such an extent.

3 Applications to Rubik's cube

The problem of determining short solutions to Rubik's cube instances has been looked at by many hobbyists and group theorists, but has not been completely solved so far. The problem is essentially the representation problem in a particular permutation group generated by 18 basic permutations (the rotation of each one of the six faces by 90, 180 or 270 degrees) operating on the 48 movable subfaces of the cube. A simple counting argument shows that at least 17 moves are required to solve most instances of the cube, but Berlekamp, Conway and Guy [BCG81] used a refined analysis to show that some instances require at least 18 moves. Singmaster [Sin79] conjectured that "God's algorithm" can solve all the instances in at most 20 moves, but did not provide any experimental evidence to support this conjecture. The fastest known cube algorithm (due to Thistlethewaite [BCG81]) requires 52 moves.

The number of non-redundant sequences of 20 moves (i.e. those which do not contain two successive rotations of the same face) exceeds 2^{78} and thus all the previously known algorithms could not find optimal solutions in reasonable time and space complexities. However, our new algorithm can find 20 move solutions to Rubik's cube instances in $O(2^{40})$ time and $O(2^{20})$ space: all we have to do is to create the list L of all the 911,250 permutations which can be generated by non-redundant compositions of 5 face rotations, and to use our 4-list algorithm to find the given instance g in $LLLL$. It is not difficult to show that this algorithm will automatically find shorter solutions whenever they exist, and thus if Singmaster's conjecture is correct, we have a feasible implementation of 'God's algorithm"!

An interesting application of the new algorithm (first pointed out to us by Stuart Kurtz) is to obtain a probabilistic upper bound on the diameter of a given permutation group G. The basic idea is to compute the length of sufficiently many random $g \in G$ (such g can be chosen with uniform probability distribution in polynomial time by using the Sims representations). We cannot use the largest discovered length as our estimate of the diameter, since G may contain very few permutations of maximal length. However, it is easy to show:

Lemma 5 *If more than half of the permutations $g \in G$ have length bounded by c, then the diameter of G is at most $2c$.*

As a result, any probabilistic algorithm for bounding the median length of random permutations in G can be transformed into a probabilistic algorithm for bounding the diameter of G. If this median length in Rubik's group can be experimentally shown to be at most 18, we can conclude that the diameter of G is at most 36. While this derived bound is considerably higher than Singmaster's conjectured bound of 20, it is considerably smaller than Thistlethwaite's best proven bound of 52.

In addition to finding optimal solutions to given instances of the cube, we can use the new algorithm to derive optimal subroutines for the manual solution of the cube. Solutions of this type usually solve the cube in stages from bottom to top, and at each stage the goal is to flip particular subcubes without moving the lower subcubes but without caring about the effect of the subroutine on the higher subcubes. This is a special case of 'planning with dont cares" in which the desired permutation g has the form $g =< z_1, z_2, \ldots, z_i, ??, \ldots ? >$, and we are looking for the shortest representation of any permutation g' which agrees with g on the first i arguments. Even though the number of compatible g' permutations may be exponential and the length of their shortest representations may fluctuate wildly, we can solve this problem with essentially the same complexity as in the fully specified case:

Theorem 6 *The representation problem for partially specified permutations can be solved in $O(R^{1/2})$ time and $O(R^{1/4})$ space.*

Another interesting application of our algorithm is to explore Rubik's group by enumerating all its identities. For example, we have enumerated all the inequivalent nontrivial identities of length 16 in this group, such as:

$$B3\,U2\,L3\,B1\,F1\,D2\,U2\,B3\,F3\,R1\,U2\,B1\,F1\,D2\,U2\,F3$$
$$D1\,F1\,L2\,U2\,L1\,U3\,F3\,U3\,L1\,U1\,L1\,D3\,L3\,F1\,U3\,F3$$

There are exactly 1190 such indentities, and their list is available upon request.

4 The Learning Algorithm

As stated in the introduction, our learning problem is essentially the problem of solving a system of equations of the form $iw = j$ where i and j are states in S and w is a word over the unknown permutations in $\Sigma \cup \Sigma^{-1}$. Without loss of generality, we can assume that the given words w are reduced.

Example 2 *Let $S = \{1,2,3\}$ and $\Sigma = \{x,y\}$. Consider the following set of equations:*

$$1xy = 1, \quad 2y^{-1}x = 3, \quad 1xy^{-1} = 2,$$
$$3yxy = 1, \quad 2yxx = 3, \quad 3x^{-1}yx^{-1} = 2$$

The problem is to deduce their unique solution $x =< 3,2,1 >$ and $y =< 2,3,1 >$.

