The Computational Complexity Column

by

Jacobo Torán

Dept. Theoretische Informatik, Universität Ulm
Oberer Eselsberg, 89069 Ulm, Germany
toran@informatik.uni-ulm.de
http://theorie.informatik.uni-ulm.de/Personen/jt.html

Isomorphism Testing: Perspective and Open Problems

V. Arvind* Jacobo Torán †

Abstract

For over three decades the graph isomorphism problem has tantalized researchers in algorithms and complexity. The study of this problem has stimulated a lot of research and has led to the discovery of important concepts in the area. In this article we take a fresh look at isomorphism problems and highlight some open questions.

*Institute of Mathematical Sciences, C. I. T. Campus, Chennai 600 113, India Email: arvind@imsc.res.in

†Abt. Theoretische Informatik, Universität Ulm, Oberer Eselsberg, 89069 Ulm, Germany. Email: toran@informatik.uni-ulm.de

1 Introduction

The graph isomorphism problem, GI, consists in deciding whether two given graphs are isomorphic. In other words, the problem is to test whether there is a bijective function mapping the vertices of the first graph to the nodes of the second graph and preserving the adjacency relation. GI has received considerable attention since it is one of the few problems in NP that is neither known to be computable in polynomial time nor to be NP-complete. GI is the best-known example of a family of isomorphism problems on algebraic structures like groups and rings that have a similar intermediate status, between P and NP-complete. Isomorphism questions have proved in the past to be an important tool for exploring the tight interplay between computational problems and complexity classes. Often, these problems do not quite fit in standard complexity classes, in terms of completeness for example. The study of this peculiarity has motivated important advances in complexity theory: Arthur-Merlin games, lowness, interactive proof systems, counting classes or derandomization. From an algorithmic perspective, the attempts at discovering a polynomial-time algorithm for graph isomorphism has enriched the field with algebraic techniques, particularly from the theory of permutation groups.

In this column we briefly survey the status of some important open questions related to isomorphisms of graphs (also rings and groups). We do not attempt to be comprehensive. Rather, our goal is to focus on a few topics and to identify interesting open questions for which, hopefully, the answers do not lie too far beyond reach. In a brief survey of this nature it is difficult to touch upon ramifications of the area in computational group theory, which is a subject by itself. Also, a certain bias due to our research interests in complexity theory is unavoidable.

2 Preliminaries

By graphs we mean finite simple graphs, usually denoted by $X = (V, E)$, where V is the vertex set and $E \subseteq \binom{V}{2}$. We say two graphs X_1 and X_2 are *isomorphic* if there is a bijection $\varphi : V_1 \longrightarrow V_2$ such that $(u, v) \in E_1$ iff $(\varphi(u), \varphi(v)) \in E_2$. We write $X_1 \cong X_2$ and call φ an isomorphism. An *automorphism* of a graph X is an isomorphism from X to X. Automorphisms are permutations on the set V, and the set of automorphisms Aut(X) forms a group under permutation composition. More precisely, if $|V| = n$ then Aut(X) is a subgroup of S_n the symmetric group on n elements. It is well known that graph isomorphism testing is polynomial time equivalent to find-

ing a polynomial-size generator set for the automorphism group of a graph. We now recall some relevant permutation group theory.

In general, $\mathrm{Sym}(\Omega)$ denotes the symmetric group on the *finite* set Ω. A *permutation group* on Ω is a subgroup of $\mathrm{Sym}(\Omega)$. For $|\Omega| = n$, we let $\Omega = [n]$ and simply write S_n of all permutations on $[n] = \{1, 2 \ldots, n\}$ to denote $\mathrm{Sym}(\Omega)$. Given $g \in S_n$ and $i \in [n]$, we denote by i^g the image of i under permutation g. This a convenient notation to express the left to right composition $g_1 g_2$ of permutations $g_1, g_2 \in S_n$. More precisely, we can write $i^{g_1 g_2} = (i^{g_1})^{g_2}$ for all $i \in [n]$. For $\Delta \subseteq [n]$ and $g \in S_n$ we write Δ^g for its image under g: $\Delta^g = \{j \mid j = i^g\}$. For $\Delta \subseteq [n]$, $G^{(\Delta)}$ denotes the subgroup of G that fixes each element of Δ, and G_Δ denotes the subgroup $\{g \in G \mid \Delta^g = \Delta\}$.

