Counting Complexity of Solvable Black-box Group Problems

N. V. Vinodchandran*
Department of Computer Science and Engineering
University of Nebraska, Lincoln, NE 68588
email: `vinod@cse.unl.edu`

Abstract

We place many computational problems over solvable black-box groups in the counting complexity classes SPP or LWPP. The classes SPP and LWPP are considered as classes of low counting complexity. In particular, SPP is *low* (powerless as oracles) for all gap-definable counting classes (PP, $C_=P$, Mod_kP etc) and LWPP is is low for PP and $C_=P$. The results improve the upper bounds proved in [3] for these problems where the authors place these problems in randomized versions of SPP and LWPP. Because of the randomization, upper bounds in [3] only implied lowness for the class PP. The results in this paper favor the belief that these problems are unlikely to be complete for NP.

Key words: Computational Complexity Theory, Complexity Classes, Computational Group Theory.

1 Introduction

Computational problems in group theory have recently been of considerable interest in computational complexity theory. One of the main reasons for this is that the exact complexity of many of the computational problems that arise from group theory are not exactly characterized. For example, consider the basic problem of testing membership in matrix groups over finite fields represented by a set of generators. No efficient algorithm is known for this problem even for the simplest case when the group involved in the problem instance is cyclic. While no polynomial time algorithm exists for many problems over matrix groups, it is not known whether it is hard for NP. The associated group theoretic structure makes reductions to these problems difficult. Do these problems have complexity intermediate between P and NP? The results of this paper supports this belief. It is interesting to note that, while most of the natural problems that are not known to be in P, are proved to be NP-complete, candidates for natural problems that are of intermediate complexity are very few. The well-known Graph Isomorphism problem, the problem of testing whether two labeled graphs are isomorphic, is one such candidate.

In order to study the complexity of computational group theoretic problems in a generalized framework, Babai and Szemerédi [6] introduced the theory of *black-box* groups. Intuitively speaking, in this framework we have an infinite family of abstract groups. The elements of each group in the family are uniquely encoded as strings of uniform length. The group operations (product, inverse etc) are assumed to be provided by a group oracle and hence are easily computable. Black-box groups are subgroups of groups from such a family and they are presented by

*Most of the research done while the author was at the Institute of Mathematical Sciences, Chennai, India. Most of the results were presented in the 1997 FSTTCS conference [19].

generator sets. For example, matrix groups over finite fields and permutation groups presented by generator sets can be seen as examples of black-box groups. It is shown in [6] that many computational problems over black-box groups including Membership Testing are in NP.

A central problem considered in the papers of Babai and Szemerédi [6, 4] is Order Verification: given a black-box group G presented by a generator set and a positive integer n, verify that $|G| = n$. This problem is important because it turns out that several other problems reduce to Order Verification. In [4], using randomization to compute approximate lower bounds and sophisticated group-theoretic tools, it is shown that Order Verification for general black-box groups is in AM. As a consequence, it turns out that several problems for black-box groups are in NP $\cap$ co-AM. It follows that these problems cannot be NP-complete unless the polynomial hierarchy collapses to Σ_2^p [8, 18]. These results strongly indicate that these group-theoretic problems may be of intermediate complexity between P and NP.

In order to better understand the complexity status of these problems researchers have taken alternative approaches. Investigating the *counting complexity* of these problems is one such approach. In these investigations, the classes of interest are counting complexity classes SPP and LWPP, defined and studied by Fortnow et.al. [11]. These classes can be seen as structural generalizations of the class UP. It holds that P $\subseteq$ UP $\subseteq$ SPP $\subseteq$ LWPP. The classes SPP and LWPP are of low counting complexity in the sense they do not add power to other counting classes such as PP and $\mathrm{C_{=}P}$ when used as oracles. Membership of a problem in the class SPP or LWPP can be seen as an evidence that the problem is unlikely to be hard for the class NP. Firstly, intuitively we can say that problems that are in SPP or LWPP are of low counting complexity and hence it is unlikely that these problems are NP-hard. Secondly, the classes SPP and LWPP are defined by imposing strong restrictions on the computation tree of nondeterministic Turing machines accepting languages in them. It would be surprising if all problems in NP can be accepted by Turing machines with such restrictions.

Köbler et.al. [15] studied the counting complexity of Graph Isomorphism and certain group-theoretic problems such as Group Intersection, Group Factorization etc, over permutation groups. They showed that these problems are in the counting class LWPP. Very recently, this upper bound has been improved to SPP by Arvind and Kurur [2].

The study of the counting complexity of black-box group problems was initiated by Arvind and Vinodchandran in [3]. There the authors study the counting complexity of a number of problems over a large subclass of groups called *solvable* black-box groups. Since solvable groups are a generalization of abelian groups, the authors first consider problems over *abelian* groups. Using a constructive version of a fundamental theorem on the structure of finite abelian groups, they show that, over abelian groups, the problems Membership Testing, Order Verification, Group Isomorphism and Group Intersection are in the class SPP. They also show lowness of the problems Group Factorization, Coset Intersection and Double-Coset Membership over abelian groups, for PP and $\mathrm{C_{=}P}$.

In the case of solvable groups, using a randomized algorithm for computing the commutator series of a solvable black-box group from [5] and a constructive version of the fundamental theorem for abelian *factor* groups, in order to construct a special set of generators called *canonical generator set*, the authors of [3] show that all the above-mentioned problems are in *randomized versions* of SPP or LWPP. While these randomized versions of SPP and LWPP are low for the class PP, the lowness properties of these problems for classes such as $\mathrm{C_{=}P}$, $\mathrm{Mod}_k\mathrm{P}$, and other gap-definable counting classes, were not clear from [3].

The present paper overcomes this randomization bottle-neck. We show an upper bound of SPP (and in some cases LWPP) for a whole host of problems over solvable black-box groups. The main technical contribution of this paper is the design of a *deterministic* oracle Turing machine for converting a set of generators of a solvable group to a canonical generator set. This oracle algorithm can then be combined with the constructions in [3] to obtain the upper bounds for various problems. In particular, we show that, over solvable black-box groups, the problems Solvability Testing, Membership Testing, Subgroup Testing, Normality Testing, Order Verification, Nilpotence Testing, Group Isomorphism, and Group Intersection are in the class SPP. We also show that the problems Group Intersection, Group Factorization, Coset Intersection, and Double-Coset Membership over solvable groups are in the class LWPP. Thus, at least with respect to their counting complexity, problems over solvable groups may not be harder than their counter-parts over abelian groups.

The rest of the paper is organized as follows. In Section 2, we give complexity-theoretic and group-theoretic notations and definitions which are necessary for the paper. The definition of a canonical generator set for solvable groups and results relating to it are also given. Section 3 is devoted to the design of a deterministic oracle algorithm, CanonicalGenerator, for computing a canonical generator set for a solvable group, from an arbitrary generator set. In Section 4, we give definitions of the computational problems that we are interested in, and improve the upper bounds on the counting complexity of these problems using the algorithm given in Section 3. Finally, in Section 5, we conclude the paper with some open problems.

2 Preliminaries

In this section, we give relevant definitions from complexity theory and group theory. We also present complexity-theoretic as well as group-theoretic techniques that we use in this paper.

2.1 Complexity Theory

We refer the reader to [7] for standard complexity-theoretic definitions. Here we give only the minimal notations and definitions. We fix the finite alphabet $\Sigma = \{0, 1\}$. Let $A, B \subseteq \Sigma^*$ be two languages. The language $0A \cup 1B$ is denoted by $A \oplus B$. In [11], a uniform method for defining a number of counting classes using GapP, the class of gap-definable functions, is given. The classes that can be defined in this manner are called *gap-definable* classes. Refer to [11] for details. We give explicit definitions of only two classes, SPP and LWPP. Let GapP [11] denote the class of gap-definable functions. A language L is in LWPP if there are functions $f \in$ GapP and $h \in$ FP (h is positive) such that; $x \in L$ implies that $f(x) = h(|x|)$, and $x \notin L$ implies that $f(x) = 0$. A language L is in SPP if there is an $f \in$ GapP such that: $x \in L$ implies that $f(x) = 1$, and $x \notin L$ implies that $f(x) = 0$. It follows that SPP $\subseteq$ LWPP.

The concept of lowness is a well-studied notion in structural complexity theory. Intuitively, we say that the class $\mathcal{L}$ is low for class $\mathcal{C}$, if $\mathcal{L}$ is powerless as an oracle to a machine accepting languages in $\mathcal{C}$. Next we give the formal definition of lowness.

