# Chapter 1

# Introduction

A central aim of computational complexity theory is to classify computational problems according to the resources (usually time and space) required for solving them on a given model of computation. Most computational problems arising in practice fall naturally into different *complexity classes*, depending on the resource bounds of interest. Using the fundamental concept of resource-bounded reductions, it is possible to meaningfully compare the relative difficulty of two given problems within a complexity class. This gives rise to the notion of *complete problems* for each complexity class: complete problems are the hardest problems in a complexity class. The theory of NP-completeness illustrates the usefulness of these notions. Natural problems in the class NP, which are computationally intractable in practice, turn out to be NP-complete. Thus, within the complexity class NP, we have the subclass P of problems that can be feasibly solved (i.e. in polynomial time) and the subclass of NP-complete problems. A question that arises is whether there could be problems of "intermediate" difficulty. Might it not be the case that NP consists solely of P and NP-complete problems? Assuming P$\neq$NP, it was shown by Ladner [Lad75] that there are many problems in NP that are neither in P nor NP-complete.

In spite of the fact that there are a large number of problems of intermediate complexity in NP as shown by Ladner's theorem, there are only a few *natural* candidates for problems of intermediate complexity (in contrast, the number of natural

NP-complete problems abound, see [GJ78]). In fact, to date, none of them are provably of intermediate complexity even if we assume P$\neq$NP. A well-known example is the problem of testing whether two graphs are isomorphic (in short, GI). Researchers believe that GI is not in P since there is no subexponential time algorithm known for this problem. On the other hand, it is not known whether GI is NP-complete. The theory of reductions and completeness does not give us any further insight into the computational complexity of problems such as GI.

A useful tool that provides some understanding about problems of intermediate complexity like GI is the notion of *lowness* for complexity classes introduced by Schöning [Sch83]. To make this notion precise, let $\mathcal{C}$ be any relativizable complexity class. A language $A$ is said to be *low* for $\mathcal{C}$ if $\mathcal{C}^A = \mathcal{C}$, where $\mathcal{C}^A$ denotes the relativized version of $\mathcal{C}$ with $A$ as oracle. Lowness of $A$ for $\mathcal{C}$ intuitively means that $A$ is powerless as an oracle to $\mathcal{C}$. It is easy to see that any problem in NP which is low for some level of the polynomial-time hierarchy (PH) is not NP-complete unless PH collapses. Thus, lowness of a problem for some level of PH is an evidence that the problem is unlikely to be complete for NP. It is shown in [Sch88, BHZ87] that GI is low for $\Sigma_2^p$, the second level of PH. Since the introduction of this notion the study of lowness of problems to various complexity classes has been of much interest (see [Köb95] for a survey on lowness).

## 1.1 Counting complexity classes and lowness

Among various complexity classes, the study of *counting classes* has received considerable attention. In general, these are the language classes related to the function class #P. Valiant [Val79] introduced the class #P as a class consisting of functions computing the number of accepting paths of polynomial-time nondeterministic Turing machines and showed that computing the permanent of a 0-1 matrix is complete for #P. The study of counting classes has been a major research area since this seminal result. Here we briefly survey some important counting classes. Some of these classes are defined later in the thesis. See the survey article [For97] and references therein for more details.

Among counting complexity classes, considerable research has gone into understanding the structure of the class PP (Probabilistic polynomial-time). This class was originally defined by J. Gill [Gil77] and independently by J. Simon [Sim75]. The class PP is very closely related to the class #P. Indeed, it is easy to show that the closure of PP and #P under polynomial-time Turing reductions coincide.

PP is computationally a hard class; the class NP is contained in PP. The hardness of PP was further established by a celebrated result due to S. Toda [Tod91]. He showed that the entire polynomial-time hierarchy is contained in $\mathrm{P}^{\mathrm{PP}}$. PP also enjoys many nice closure properties. It is closed under complementation [Gil77]. The question posed by Gill in his seminal work [Gil77], whether PP is closed under intersection (or union), was settled in the affirmative by Beigel et. al. in [BRS95]. The techniques used in [BRS95] were extended by Fortnow et. al. [FR96] to show that PP is also closed under polynomial-time truth-table reductions.