The permutation group *generated* by a subset A of S_n is the smallest subgroup of S_n containing A and is denoted $\langle A \rangle$. We assume that subgroups of S_n are presented by generator sets. Since any finite group G has a generator set of size $\log |G|$, subgroups of S_n have generator sets of size polynomial in n. The identity permutation is denoted by 1 (we use 1 to denote the identity of all groups).

For a subgroup G of S_n (denoted $G \leq S_n$) the set $i^G = \{i^g \mid g \in G\}$ for $i \in [n]$ is the *G-orbit* of i, and G is *transitive* on $[n]$ if $i^G = [n]$ for $i \in [n]$. Let $G \leq \mathrm{Sym}(\Omega)$ be transitive on Ω. A *G-block* is a subset Δ of $[n]$ such that for every $g \in G$ either $\Delta^g = \Delta$ or $\Delta^g \cap \Delta = \emptyset$. For a transitive group G, the set $[n]$ and the singleton sets $\{i\}$, $i \in [n]$ are *trivial* blocks. A transitive group G is *primitive* if it does not have any nontrivial blocks otherwise it is called *imprimitive*.

Let G_1 and G_2 be two finite groups. We say that G_1 and G_2 are isomorphic if there is a bijection $\varphi : G_1 \longrightarrow G_2$ that preserves the group operation. Likewise, for two finite rings R_1 and R_2, we say that they are isomorphic if there is a bijection $\varphi : R_1 \longrightarrow R_2$ that preserves the ring operations. As for graphs, automorphisms are isomorphisms from an algebraic structure to itself, and the automorphisms form a group under the composition operation.

We briefly recall the definitions and notation for some standard complexity classes. Details can be found in a textbook like [14]. Let P denote the class of languages (decision problems) that are accepted by deterministic Turing machines in time bounded by a polynomial in input size, and NP denote the class of languages accepted by nondeterministic Turing machines in polynomial time. We denote the class of functions computable in polynomial time by FP.

A function $f : \{0,1\}^* \to \mathbb{N}$ is said to be in the counting class #P if there is a polynomial time nondeterministic Turing machine M such that $f(x)$ is the number of accepting paths of M on input x.

A function f in the class FP^A is computable by polynomial-time deterministic *oracle* Turing machine M which has access to oracle A: M can enter a special query state and query the membership of a string y in A. We can similarly define FP^f for a function oracle f. Let $\mathcal{C}$ be a relativizable complexity class. A language A is said to be *low* for $\mathcal{C}$ if $\mathcal{C}^A = \mathcal{C}$.

3 Hardness

GI has several properties that are not known to hold by NP-complete problems. For example, the counting version of GI is reducible to its decision version [28]. Moreover, it is known that graph non-isomorphism, the complement of GI, belongs to the class AM of decision problems whose "yes" instances have short membership proofs in a probabilistic sense [7]. This implies that if the problem were NP-complete, then the polynomial time hierarchy would collapse to its second level [15, 34]. Because of these facts, we do not believe that GI is NP-complete. On the other hand GI is not known to be in P and we might ask what is the largest complexity class $\mathcal{C}$ for which we can prove that GI is hard for $\mathcal{C}$? Or more specifically:

Problem 1. *Is GI hard for P?*

The first hardness results for GI were given in [20] where it was shown that GI is hard for NC^1 the class of problems computable by uniform circuit families of polynomial size and logarithmic depth, and for L, logarithmic space. The hardness for NC^1 is proved by essentially "simulating" a logarithmic depth circuit with AND and OR gates by an isomorphism question. For each gate g in the circuit, a pair of graphs (G_g, H_g) is constructed in such a way that the graphs are isomorphic if and only if the gate has value 1. This is easy to do for the input gates. For the circuit gates, the AND and OR functions for GI are used. An AND function for a problem A is a function f that is easy to compute and such that on input x, y, $f(x, y) \in A$ if and only if $x \in A$ AND $y \in A$. The OR function is defined analogously. It is known that GI has AND and OR functions. This property can be used as sketched above to finally build a pair of graphs (G, H) corresponding to the output gate, such that they are isomorphic if and only if the circuit outputs 1. A natural question is: why cannot this method be applied to similarly "simulate" polynomial-size monotone circuits? If this were possible it would follow that GI is hard for P. Unfortunately, the difficulty lies with the known OR function construction for GI: the OR function doubles the size of its inputs. Therefore, in order to keep the output of the reduction polynomial in size, the above method can only be applied to for circuits having a logarithmic

number of OR-gates in any path from an input to the output gate. A natural question in this context is the following.