Given a system of equations over an arbitrary domain, we can define its "solvability" in two different ways:

1. Semantic solvability: Exactly one set of values satisfies all the equations.

2. Syntactic solvability: A unique solution of the form variables=values can be derived as additional equations from the given equations by a finite sequence of equality-preserving transformations.

The semantic solvability of systems of equations over permutations groups is an extremely difficult decision problem:

Lemma 7 *(i) Deciding whether the system has at least one solution is NP-complete.*

(ii) Deciding whether the system has at most one solution is co-NP-complete.

Since the only viable way to check the semantic solvability of systems of equations over permutation groups seems to be exhaustive search, handling even a small set of equations with $k = 2$ unknown permutations over $q = 16$ elements may require $(q!)^k = 2^{88}$ operations.

However, learning problems are usually characterized by an abundance of equations obtained from a long series of observations. The first few equations give rise to an underdefined system which is neither semantically solvable nor syntactically solvable. As we discover more equations, the system becomes semantically solvable, but it is difficult to discover the transition or to solve the equations. By adding even more questions, we can make the system syntactically solvable. As shown in this paper, we can then demonstrate this fact and actually find the solution by an almost linear time algorithm. Finally, if we add at this stage too many contradictory equations, we can prove the unsolvability of the system by deriving an impossible equation of the form $i = j$ for two different values in S.

The equality preserving transformations we allow in our definition of syntactic solvability are all the group theoretic operations:

1. $ie = i$ is an equation for any $i \in S$.
2. If $iw'w" = j$ is an equation and x is an operation in Σ, then $iw'xx^{-1}w" = j$ and $iw'x^{-1}xw" = j$ are also equations.
3. If $iw'xx^{-1}w" = j$ or $iw'x^{-1}xw" = j$ are equations, then $iw'w" = j$ is also an equation.
4. If $iw = j$ is an equation, then $jw^{-1} = i$ is also an equation.
5. If $iw' = j$ and $jw" = k$ are equations, then $iw'w" = k$ is also an equation.

We denote the set of all the original and implied equations by E, and the subset of E in which the words w are reduced by D.

Our learning algorithm is a natural extension of the coset enumeration techniques used by combinatorial group theorists (e.g., Todd and Coxeter [TC36] and Mendelsohn [M64]) to study abstract groups presented by generators $x, y, \ldots$ and identity relations $w = e$, and the graphical technique used by Angluin [A82] to learn reversible regular languages. Given a system of equations, we construct a directed graph G whose edges are labelled by letters from Σ. We interpret any undirected path in G as a word $w \in (\Sigma \cup \Sigma^{-1})^*$ by using the original letters for edges traversed in the forwards direction and their inverse letters for edges traversed in the backwards direction. The graph contains q special vertices, which denote the q possible values in S. An undirected path $w \in (\Sigma \cup \Sigma^{-1})^*$ from special vertex i to special vertex j corresponds to the equation $iw = j$ (by our conventions, we get the equivalent equation $jw^{-1} = i$ by following the same path in the reverse direction). Initially the graph contains only the q special vertices and no edges (and thus it represents only the trivial equations $ie = i$). To accomodate a new equation $iw = j$, we just add a new path from i to j (via new intermediate vertices) in which the labels and directions of the edges correspond to the successive letters in w.

Example 3 *Consider once more the six equations over* $\Sigma = \{x, y\}$ *and* $S = \{1, 2, 3\}$*:*

$$1xy = 1, \quad 2y^{-1}x = 3 \quad 1xy^{-1} = 2$$
$$3yxy = 1, \quad 2yxx = 3, \quad 3x^{-1}yx^{-1} = 2.$$

The graph G generated by these equations is given in Figure 1.

G:

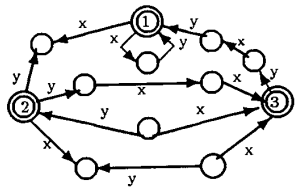

Figure 1: The graph G generated by example 3.

To show that the equation $1y = 2$ is implied by this graph, we have to find some path w from 1 to 2 whose reduced form is the single letter y. The inherent non-determinism of the reduction process makes this a non-trivial task: the shortest path of this type is $w = xyxyy^{-1}x^{-1}y^{-1}x^{-1}y$, obtained by starting at 1, proceeding twice around the loop, continuing from 1 to 3, and finally going to 2 via the middle path. This path is generated by composing two instances of the first

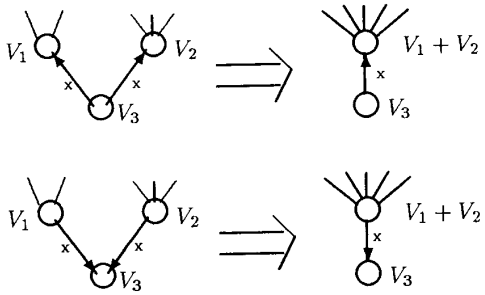

Figure 2: Merging Vertices.

equation, the inverse of the fourtn equation, and the inverse of the second equation. Note in particular that this w is much longer than either the original or the final equations. □

Given the graph G which represents all the original equations, our algorithm repeatedly merges any pair of (special or non-special) vertices V_1 and V_2 for which there exists some V_3 which is connected to both V_1 and V_2 by edges with the same label and direction. (See Figure 2).