The notion of lowness, originally defined in relation to the polynomial-time hierarchy [Sch83], was first studied for the class PP by Torán in [Tor88]. He gave a sufficient condition for languages to be low for PP. More precisely, he showed that languages in CH [Wag86] (Counting hierarchy; a hierarchy of classes over PP) which are #P-rankable are low for PP. Since then, many researchers have studied the structure of low sets for PP. It was shown in [KSTT92] that the class Few (introduced in [CH90] as a generalization of FewP [All86]) is low for PP. In [KSTT92] the authors also show that the probabilistic class BPP is also low for PP. Since these lowness proofs relativize, it is easy to get more and more complex sets that are low for PP. Another interesting lowness result shown in [KSTT92] is that the sparse sets in NP are low for PP.

Other counting classes that have been studied include classes $\mathrm{C}_=\mathrm{P}$ (defined in [Wag86]) and $\mathrm{Mod}_k\mathrm{P}$ for $k \geq 2$ ([PZ83, BG92, Her90, CH90]). These are also computationally hard classes; for example, from the definition of $C_=P$, it follows that co-NP $\subseteq C_=P$. Also, as an intermediate step in the proof of Toda's theorem it is shown that PH $\subseteq$ BP. $\oplus$ P (the class obtained by applying the BP operator to $\oplus$P) [Tod91]. In general, it holds that PH $\subseteq$ BP.Mod$_k$P [TO92].

The class UP introduced by Valiant [Val76] is another important complexity class. UP consists of those languages in NP accepted by nondeterministic polynomial-time machines having at most one accepting path. Valiant defined UP for studying the relative complexity of checking and evaluating. This class later found applications in the areas of one-way functions and cryptography; for example, in [GS84] it is shown that P is different from UP if and only if one-way functions exist. More recently, it is shown in [FK92] that the problem of deciding primality is in UP $\cap$ co-UP. UP is also low for many counting classes like PP, $C_=P$ and Mod$_k$P for $k \geq 2$.

The class GapP, studied by Fenner et. al. in [FFK94], is an important function complexity class. The main motivation for defining this class is from the observation that the class #P cannot take negative values. This led to the introduction of the class GapP as the closure of #P under subtraction. The class GapP satisfies many algebraic closure properties. Specifically, it is closed under exponential summation and polynomial multiplication. Using the notion of gap-definability, Fenner et. al. have given a uniform method for defining different counting complexity classes (gap-definable counting classes). These new definitions have proved to be very convenient to work with.

The notion of gap-definability also gives rise to some new counting complexity classes. Of special interest to us is the class SPP [FFK94]. (This class is also independently studied in [OH93] under the name XP and in [Gup95] under the name $\mathcal{Z}$UP). The class SPP can be seen as the *gap* analogue of the class UP. More precisely, SPP consists of languages that are accepted by polynomial-time nondeterministic Turing machines such that, for inputs in the language the difference in the number of accepting and rejecting paths of these machines is exactly one, and for inputs not in the language this difference is zero.

The class SPP is large enough to include FewP and it is contained in the classes PP, $C_=P$ and $\mathrm{Mod}_kP$ for $k \geq 2$. This class is interesting mainly because of its lowness properties. It is shown in [FFK94] that SPP is exactly the class of languages that are low for the function class GapP [FFK94]. From this result it easily follows that SPP is low for PP, $C_=P$ and $\mathrm{Mod}_kP$ for $k \geq 2$ and even SPP itself. In particular, it is closed under polynomial-time Turing reductions. Hence, intuitively, SPP is a class of "low counting complexity".

Another complexity class which is of interest is the class LWPP [FFK94]. LWPP is a generalization of SPP. From the definitions, it follows that SPP $\subseteq$ LWPP. LWPP also enjoys many of the lowness properties of SPP. For example, LWPP is low for PP and $C_=P$ and is closed under polynomial-time Turing reductions.