Definition 2.1 Let $\mathcal{C}$ be any complexity class which allows natural relativization. Then the class $\mathcal{L}$ is said to be *low* for $\mathcal{C}$, if for all $L \in \mathcal{L}$, $\mathcal{C}^L = \mathcal{C}$, where $\mathcal{C}^L$ denotes the complexity class that is obtained by relativizing $\mathcal{C}$ with respect to the language L.

The main interest in these classes is because of the following theorem proved in [11] regarding their complexity. This result indicates that SPP and LWPP are counting classes of low complexity. It is believed that those problems in SPP or LWPP cannot be complete for NP.

Theorem 2.2 ([11]) *The class* SPP *is low for all gap-definable counting classes. The class* LWPP *is low for* PP *and* $\mathrm{C_=P}$.

Let M be an oracle NP machine and let $A \in$ NP be accepted by an NP machine N and let $f \in$ FP. We say that M makes $f(n)$-*guarded* queries to A if, on length n inputs, M only asks queries y for which $N(y)$ has either 0 or $f(n)$ accepting paths, for each n. In this terminology, we state a weaker version of a theorem from [15] which gives a method for proving membership of languages in the classes SPP and LWPP.

Theorem 2.3 ([15]) *Let M be a deterministic polynomial-time oracle machine and let f be a polynomial-time computable function. If $A \in \mathrm{NP}$ such that M makes $f(n)$-guarded queries to A, then there is a polynomial q such that the function h, where $h(x) = f(|x|)^{q(|x|)}$ if M on input x accepts and $h(x) = 0$ if M on input x rejects, is in* GapP.

For proving our upper bounds we use the following corollary of the above theorem.

Corollary 2.4 *Let L be a language accepted by a deterministic polynomial-time oracle machine M with queries to a language A in* NP. *Let f be a polynomial-time computable function.*

1. *If M makes 1-guarded queries to A then $L \in \mathrm{SPP}$.*

2. *If M makes $f(n)$-guarded queries to A then $L \in \mathrm{LWPP}$.*

2.2 Group Theory

Here we give some notations and basic definitions from group theory. We also state some basic results. For further results and their proofs, please refer to standard textbooks [9, 12].

Let G be a group. A subset H of G is called a *subgroup* of G (denoted $H < G$ or $G > H$) if H is a group under the group operation of G. For a subset S of G, the smallest subgroup of G containing S is called the group *generated by* S and is denoted by $\langle S \rangle$. This group is the same as the set of all finite products of elements from S. A subset S of G is a generator set for G if G is identical to $\langle S \rangle$. A group G is *finite* if the cardinality of the set G is finite. In this paper, we are only interested in finite groups. Henceforth by a group we mean a finite group. The *order* of G is defined as the cardinality of the set G and is denoted by $|G|$. A group is said to be cyclic if it is generated by a single element. For an element $g \in G$, the *order* of g (denoted as $o(g)$) is the order of the cyclic subgroup generated by g. This is the same as the smallest positive integer k such that $g^k = e$, (g^k denotes the product of g, k times) where e is the identity of G. Two groups G and H are said to be isomorphic if there is a bijection (set-theoretic) ϕ from G to H such that for any $x, y \in G$, $\phi(xy) = \phi(x)\phi(y)$. Isomorphism preserves all structural properties of groups.

A fundamental theorem in finite group theory, due to Lagrange, states that if $H < G$ then $|H|$ divides $|G|$. This theorem has a large number of algorithmic applications. For example, it follows from Lagrange's theorem that any finite group G is generated by a set of group elements of cardinality bounded by $\log |G|$.

Let H be a subgroup of a group G. For $g \in G$ the set $\{hg \mid h \in H\}$, denoted by Hg, is called a *right coset* of H in G. Similarly, the set $gH = \{gh \mid h \in H\}$ is called a *left coset* of H in G. H is a *normal* subgroup of G if for all $g \in G$ it holds that $Hg = gH$. A fundamental result in the theory of groups is that if H is a normal subgroup of G, then the set of right cosets of H in G forms a group (called the *factor group* or a *quotient group* induced by H and denoted by G/H) under the binary operation $\cdot$ defined as $Hx \cdot Hy = Hxy$. The identity element of this group is the coset $He = H$. For a set $X \subseteq G$, the *normal closure* of X is the smallest normal subgroup containing X.

Solvable Groups

Here we give the definition of a *solvable* group and state some properties of them. Intuitively solvable groups can be thought of as a generalization of *abelian groups*. A group G is *abelian* if $\forall x, y \in G : xy = yx$; that is $xyx^{-1}y^{-1} = e$. In general, the element $xyx^{-1}y^{-1}$ is called the *commutator* of elements x and y in G. The subgroup of G generated by the set $\{xyx^{-1}y^{-1} \mid x, y \in G\}$ is called the *commutator subgroup* of G. We denote this subgroup by G'. Observe that if G is abelian, G' is the trivial group containing only the identity element. The commutator subgroup G' is actually a normal subgroup of G and the factor group G/G' is abelian. For a group G, the sequence $G = G_0 > G_1 > \ldots$ is called the commutator sequence, where each group G_i is the commutator subgroup of G_{i-1}. G is solvable if the commutator sequence terminates in the trivial subgroup $\langle e \rangle$ in finitely many steps. This intuitively means that any solvable group can be decomposed into a series of abelian factor groups. Therefore understanding the structure of a solvable group boils down to understanding the structure of the abelian factor groups involved. Next we state a structure theorem for abelian groups which essentially states that any abelian group can be uniquely decomposed into a set of cyclic groups. We need some more definitions to state the theorem.

Let p be a prime. A p-group is a finite group whose order is a power of p. Let G be finite group such that $|G| = p_1^{e_1} p_2^{e_2} \ldots p_r^{e_r}$. The existence of subgroups in G which are p-groups, is given by Sylow's theorem. That is, for each i there is a subgroup of G of order $p_i^{e_i}$. A subgroup of G of order $p_i^{e_i}$ is referred to as a *p_i-Sylow subgroup* of G.

Theorem 2.5 [9] *Let G be a finite abelian group such that $|G| = p_1^{e_1} p_2^{e_2} \ldots p_r^{e_r}$, where the p_i's are distinct primes. The group G can be expressed as the direct product of its Sylow subgroups $S(p_1), S(p_2), \ldots, S(p_r)$ where $|S(p_i)| = p_i^{e_i}$ for $1 \leq i \leq r$. Furthermore, for $1 \leq i \leq r$, each Sylow subgroup $S(p_i)$ can be uniquely expressed as the direct product of cyclic groups of orders $p_i^{e_{i1}}, p_i^{e_{i2}}, \ldots, p_i^{e_{is_i}}$ such that $e_{i1} \geq e_{i2} \geq \ldots \geq e_{is_i}$ and $\Sigma_{j=1}^{s_i} e_{ij} = e_i$. This decomposition of G is unique upto isomorphism.*

Solvable groups form a large subclass of all finite groups. In fact, a celebrated result due to Fiet and Thompson says that any finite group of odd order is solvable. Subgroups of solvable groups are solvable. It follows from Lagrange's theorem that, if G is solvable, then length of the commutator sequence is bounded by $\log |G|$. From a computational viewpoint this fact is very useful. It also holds that two solvable groups G and H are isomorphic if and only if the factor group H_{i-1}/H_i is isomorphic to G_{i-1}/G_i for all i where H_i (G_i) is the i^{th} element in the commutator series of H (respectively, G).

We will also consider nilpotent groups. Let H_1, H_2 be two subgroups of G. The *mutual commutator subgroup* of H_1 and H_2 is the group generated by the set $\{xyx^{-1}y^{-1} | x \in H_1, y \in H_2\}$ and is denoted by $[H_1, H_2]$. Note that using this notation the commutator subgroup G' of G is $[G, G]$. The chain of subgroups $G = L^0(G) > L^1(G) > \ldots$ where $L^i(G) = [G, L^{i-1}(G)]$ is called the *lower central series* of G. G is called *nilpotent* if the series terminates in the trivial subgroup $\{e\}$. A nilpotent group is also solvable.

Computational Problems over Black-box Groups

Now we define the notion of black-box groups.

Definition 2.6 A *group family* is a countable sequence $\mathcal{B} = \{B_m\}_{m \geq 1}$ of finite groups B_m, such that there are polynomials p and q satisfying the following conditions. For each $m \geq 1$, elements of B_m are uniquely encoded as strings in $\Sigma^{p(m)}$. The group operations (inverse, product and testing for identity) of B_m can be performed in time bounded by $q(m)$, for every $m \geq 1$. The order of B_m is computable in time bounded by $q(m)$, for each m. We refer to the groups B_m of a group family and their subgroups (presented by generators sets) as *black-box* groups. A class $\mathcal{C}$ of finite groups is said to be a *subclass* of $\mathcal{B}$ if every $G \in \mathcal{C}$ is a subgroup of some $B_m \in \mathcal{B}$.

