Graph Isomorphism is in SPP

V. Arvind and Piyush P Kurur
Institute of Mathematical Sciences, C.I.T Campus
Chennai 600113, India
email: {arvind,ppk}@imsc.ernet.in

Abstract

We show that Graph Isomorphism is in the complexity class SPP, and hence it is in $\oplus$P (in fact, it is in Mod$_k$P for each $k \geq 2$). We derive this result as a corollary of a more general result: we show that a *generic problem* FIND-GROUP has an FP$^{\mathrm{SPP}}$ algorithm.

This general result has other consequences: for example, it follows that the *hidden subgroup problem* for permutation groups, studied in the context of quantum algorithms, has an FP$^{\mathrm{SPP}}$ algorithm. Also, some other algorithmic problems over permutation groups known to be at least as hard as Graph Isomorphism (e.g. coset intersection) are in SPP, and thus in Mod$_k$P for each $k \geq 2$.

1 Introduction

The Graph Isomorphism problem —of testing if two graphs are isomorphic— is a well-studied algorithmic problem in the class NP. Formally, the decision problem GI can be defined as:

$$\mathrm{GI} = \{\langle X_1, X_2\rangle \mid X_1 \text{ and } X_2 \text{ are isomorphic graphs}\}.$$

Graph Isomorphism has attracted a lot of research because there is no known polynomial-time algorithm for it and on the other hand there is strong evidence that it is not NP-complete. In [6] it was shown that Graph Nonisomorphism is in AM implying that GI is in NP $\cap$ coAM. It follows that GI cannot be NP-complete unless the polynomial hierarchy collapses to Σ_2^p [6, 18]. Schöning, who introduced the notion of lowness in complexity theory, pointed out in [18] that GI is *low* for Σ_2^p. I.e. GI is powerless as oracle for Σ_2^p.

Subsequently, it was shown in [13] that GI is also *low* for the class PP (the language class corresponding to #P). This result is proven using the machinery of GapP functions introduced in the seminal paper of [7]. In [7] the languages classes SPP and LWPP are introduced as generalizations of Valiant's class UP. It is shown in [7] that UP $\subseteq$ SPP $\subseteq$ LWPP, and LWPP is low for PP.

SPP is perhaps the most important and natural of the counting classes. It is known that SPP is in and low for the classes $\oplus$P, C$_=$P, PP etc. SPP has a host of other nice properties as well (see [7] for details). For instance, SPP is characterized exactly as the class of languages low for GapP. In summary, SPP can be seen as the GapP analogue of UP. In [7] it is also shown that SPP is the *smallest* reasonable gap-definable class.

Coming back to [13], the result that GI is low for PP is shown in that paper by proving that GI is in LWPP. It is also shown in [13] that GA (testing if a given graph has a nontrivial graph automorphism) is in SPP. It is known that GA is polynomial-time reducible to GI, but the converse is open.

1.1 Summary of new results

In this paper, we show that GI is in the class SPP. This was left as an open question in [13] (also see [7]).

As a consequence it follows that GI is in and low for $\oplus$P (in fact, GI $\in$ Mod$_k$P for each $k \geq 2$), C$_=$P etc. Previously, only a special case of Graph Isomorphism, namely Tournament Isomorphism, was known to be in $\oplus$P.[1]

What we prove is a more general result: we show that a generic problem FIND-GROUP is in FP$^{\mathrm{SPP}}$ as a consequence of which GI and some other algorithmic problems on permutation groups that are not known to have polynomial-time algorithms also turn out to be in SPP. In particular, as another corollary, we show that the hidden subgroup problem (HSP) over permutation groups is in FP$^{\mathrm{SPP}}$. The hidden subgroup problem is of interest in the area of quantum algorithms.

2 Preliminaries and Notation

Let $\Sigma = \{0, 1\}$ be the finite alphabet. Let log denote logarithm to base 2. Let FP denote the class of polynomial-time computable functions and NP denotes all languages accepted by polynomial-time nondeterministic Turing machines.

Let $\mathbb{Z}$ denotes the set of integers. A function $f : \Sigma^* \to \mathbb{Z}$ is said to be *gap-definable* if there is an NP machine M such that, for each $x \in \Sigma^*$, $f(x)$ is the difference between the number of accepting paths and the number of rejecting paths of M on input x. Let GapP denote the class of gap-definable functions [7]. For each NP machine M let gap_M denote the GapP function defined by it. The language class PP is defined as follows: L is in PP if there is an $f \in$ GapP such that $x \in L$ if and only if $f(x) > 0$.