Though SPP and LWPP are structurally defined classes, it is shown that some natural, computationally hard problems fall in these classes. In [KST92], it is shown that GI is in LWPP. They also show that Graph Automorphism problem (GA; the problem of deciding whether a graph has a nontrivial automorphism or not) is in SPP.

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, as mentioned before, 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 will be surprising that all the problems in NP can be accepted by Turing machines with such restrictions. In view of these explanations, membership of GI in LWPP shown in [KST92] gives additional evidence that GI is unlikely to be complete for NP.

## 1.2 Contribution of the Dissertation

The main contribution of this dissertation is in proving upper bounds on the counting complexity of some computational problems that arise from group theory. Broadly, it consists of two parts. The first part concentrates on studying the complexity of three basic, computationally hard group-theoretic problems over black-box groups. These problems are Membership Testing, Order Verification and Isomorphism Testing. We investigate their counting complexity over a class of groups called *solvable* groups. Solvable groups form a large subclass of finite groups. Indeed, a celebrated theorem in group theory due to Fiet and Thompson says that all finite groups of odd order are solvable. It is shown that all these problems over solvable black-box groups are in the complexity class SPP. This result provides additional evidence that these problems are unlikely to be hard for NP. Another aspect of this result is that it adds to the list of natural problems that are in SPP but not known to be in P. Graph Automorphism and permutation Group Intersection are among the few members that were already known to be in SPP but not known to be in P [KST92].

The main focus of the second part of the thesis is to investigate the complexity of a computational problem over finite groups which can be thought of as an "inverse problem" of Membership Testing. The problem can be informally stated as follows. Given an *unknown* permutation group $G$ and an oracle to test membership in $G$; compute a generator set for the group. We observe that this problem is very closely related to the problem of learning representation classes in computational learning theory.

Motivated by this, in the second part of the thesis, we focus on studying the complexity of various representation classes with respect to their learnability. Of particular interest to us is the learnability of some group theoretic and linear algebraic concept classes. We compare the complexity of learning these algebraic concept classes to that of learning boolean functions represented in conjunctive normal form. One of the learning models that we are interested in is the exact learning model proposed by Angluin [Ang88]. It turns out that Angluin's model of exact learning is inadequate for distinguishing the complexity of learning various classes of interest to us. In order to do a finer classification of the complexity of exact learning, we propose a refinement of Angluin's model called the *teaching assistant* model of exact learning. The main ingredient of this model is a new agent namely the teaching assistant. A teaching assistant acts as an interface between the teacher and the learner. In order to classify the complexity of learning, we define the notion of teaching assistant classes. These classes are defined in exact analogy with known complexity classes. We show that the complexity of exactly learning algebraic concepts like permutation groups or linear spaces is different from the complexity of learning bounded CNF formulas in the teaching assistant model. More specifically, as one of our main results, we show that while permutation groups (linear spaces over fixed finite fields) can be learned using a teaching assistant in the assistant class analogous to LWPP (respectively, SPP), it is unlikely that the class 3-CNF can be learned using an LWPP or SPP assistant (unless NP $\subseteq$ LWPP or NP $\subseteq$ SPP). We also investigate the power of various assistant classes and exhibit representation classes which separate these assistant classes.

Although the two parts of the thesis are addressing apparently two different issues, there are some underlying connections between the two parts. Firstly, in both the parts, we are addressing the issue of analyzing the complexity of some computational problems over finite groups, although the exact nature of the problems differ. Secondly and importantly, while in the first part the complexity class of interest to us is SPP, in the second part, our interest is in the teaching assistant classes related to low complexity counting classes SPP and LWPP. Hence, the learning algorithms we design with LWPP and SPP teaching assistants are built on the complexity-theoretic ideas used in showing the upper bound results in the first part of the thesis.

In the next two sections, we briefly explain the technical contents of the thesis more formally. Most of the group-theoretic and complexity-theoretic notations that we use are given in the next chapter.