Remark. The black-box groups we consider are somewhat a restricted version of the black-box groups introduced in [6]. The main difference is that, while we require identity testing to be in polynomial time, they only require it to be in *nondeterministic* polynomial time. This naturally places factor groups within the framework of black-box groups. Since in [6] the authors focus on proving many group-theoretic problems in NP, this generality does not cause additional complications. Since we are interested on certain "uniqueness" properties, we need this restriction.

Let S_n denote the permutation group on n elements. Then, $\{S_n\}_{n\geq 1}$ is a group family of all permutation groups S_n. As another example let $GL_n(q)$ denote the group of all $n \times n$ invertible matrices over the finite field F_q of size q. The collection $\mathrm{GL(q)} = \{GL_n(q)\}_{n\geq 1}$ is a group family. The class of all solvable subgroups, $\{G \mid G < GL_n(q)$ for some n and G is solvable $\}$ is a subclass of GL(q).

The following decision problems which we consider in this paper are well-studied in computational group theory [4, 6, 10, 15]. Let $\mathcal{B} = \{B_m\}_{m>0}$ be a group family.

Solvability Testing $\stackrel{\triangle}{=} \{(0^m, S) \mid \langle S\rangle < B_m$ and $\langle S\rangle$ is solvable $\}$.
Membership Testing $\stackrel{\triangle}{=} \{(0^m, S, g) \mid \langle S\rangle < B_m$ and $g \in \langle S\rangle\}$.
Subgroup Testing $\stackrel{\triangle}{=} \{(0^m, S_1, S_2) \mid \langle S_1\rangle, \langle S_2\rangle < B_m$ and $\langle S_1\rangle$ is a subgroup of $\langle S_2\rangle\}$.
Normality Testing $\stackrel{\triangle}{=} \{(0^m, S_1, S_2) \mid \langle S_1\rangle, \langle S_2\rangle < B_m$ and $\langle S_1\rangle$ is a normal subgroup of $\langle S_2\rangle\}$.
Nilpotence Testing $\stackrel{\triangle}{=} \{(0^m, S) \mid \langle S\rangle < B_m$ and $\langle S\rangle$ is nilpotent $\}$.
Order Verification $\stackrel{\triangle}{=} \{(0^m, S, n) \mid \langle S\rangle < B_m$ and $|\langle S\rangle| = n\}$.
Group Isomorphism $\stackrel{\triangle}{=} \{(0^m, S_1, S_2) \mid \langle S_1\rangle, \langle S_2\rangle < B_m$ and are isomorphic$\}$.
Group Intersection $\stackrel{\triangle}{=} \{(0^m, S_1, S_2) \mid \langle S_1\rangle, \langle S_2\rangle < B_m$ and $\langle S_1\rangle \cap \langle S_2\rangle \neq (e)\}$.
Group Factorization $\stackrel{\triangle}{=} \{(0^m, S_1, S_2, g) \mid \langle S_1\rangle, \langle S_2\rangle < B_m$ and $g \in \langle S_1\rangle\langle S_2\rangle\}$.
Coset Intersection $\stackrel{\triangle}{=} \{(0^m, S_1, S_2, g) \mid \langle S_1\rangle, \langle S_2\rangle < B_m$ and $\langle S_1\rangle g \cap \langle S_2\rangle \neq \emptyset\}$.
Double Coset Memb $\stackrel{\triangle}{=} \{(0^m, S_1, S_2, g, h) \mid \langle S_1\rangle, \langle S_2\rangle < B_m$ and $g \in \langle S_1\rangle h\langle S_2\rangle\}$.

The problems Group Factorization, Coset Intersection, and Double Coset Memb are very closely related. In particular, over permutation groups these problems are Turing equivalent to each other [14, 17]. Also Graph Isomorphism problem is a special case of Double Coset Memb over permutation groups [13].

Group-theoretic Techniques

We are interested in the counting complexity of the problems when the groups involved are solvable. Since solvable groups are a generalization of abelian groups, some remarks about the complexity of these problems over abelian black-box groups are in order. For proving tight upper bounds on the counting complexity of the above-mentioned problems over abelian groups in [3], the authors employ a constructive version of the structure theorem (Theorem 2.5). One of the immediate consequences of this theorem is the existence of a special generator set, called *independent generator set*, for any abelian group. To be precise, let G be a finite abelian group. An element $g \in G$, is said to be *independent* of a set $X \subseteq G$ if $\langle g\rangle \cap \langle X\rangle = \{e\}$. A generator set S of G is an *independent generator set* for G, if all $g \in S$ is independent of $S - \{g\}$. A set of generators of the cyclic groups involved forms an independent generator set for any abelian group. One of the very useful properties of independent generator sets is the following. Let S be an independent generator set for an abelian group G. Then for any $g \in G$, there exist *unique* indices l_h for $h \in S$; $l_h < o(h)$ such that $g = \prod_{h\in S} h^{l_h}$. Hence membership testing in G can be done in a 1-guarded manner if G is presented by an independent generator set. In [3], an algorithm for converting a given generator set to an independent generator set is given, which is used in proving the upper bounds on the counting complexity for problems over abelian black-box groups.

For proving the upper bounds for problems over solvable black-box groups in [3], the authors introduce a generalization of the notion of independent generator set, called *canonical generator set* for any class of finite groups. We now give the definition of a canonical generator set. The existence of canonical generators for the class of solvable groups is shown in [3].

Definition 2.7 Let $\mathcal{B} = \{B_m\}_{m>0}$ be any group family. Let $\mathcal{C}$ be a subclass of $\mathcal{B}$. The class of groups $\mathcal{C}$ has *canonical generator sets* if for every $G \in \mathcal{C}$, if $G < B_m$ then there is an *ordered* set $S = \{g_1, g_2, \ldots, g_s\} \subseteq G$ such that each $g \in G$ can be *uniquely* expressed as $g = g_1^{l_1} g_2^{l_2} \ldots g_s^{l_s}$, where $0 \leq l_i < o(g_i), 1 \leq i \leq s$. Furthermore, $s \leq q(m)$ for a polynomial q. S is called a *canonical generator set* for G.

Notice that the above definition is a generalization of the definition of an independent generator set in the sense that the uniqueness property of the indices is preserved. Define a language L as follows.

$L = \{(0^m, S, g) | S \subseteq B_m, g \in B_m; \forall h \in S\ \exists l_h; 0 \leq l_h < o(h) \text{ and } g = \prod_{h \in S} h^{l_h}\}$

The following proposition brings out the fact that the language L can act as a "pseudo Membership Testing" in the sense that if S is a canonical generator set, then $(0^m, S, g) \in L$ if and only if $g \in \langle S \rangle$. More importantly in this case, the NP machine M_u (given in the proposition) will have a *unique* accepting path for those instances inside L.

Proposition 2.8 *Let $\mathcal{B} = \{B_m\}_{m>0}$ be any group family. Then there exists an* NP *machine M_u witnessing $L \in$ NP with the following property. Let $\mathcal{C}$ be a subclass of $\mathcal{B}$ which has a canonical generator set and suppose the input to M_u satisfies the promise that S is a canonical generator set for $\langle S \rangle \in \mathcal{C}$. Then M_u on input $(0^m, S, g)$ will have a unique accepting path if $g \in \langle S \rangle$ and M_u will have no accepting path if $g \notin \langle S \rangle$. The behavior of M_u is unspecified if the input does not satisfy the promise.*

Proof. We know that that checking whether a number is prime or not is in P [1]. Using this, one can easily design an unambiguous nondeterministic polynomial-time transducer which computes the prime factorization of any number. Let M' be such a machine. Now, It is easy to see that the order of any $g \in B_m$ can be computed if the prime factorization of $|B_m|$ is given. So, M_u first computes $|B_m|$ in polynomial time. Then by simulating M', it computes the prime factorization of $|B_m|$ and computes the order $o(h)$ for all $h \in S$. Now, M_u guesses indices l_h such that $1 \leq l_h < o(h)$ and accepts if $g = \prod_{h \in S} h^{l_h}$ and rejects otherwise. From the definition of canonical generator set, it follows that M_u has the behavior as described in the proposition. ∎

Next lemma shows the existence of canonical generator sets for any solvable group.