Problem 2. *Does graph isomorphism have an efficiently computable* OR *function f such that $f(x,y)$ has size at most $c(|x|+|y|)$, where $c<2$?*

The hardness results for GI from [20] were improved in [35] to other complexity classes using a different method. In order to simulate a certain kind of circuit gate g with inputs x and y, a graph gadget is constructed having some vertices related to the inputs of g and some vertices related to the outputs. An automorphism in the gadget graph with certain restrictions encoding the input values of g is forced to map the nodes related to the output in a way encoding $g(x,y)$. An example of such a gadget encoding a parity gate is given in Figure 1.

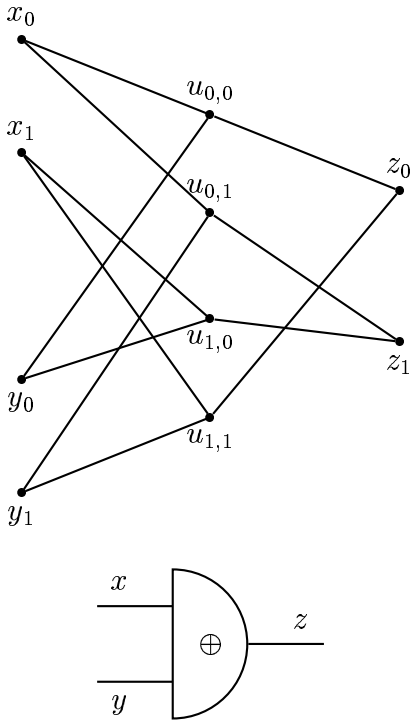

Figure 1: A graph gadget simulating a parity gate.

For the $\oplus$-gate considered in the figure, the input values can be encoded in the gadget graph automorphism as follows: if x has value $a \in \{0,1\}$ then we

restrict the set of considered automorphisms to those mapping vertex x_0 to x_a, and the same for y. It is not hard to see that any automorphism mapping x_0 to x_a and y_0 to y_b for $a, b \in \{0, 1\}$, must map z_0 to $z_{a \oplus b}$ thus computing the output of the gate. A gadget is constructed for each gate and they are connected as in the circuit. The constructed graph has an automorphism of the kind encoding the input values of the circuit and mapping the output vertex to a vertex encoding value 1 if and only if the value produced by the circuit is 1. This question can be reduced to GI. In [35] it is shown that such graph gadgets can be constructed for every modular addition gate. More generally, for any commutative group this gadget can simulate the group operation. But this does not seem to suffice for capturing the whole class P. It is known that the circuit value problem for polynomial size circuits with gates computing multiplication in S_5, the group of permutations over five elements, is complete for P. But S_5 is not an abelian group, and therefore the technique from [35] cannot be applied here.

The largest complexity class known to be reducible to GI is DET [35], the class of problems that are NC^1 reducible to computing the integer determinant [16]. DET belongs to NC^2 and therefore there is still a large gap for proving hardness of GI for P. An immediate and natural open question is whether GI is hard for LOGCFL (LOGCFL is the subclass of NC^2 consisting of problems that are logspace reducible to a context-free language).

Problem 3. *Is graph isomorphism hard for* LOGCFL*?*

A different approach to make progress on this problem is to consider isomorphism of other algebraic structures (see Section 6 where we consider some of these in detail). Problems like ring isomorphism and group isomorphism appear to be harder than GI. It should be easier to show that these are hard for P.

Problem 4. *Are the isomorphism problems for rings, permutation groups or black-box groups hard for* P*?*

4 Graph Isomorphism for Restricted Classes

The graph isomorphism problem for certain special classes of graphs is known to have polynomial-time algorithms. One such restriction that is well-studied is the *bounded color multiplicity graph isomorphism* problem (BCGI_b): for a pair of vertex-colored graphs (G_1, G_2) such that there are at most a constant b many vertices of any given color in each graph, test if there is a color-preserving isomorphism between G_1 and G_2.