## 1.3 Solvable Black-box Group Problems

In this section, we explain the contents of the first part of the thesis in more detail. This part is devoted to the investigation of the counting complexity of three basic computational problems over solvable black-box groups. Before giving the exact definitions of problems of interest to us, we explain the framework of black-box groups. Intuitively, 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 easily computable. Black-box groups are subgroups of groups from such a family and they are presented by generator sets.

**Remark.** We would like to note here that the black-box groups we define above are a slightly restricted version of black-box groups introduced in [BS84]. The definition in [BS84] is technically more general so as to incorporate factor groups.

**Definition 1.3.1** 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 generator sets) as *black-box* groups. A class $\mathcal{C}$ of black-box groups is said to be a *subclass* of $\mathcal{B}$ if every $G \in \mathcal{C}$ is a subgroup of some $B_m \in \mathcal{B}$.

For example, 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 $\{GL_n(q)\}_{(n,q)}$ is a group family. For any group family $\mathcal{B}$, the class of all abelian (solvable) subgroups $\{G \mid G < B_m$ for some $m$ and $G$ is abelian (respectively solvable)$\}$ is a subclass of $\mathcal{B}$.

Let $\mathcal{B} = \{B_m\}_{m>0}$ be a group family. We consider the following three basic computational problems over black-box groups.

Membership Testing $\triangleq \{\langle m, S, g\rangle \mid \langle S\rangle < B_m$ and $g \in \langle S\rangle\}$.
Order Verification $\triangleq \{\langle m, S, n\rangle \mid \langle S\rangle < B_m$ and $|\langle S\rangle| = n\}$.
Group Isomorphism $\triangleq \{\langle m, S_1, S_2\rangle \mid \langle S_1\rangle$ and $\langle S_2\rangle$ are isomorphic subgroups of $B_m\}$.

In Chapter 4, we prove the following theorem.

**Theorem** *Over any group family $\mathcal{B}$, the problems* Membership Testing, Order Verification, Group Isomorphism *over the subclass of solvable groups are in* SPP *and hence low for the classes* PP, $\mathrm{C_{=}P}$, *and* $\mathrm{Mod}_k\mathrm{P}$ *for $k \geq 2$.*

A few remarks about the above theorem are in order. Using the fact that testing for primality is in UP [FK92], it can be shown that the above-mentioned problems when defined over *cyclic* groups are in the class UP. So the above result can be seen as a nontrivial extension of the upper bound for cyclic groups to solvable groups. Note that the class SPP is a generalization of the class UP.

We very briefly explain the main ideas that go into the proof of the above theorem. Since solvable groups can be viewed as a series of abelian factor groups, we first consider the above problems over abelian groups. Using a constructive version of a fundamental theorem on the structure of abelian groups, we first show the same upper bound for abelian factor groups. Then, using a procedure for computing normal closure to get generator sets for the commutator group of a finite group, we extend the upper bound for abelian groups to solvable groups.

## 1.4 Complexity of Exact Learning

In this section, we explain the contents of the second part of the thesis. As mentioned before, in this part we are interested in analyzing the complexity of learning some group-theoretic representation classes. Before we state our main results, we give a very brief introduction to learning theory.

Learning theory is concerned with providing mathematical models for machine learning and analyzing the learnability/non-learnability of various classes of concepts in these models. Among various models that have been proposed, Valiant's PAC learning model [Val84] and Angluin's exact learning model [Ang88] have received considerable attention. In both these models the concept classes are sets of subsets of finite strings over a finite alphabet along with a short representation for each of the sets (concepts) in the class. A concept class along with a representation for each concept is called a representation class. By learning a representation class we mean to find a representation for the concept of interest with limited access to it. Among the representation classes which have received much attention are DFAs, DNFs and CIRCUITS. See the article by Angluin [Ang92] for a survey on learning theory.