The language classes UP, SPP and LWPP are defined using GapP functions [7]. L is in UP if there is an NP machine M accepting L such that M has at most one accepting path on any input. L is in SPP if there is an NP machine M such that $x \in L$ implies that $gap_M(x) = 1$, and $x \notin L$ implies that $gap_M(x) = 0$. L is in LWPP if there are an NP machine M and $h \in$ FP such that $x \in L$ implies that $gap_M(x) = h(0^{|x|})$, and $x \notin L$ implies that $gap_M(x) = 0$. For $L \in$ SPP (or in LWPP) we say that the language L is *accepted* by the machine M. The containments UP $\subseteq$ SPP $\subseteq$ LWPP is shown in [7].

We say that f is in GapP^A, for oracle $A \subseteq \Sigma^*$, if there is an NP^A machine M^A such that, for each $x \in \Sigma^*$, $f(x)$ is the difference between the number of accepting paths and the number of rejecting paths of M^A on input x. For any oracle A, we can define the standard relativized classes UP^A, SPP^A, and LWPP^A and we can easily see the containments UP$^A \subseteq$ SPP$^A \subseteq$ LWPP^A for any oracle A.

We say that $A \subseteq \Sigma^*$ is *low* for PP if PP^A = PP. In [7] it is shown that every language in LWPP is low for PP.

Similarly, we say that $A \subseteq \Sigma^*$ is *low* for GapP if GapP^A = GapP. Again, it is shown in [7] that A is low for GapP if and only if $A \in$ SPP.

Let M be an oracle NP machine, let $A \in$ NP be accepted by an NP machine N. We say that M^A makes *UP-like queries* to A if on all inputs x, $M^A(x)$ makes *only* such queries y for which $N(y)$ has *at most* one accepting path. Effectively, it is like M having access to a UP oracle. We state a useful variant of a result from [13, 14].

Theorem 2.1 ([13]) *Let M be a nondeterministic polynomial-time oracle machine with oracle $A \in$ NP such that M^A makes UP-like queries to A then the function $h(x) = gap_{M^A}(x)$ is in* GapP.

Next, we recall an important property of the class SPP shown in [7].

Theorem 2.2 ([7]) *If L is in* SPP^A *for some oracle $A \in$* SPP *then $L \in$* SPP. *I.e.* SPP$^{\mathrm{SPP}}$ = SPP.

The following lemma, which is a straightforward consequence of Theorem 2.1 and of Theorem 2.2, is in a form useful for this paper.

Lemma 2.3

[1] Tournament Isomorphism in $\oplus$P follows because any tournament has an odd number of automorphisms. There are special cases of Graph Isomorphism, e.g. Graph Isomorphism for bounded-degree graphs or bounded genus graphs, that have polynomial-time algorithms.

- *Suppose L is in* SPP^A *accepted by the nondeterministic polynomial-time oracle machine* M^A *with oracle* $A \in \mathrm{NP}$ *(i.e. $x \in L$ implies that $gap_{M^A}(x) = 1$, and $x \notin L$ implies that $gap_{M^A}(x) = 0$), such that the machine M^A makes UP-like queries to A, then L is in* SPP.

- *Suppose a function $f : \Sigma^* \to \Sigma^*$ is in* FP^A *(i.e. f is computed by a polynomial-time oracle transducer M^A) where $A \in \mathrm{NP}$, such that the machine M^A makes UP-like queries to A, then f is in* $\mathrm{FP}^{\mathrm{SPP}}$.

2.1 Permutation group preliminaries

In general, $\mathrm{Sym}(\Omega)$ denotes the symmetric group on the *finite* set Ω. A *permutation group* on Ω is a subgroup of $\mathrm{Sym}(\Omega)$. For $|\Omega| = n$, in this paper we let $\Omega = [n]$ and identify $\mathrm{Sym}(\Omega)$ with the group S_n of all permutations on $[n] = \{1, 2 \ldots, n\}$.

We use letters $g, h, \ldots, \sigma, \tau, \pi, \ldots$ with subscripts and superscripts to denote elements of S_n and i, j and k for the elements of the set $\Omega = \{1, 2 \ldots, n\}$. Subgroups (and in general subsets) of $\mathrm{Sym}(\Omega)$ will be usually denoted by capital letters A, G, H etc. We use the following notation which is standard in permutation group theory [22, 15]. Given $g \in S_n$ and $i \in [n]$, we denote by i^g the image of i under permutation g. The composition $g_1 g_2$ of permutations $g_1, g_2 \in S_n$ is defined *left to right*: i.e. applying g_1 first and then g_2. More precisely, $i^{g_1 g_2} = (i^{g_1})^{g_2}$ for all $i \in [n]$. For subset $A \subseteq S_n$ and $x \in \Omega$ we use x^A to denote the set $\{x^g | g \in A\}$. In particular if A is a subgroup of S_n, x^A is the orbit of x under the action of A on Ω.