Luks in [26] gave a remarkable NC algorithm for the BCGI_b problem. On the other hand, we can see that several of the hardness results for GI [35] (see Section 3) are hardness results for BCGI_b. More precisely, it is known that BCGI_b is AC^0-many one hard for the logspace counting class Mod$k$ for each constant k. The construction in [35] requires b to be k^2. Building on [26], in [5] the gap between the upper and lower bound results for BCGI_b is in some sense closed by proving that BCGI_b is in Mod$k$ hierarchy and noting that the hardness results for BCGI_b extend to the Mod$k$ hierarchy, where the constant k and the level of the hierarchy in which BCGI sits depends on b.

Tight characterizations are also known for tree isomorphism in two different representations (see e.g. [20] for this and other examples).

This opens up similar complexity-theoretic questions for other classes of graphs for which GI has a polynomial-time algorithm. The problem is to precisely classify the restricted problem inside P by giving matching upper and lower bounds. Recall that inside P there is a rich tapestry of natural complexity classes. Particularly, natural problems like computing integer determinant abound within NC^2; the classification of these problems are mainly a result of insights into logspace counting classes (see e.g. the survey article [2]). Thus it is natural to seek the precise classification of restricted versions of GI. Notable examples are (i) graphs of bounded degree [25], (ii) graphs of bounded genus [29], and (iii) graphs of bounded eigenvalue multiplicity [11] which all have polynomial time algorithms for GI.

Of these, we focus on graph isomorphism for bounded degree graphs (BDGI), where the maximum degree of the input graphs is bounded by a constant. Luks in [25] gave a polynomial-time algorithm for this problem. This paper was a major breakthrough, introducing methods from permutation group theory which have since become central techniques in the area of isomorphism testing as well as in the design of permutation group algorithm. However, it is still open if BDGI is in NC (or even RNC).

Luks has observed in [26] that BDGI can be NC reduced to the set-stabilizer problem for groups in Γ_d. We recall the definitions to introduce the ideas involved.

Definition 1. *A finite group G is said to be in the class Γ_d if for any composition series $G = G_0 \rhd G_1 \ldots \rhd G_t = 1$ each composition factor G_i/G_{i+1} is either abelian or is isomorphic to a subgroup of S_d.*

The class Γ_d of finite groups is algorithmically important. It has played an important role in proving time bounds for several permutation group algorithms, including the current best algorithm for the graph isomorphism problem (e.g. see [27]).

Given a permutation group $G \leq S_n$ by a generating set A and a subset Δ of $\{1, 2, \cdots, n\}$, the *set stabilizer problem* is to compute a generating set for the stabilizer group G_Δ. Set stabilizer is of interest because GI is reducible to it. To see it note that it suffices to show that finding the automorphism group is reducible to set stabilizer. For a graph $X = (V, E)$, let $G = S_n$ act on the pairs $\binom{V}{2}$ where $|V| = n$. Clearly, for $\Delta = E$ we have G_Δ is Aut(X).

Proposition 2. *Graph isomorphism is polynomial-time reducible to set stabilizer.*

A more involved reduction in [26] shows that BDGI is NC reducible to permutation groups in Γ_d. Therefore, one way to put BDGI in NC would be to show that the set stabilizer problem is in NC.

Problem 5. *Precisely classify the complexity of the set stabilizer problem for groups in Γ_d (or even solvable groups).*

It follows from the results of Babai, Luks, and Séress [11, 8] that graph isomorphism for the bounded eigenvalue multiplicity case is in NC.

Problem 6. *Classify the complexity of graph isomorphism for graphs with bounded eigenvalue multiplicity.*

5 Graph Canonization

Let $\mathcal{G}_n$ denote the set of all simple undirected graphs on n vertices. A *canonizing function* for $\mathcal{G}_n$ is a function f from $\mathcal{G}_n$ to $\mathcal{G}_n$ such that

- For any graph $G \in \mathcal{G}_n$, $f(G)$ is isomorphic to G.
- For $G_1, G_2 \in \mathcal{G}_n$, $f(G_1) = f(G_2)$ if and only if G_1 is isomorphic to G_2.

In other words, a canonizing function assigns a *canonical form* to each isomorphism class of graphs.

For example, the function f such that $f(G)$ is the lexicographically least graph in the isomorphism class containing G is a canonizing function. However, as observed in [10, 27], this canonizing function is NP-hard. Notice that this function can be computed in $\mathrm{FP}^{\mathrm{NP}}$ by a simple prefix search algorithm.