Lemma 2.9 ([3], Lemma 3.4) *Let $\mathcal{B} = \{B_m\}_{m>0}$ be a group family such that $|B_m| \leq 2^{q(m)}$ for a polynomial q. Let $G < B_m$ be a finite solvable group and $G = G_0 > G_1 > \ldots > G_{k-1} > G_k = e$ be the commutator series of G. Let $T_i = \{h_{i1}, h_{i2}, \ldots, h_{is_i}\}$ be a set of distinct coset representatives corresponding to an independent set of generators for the abelian group $H_i = G_{i-1}/G_i$. Then for any i, $1 \leq i \leq k$, the ordered set [1] $S_i = \bigcup_{j=i}^{k} T_j$ forms a canonical generator set for the group G_i and $|S_i| \leq q(m)$. Thus the class of solvable groups from $\mathcal{B}$ has canonical generator sets.*

[1]The elements of the set $\bigcup_{j=i}^{k} T_j$ are ordered on increasing values of the index j, and lexicographically within each set T_j.

The basic steps implicitly involved in the upper bound proofs given in [3], for problems over solvable black-box groups, are the following.

1. A deterministic oracle algorithm (let us call it Canonize) is developed, which takes an arbitrary set of generators for the commutator series of a solvable black-box group as input, and converts it into a canonical generator set by making 1-guarded queries to a language in NP.

2. By carefully combining the algorithm Canonize with a randomized algorithm from [5] for computing generator sets of the commutator series for any solvable black-box group, a randomized oracle algorithm (let us call it RandCanonicalGenerator) for converting a generator set for any solvable group to a canonical generator set (which makes 1-guarded queries to an NP language) is given.

3. RandCanonicalGenerator is then easily modified to give membership of many computational problems over solvable groups in randomized counting classes which are low for PP.

In this paper, we avoid randomization involved in step 2. More precisely, by using the algorithm Canonize as subroutine, we give a *deterministic* oracle algorithm CanonicalGenerator (which makes 1-guarded queries to an NP language) for converting an arbitrary generator set to a canonical generator set, for any solvable black-box group G. This will immediately give improved upper bounds on the counting complexity of many problems over solvable groups. In the next section we present the algorithm CanonicalGenerator for converting an arbitrary generator set to a canonical generator set, for any solvable group.

Since we will be using the algorithm Canonize as subroutine in CanonicalGenerator, we describe the behavior of Canonize as a theorem. First we state a result from [3] (Corollary 4.9 from [3]) regarding abelian factor groups which is the building-block for Canonize.

Lemma 2.10 ([3], Corollary 4.9) *Let $\mathcal{B} = \{B_m\}_{m>0}$ be a group family. There is a deterministic oracle machine M_f that takes as input tuples $(0^m, X, Y)$, where $X, Y \subseteq B_m$ are finite sets, and a language $L_f \in \mathrm{NP}$ as oracle. Let $G = \langle X \rangle$ and $H = \langle Y \rangle$ and suppose the input $(0^m, X, Y)$ satisfies the properties.*

1. *Y is a canonical generator set for H.*

2. *H is a normal subgroup of G and the factor group G/H is abelian.*

Then the machine outputs a list of coset representatives corresponding to an independent generator set for G/H and also outputs $|G/H|$. Furthermore, it runs in time polynomial in input length, and it makes 1-guarded queries to the NP *oracle L_f. The behavior of the machine is not specified if the input does not satisfy the properties.*

The following theorem describes the behavior Canonize. The proof essentially follows from an iterative application of the above lemma. We omit the details.

Theorem 2.11 *Let $\mathcal{B} = \{B_m\}_{m\geq 0}$ be a group family. Then there is a deterministic oracle machine* Canonize *and a language $L' \in \mathrm{NP}$ such that* Canonize *takes $\langle 0^m, S_0, \ldots, S_k \rangle$, $S_i \subseteq B_m$ as input and L' as oracle. Suppose the input satisfies the promise that $\langle S_0 \rangle$ is solvable and for $0 \leq i \leq k$, S_i generates the i^{th} commutator subgroup of $\langle S_0 \rangle$. Then* Canonize *outputs canonical generator sets for $\langle S_i \rangle$ for $0 \leq i \leq k$. Moreover,* Canonize *runs in time polynomial in the length of the input and makes only 1-guarded queries to L'. The behavior of* Canonize *is unspecified if the input does not satisfy the promise.*

3 Computing a Canonical Generator Set

This section is devoted to the proof of the following theorem.

Theorem 3.1 *Let $\mathcal{B} = \{B_m\}_{m\geq 0}$ be a group family. Then there is a language $L_{ca} \in$ NP and a deterministic oracle machine* CanonicalGenerator *that takes $(0^m, S)$ as input and L_{ca} as oracle, and outputs canonical generator set for all the groups in the commutator series of $\langle S\rangle$ if $\langle S\rangle$ is solvable and outputs* NOT SOLVABLE *otherwise. Moreover,* CanonicalGenerator *runs in time polynomial in the length of the input and makes only 1-guarded queries to L_{ca}.*

Before going into the formal proof of the theorem, we give basic ideas behind the proof. Let S be a set of generators for a solvable group. Let $\langle S\rangle = G_0 > \ldots > G_i > \ldots > G_k = \{e\}$ be the commutator series of $\langle S\rangle$. We are interested in computing generator sets for all G_i. The following theorem based on normal closure provides a method for computing a generator set for the commutator subgroup of any group. Recall that the normal closure of $X \subseteq G$ is the smallest normal subgroup of G which contains X.

Theorem 3.2 *Let G be a finite group generated by the set S. Then, the commutator subgroup of G is the normal closure of the set $\{ghg^{-1}h^{-1} \mid g, h \in S\}$ in G.*

Proof. The proof uses standard group-theoretic argument. Let N be the normal closure of the set $X = \{ghg^{-1}h^{-1} \mid g, h \in S\}$ in G and G' be the commutator subgroup of G. Recall that $G' = \langle\{ghg^{-1}h^{-1} \mid g, h \in G\}\rangle$. It is a well known group-theoretic fact that G' is normal in G and G/G' is abelian. Moreover if H is a normal subgroup of G such that G/H is abelian, then $G' < H$. Now, from the definition of N and G' and the fact that G' is normal in G, it follows that $N < G'$. Hence to show that $N = G'$, it is enough to show that G/N is abelian. Firstly observe that G/N is generated by $\{gN \mid g \in S\}$. But for all $g, h \in S$, $(gN)(hN)(gN)^{-1}(hN)^{-1} = N$. Hence the theorem. ∎

The above theorem gives us the following easy polynomial-time oracle algorithm CommutatorSubgroup which takes $(0^m, S)$ as input and Membership Testing as oracle and computes a generator set for the commutator group of $\langle S\rangle$.

CommutatorSubgroup$(0^m, S)$

$X \leftarrow \{ghg^{-1}h^{-1} \mid g, h \in S\}$
while $\exists g \in S;\ x \in X$ such that $(0^m, X, gxg^{-1}) \notin$ Membership Testing
  do $X \leftarrow X \cup gxg^{-1}$
end-while
Output X

Claim 3.2.1 *On input $(0^m, S)$,* CommutatorSubgroup *runs in polynomial time and outputs a generator set for the commutator subgroup of $\langle S\rangle$.*

Proof. We can argue that CommutatorSubgroup outputs a generator set for the commutator subgroup of $\langle S\rangle$ as follows. From the Theorem 3.2, it follows that we need a generator set for the smallest normal subgroup of $\langle S\rangle$ containing X. Moreover it easily follows from the definition that, in order to check whether $H = \langle X\rangle$ is normal in $G = \langle S\rangle$ it is enough to check for all pairs $g \in S$ and $x \in X$ whether $gxg^{-1} \in H$. The algorithm adds new elements of the form gxg^{-1} to X until this condition is satisfied. Therefore the output of this algorithm generates a normal subgroup of $\langle S\rangle$ containing X. Since all the elements added should be contained in any normal subgroup of $\langle S\rangle$ containing X, the output generates the smallest such normal subgroup.

To argue that CommutatorSubgroup runs in polynomial time, let X_i be the set X at the beginning of the i^{th} iteration of the **while**-loop. If, after the i^{th} iteration, no new element is added to X_i, then X_i is output. Otherwise, if $X_{i+1} = X_i \cup \{g\}$, it follows from Lagrange's theorem that $|\langle X_{i+1}\rangle| \geq 2|\langle X_i \rangle|$. Hence the number of iterations of the **while**-loop is bounded by the polynomial $p(m)$. ∎

Since in the above algorithm, the queries to Membership Testing oracle may not be 1-guarded, a straightforward adaptation of the algorithm for computing a generator set for all elements in the commutator series seems difficult. Suppose we can make sure that whenever a query $y = \langle 0^m, X, g\rangle$ to Membership Testing is made, X is a canonical generator set for the solvable group $\langle X \rangle$. Then from Proposition 2.8 it follows that we can replace Membership Testing oracle with the NP language L and the query y will be 1-guarded. We ensure this promise, by constructing the commutator series in stages.