For a subset Δ of $[n]$, let G_Δ denote the subgroup of G that fixes each element of Δ. In particular, if $G \leq S_n$ then for each $i \in [n]$, we let $G^{(i)}$ denote the subgroup $\{g \in G \mid j^g = j \text{ for each } j \in [i]\}$. $G^{(i)}$ is called the *pointwise stabilizer* of $[i]$ in G.

The identity permutation is denoted by 1 (we use 1 to denote the identity of all groups) and the subgroup consisting of only 1 is denoted $\mathbf{1}$. The permutation group *generated* by a subset A of S_n is the smallest subgroup of S_n containing A and is denoted $\langle A \rangle$. We assume that subgroups of S_n are presented by generator sets. For a generator set $A \subseteq S_n$, each permutation $\psi \in A$ is a list of n ordered pairs $\langle i, j \rangle \in [n] \times [n]$.

For permutation groups G and H, the expression $H \leq G$ means that H is a subgroup of G (not necessarily a proper subgroup). For $\varphi \in G$ the subset $H\varphi = \{\pi\varphi : \pi \in H\}$ of G is a *right coset* of H in G. Two right cosets of H in G are either disjoint or identical. Thus, the right cosets of H in G form a partition of G written as $G = H\varphi_1 + H\varphi_2 + \ldots + H\varphi_k$. Each right coset of H has cardinality equal to $|H|$ and the set $\{\varphi_1, \varphi_2, \ldots, \varphi_k\}$ is a set of coset representatives of H in G.

As developed by Sims [20], pointwise stabilizers are fundamental in the design of algorithms for permutation group problems. The structure used is the chain of stabilizers subgroups in G given by: $\mathbf{1} = G^{(n)} \leq G^{(n-1)} \leq \ldots \leq G^{(1)} \leq G^{(0)} = G$. Let C_i be a complete set of right coset representatives of $G^{(i)}$ in $G^{(i-1)}, 1 \leq i \leq n$. Then $\bigcup_{i=1}^{n-1} C_i$ forms a generator set for G. Such a generator set is called a *strong generator set* for G [20, 9]. Any $g \in G$ has a unique factorization $g = g_1 g_2 \ldots g_n$, with $g_i \in C_i$.

We now recall two basic algorithmic results concerning permutation groups. Given as input the generator set S for a permutation group $G \leq S_n$, the following two basic algorithmic tasks can be implemented in time polynomial in n (see e.g. [20, 9] for these and other results and [15, 11] for a comprehensive treatment).

Theorem 2.4

1. *For each element $i \in [n]$ the orbit of i, defined as $\{i^g \mid g \in G\}$, can be computed in polynomial time.*

2. *The tower of subgroups $\mathbf{1} = G^{(n)} \leq G^{(n-1)} \leq \ldots \leq G^{(1)} \leq G$ can be computed in time polynomial in n. (I.e. the right coset representative sets C_i for the groups $G^{(i)}$ in $G^{(i-1)}, 1 \leq i \leq n$ can be computed in polynomial time giving a strong generator set for each $G^{(i)}$ including G).*

3 Computing the least element of a right coset

In this section we describe a simple polynomial-time algorithm that takes as input a permutation group $\langle A \rangle = G \leq S_n$ and a permutation $\sigma \in S_n$ and computes the lexicographically least element of the right coset

$G\sigma$ of G in S_n. Here, we use the standard lexicographic ordering of permutations in S_n given by the ordering of the set $[n] = \{1, 2, \ldots, n\}$. This algorithm is a crucial ingredient in the proof of the main theorem in the next section.

Theorem 3.1 *There is a polynomial-time algorithm that takes as input a permutation group $\langle A \rangle = G \leq S_n$ and a permutation $\sigma \in S_n$ and computes the lexicographically least element of the right coset $G\sigma$.*

Proof.
We describe the easy algorithm and then argue its correctness.

```
Input: $G \leq S_n$, $\sigma \in S_n$
Output: Lexicographically least element in $G\sigma$
Let $G^{(n)} \leq G^{(n-1)} \leq \ldots \leq G^{(1)} \leq G$ be the tower of subgroups of $G$ where, by Theorem 2.4, the generator set for each $G^{(i)}$ and the strong generator set for $G$ can be computed in polynomial time;
$\pi_0 = \sigma$;
for $i := 0$ to $n - 1$ do
    let $x := i + 1$;
    find the element $y$ in $x^{G^{(i)}}$ such that $y^{\pi_i}$ is minimum;
    { This can be done in polynomial time as the entire orbit $x^{G^{(i)}}$ of $x$ in $G^{(i)}$, which is a set of size at most $n - i$, can be computed in polynomial time by applying Theorem 2.4, and finding the minimum in the orbit takes linear time.};
    Let $g_i \in G^{(i)}$ be such that $x^{g_i} = y$;
    $\pi_{i+1} := g_i \pi_i$;
end
Result: $\pi_n$
```

Algorithm 1: **Lexicographically least in a Right Coset**

Since $\pi_0 = \sigma$ and $G^{(n-1)} = \{1\}$, it suffices to prove the following claim in order to show that the algorithm computes the lexicographically least element of $G\sigma$.