A major area of research in computational learning theory is the classification of different representation classes with respect to the difficulty of learning them in any reasonable learning model. In this direction many interesting results are known. In Angluin's model (this is the model of interest to us), one way to quantify the complexity of a representation class is to consider the type and number of queries that a learner has to ask the teacher in order to learn any concept in the representation class. The two types of queries that Angluin had introduced in her model are membership and equivalence queries. There have been successful attempts to capture the complexity of representation classes with respect to the type of queries needed to learn them in this model, by combinatorial properties like approximate fingerprints, polynomial certificates etc. [Ang90, BCG+96, HPRW96, Heg95]. Also, Watanabe [Wat90] used complexity-theoretic ideas to analyze the complexity of query learning. He defined the notion of *machine types* to capture various types of queries in Angluin's model (see also [WG94]).

Our focus is to further investigate the complexity of exact learning (henceforth, by learning we mean exact learning). Our approach towards classifying the complexity of learning representation classes is built on ideas from complexity theory. To motivate our study, consider the scenario where there are two representation classes both of which can be learned using polynomially many equivalence queries but cannot be learned using polynomially many membership queries. What further can we say about their learnability? It could be the case that one of the classes is easier to learn than the other because the full power of equivalence query may not be necessary to learn the former although it is required to learn the latter. To frame the above question more formally, we first give a brief introduction to Angluin's exact learning model.

Let $\mathcal{P}$ be a representation class (for example, the class CNF: in this case the boolean functions are the concepts of interest and the representation of a function is by conjunctive normal form formula). A learner, usually a deterministic Turing machine, has to output the representation of the concept fixed by the teacher. During the computation, the learner can ask two types of queries to the teacher about the concept of interest; membership queries and equivalence queries. To a membership query $x$ asked, teacher gives YES/NO answer depending on whether $x$ is in the concept or not. An equivalence query is a string $y$ which is a representation for some concept in the class and the teacher answers YES if $y$ represents the concept of interest or produces a string $x$ in the symmetric difference of the concept of interest and the concept represented by $y$, as a counter example. An efficient learner is one which outputs a representation of the concept, in time polynomial in the length of the minimal representation of the concept.

Now consider the three representation classes; SYM of permutation groups (a subgroup $G$ of $S_n$ is represented by a generator set for $G$), LS$(p)$ of linear spaces over finite fields (represented by a basis) and 3-CNFs (the exact definitions of these classes are given in Chapter 5). We have the following theorem on the complexity of learning these classes in Angluin's model.

**Theorem** *The classes* 3-CNF, SYM *and* LS($p$) *are polynomial-time learnable with equivalence queries but not polynomial-time learnable with only membership queries.*

An immediate question that arises is whether we can say more about the learnability of these classes. Intuitively, we can expect that the algebraic classes SYM and LS($p$) may be *easier* to learn than 3-CNF due to their inherent algebraic structure. Our goal is to investigate this possibility. We develop a new exact learning model, called the *teaching assistant* model of exact learning. This can be seen as a refinement of Angluin's model. The new ingredient in our model is the concept of a *teaching assistant*. This model allows us to make a finer classification of the complexity of exact learning than what is possible in Angluin's model.

The motivation for the definition of this model is the following. In Angluin's model of exact learning the learner communicates with the teacher through equivalence and membership queries in order to learn a concept. It is easy to show that an equivalence query can be replaced by a series of queries to an NP oracle where the machine accepting this oracle is a non-deterministic oracle Turing machine with access to the concept of interest (a similar result can be seen in [WG94]).[1] So, intuitively we can think of the NP oracle as an intermediate agent (a teaching assistant) between the learner and the teacher. To this teaching assistant the learner can make queries and the Turing machine accepting this assistant is allowed to make membership queries to the teacher. Hence informally, we can say that a representation class which is polynomial-time learnable (hereafter, we use the notation FP-learnable) using equivalence queries is also FP-learnable with an assistant in NP. Now, this idea can be extended to other well-studied complexity classes also; for example the

[1] The above statement is informal. In Chapter 5, after defining teaching assistants, we will give a more formal proof of this.

classes NP $\cap$ co-NP, UP, SPP, LWPP etc. This immediately gives a framework for comparing the complexity of exactly learning various representation classes.