Note that by merging V_1 and V_2 we can create new mergeable pairs of vertices which were not mergeable in the original graph. Since each merge operation reduces the number of vertices by 1, the number of merge operations is bounded by the size of the original graph which is bounded by q plus the total number of symbols in all the original equations. Since this reduction process has the Church-Rosser property, the order of the merges does not affect the final reduced graph, which is denoted by $\overline{G}$.

Example 4 *Consider the graph G in Example 3. The graphs in figure 3 represent the successive generations of intermediate graphs obtained during the reduction process (for the sake of clarity, we connect all the mergeable pairs of vertices in the graph by dotted lines).*

The last graph is reduced since none of its vertices are mergeable, and thus it is the final $\overline{G}$. □

The crucial property of this reduction is:

Theorem 8 *Let E_{ij} be the set of equations $iw = j$ associated with the undirected paths from i to j in G, and let D_{ij} be the same set in which each word w is replaced by its reduced form. Then:*

(i) The reduction process can affect the E_{ij} but leaves the D_{ij} unchanged.

(ii) In the final reduced graph $G, E_{ij} = D_{ij}$.

Since $\overline{G}$ is a deterministic finite automation, all its paths are reduced by definition, and thus we can determine whether a given equation $iw = j$ with reduced w is syntactically implied by the original equations simply by checking whether the path w which starts at i ends at j. In particular, we can deduce the following corollaries:

Theorem 9 *The system of equations is syntactically solvable iff in the subgraph of $\overline{G}$ induced by the q special vertices each vertex has exactly one outgoing edge with each label.*

Theorem 10 *If the reduction process attempts to merge two special vertices, the original system of equations is contradictory.*

Example 5 *The system of equations in Example 3 is syntactically solvable, since the reduced graph in Example 5 has all the required edges between special vertices. By examining these edges, we can easily conclude that the unique solution of this system of equations is $x =< 3,2,1 >$ and $y =< 2,3,1 >$. Adding the equation $1yyy = 2$ to the original equations will cause the corresponding G to collapse to a single vertex, and thus the expanded system of equations is contradictory.* □

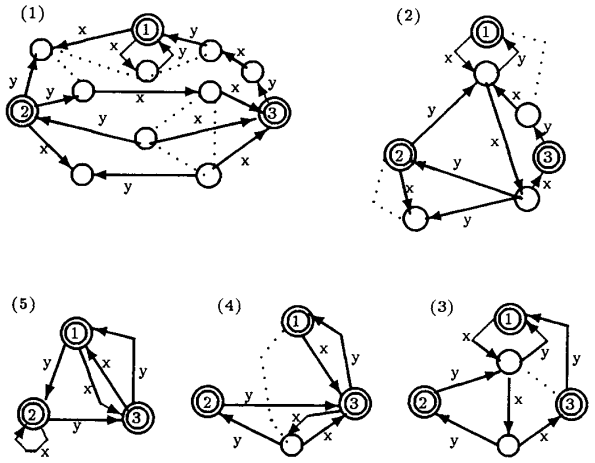

Figure 3: Reduction process of example 4

Remark: In some learning problems, the given equations contain only uninverted symbols from Σ. Our learning algorithm can handle such problems without difficulty, but its proof of correctness still depends on the assumption that the unused inverse operations exist.

By carefully optimizing the various subroutines in our algorithm, we can prove:

Theorem 11 *The syntactic solvability of a given system of equations can be decided in $O(m\alpha(m))$ time and $O(m)$ space, where m is q plus the total number of symbols in all the equations and $\alpha(x)$ is the inverse to the Ackerman function.*

This almost linear complexity enabled us to solve huge systems with up to one million equations in reasonable time and space complexities.

The special case of syntactic solvability with $q = 1$ has a particularly interesting interpretation. For this value of q the permutations $x, y \ldots$ have no internal structure, and thus they can be considered as the generators of a free group. Instead of equations $iw = j$, we are given an initial set of "known" words w, and asked to close this set under the group operations of products, inverses and reductions (in particular, the learning problem is to determine whether the original generators $x, y \ldots$ are in this closure). Note that this problem is not the standard word problem in free groups, since the words w are "known" but are not necessarily the identity, and thus we are not allowed to delete occurrences of w from the middle of other words. This slight change of interpretation suffices to make the "knowledge closure" solvable in almost linear time, whereas the "identity closure" is undecidable!