Claim 3.2 *For all $0 \leq i < n - 1$ the lexicographically least element of $G^{(i)}\pi_i$ is in $G^{(i+1)}\pi_{i+1}$.*

Proof of Claim. Let x^H denote the orbit of any point $x \in [n]$ under the action of $H \leq S_n$. By definition, $\pi_{i+1} = g_i \pi_i$, where g_i is in $G^{(i)}$ such that g_i maps $i+1$ to $y \in (i+1)^{G^{(i)}}$ and such that $y^{\pi_i} = x$ is the minimum element in $\{z^{\pi_i} \mid z \in (i+1)^{G^{(i)}}\}$. Since $G^{(i)}$ fixes each element in the set $[i]$ and since $g_i \in G^{(i)}$, we can see that for every $1 \leq k \leq i$, for each $g \in G^{(i)}$ and $h \in G^{(i+1)}$, we have $k^{h\pi_{i+1}} = k^{\pi_{i+1}} = k^{g_i \pi_i} = k^{\pi_i} = k^{g\pi_i}$. In particular if ρ is the lex-least element of $G^{(i)}\pi_i$, every element in $G^{(i+1)}\pi_{i+1}$ agrees with ρ on the first i elements.

Furthermore, for each $g \in G^{(i+1)}$ notice that $(i+1)^{g\pi_{i+1}} = (i+1)^{\pi_{i+1}} = (i+1)^{g_i \pi_i} = x$, where x is defined above. It is clear that $G^{(i+1)}\pi_{i+1}$ is precisely the subset of $G^{(i)}\pi_i$ each of whose elements maps $i+1$ to x. Together with the fact that $(i+1)^{\rho} = x$ (by the lex-least property of ρ), we get the desired conclusion.

By induction and the above Claim it follows that the lex-least element of $G\sigma = G^{(0)}\pi_0$ is in $G^{(n)}\pi_n = \{\pi_n\}$. Hence π_n is the desired lex-least element of $G\sigma$. ■

We can easily extend the above result to show the following.

Theorem 3.3 *There is a polynomial-time algorithm that takes as input a permutation group $\langle A \rangle = G \leq S_n$ and two permutations $\tau, \sigma \in S_n$, and computes the lexicographically least element of $\tau G \sigma$.*

Remark. The above theorem implies, in particular, that the lexicographically least element of a left coset τG can be computed in polynomial time.

4 Graph Isomorphism in SPP

We are ready to prove the main theorem of the paper. Recall that the Graph Isomorphism problem is the following decision problem: $\mathrm{GI} = \{(X_1, X_2) \mid X_1 \text{ and } X_2 \text{ are isomorphic}\}$. A related problem is AUTO which is a functional problem: given a graph X as input the problem is to output a strong generator set for $\mathrm{Aut}(X)$. It is well-known from the result of Mathon [16] (see e.g. [14]) that GI and AUTO are polynomial-time Turing equivalent.

Thus, in order to show that $\mathrm{GI} \in \mathrm{SPP}$ it suffices to show that $\mathrm{AUTO} \in \mathrm{FP}^{\mathrm{SPP}}$. In other words, it suffices to show that there is a deterministic polynomial-time Turing machine M with oracle $A \in \mathrm{SPP}$ that takes a graph X as input and outputs a strong generator set for $\mathrm{Aut}(X)$.

We observe here that the problem AUTO itself is one among a class of problems, each of which we will show is in $\mathrm{FP}^{\mathrm{SPP}}$ by giving such an algorithm for the following *generic* problem FIND-GROUP which we formally describe below:

To each instance $\langle x, 0^n \rangle$ of FIND-GROUP there is associated an unknown subgroup $G_x \leq S_n$ for which there is polynomial time membership test. More precisely, a polynomial-time function $\mathrm{MEMB}(x, g)$ is given, that takes x and $g \in S_n$ as input and evaluates to **true** if and only if $g \in G_x$. The FIND-GROUP problem is to compute a strong generator set for G_x given $\langle x, 0^n \rangle$ as input.

Notice that AUTO is an example of the generic FIND-GROUP problem, as checking whether $g \in S_n$ is an automorphism of a graph X on n nodes can be done in time polynomial in n.

Remark. The advantage of solving the generic problem FIND-GROUP is that it allows us to show at one stroke that several group-theoretic problems apart from GI are all in SPP. In particular, as a corollary to Theorem 4.1 we will show in the next section that the hidden subgroup problem (of interest in quantum computing) in the case of permutation groups is also in the class $\mathrm{FP}^{\mathrm{SPP}}$.