Let S_l^j denote the partial generator set for the l^{th} element in the commutator series of G_0 constructed at the beginning of *stage* j. At *stage 1* we have $S_0^1 = S$ and $S_i^1 = \{e\}$ for $1 \leq l \leq p(m)$, where p is the polynomial bounding the length of any element in the group family. Input to *Stage* j is the tuple $\langle i, S_i^j, \ldots, S_{p(m)}^j \rangle$ such that for $k > i$, S_k^j is a canonical generator set for the solvable group $\langle S_k^j \rangle$. At this stage, the only oracle queries made are with generator sets S_k^j for $k > i$ and in particular, will not involve S_i^j. This allows us to make only queries that are 1-guarded. At the end of the stage, we update each S_l^j to S_l^{j+1} such that S_l^{j+1} is still a subgroup of G_l, the l^{th} commutator subgroup of G_0. To keep the running time within polynomial bound, we make sure that after $p(m)$ stages, there exists k, such that the k^{th} partial commutator subgroup doubles in size. Then from Lagrange's theorem, it will follow that the commutator series will be generated after $p^3(m)$ stages. We now formally prove the theorem.

Proof. (*of Theorem 3.1*) We first give the formal description of the algorithm CanonicalGenerator and then prove the correctness.

CanonicalGenerator uses oracle algorithms CheckCommutator and Canonize as subroutines. CheckCommutator takes as input $(0^m, X, Y)$ such that $X, Y \subseteq B_m$ and checks whether $\langle Y \rangle$ contains the commutator subgroup of $\langle X \rangle$. This is done by first checking whether the commutators of all the elements in X are in $\langle Y \rangle$. If this is not the case, the algorithm returns a commutator not in $\langle Y \rangle$. Otherwise, it further checks whether $\langle Y \rangle$ is normal in $\langle X \rangle$. Notice that, to do this it is enough to verify that $\forall x \in X; y \in Y$, $xyx^{-1} \in \langle Y \rangle$. If this condition is false, the algorithm returns an element $xyx^{-1} \notin \langle Y \rangle$. If both the conditions are true, it follows from Theorem 3.2 that $\langle Y \rangle$ contains the commutator subgroup of $\langle X \rangle$.

CheckCommutator makes oracle queries to the language L (defined in the previous section) for testing membership in $\langle Y \rangle$. It should be noted that, for CheckCommutator to work as intended, Y should be a canonical generator set for the group $\langle Y \rangle$. Also the definition of normality requires $\langle Y \rangle < \langle X \rangle$. We will make sure that CanonicalGenerator makes calls to CheckCommutator with $(0^m, X, Y)$ as input, only when Y is a canonical generator set for the solvable group $\langle Y \rangle$ and $\langle Y \rangle < \langle X \rangle$. A formal description of the subroutine CheckCommutator is given below.

CheckCommutator($0^m, X, Y$)

```
if ∃x_1, x_2 ∈ X, such that (0^m, Y, x_1 x_2 x_1^{-1} x_2^{-1}) ∉ L
  then g ← x_1 x_2 x_1^{-1} x_2^{-1}
       Return g
  else if ∃x ∈ X, y ∈ Y such that (0^m, Y, x y x^{-1}) ∉ L
         then g ← x y x^{-1}
              Return g
         else g ← YES
              Return g
       end-if
end-if
```

The subroutine Canonize is the algorithm promised by Theorem 2.11 for computing a canonical generator set for a solvable black-box group G, given an arbitrary generator set for the commutator series of G. Canonize makes 1-guarded queries to the NP language L' if the input satisfies the promise given in Theorem 2.11. For $0 \leq k \leq l \leq p(m)$, let $[\text{Canonize}(S^j_k, \ldots, S^j_{p(m)})]_l$ denote the generator set produced by Canonize for the group $\langle S^j_l \rangle$.

Following is the description of the algorithm CanonicalGenerator. Define the language L_{ca} as; $L_{ca} = L' \oplus L$. Notice that the oracle access to L_{ca} is implicit in the description. That is, CanonicalGenerator queries L' through the subroutine Canonize and L through CheckCommutator.

CanonicalGenerator($0^m, S$)

```
S^1_0 ← S;  S^1_i ← {e} for 1 ≤ i ≤ p(m)
i ← 0
j ← 1
Stage j (Input to this stage is ⟨i, S^j_i, ..., S^j_{p(m)}⟩)
k ← i
g ← CheckCommutator(0^m, S^j_k, S^j_{k+1})
while g ≠ YES
   do S^j_{k+1} ← S^j_{k+1} ∪ {g}
      k ← k + 1
      if k = p(m)
        then Output NOT SOLVABLE
      end-if
      g ← CheckCommutator(0^m, S^j_k, S^j_{k+1})
end-while
if k = 0
  then Output [Canonize(S^j_0, S^j_1, ..., S^j_{p(m)})]_l for all 0 ≤ l ≤ p(m)
  else S^{j+1}_l ← S^j_l for 1 ≤ l ≤ (k − 1)
       S^{j+1}_l ← [Canonize(S^j_k, S^j_{k+1}, ..., S^j_{p(m)})]_l for k ≤ l ≤ p(m)
       i ← (k − 1)
        goto Stage j + 1
end-if
```

Now we are ready to prove the correctness of CanonicalGenerator. We first prove a series of claims, from which the correctness will follow easily.

Claim 3.2.2 *In the algorithm* CanonicalGenerator*, at any stage j, it holds that $\forall i; 1 \leq i < p(m)$, $\langle S_{i+1}^j \rangle < \langle S_i^j \rangle'$.*

Proof. We prove this by induction on the stages. For the base case, when $j = 0$, it is clear that the claim holds. Assume that it is true for $(j-1)^{\text{th}}$ stage. Now consider S_{i+1}^j and S_i^j. Depending on how the sets S_{i+1}^j and S_i^j are updated in *lines*-17,18 of CanonicalGenerator, we have the following cases.

Case 1. $S_i^j = S_i^{j-1}$; $S_{i+1}^j = S_{i+1}^{j-1}$. In this case, from the induction hypothesis, it is clear that $\langle S_{i+1}^j \rangle < \langle S_i^j \rangle'$.

Case 2. $S_i^j = S_i^{j-1} \cup \{g_i\}$; $S_{i+1}^j = S_{i+1}^{j-1}$. From the induction hypothesis, it follows that $\langle S_{i+1}^j \rangle = \langle S_{i+1}^{j-1} \rangle < \langle S_i^{j-1} \rangle' < \langle S_i^j \rangle'$.

Case 3. $S_i^j = S_i^{j-1}$; $S_{i+1}^j = S_{i+1}^{j-1} \cup \{g_{i+1}\}$. The element g_{i+1} is added to the set S_{i+1}^{j-1} at *line*-8 of the algorithm, where g_{i+1} is the element returned by the subroutine CheckCommutator. Suppose g_{i+1} is a commutator of the set $S_i^j = S_i^{j-1}$. Then $g_{i+1} = xyx^{-1}y^{-1}$ for some elements $x, y \in S_i^j$. From induction hypothesis we have that $\langle S_{i+1}^{j-1} \rangle < \langle S_i^{j-1} \rangle' = \langle S_i^j \rangle'$. Also $g_{i+1} \in \langle S_i^j \rangle'$ since g_{i+1} is a commutator of S_i^j. Therefore $\langle S_{i+1}^j \rangle = \langle S_{i+1}^{j-1} \cup \{g_{i+1}\} \rangle < \langle S_i^j \rangle'$. On the other hand, suppose g_{i+1} is of the form xyx^{-1} for some $x \in S_i^j = S_i^{j-1}$ and $y \in S_{i+1}^{j-1}$. We have, $\langle S_{i+1}^{j-1} \rangle < \langle S_i^{j-1} \rangle' = \langle S_i^j \rangle'$. But we know that $\langle S_i^j \rangle'$ is normal in $\langle S_i^j \rangle$. So, in particular $g_{i+1} \in \langle S_i^j \rangle'$. Therefore in this case also $\langle S_{i+1}^j \rangle = \langle S_{i+1}^{j-1} \cup \{g_{i+1}\} \rangle < \langle S_i^j \rangle'$.