A natural question (which apparently has not been addressed so far in the literature) is to determine the threshold

number t_n of random reduced words in $(\Sigma \cup \Sigma^{-1})^n$ we have to choose to make their closure under group operations the whole $(\Sigma \cup \Sigma^{-1})^*$.

Example 6 *The following approximations of t_n were experimentally found by running the learning algorithm on randomly generated sets of words of length n over $\Sigma = \{x, y\}$:*

$t_5 \approx 5$, $t_9 \approx 24$, $t_{13} \approx 130$, $t_{17} \approx 778$,
$t_{21} \approx 5272$, $t_{25} \approx 37987$, $t_{29} \approx 282159$.

Let $R = 2k(2k-1)^{n-1}$ be the number of possible reduced words of length n over a k-letter alphabet. The following upper bound on the asynptotic behavior of t_n is a simple corollary of the birthday paradox:

Lemma 12 $t_n = O(R^{1/2})$.

Proving that this bound is tight is much harder, and requires a very delicate counting argument:

Theorem 13 $t_n = \Omega(R^{1/2-\varepsilon})$ *for any* $\varepsilon > 0$.

In the context of our learning algorithm in permutation groups with $q > 1$, we can show:

Theorem 14 *The expected number of random equations of length n over q states required to make the system syntactically solvable is $O(t_n q)$.*

It is important to note that while this number increases exponentially with the word length n, it increases only linearly with the number q of states in S. In typical cases, the learning entity applies only a few consecutive operations to its environment between any pair of observations, but the number of possible states of the environment is very large. For $n \leq 15$, the constant of proportionality t_n in the linear function of q is quite small.

To demonstrate the usefulness of the new algorithm, we consider the well known problem of testing ping-pong communication protocols. In this problem, the opponent has a repertoire of invertible operations x_i which he can apply by himself, and a repertoire of operation words w_i which he can trick the other users to apply to his arguments. The first user applies an initial word w_0 to the message M, and the question is whether the opponent can recover M by inverting the effect of w_0. Dolev and Yao [DY83] were the first to analyse this problem, and described a $O(m^8)$ algorithm for solving it. This was later improved to an $O(m^3)$ algorithm by Dolev Even and Karp [DEK82]. However, the analysis of ping-pong protocols is just a special case of our learning problem, in which we have to decide whether w_0^{-1} belongs to the "knowledge closure" of all the x_i and w_i. We have thus proved:

Theorem 15 *Ping-pong protocols can be tested in almost linear time.*

There are many interesting research problems left open by our learning algorithm. For example:

1. The algorithm is very sensitive to erroneous equations. How can we make it more robust?
2. The algorithm can result in a partially reduced graph. How can we use the structure of this graph to conduct an intelligent search for solutions?
3. The states may not be fully observable. How can we augment the algorithm to handle partially specified states?

References

[A82] D. Angluin, Inference of Reversible Languages, J. ACM, July 1982.

[BCG81] E.R. Belekamp, J.H. Conway, and R.K. Guy, *Winning Ways*, Academic Press, London, 1981.

[DEK82] D. Dolev, S. Even, and R.M. Karp, On the Security of Ping-Pong Protocols, Information and Control, Vol. 55, 1982.

[DY83] D. Dolev and A.C. Yao, On the Security of Public-Key Protocols, IEEE Trans. on Information Theory, Vol IT-29, 1983.

[EG81] S. Even and O. Goldriech, The Minimum Length Generator Sequence Problem is *NP*-hard, J. Algorithms, 2, pp. 311–313, 1981.

[FHL80] M. Furst, J. Hopcroft and E. Luks, Polynomial Time Algorithms for Permutation Groups, in *Proceedings of the 21st IEEE Annual Symposium on Foundations of Computer Science*, 1980.

[J85] M.R. Jerrum, The Complexity of Finding Minimum Length Generator Sequences, TCS, April 1985.

[M64] N.S. Mendelsohn, An Algorithmic Solution for a Word Problem in Group Theory, Canad. J. Math, 16, 1964, pp. 509–516.

[SS81] R. Schroeppel and A. Shamir, A $T = O(2^{n/2})$, $S = O(2^{n/4})$ Algorithm for Certain *NP*-complete Problems, SIAM J. Computing, 1981.

[Sim70] C.C. Sims, Computational Methods in the Study of Permutation Groups, in *Computational problems in Abstract Algebra*, J. Leech, Ed., Pergamon Press, Elmsford, NY.

[Sin79] D. Singmaster, Notes on the "magic cube", Polytechnic of the South Bank, London SE1 OAA, 1979.

[TC36] J.A. Todd and H.S.M. Coxeter, Proc. Edinburgh Math Soc., (2), 5, 1936, pp. 26–36.