Theorem 4.1 *There is an* $\mathrm{FP}^{\mathrm{SPP}}$ *algorithm for the* FIND-GROUP *problem.*

Proof. Let $\langle x, 0^n \rangle$ be an input instance of FIND-GROUP. The goal is to compute a strong generator set for $G_x \leq S_n$ using MEMB as subroutine. As we have fixed the input, we will sometimes drop the subscript and write G instead of the group G_x.

Our goal is to design an $\mathrm{FP}^{\mathrm{SPP}}$ algorithm for finding the coset representatives of $G^{(i)}$ in $G^{(i-1)}$ for each i in the tower of subgroups $\mathbf{1} = G^{(n-1)} \leq G^{(n-2)} \leq \ldots \leq G^{(1)} \leq G^{(0)} = G$. Starting with $G^{(n-1)}$, which is trivial, the algorithm will build a strong generator set for $G^{(i)}$ in decreasing order of i until finally it computes a strong generator set for $G^{(0)} = G$. Thus, it suffices to describe how the algorithm will compute the coset representatives of $G^{(i)}$ in $G^{(i-1)}$ assuming that a strong generator set for $G^{(i)}$ is already computed. To this end we first define a language:

$L = \{\langle x, 0^n, S, i, j, \pi\rangle \mid \pi$ is a partial permutation that pointwise fixes $1, \ldots, i-1$ and maps i to $j, S \subseteq G_x$, and $\langle S\rangle$ pointwise fixes $[i]$, and $\exists\, g \in G_x^{(i-1)}$ such that $i^g = j$, lex−least$(\langle S\rangle g)$ and extends $\pi\}$.

Here, we use lex−least(Hg) for a group H to denote the lexicographically least permutation in the coset Hg.

Partial permutation π is part of instance $\langle x, 0^n, S, i, j, \pi\rangle$, as we will be using L as oracle to do a prefix search for the lexicographically least $g \in G^{(i-1)}$ such that $i^g = j$. We now describe an NP machine N that accepts L.

```
Description of Machine $N$ ;
Input: $\langle x, 0^n, S, i, j, \pi\rangle$
Verify using MEMB that $S \subseteq G^{(i)}$;
Guess $g \in S_n$;
if $g \in G^{(i-1)}$ and $i^g = j$ and $g$ extends $\pi$ and $g = $ lex−least$(\langle S\rangle g)$ then
    ACCEPT ;
end
else
    REJECT ;
end
```

Clearly, N is an NP machine that accepts L. The crucial point is that if $i^g = j$ then for every element $h \in \langle S\rangle g$, $i^h = j$. Also, using the algorithm in Theorem 3.1 the lexicographically least element of $\langle S\rangle g$ can be computed in polynomial time.

Claim 4.2 *If $\langle S\rangle = G^{(i)}$ then the number of accepting paths of N on input $\langle x, 0^n, S, i, j, \pi\rangle$ is either 0 or 1. In general, on input $\langle x, 0^n, S, i, j, \pi\rangle$, N has either 0 or $\frac{|G^{(i)}|}{|\langle S\rangle|}$.*

Proof of Claim. Suppose $\langle x, 0^n, S, i, j, \pi\rangle$ is in L and $\langle S\rangle = G^{(i)}$. Notice that if for some $g \in G^{(i-1)}$ we have $i^g = j$ (for $j > i$), then $\langle S\rangle g$ consists of all elements in $G^{(i-1)}$ that map i to j. Thus the only guessed element $g \in S_n$ by the machine N that leads to acceptance corresponds to the unique lexicographically least element of $\langle S\rangle g$.

On the other hand, if $\langle S\rangle$ is a proper subgroup of $G^{(i)}$ then we can easily see that $G^{(i)}g$ can be written as a disjoint union of $|G^{(i)}|/|\langle S\rangle|$ many right cosets of $\langle S\rangle$. Thus, in general N would have $|G^{(i)}|/|\langle S\rangle|$ many accepting paths if $\langle x, 0^n, S, i, j, \pi\rangle$ is in L.