The second part of the thesis spreads over into two chapters, Chapters 5 and 6. In Chapter 5, one of our main contributions is the precise definition of the new teaching assistant model and that of FP-learnability using various assistant classes. The teaching assistant classes that we define here are the ones analogous to the classes P, NP $\Sigma_2^p$, SPP and LWPP. After giving definitions of all the notions involved in our model, we formulate the notion of learning any representation class efficiently with assistants from a teaching assistant class $\mathcal{C}$ (FP-learnability with a $\mathcal{C}$-assistant). The main result we show here is stated as the following theorem.

**Theorem** *The representation class* SYM *is* FP*-learnable with an* LWPP*-assistant. The representation class* LS$(p)$ *over any fixed prime $p$ is* FP*-learnable with an* SPP*-assistant. The class 3-*CNF *is not* FP*-learnable with an* LWPP*-assistant* (SPP*-assistant*) *unless* NP $\subseteq$ LWPP (*respectively* NP $\subseteq$ SPP).

This theorem illustrates the possibility of a finer classification of the complexity of exact learning than what is possible in Angluin's model. For proving the upper bounds, we make use the algebraic structure of these representation classes. In particular, in the case of representation class of permutation groups, we make use of the properties of special generator sets called *strong generator sets* for any permutation group.

In Chapter 6, we further investigate the fine inclusion structure that is possible among various teaching assistant classes. In particular, we consider the FP-learnability of some subclasses of SYM, with teaching assistants from assistant classes UP$\cap$co-UP, UP and NP$\cap$co-NP. We show upper bounds and absolute lower bounds on the learnability of these representation classes.

## 1.5 Organization of the Thesis

This thesis consists of seven chapters. In Chapter 1, we have already seen the basic motivation for studying the complexity of computational problems over finite groups. Chapter 2 consists of the necessary notations and definitions which we use throughout the thesis; both from complexity theory and group theory. Most of these are standard. At the end of Chapter 2 we prove a lemma (Lemma 2.0.2) which is the basic complexity-theoretic technique (both the lemma and the proof method) we use in almost all our upper bound proofs. So a good understanding of this lemma as well as the proof will considerably aid in understanding most of the upper bound proofs.

Chapters 3 and 4 contain the first part of the thesis. In Chapter 3, we give a very brief survey of computational group theory. Chapter 4 is devoted completely for proving the upper bound of SPP for the three basic computational problems over solvable black-box groups; namely Membership Testing, Order Verification and Group Isomorphism. The proof of this upper bound is built on the fundamental theorem of finite abelian groups (Theorem 4.2.1). The proof of the fundamental theorem given in [Bur55] is recommended for easily understanding the proof of our upper bound result. A proof of Lemma 4.2.6, which is the basic ingredient of the proof of the fundamental theorem, is given in the Appendix. Though the upper bounds we prove here are for solvable groups, we devote a large part of this chapter for dealing with abelian groups. This helps in understanding the basic ideas involved in the proofs for solvable groups.

Chapters 5 and 6 contain the second part of the thesis. These can be read independently of Chapters 3 and 4. In the first two sections of Chapter 5, we develop the teaching assistant model of learning. Naturally, these sections are basic requirements for the rest of the results in Chapters 5 and 6. Among all the learning algorithms that we design in Chapters 5 and 6 , the design of the learning algorithm for the class of permutation groups, SYM, is the most involved (Theorem 5.5.1).

For proving the correctness of this algorithm, the notion of a strong generator set for a permutation group and its properties are crucial.

Finally in Chapter 7, we conclude the thesis and state some open questions that arise from this investigation.

Much of the group theory (and some linear algebra) that we use are elementary and can be derived fairly easily from basic definitions. But, for completeness sake, in the Appendix we give proofs for most of the group-theoretic and linear-algebraic results we use in this thesis.