The intriguing open question is whether there is *some* canonizing function for graphs that can be computed in polynomial time. No better upper bound than $\mathrm{FP}^{\mathrm{NP}}$ is known for general graphs (for any canonizing function). Thus, it is a basic complexity-theoretic question to classify the complexity of canonization. Is this problem low for any level of the polynomial hierarchy?

We note that obtaining canonical forms for algebraic objects is a natural and fruitful pursuit in mathematics. For instance, we have the Jordan Canonical Form for matrices under similarity transformations. Likewise, we have the Hermite Normal Form for lattices under unimodular transformations. These normal forms can play an important role in the design of efficient algorithms for problems.

On first sight it would appear that graph canonization is closely related to the problem of isomorphism testing. Indeed, for one direction we can observe that isomorphism testing for graphs is polynomial-time reducible to graph canonization. How about the converse?

Problem 7. *Is graph canonization polynomial time reducible to graph isomorphism?*

There is interesting evidence supporting a positive answer in the results of Babai and Luks [10]. Building on the earlier seminal work of Luks [25], Babai and Luks take an algebraic approach to the canonization problem. We recall some definitions before we explain a key result in their paper.

First we can assume by encoding that we are working with strings (over a finite alphabet, say $\{0,1\}$) as our objects instead of graphs. Let $G \leq S_n$ be a subgroup of the symmetric group acting on $\{0,1\}^n$ as follows: for a permutation $g \in G$ and $x = x_1 x_2 \cdots x_n \in \{0,1\}^n$, g maps x to y (denoted $x^g = y$), where $y = x_{i_1} x_{i_2} \cdots x_{i_n}$ such that $i_k = k^g$ for $1 \leq k \leq n$.

Now, the general problem can be stated as follows: We say that two strings x and y are G-isomorphic if $x^g = y$ for some $g \in G$. We say that $f : \{0,1\}^n \longrightarrow \{0,1\}^n$ is a canonizing function w.r.t. the group G, if $f(x)$ and x are G-isomorphic for all x and $f(x) = f(y)$ iff x and y are G-isomorphic. In any case, lexicographic canonization remains hard even for very simple groups G.

Proposition 3. [10] *The lexicographic canonizing function w.r.t. arbitrary groups G for strings is* NP*-hard even if G is restricted to be an elementary abelian 2-group.*

The idea of the proof is to give a polynomial-time reduction from the maximum clique problem to the lexicographic canonization problem.

More important, on the positive side, Babai and Luks give an algorithm for computing a canonizing function *that depends* on the structure of the group G. This algorithm is based on a divide-and-conquer strategy along the same lines as developed by Luks in [25]: the divide and conquer is done on the group G based on its internal structure (transitive constituents, primitivity and imprimitivity structure).

If G is a permutation group in the class Γ_d, it turns out that this canonization algorithm runs in polynomial time. Crucially, the fact that primitive groups in Γ_d are of size at most $n^{O(d)}$ is used for this analysis. To summarize:

Theorem 4. [10] *Given a group $G \leq S_n$ such that $G \in \Gamma_d$, there is an $n^{O(d)}$ algorithm that computes a canonizing function for $\mathcal{G}_n$.*

In [10] it is also shown that for general graphs there is a $c^{n^{1/2+o(1)}}$ canonizing algorithm which closely matches the running time of the best known isomorphism test for general graphs. However, from a complexity-theoretic point of view the relative difficulty is not clear even for the problem of G-isomorphism testing for a group $G \in \Gamma_d$. In particular, we would like to know an answer for the following question. We conjecture that the answer should be positive.

Problem 8. *Let $G \leq S_n$ be a permutation group that is in Γ_d. Is the problem of testing if two strings x and y are G-isomorphic NC equivalent to the corresponding canonization problem? We can also ask a similar question for solvable permutation groups, which is a subclass of Γ_d.*

A more general problem is the following.