We are now ready to describe an FP^L algorithm for FIND-GROUP. The algorithm is designed in a way that it will query L for some $\langle x, 0^n, S, i, j, \pi\rangle$ *only if* $\langle S\rangle = G^{(i)}$, thereby ensuring that it makes only UP-like queries to L. Finally, by Lemma 2.3 we can convert this algorithm to an $\mathrm{FP}^{\mathrm{SPP}}$ algorithm.

```
$C_i := \emptyset$ for every $0 \leq i \leq n-2$;
$\{C_i$ will finally be a complete set of coset representatives of $G^{(i+1)}$ in $G^{(i)}$. $\}$
$D_i := \emptyset$ for every $0 \leq i \leq n-2$;
$D_{n-1} = 1$;
$\{D_i$ will finally be a strong generator set for $G^{(i)}$ for each $i$. $\}$
for $i := n-1$ down to $1$ do
    $\{$ $D_i$ is already computed at the beginning of the $i^{th}$ iteration and at the end of the $i^{th}$ iteration we
    have $D_{i-1}$ $\}$
    Let $\pi : [i-1] \to [n]$ be the partial permutation that fixes all elements from 1 to $i-1$
    $\{$ in case $i = 1$ this is the everywhere undefined partial permutation $\}$
    for $j := i+1$ to $n$ do
        $\pi' := \pi[i := j]$;
        if $\langle x, 0^n, D_i, i, j, \pi' \rangle \in L$ then
            $\{$ There is an element in $G^{(i-1)}$ that maps $i$ to $j$. We will find it by a prefix search that extends
            the partial permutation $\pi'$
            $\}$
            for $k := i+1$ to $n$ do
                find the element $l$ not in the range of $\pi'$ such that
                $\langle x, 0^n, D_i, i, j, \pi'[k := l] \rangle \in L$;
                $\pi' := \pi'[k := l]$;
            end
            $\{$ At this point $\pi'$ will be a permutation in $S_n\}$
            $C_{i-1} := C_{i-1} \cup \{\pi'\}$;
        end
    end
    $\{$ At this point $C_{i-1}$ is a complete set of coset representatives of $G^{(i)}$ in $G^{(i-1)}\}$
    $D_{i-1} = D_i \cup C_{i-1}$
end
Result: $D_0$
```

Algorithm 2: FP^L algorithm CONSTRUCT($\langle x, 0^n \rangle$)

We claim that a call to the FP^L algorithm CONSTRUCT($\langle x, 0^n \rangle$) outputs a strong generator set D_0 for the group $G = G_x$. We show this by induction. Initially, $D_{n-1} = 1$ clearly generates $G^{(n-1)} = \mathbf{1}$. Suppose at the beginning of the i^{th} iteration it holds that D_i is a strong generator set for $G^{(i)}$. It suffices to show that at the end of the i^{th} iteration $D_{i-1} = D_i \cup C_{i-1}$ is a strong generator set for $G^{(i-1)}$. For each $j : i+1 \leq j \leq n$, the query $\langle x, 0^n, D_i, i, j, \pi' \rangle \in L$ checks if there is an element in $G^{(i-1)}$ that maps i to j. The subsequent prefix search with queries to L computes the lexicographically least element in $G^{(i-1)}$ that maps i to j. Furthermore, by Claim 4.2, as D_i generates $G^{(i)}$, all queries made to L are UP-like. Thus, at the end of the i^{th} iteration C_{i-1} is a complete set of coset representatives for $G^{(i)}$ in $G^{(i-1)}$ and hence D_{i-1} is a strong generator set for $G^{(i-1)}$. Thus at the end D_0 is a a strong generator set for G. Therefore, we have an FP^L algorithm problem for FIND-GROUP.

Finally, since the FP^L algorithm makes only UP-like queries to the NP oracle L, it follows from Lemma 2.3 that FIND-GROUP has an $\mathrm{FP}^{\mathrm{SPP}}$ algorithm. ■

Remark. We note that there is alternative way to conceive of an $\mathrm{FP}^{\mathrm{SPP}}$ algorithm for the FIND-GROUP problem: we can first design an $\mathrm{UPSV}^{\mathrm{SPP}}$ algorithm, where the prefix search that we do in CONSTRUCT($\langle x, 0^n \rangle$) is replaced by directly guessing a permutation in the right coset (consisting of elements that fix 1 to $i-1$ and map i to j) and rejecting along all paths on which we do not guess the lexicographically least element of the coset. Then, by a general prefix search argument we can see that $\mathrm{FP}^{\mathrm{SPP}}$ and $\mathrm{UPSV}^{\mathrm{SPP}}$ are the same and hence conclude that FIND-GROUP is in $\mathrm{FP}^{\mathrm{SPP}}$.

As we already noted, GI and AUTO are polynomial-time equivalent and AUTO, being an instance of FIND-GROUP has an $\mathrm{FP}^{\mathrm{SPP}}$ algorithm by Theorem 4.1. Since $\mathrm{SPP}^{\mathrm{SPP}} = \mathrm{SPP}$ and $\mathrm{SPP} \subseteq \mathrm{Mod}_k\mathrm{P}$ for each $k \geq 2$, the next corollary is an immediate consequence.

Corollary 4.3 *Graph Isomorphism is in* SPP *and hence in* $\mathrm{Mod}_k\mathrm{P}$ *for every* $k \geq 2$.

5 Hidden subgroup problem and other applications

We recall the general definition of the hidden subgroup problem.