Case 4 . $S_i^j = S_i^{j-1} \cup \{g_i\}$; $S_{i+1}^j = S_{i+1}^{j-1} \cup \{g_{i+1}\}$. From induction hypothesis, we have $\langle S_{i+1}^{j-1} \rangle < \langle S_i^{j-1} \rangle'$. It follows that $\langle S_{i+1}^{j-1} \rangle < \langle S_i^{j-1} \cup \{g_i\} \rangle' = \langle S_i^j \rangle'$. Now we are in the situation as in Case 3 and using an identical argument as in Case 3 we have $\langle S_{i+1}^j \rangle < \langle S_i^j \rangle'$.

Hence the claim. ∎

Claim 3.2.3 *In* CanonicalGenerator*, the input $\langle i, S_i^j, S_{i+1}^j, \ldots, S_{p(m)}^j \rangle$ to any stage j, is such that for all $i < t \leq p(m)$, S_t^j is a canonical generator for the solvable group $\langle S_t^j \rangle$.*

Proof. We prove this by induction. For $j = 1$, it is easily verified that the claim is true. Let us assume that the claim is true for the j^{th} stage. Let $\langle i, S_i^j, \ldots, S_{p(m)}^j \rangle$ be the input to the j^{th} stage. Suppose the **while**-loop is exited through *line*-14 after l iterations with the value of g='YES'. (If the loop is exited through *line*-11, then there are no more stages to be considered). Then the value of $k = i + l$ and for $t > k$, S_t^j is not updated inside the loop, and hence by induction hypothesis, it remains a canonical generator set for the solvable group $\langle S_t^j \rangle$. Since the value of $g =$ CheckCommutator$(0^m, S_k^j, S_{k+1}^j)$ is 'YES' we have that $\langle S_k^j \rangle' < \langle S_{k+1}^j \rangle$. From Claim 3.2.2 we have $\langle S_{k+1}^j \rangle < \langle S_k^j \rangle'$. Hence $\langle S_{k+1}^j \rangle = \langle S_k^j \rangle'$. It follows that $\langle S_k^j \rangle$ is solvable and S_t^j for $k \leq t \leq p(m)$ are generator sets for the commutator series of $\langle S_k^j \rangle$. Hence at *line*-18, Canonize will output a canonical generator set for each of the elements in the commutator series of $\langle S_k^j \rangle$. At *line*-19, i is updated to $k-1$, and the input to the $j+1^{\text{th}}$ stage is $\langle k-1, S_{k-1}^{j+1}, S_k^{j+1}, \ldots, S_{p(m)}^{j+1} \rangle$ where S_t^{j+1} is a canonical generator set for the solvable group $\langle S_t^{j+1} \rangle$ for $k \leq t \leq p(m)$. Hence the claim[2] ∎

Claim 3.2.4 *In the algorithm* CanonicalGenerator*, for any stage j, it holds that $\exists i$ such that $|\langle S_i^{j+p(m)} \rangle| \geq 2|\langle S_i^j \rangle|$ if stage $j + p(m)$ exists.*

[2]Technically, one should establish Claims 3.2.2 and 3.2.3 together. We separated them for clarity of presentation.

Proof. Notice that, if the algorithm at stage j enters the **while**-loop, then there $\exists i$ such that $S_i^{j+1} = S_i^j \cup g$ for a $g \notin \langle S_i^j \rangle$ and the claim follows from Lagrange's theorem. So, it is enough to show that the **while**-loop is entered at least once after every $p(m)$ stages, if such a stage exists. Suppose the stage j is entered with the value of $i = i'$. Note that in a stage where the algorithm does not enter the **while**-loop, the variable k remains at i. So if the algorithm never enters the **while**-loop in the next $p(m)$ stages, at stage $(j + p(m) + 1)$, the value of $k = i = i' - (p(m) + 1) < 0$, for all $i' \leq p(m)$. But when $k = 0$ (*line* 15) the algorithm stops. Therefore it is not possible that $k < 0$. Hence the claim. ∎

To complete the proof of the theorem, first we shall see that the algorithm CANONICAL-GENERATOR runs in time polynomial in the length of the input. Observe that it is enough to show that the number of stages executed by the algorithm is bounded by a polynomial, since the number of iterations of the **while**-loop in *lines* 7-14 is bounded by $p(m)$. Now, the claim is that the the number of stages executed by the algorithm is bounded by $2p^3(m)$. Firstly, notice that for any $H < B_m$, $|H| \leq 2^{p(m)}$. Hence for any j, $\prod_{i=1}^{p(m)} |\langle S_i^j \rangle| \leq 2^{p^2(m)}$. Suppose the claim is false. Now from Claim 3.2.4 it follows that, $\prod_{i=1}^{p(m)} |\langle S_i^{j+p(m)} \rangle| \geq 2 \prod_{i=1}^{p(m)} |\langle S_i^j \rangle|$. Hence $\prod_{i=1}^{p(m)} |\langle S_i^{2p^3(m)} \rangle| > 2^{p^2(m)}$, a contradiction.

Now we shall see that CANONICALGENERATOR makes only 1-guarded queries to L_{ca}, where $L_{ca} = L' \oplus L$. Let us first see that the queries to L through CHECKCOMMUTATOR are 1-guarded. It is enough to show that whenever CANONICALGENERATOR calls CHECKCOMMUTATOR with argument $(0^m, S_k^j, S_{k+1}^j)$ in *stage* j, S_{k+1}^j is a canonical generator set. But from Claim 3.2.3, the input $\langle i, S_i^j, S_{i+1}^j, \ldots, S_{p(m)}^j \rangle$ to any stage j, is such that for all $i < t \leq p(m)$, S_t^j is a canonical generator for the solvable group $\langle S_t^j \rangle$. In this stage CANONICALGENERATOR calls CHECKCOMMUTATOR only with argument $(0^m, S_k^j, S_{k+1}^j)$ for $k \geq i$. Since S_{k+1}^j for $k \geq i$ is a canonical generator set, the queries made to L will be 1-guarded.

To see that the queries to L' through CANONIZE are 1-guarded, notice that calls to CANONIZE are made outside the **while**-loop. This means that CHECKCOMMUTATOR with input $(0^m, S_k^j, S_{k+1}^j)$ returns YES. That is $\langle S_k^j \rangle' < \langle S_{k+1}^j \rangle$. Hence $\langle S_k^j \rangle' = \langle S_{k+1}^j \rangle$ from Claim 3.2.2. So it follows that calls to CANONIZE with argument $(S_i^j, S_{i+1}^j, \ldots, S_{p(m)}^j)$ will be such that S_l^j for $i \leq l \leq p(m)$ will generate the commutator series of S_i^j for all i. It follows from Theorem 2.11 that queries to L' will be 1-guarded.

Finally, we show that the above algorithm on input $(0^m, S)$, outputs a canonical generator set for the group $G = \langle S \rangle$ if G is solvable and outputs NOT SOLVABLE otherwise. Now, observe that if $H_1 < H_2$ are two finite groups, $H_1' < H_2'$. Hence it follows from Claim 3.2.2 that, $\langle S_i^j \rangle < G_i$ for any i at any stage j, where G_i is the i^{th} element in the commutator series of G. We know that after the execution of $2p^3(m)$ stages, the algorithm outputs either a set $X \subseteq B_m$ or NOT SOLVABLE. Suppose it outputs NOT SOLVABLE in stage j. This happens after the value of the variable inside the **while**-loop is assigned $p(m)$. From *line* 8 of the algorithm it follows that $S_{p(m)}^j \neq \{e\}$. But if G where solvable, then we know that $G_{p(m)} = \{e\}$ and since $\langle S_{p(m)}^j \rangle < G_{p(m)}$ from Claim 3.2.2, we have a contradiction.

Suppose the algorithm outputs a set $X \subseteq B_m$ at *line*-16 in stage j. Thus the value of the variable k is 0. Notice that, inside the **while**-loop, the value of k is only incremented. This implies that at stage j the **while**-loop is not entered (the value of i could not have become 0 at a previous stage). So input to stage j is $\langle 0, S_0^j, \ldots, S_{p(m)}^j \rangle$. From Claim 3.2.3, it follows that for all $1 \leq t \leq p(m)$, $\langle S_t^j \rangle$ is solvable and S_t^j is a canonical generator set for the group $\langle S_t^j \rangle$. From the value of g =YES and Claim 3.2.2, it follows that $\langle S_1^j \rangle' = \langle S_2^j \rangle$. Also, since $S_0^j = S$ for any stage j, it follows that S_i^j generates the i^{th} element in the commutator series of $\langle S \rangle = G$. Hence, from Theorem 2.11, it follows that $[\text{CANONIZE}(S_1^j, \ldots, S_{p(m)}^j)]_l$ is a canonical generator set for G_l. Hence the theorem. ∎

4 Applications to Computational Problems

In this section we will use the algorithm CanonicalGenerator to prove upper bounds on the complexity of computational problems over solvable black-box groups that we defined in Section 2.