Problem 9. *For different restricted graph classes considered in Section 4, what is the relative complexity of isomorphism and canonization?*

We next briefly discuss canonization for finite groups and rings. What is the appropriate notion for groups? For abelian groups, the structure theorem decomposing any finite abelian group into a direct product of cyclic groups is a natural canonical form and it can be used to test the isomorphism of two abelian groups. Thus, the problem of canonization for finite abelian groups boils down to computing the cyclic group decomposition. Of course, the question arises whether there could be canonical forms that are *easier* to compute. For nonabelian groups, it does not appear that there is any such intrinsic canonical form. It is tempting to use the composition series (or some other series for groups) but these are only partial isomorphism invariants. Of course, the lexicographic canonical form can always be defined for finite groups (and rings). But one would suspect that it is NP-hard to compute. Turning to finite rings, we can try to use Wedderburn's decomposition theorem for semisimple rings to define canonical forms. To summarize, we have the following open-ended question.

Problem 10. *What are the suitable canonical forms for finite groups and rings and what is complexity of computing these canonical forms?*

6 Ring and Group Isomorphism

We look now at the complexity of isomorphism testing for rings and groups. These questions have evoked interest due to the recent work by Kayal and Saxena [21] relating the complexity of ring isomorphism to both graph isomorphism and integer factoring. More recently, Agrawal and Saxena in a fascinating article [1] have highlighted the importance of finite rings and their automorphisms for computational problems in algebra with various examples.

We discuss the main results about ring isomorphism and automorphism from [21]. Alongside, we make some new observations for the group isomorphism problem to draw comparisons and formulate open questions.

Recall that a ring $(R, +, .)$ with unity is a commutative group under the addition operation with 0 as identity and is a monoid under multiplication with 1 as multiplicative identity, together with multiplication distributing over addition.

The complexity of isomorphism problems might change depending on the way the input instances are represented. We first consider the representation of a finite ring R. One way is to describe R explicitly by its addition and multiplication tables. This *table representation* is of size $O(k|R|^2)$, where elements of R are encoded as strings of length k.

A more compact *basis representation* would be to describe R by giving a *basis* for R. The basis is an *independent* generating set $\{e_1, e_2, \cdots, e_m\}$ for the additive group $(R, +)$. Clearly, $(R, +)$ has generating sets of size $m = O(\log |R|)$. Additionally, to describe the multiplicative structure, "structural constants" of the ring $\alpha_{ijk} \in \mathbb{Z}, 1 \leq i, j, k \leq m$ are given, where $e_i \cdot e_j = \sum_k \alpha_{ijk} e_k$. Since the *characteristic* of R is bounded by $|R|$, each structural constant is m bits long. Now, suppose that the elements of R are encoded as strings of length k. Clearly the entire representation is of size $O(km^4)$.

Likewise, consider finite groups G whose elements are encoded as strings of length k. We could describe G by the *table representation* by giving the multiplication table of size $O(k|G|^2)$. Again, a more compact representation for finite groups would be to give a generating set $\{g_1, g_2, \cdots, g_k\}$ for G, where the multiplication operation is implicitly described by a "black-box" [13, 9]. The "black-box" model introduced by [13, 9] is a convenient setting to study the complexity of group-theoretic problems that do not take advantage of the actual group operation (permutation groups or matrix groups etc).

A third possibility for representing finite abelian groups by giving *independent* generating sets. Whether an arbitrary generating set can be transformed to an *independent* generating set in polynomial time is open. It is related to membership testing and the discrete log problem. However, this

transformation can be done by a polynomial time quantum algorithm. Indeed, isomorphism testing for abelian black-box groups given by generating sets can be done in quantum polynomial time (see [31] for example).

Problem 11. *What is the complexity of converting a generator representation to a basis representation for finite rings?*

Notice that the basis representation for finite rings is more structured than the generator representation for finite groups. The nicer representation is basically due to the fact that the additive group of a finite ring is commutative. Indeed, if a finite group G is given by a generator set $\langle g_1, g_2, \cdots, g_k \rangle$, in general it is not possible to express an arbitrary element $g \in G$ as a polynomial-size product $\prod_{j=1}^{m} g_{i_j}$! However the reachability lemma of [13] shows that it is possible to express g as a polynomial-size straight-line program over the generators.

In this section we focus on the basis and generator representation for rings and groups. We will discuss the table representation for these problems in Section 7.

Kayal and Saxena [21] study the complexity of ring isomorphism. We recall their main results here. For rings input in the basis representation, it is shown in [21] that the problems of *finding a ring automorphism*, *counting ring automorphisms*, *ring isomorphism testing*, and *finding a ring isomorphism* are all essentially in $\mathrm{AM} \cap \mathrm{coAM}$. More precisely, the functional versions of these problems are in $\mathrm{FP}^{\mathrm{AM} \cap \mathrm{coAM}}$. Curiously, the problem deciding if a ring has a nontrivial automorphism is in P [21]. This is essentially because those finite rings that do not have nontrivial automorphisms have a nice mathematical description which can be tested in polynomial time.