Definition 5.1 *The* hidden subgroup problem HSP *has an input instance a finite group G (presented by a finite generator set) and we are given (in the form of an oracle) a function f from G to some finite set X such that f is constant and distinct on different right cosets of a hidden subgroup H of G. The problem is to determine a generator set for H.*

Many natural problems like Graph Isomorphism, integer factorization etc, can be cast as a special case of HSP. An efficient quantum algorithm for the general problem will result in efficient quantum algorithm for all these. Based on suitable generalizations of Shor's technique [19], the above problem has efficient quantum algorithms for the case when G is an abelian group (see e.g. [17] for an exposition). However, the status of HSP is open for general nonabelian groups, except for some special cases where it is settled (see, e.g. [10, 12]). In particular, even when we restrict attention to G being the permutation group S_n, it is not known if HSP has quantum polynomial time algorithms except in special cases.

Independently, it is shown by Fortnow and Rogers [8] that the class BQP of languages that have polynomial-time quantum algorithms is closely connected with language classes that are low for PP. In particular, it is shown in [8] that $\mathrm{BQP} \subseteq \mathrm{AWPP}$ where AWPP is a language class that generalizes both BPP and LWPP.

Theorem 5.2 [8] $\mathrm{BQP} \subseteq \mathrm{AWPP}$ *and hence* BQP *is low for* PP.

In this section we show as a corollary to Theorem 4.1 that there is an $\mathrm{FP}^{\mathrm{SPP}}$ algorithm for the HSP problem over permutation groups.

Theorem 5.3 *There is an* $\mathrm{FP}^{\mathrm{SPP}}$ *algorithm for the* HSP *problem over permutation groups, and hence* HSP *over permutation groups is low for* PP*,* GapP*,* $\oplus$P*,* $\mathrm{C}_=\mathrm{P}$ *etc.*

Proof Sketch. We are given (in the form of an oracle) a function f from S_n to a finite set X such that f is constant and distinct on different right cosets of a hidden subgroup H of S_n. The $\mathrm{FP}^{\mathrm{SPP}}$ will first compute $f(1)$ with one query to f. Now, notice that f gives a membership test for the unknown subgroup H, because a permutation $g \in S_n$ is in H if and only if $f(g) = f(1)$. Thus we essentially have a membership test as required for the FIND-GROUP problem of Theorem 4.1. The result now follows by invoking the algorithm described in the proof of Theorem 4.1. Lowness for PP also follows as SPP is low for PP. ■

5.1 Other applications

Using the $\mathrm{FP}^{\mathrm{SPP}}$ algorithm for the FIND-GROUP problem we can show that other algorithmic problems on permutation groups [15] which are not known to have polynomial-time algorithms are also in SPP. Among the different problems mentioned in [15] we pick the following two examples as most other problems are known to be polynomial time reducible to these.

The input instance to the CONJ-GROUP problem consists of three subgroups $\langle S\rangle = G$, $\langle S_1\rangle = H_1$, and $\langle S_2\rangle = H_2$ of S_n, and the problem is to determine if there is a $g \in G$ such that $gH_1g^{-1} = H_2$ (i.e. H_1 and H_2 are G-conjugate).

A closely related problem NORM has input instance two subgroups G and H of S_n, and the problem is to determine a generator set for the normalizer subgroup $N_G(H) = \{g \in G \mid gHg^{-1} = H\}$. Just as GI and AUTO are polynomial-time equivalent, it turns out that CONJ-GROUP and NORM are also polynomial-time equivalent [15].

Theorem 5.4 *The problem* NORM *is in* $\mathrm{FP}^{\mathrm{SPP}}$ *and the problem* CONJ-GROUP *is in* SPP.

Proof. We show that NORM is an example of the generic problem FIND-GROUP. The theorem will follow as a direct consequence of Theorem 4.1. It suffices to observe that given subgroups $\langle S\rangle = G$ and $\langle T\rangle = H$ of S_n, testing if $g \in N_G(H)$ (i.e. $gHg^{-1} = H$) can be carried out in polynomial time. More precisely, it is clear that $gHg^{-1} = H$ if and only if $gtg^{-1} \in H$ for every $t \in T$, which can be checked in polynomial time by Theorem 2.4. ■

As already mentioned, a consequence of the above theorem is that several other decision problems in permutation groups (e.g. coset intersection, double coset equality, set transporter) which are polynomial-time many-one reducible to CONJ-GROUP are also in SPP.

6 Conclusion

In this paper we have shown that Graph Isomorphism is in SPP. We have also shown that several other problems on permutation groups are in SPP. All these results are byproducts of the $\mathrm{FP}^{\mathrm{SPP}}$ algorithm for the problem FIND-GROUP. We would like to know if better upper bounds can be shown for the complexity of special cases of graph isomorphism especially tournament isomorphism. Specifically, is tournament isomorphism in UP? It is known that the automorphisms of a tournaments forms a solvable group and has odd order. Can this additional property be somehow exploited?