Theorem 4.1 *Over any group family,* Solvability Testing *is in* SPP *and hence low for all gap-definable counting classes.*

Proof. We can easily modify CanonicalGenerator to get an oracle machine which makes only 1-guarded queries to an NP oracle to test whether the input group is solvable or not. Now applying Corollary 2.4 we get that Solvability Testing is in SPP. ∎

Remark. In [5], a co-RP algorithm is given for Solvability Testing. While this upper bound gives lowness for PP [16], it does not show lowness for other gap-definable counting classes.

In view of the above theorem and the fact that $\mathrm{P}^{\mathrm{SPP}} = \mathrm{SPP}$, for all the problems that we consider here, we assume without loss of generality that the groups encoded in the problem instances are solvable.

Theorem 4.2 *Over any group family,* Membership Testing *for the subclass of solvable groups is in* SPP *and hence low for all gap-definable counting classes.*

Proof. A direct application of Theorem 3.1, Proposition 2.8, and Corollary 2.4 gives that Membership Testing over solvable groups is in SPP. ∎

Theorem 4.3 *Over any group family,* Subgroup Testing *for the subclass of solvable groups is in* SPP *and hence low for all gap-definable counting classes.*

Proof. Let S_1 and S_2 be generator sets for groups G_1 and G_2. Then G_1 is a subgroup of G_2 if and only if $\forall g \in S_1$, $g \in \langle S_2 \rangle$. This testing can be done in $\mathrm{P}^{\mathrm{SPP}}$ since Membership Testing is in SPP. Since $\mathrm{P}^{\mathrm{SPP}} = \mathrm{SPP}$, it follows that Subgroup Testing is in SPP. ∎

Theorem 4.4 *Over any group family,* Normality Testing *for the subclass of solvable groups is in* SPP *and hence low for all gap-definable counting classes.*

Proof. Let S_1 and S_2 be generator sets for groups G_1 and G_2. In order to check whether G_2 is normal in G_1 it is enough to check for pairs $g_1 \in S_1$ and $g_2 \in S_2$ whether $g_1 g_2 g_1^{-1} \in G_2$. This can be done in $\mathrm{P}^{\mathrm{SPP}}$ since Membership Testing is in SPP. Since $\mathrm{P}^{\mathrm{SPP}} = \mathrm{SPP}$, it follows that Normality Testing is in SPP. ∎

We have seen that checking whether a group is solvable or not is in SPP. Using this result we can design an SPP algorithm for testing a group is nilpotent or not. We need the following generalization of Theorem 3.2. We omit the proof of this group-theoretic result.

Theorem 4.5 *Let G be a group and H_1 and H_2 are normal subgroups of G generated by S_1 and S_2. Then the mutual commutator subgroup $[H_1, H_2]$ is the normal closure of the set $\{g_1 g_2 g_1^{-1} g_2^{-1} | g_1 \in S_1, g_2 \in S_2\}$.*

Using this result we can design an oracle algorithm for computing a generator set for the mutual commutator subgroup for a given pair of groups. Following is the subroutine for computing a generator set for the mutual commutator subgroup. We will be interested in the case when the groups involved are solvable. This subroutine will use Membership Testing, Normality Testing, and Solvability Testing as oracles.

MutualCommutator($0^m, S, S_1, S_2$)
if $\langle S\rangle$ or $\langle S_1\rangle$ or $\langle S_2\rangle$ not Solvable
  then Output NA
if $\langle S_1\rangle$ is not Normal in $\langle S\rangle$ or $\langle S_2\rangle$ is not Normal in $\langle S\rangle$
  then Output NA
$X \leftarrow \{ghg^{-1}h^{-1} \mid g \in S_1, h \in S_2\}$
while $\exists g \in S;\ x \in X$ such that $(0^m, X, gxg^{-1}) \notin$ Membership Testing
  do $X \leftarrow X \cup gxg^{-1}$
end-while
Output X

Theorem 4.6 *Over any group family,* Nilpotence Testing *is in* SPP *and hence low for all gap-definable counting classes.*

Proof. We can use the subroutine MutualCommutator for designing the following $\mathrm{P}^{\mathrm{SPP}}$ algorithm for Nilpotence Testing as follows.

NilpotenceTester($0^m, S$)
1 **if** $\langle S\rangle$ is not Solvable
2   **then Output** NOT NILPOTENT
3 $L^0 \leftarrow S$
4 **for** $i = 1$ to $p(m)$
5   **do** $L^i \leftarrow$ MutualCommutator($0^m, S, S, L^{i-1}$)
6   **if** $\langle L^i\rangle = \langle L^{i-1}\rangle \neq \{e\}$
7   **then Output** NOT NILPOTENT
8 **end-for**
9 **Output** NILPOTENT

The above algorithm first tests whether the group is solvable. Since any nilpotent groups is also solvable, in the remaining computation we only need to worry about groups which are solvable. Hence we can use any of the problems that we have already shown to be in SPP as oracles. The algorithm essentially computes the lower central series of the group. If the lower central series is not terminating then for some i between 1 and $p(m)$ (where p is the polynomial bounding the size of B_m in the definition of the group family) the mutual commutator subgroup L^i and L^{i-1} must be equal and nontrivial for some i. This is tested in *line* 6. Note that testing whether two solvable groups are equal can be done with two queries to Subgroup Testing oracle which we have shown to be in SPP. Hence the overall algorithm is a $\mathrm{P}^{\mathrm{SPP}}$ algorithm which can be converted into an SPP algorithm. ■

Theorem 4.7 *Over any group family,* Order Verification *for the subclass of solvable groups is in* SPP *and hence low for all gap-definable counting classes.*

Proof. Let S be the generator for a solvable group G. Let the sequence $G = G_0 > G_1 > \ldots G_k = \{e\}$ be the commutator sequence of G. Then we have $|G| = \prod_{i=1}^{k} |\frac{G_{i-1}}{G_i}|$. Consider an oracle machine M_o which uses oracle L_o as follows. M_o first simulated CanonicalGenerator to construct a canonical generator set for G. Then it simulates the oracle algorithm M_f provided in Lemma 2.10 to compute $|\frac{G_{i-1}}{G_i}|$ for each i using the canonical generator for G_{i-1} and G_i. Finally, it computes $\prod_{i=1}^{k} |\frac{G_{i-1}}{G_i}|$. The oracle L_o is the disjoint union of the NP oracles used by CanonicalGenerator and M_f. CanonicalGenerator makes only 1-guarded queries. Also since inputs to M_f are canonical generator sets, M_f makes only 1-guarded queries. Therefore M_o makes only 1-guarded queries to L_o. Hence by Corollary 2.4, Order Verification for the subclass of solvable groups is in SPP. ∎

Theorem 4.8 *Over any group family,* Group Isomorphism *for the subclass of solvable groups is in* SPP *and hence low for all gap-definable counting classes.*

Proof. We know that two solvable groups G and H are isomorphic if and only if the factor group H_{i-1}/H_i is isomorphic to G_{i-1}/G_i for all i where H_i (G_i) is the i^{th} element in the commutator series of H (respectively, G). Isomorphism among abelian groups can be tested by computing the orders of the elements in the independent generators. Two abelian groups are isomorphic if the orders of the elements in their independent generators (arranged in some fixed order, say ascending) are the same. This suggests the following oracle algorithm M_{is} for testing isomorphism. M_{is} uses a language $L_{is} \in$ NP as oracle.

Let S_1 and S_2 be the generator sets for solvable groups G and H. M_{is} first computes canonical generator sets for both G and H using CanonicalGenerator. Then it simulates the oracle algorithm M_f provided in Lemma 2.10 to compute independent generators sets for the abelian factor groups H_{i-1}/H_i and G_{i-1}/G_i for all i. For each of these groups it computes the orders of all the elements in the independent generator set, say by using M_f. Then it compares whether the orders of the elements of independent generator of H_{i-1}/H_i is same as orders of the elements of independent generator of G_{i-1}/G_i for all i to decide whether G and H are isomorphic. The oracle L_{is} is the disjoint union of the NP oracles used by CanonicalGenerator and M_f. CanonicalGenerator makes only 1-guarded queries. Also since inputs to M_f are canonical generator sets, M_f makes only 1-guarded queries. Therefore M_{is} makes only 1-guarded queries to L_{is}. Hence by Corollary 2.4, Group Isomorphism for the subclass of solvable groups are in SPP. ∎

For proving upper bound for Group Intersection, we need a result from [3], which we state next as a lemma.