In [21] the connection between ring isomorphism and integer factoring is also studied. It is shown that counting the number of ring automorphisms is harder than integer factoring and finding a nontrivial automorphism is equivalent to integer factoring (via randomized reductions).

It is interesting to compare these results with the situation for group isomorphism. A basic difference between the two problems is in the description of an isomorphism. A ring isomorphism between two rings R_1 and R_2 in basis representation can be described by an invertible integer matrix (after suitably modifying the bases in polynomial time). However, in the case of an isomorphism φ between two finite groups G and H given by generator sets, there seems no mathematically explicit way to describe a group isomorphism. We can only describe φ by taking each generator g_i of G and expressing $\varphi(g_i)$ as a straight-line program over the generators of H.

Nevertheless, it is shown in [9] that black-box group isomorphism is in $\mathrm{AM} \cap \mathrm{coAM}$. In the case of permutation group isomorphism, where the two

input groups are permutation groups and hence more amenable, the group isomorphism problem is shown to be in $\mathrm{NP} \cap \mathrm{coAM}$.

Since the isomorphisms (or automorphisms) of finite groups given by generators do not have explicit mathematical descriptions, we do not have an $\mathrm{FP}^{\mathrm{AM} \cap \mathrm{coAM}}$ bound for counting the number of group automorphisms (equivalently isomorphisms). However, suppose a mapping $\varphi : G \longrightarrow H$ is given by $\varphi(g_i)$ as a straight-line program over the generators of H for each generator g_i of G. Then testing if φ defines an isomorphism is in $\mathrm{AM} \cap \mathrm{coAM}$ due to the order-verification interactive protocol of Babai [9]. Using this we can give a $\#\mathrm{P}^{\mathrm{NP}}$ upper bound: Suppose elements of G and H are encoded as strings of length m. For the generators $g_1, g_2, \cdots, g_k$, the #P oracle machine guesses the images $\varphi(g_i), 1 \leq i \leq k$ as strings of length m. Using an NP oracle, it then computes the straight-line programs for each $\varphi(g_i)$, over the generators of H. Now, a new group K is formed, that is generated by the pairs $(g_i, \varphi(g_i))$. Notice that K is a subgroup of $G \times H$. The order verification AM protocol of [9] can now be used to compare the orders of G and K and to accept if and only if their orders are equal. Clearly, this upper bound also holds for the complexity of computing the number of automorphisms of G. Since $\#\mathrm{P}^{\exists.\mathrm{AM}} = \#\mathrm{P}^{\mathrm{AM}} = \#\mathrm{P}^{\mathrm{NP}}$ we have the following:

Proposition 5. *Computing the number of isomorphism between two groups G and H given by generating sets is in $\#P^{\mathrm{NP}}$.*

Problem 12. *Tightly classify the complexity of computing the number of group isomorphisms, when the groups are in the generator representation. More precisely, for a finite group G given by generators in the black-box model, is the problem of computing the number of automorphism in G low for any level of PH?*

In contrast note that counting ring automorphisms is low for $\mathrm{AM} \cap \mathrm{coAM}$. However hardness questions remain.

Problem 13. *Is counting ring automorphisms harder than discrete log? Is ring isomorphism (decision or search version) harder than discrete log?*

We next consider the question of *rigidity*. Rigid finite groups are finite groups with no nontrivial automorphism. We note that the algorithmic problem is trivial here.

Proposition 6. *There are no rigid groups except groups of order* 1 *and* 2.