A related problem is Graph Canonization. Let f be a function from the family of finite graphs, $\mathcal{G}$, to itself. We say that f is a *canonization* if for every $X \in \mathcal{G}$, $f(X) \cong X$ and for every $X_1, X_2 \in \mathcal{G}$, $f(X_1) = f(X_2)$ iff $X_1 \cong X_2$. There is an $O(n^{logn})$ algorithm for Tournament Isomorphism by giving a canonization procedure for tournements [3]. The complexity of Graph Canonization is intriguing. The only known upper bound for the problem is $\mathrm{FP}^{\mathrm{NP}}$. It is known that Graph Isomorphism is polynomial-time reducible to Graph Canonization. Is the converse true, at least for tournaments? Is Graph Canonization for tournaments low for PP?

Babai and others, in a series of papers [5, 4, 2], developed a theory of black-box groups to study the complexity of group-theoretic problems in a general setting. The main results in [5, 4, 2] were to put these problems in NP ∩ coAM or AM ∩ coAM. However, lowness for PP has been addressed only for the case of *solvable* black-box groups in [1, 21], where many of these problems are shown to be in SPP. It is interesting to ask if our approach of showing membership in SPP via finding the lexicographically least element in a coset can be generalized to black-box groups. More precisely, what is the complexity of finding a canonical element in the right coset of a black-box group?

References

[1] V. Arvind and N. V. Vinodchandran. Solvable black-box group problems are low for PP. *Theoretical Computer Science*, 180(1–2):17–45, 1997.

[2] L. Babai. Bounded round interactive proofs in finite groups. *SIAM journal of Discrete Mathematics*, 5(1):88–111, February 1992.

[3] L. Babai and E. M. Luks. Canonical labeling of graphs. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pages 171–183, 1983.

[4] L. Babai and S. Moran. Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36:254–276, 1988.

[5] L. Babai and E. Szemeredi. On the complexity of matrix group problems I. In *Proceedings of the* 24^{th} *IEEE Foundations of Computer Science*, pages 229–240, 1984.

[6] R. Boppana, J. Hastad, and S. Zachos. Does co-NP have short interactive proofs. *Information Processing Letters*, 25:127–132, May 1987.

[7] S. A. Fenner, L. J. Fortnow, and S. A. Kurtz. Gap-definable counting classes. In *Structure in Complexity Theory Conference*, pages 30–42, 1991.

[8] L. J. Fortnow and J. D. Rogers. Complexity limitations on quantum computation. In *IEEE Conference on Computational Complexity*, pages 202–209, 1998.

[9] M. L. Furst, J. E. Hopcroft, and E. M. Luks. Polynomial-time algorithms for permutation groups. In *IEEE Symposium on Foundations of Computer Science*, pages 36–41, 1980.

[10] S. Hallgren, A. Russel, and A. Ta-Shma. Normal subgroup reconstruction and quantum computing using group representation. In *Proceedings of the* 32^{nd} *ACM Symposium on Theory of Computing*, pages 627–635, Portland, Oregon, 21-23 May 2000.

[11] C. M. Hoffmann. *Group-Theoretic Algorithms and Graph Isomorphism.* Springer, Berlin, Heidelberg, 1982.

[12] G. Ivanyos, F. Magniez, and M. Santha. Efficient quantum algorithms for some instances of the non-abelian hidden subgroup problem. In 13^{th} *ACM Symposium on Parallel Algorithms and Architectures*, pages 263–270, 2001.

[13] J. Köbler, U. Schöning, and J. Torán. Graph isomorphism is low for PP. *Computational Complexity*, 2(4):301–330, 1992.

[14] J. Köbler, U. Schöning, and J. Torán. *The Graph Isomorphism Problem: Its Structural Complexity.* Birkhauser, 1993.

[15] E. M. Luks. Permutation groups and polynomial time computations. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 11:139–175, 1993.

[16] R. Mathon. A note on graph isomorphism counting problem. *Information Processing Letters*, 8(3):131–132, 15 March 1979.

[17] M Mosca. *Quantum Computer algorithms.* PhD thesis, Oxford University, 1999.

[18] U. Schöning. Graph isomorphism is in the low hierarchy. In *Symposium on Theoretical Aspects of Computer Science*, pages 114–124, 1987.

[19] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[20] C. C. Sims. Computational methods in the study of permutation groups. *Computational problems in Abstract Algebra*, pages 169–183, 1970.

[21] N. V. Vinodchandran. Counting complexity of solvable black-box group problems. *SIAM Journal of Computing*, 33(4):852–869, 2004.

[22] H. Wielandt. *Finite Permutation Groups.* Academic Press, New York, 1964.