Lemma 4.9 ([3], Lemma 5.7) *Let $\mathcal{B} = \{B_m\}_{m>0}$ be a group family. There is a deterministic polynomial-time oracle machine M that takes $(0^m, X, Y, Z, K)$ as input. Let $G = \langle X \rangle$ and $H = \langle Y \rangle$. Suppose the input has the following properties.*

1. *G and H are solvable groups.*

2. *X and Y are canonical generator sets for G and H respectively.*

3. *Z is a canonical generator set for G' the commutator subgroup for G.*

4. *The number K is $|G' \cap H|$.*

Then the machine M outputs $|G \cap H|$. Furthermore, M makes $|B_m|$-guarded queries to an NP *language A and 1-guarded queries to another* NP *language B. The behavior of machine M is not specified for inputs that does not satisfy the above properties.*

Theorem 4.10 *Over any group family,* Group Intersection *for the subclass of solvable groups is in* LWPP *and hence low for the classes* PP *and* $\mathrm{C_{=}P}$*.*

Proof. Let S_1 and S_2 be the generator sets for solvable groups G and H. We will design an oracle algorithm M_{int} which first computes canonical generator sets for all the groups in the commutator series of both G and H, and then uses the machine M from the above result inductively to compute $G \cap H$ as follows. Let $G = G_0 > G_1 > \ldots > G_{k-1} > G_k = \langle e \rangle$ be the commutator series for G. To begin with, G_{k-1} is abelian and we have an independent generator set for it. Applying the algorithm M of Lemma 4.9 to the groups G_{k-1} and H, with $Z = \{e\}$ and $K = 1$, M_{int} can compute $|G_{k-1} \cap H|$. Now, M can be used inductively to compute $|G_{i-1} \cap H|$ given canonical generator sets for G_i and H as well as the number $|G_i \cap H|$, for each i. Since all the generator sets input to M are canonical generator sets, it is clear that M_{int} only makes $|B_m|$-guarded and 1-guarded queries. Also by standard techniques, 1-guarded queries can be made $|B_m|$-guarded. Now applying Corollary 2.4 we get that Group Intersection is in LWPP. ■

Finally we show that Group Factorization, Coset Intersection, Double Coset Memb for solvable groups are also in LWPP.

Theorem 4.11 *Over any group family, the problems* Group Factorization, Coset Intersection, Double Coset Memb *for the subclass of solvable groups are in* LWPP *and hence low for the classes* PP *and* $\mathrm{C_{=}P}$*.*

Proof. We give proof for Group Factorization. Proofs for other two problems are very similar and is omitted. Let $\mathcal{B} = \{B_m\}_{m \geq 1}$ be a group family with elements of B_m uniquely encoded as strings in $\Sigma^{p(m)}$ for a polynomial p. Consider the following NP machine R, which takes $\langle 0^m, S_1, S_2, g, K \rangle$ as input, where S_1 and S_2 are the generator sets for two groups, g is an element of B_m and K divides $|B_m|$.

R($0^m, S_1, S_2, g, K$)
1 Verify that K divides $|B_m|$;
2 Guess $w \in \Sigma^{p(m)}$;
3 Verify that $w \in \langle S_1 \rangle$ and $w^{-1}g \in \langle S_2 \rangle$;
4 Branch into $|B_m|/K$ paths and **accept** on each of them.

We are interested in the behavior of this machine only when the following conditions are satisfied: S_1 and S_2 are canonical generator sets for two solvable groups G and H, and $K = |G \cap H|$. *Line* 3 of this algorithm can be implemented by simulating M_u in Proposition 2.8.

Machine R accepts the input if and only if $(0^m, S_1, S_2, g)$ is a positive instance of Group Factorization. This is because $g \in GH$ if and only if there exists a w such that $w \in G$ and $w^{-1}g \in H$. Moreover, in this case R will have $\frac{|B_m|}{K} * |G \cap H|$ accepting paths since there will be exactly $|G \cap H|$ elements w satisfying $w \in G$ and $w^{-1}g \in H$. Now consider an oracle machine M_{gf} which on input $(0^m, S_1, S_2, g)$ as input, first computes canonical generator sets for G and H using CanonicalGenerator, then computes $|G \cap H|$ using the machine M_{int}, and then queries the language accepted by R the string $(0^m, S_1, S_2, g, |G \cap H|)$. It is clear that M_{gf} only makes $|B_m|$-guarded and 1-guarded queries. 1-guarded queries can be made $|B_m|$-guarded using standard techniques. Therefore, applying Corollary 2.4, we have that Group Factorization is in LWPP. ■

5 Conclusion and Open Problems

In this paper we investigated the complexity of many computational group-theoretic problems over solvable black-box groups. All the problems considered are computationally difficult: there is no known polynomial time algorithm for any of them. But, it is also not known whether they are hard for NP. We showed that these problems over solvable black-box groups are in the counting class SPP or LWPP. The SPP upper bound implies that they will not provide additional power as oracle to counting classes such as PP, $C_=P$ and Mod_kP. The upper bound of LWPP implies that problems will not provide additional power as oracle to counting classes such as PP and $C_=P$. These upper bounds support the belief that the problems considered are unlikely to be hard for NP. Another important aspect of the results is that they provide more examples of natural problems in SPP and LWPP which are not known to be in P.

There is one major open question that arises from this investigation: extend the above upper bound to problems defined over general black-box group. We believe that the problems we considered here are in SPP even over general black-box groups. The methods we used for solvable groups will not extend for the general case. We feel that new methods based on classification theorems about finite simple groups may be required for the proof of this. A more general question is to show membership of other natural problems in the counting complexity classes like SPP or LWPP.

Acknowledgments. I thank V. Arvind for discussions we had about the results in this paper and his comments. I would like to thank the referees for carefully reading the first version of this paper and giving very detailed comments and pointing out crucial omissions. Their input has helped me to greatly improve the presentation of the paper.

References

[1] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. Manuscript 2002. Available at `http://citeseer.nj.nec.com/agrawal02primes.html`

[2] V. Arvind and P. Kurur. Graph isomorphism is in SPP. *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, 743 –750, 2002.

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

[4] L. Babai. Bounded round interactive proofs in finite groups. *SIAM Journal of Discrete Mathematics,* 5: 88–111, 1992.

[5] L. Babai, G. Cooperman, L. Finkelstein, E. Luks, and Á. Seress. Fast Monte Carlo algorithms for permutation groups. In *Journal of Computer and System Sciences,* 50: 296–308, 1995.

[6] L. Babai and M. Szemerédi. On the complexity of matrix group problems I. *Proceedings of 25th IEEE Symposium on Foundations of Computer Science,* 229–240, 1984.

[7] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity – I & II.* Springer Verlag, Berlin Hiedelberg, 1988.

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

[9] W. Burnside. *Theory of Groups of Finite Order*, Dover Publications, INC, 1955.

[10] G. Cooperman and L. Finkelstein. Random algorithms for permutation groups. *CWI Quarterly*, 5 (2): 93–105, 1992.

[11] S. Fenner, L. Fortnow, and S. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences,* 48: 116–148, 1994.

[12] M. Hall. *The Theory of Groups.* Macmillan, New York, 1959.

[13] C. Hoffmann. *Group-Theoretic Algorithms and Graph Isomorphism.* LNCS Vol. 136, 1982.

[14] C. Hoffmann. Subcomplete generalizations of graph isomorphism. *Journal of Computer and System Sciences,* 25: 332–359, 1982.

[15] J. Köbler, U. Schöning, and J. Torán. Graph isomorphism is low for PP. *Journal of Computational Complexity,* 2: 301–310, 1992.

[16] J. Köbler, U. Schöning, S. Toda, and J. Toran. Turing machines with few accepting computations and low sets for PP. *Journal of Computer and System Sciences*, 44(2): 272–286, 1992.

[17] E. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences,* 25: 42–65, 1982.

[18] U. Schöning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences,* 37: 312–323, 1988.

[19] N. V. Vinodchandran. Improved lowness results for solvable black-box group problems. *Proceedings of the 17th International Conference on the Foundations of Software Technology and Theoretical Computer Science*, LNCS, Vol 1346, 220–234, 1997.