Proof. Clearly the groups of order 1 and 2 are rigid. Let G be a finite group of size more than 2. If G is nonabelian then let $g \in G$ such that g

does not commute with all elements of G. Then the *inner automorphism* τ_g defined as: $\tau_g : x \mapsto gxg^{-1}$ is clearly a nontrivial automorphism. On the other hand if G is abelian, then we use the structure theorem of abelian groups to decompose G as a direct product of cyclic groups $G_1 \times G_2 \times \cdots \times G_r$. Suppose $|G_i| = t > 2$ for one of the cyclic groups G_i. Let $a \in G_i$ be a generator. Then a^k is also a generator of G_i for each k such that $\gcd(k, t) = 1$. It is easy to see that $a \mapsto a^k$ is an automorphism of G_i if and only if $\gcd(k, t) = 1$. If $t > 2$ there is at least one such $k > 1$ so that $a \mapsto a^k$ is a nontrivial automorphism of G_i. This can be extended easily to a nontrivial automorphism of G. On the other hand, if $|G_i| = 2$ for each i, then G is a vector space over $\mathbb{F}_2$ of dimension r. Hence any nonsingular $r \times r$ matrix different from identity over $\mathbb{F}_2$ is a nontrivial automorphism of G. ■

It is shown in [21] that all rigid rings have a simple structure that can be easily recognized. The above proposition implies that group rigidity is even easier to test than testing rigidity of rings. We recall that the rigidity question for graphs is not known to be in P.

We now show that computing the number of group automorphisms is also harder than integer factoring (analogous to the result for ring automorphisms in [21]). In the case of group automorphisms the hardness is easy to show. Consider $(\mathbb{Z}_n, +)$, the additive group of integers modulo n. The group is cyclic with 1 as generator, and $1 \mapsto j$ defines an automorphism if and only if $\gcd(j, n) = 1$, since $j \in \mathbb{Z}_n$ is a generator iff it is relatively prime to n. Thus, $(\mathbb{Z}_n, +)$ has precisely $\varphi(n)$ generators, where φ is the Euler φ-function. It follows that computing $\#\mathrm{Aut}(\mathbb{Z}_n)$ implied computing $\varphi(n)$ which is equivalent to integer factoring w.r.t. randomized polynomial-time reductions.

Proposition 7. *Integer factoring is reducible to computing the number of automorphism for a finite group G given by generators.*

It would be interesting to know if the same result holds for permutation groups.

Problem 14. *Is integer factoring reducible to computing the number of automorphism of a permutation group $G \leq S_n$ given by generators?*

In [21] it is shown that finding a nontrivial ring automorphism is equivalent to integer factoring. Interestingly, for the case of groups the situation is quite different. Let G be a group given by generators. We can check if it is nonabelian (simply by checking if the generators commute with each other). If G is nonabelian, we will find a generator g such that $gg_i \neq g_ig$ for some other generator g_i of G. Clearly, the inner automorphism τ_g defined by $\tau_g : x \mapsto gxg^{-1}$ is a nontrivial automorphism.

Proposition 8. *There is a polynomial-time algorithm for finding a nontrivial automorphism of a nonabelian group given by generator set.*

Abelian groups dot have inner automorphisms. On the other hand if G is an abelian group given by an *independent* generating set $\langle g_1, g_2, \cdots, g_k \rangle$, and a multiple n of $|G|$ is known, then it is easy to find a nontrivial automorphism applying the ideas of Proposition 6. If each g_i has order 2 then G is vector space over $\mathbb{F}_2$ and any nonsingular $k \times k$ matrix is an automorphism. Otherwise, if g_i has order more than 2 then pick a positive integer $a > 1$ such that $\gcd(a, n) = 1$ by randomly picking $a \in [n-1]$. Then, with high probability $g_i \neq g_i^a$ and g_i and g_i^a have the same order. Now, $g_i \mapsto g_i^a$ and $g_j \mapsto g_j, j \neq i$ defines a nontrivial automorphism.

However, if an abelian group G is given by a generating set (not necessarily independent) then the complexity of the problem is open.

Problem 15. *For abelian groups, is the problem of finding a nontrivial automorphism harder than integer factoring? Is it harder than discrete log?*

Another observation is that the problem of group isomorphism testing is harder than the decision version of discrete log: given $a, b \in \mathbb{Z}_n^*$, the problem is to check if a is in the cyclic group generated by b (i.e. $a \in \langle b \rangle$). Clearly, $a \in \langle b \rangle$ iff $\langle a, b \rangle$ is isomorphic to $\langle b \rangle$. More generally, the membership testing problem for groups reduces to group isomorphism.

For both ring and group isomorphism the relative complexities of search and decision remains open. In the case of graph isomorphism, search is polynomial-time reducible to decision. The reduction uses graph gadgets to guide a prefix search. It is not clear how to build similar gadgets for groups and rings.

Problem 16. *Is search polynomial-time reducible to decision for group isomorphism and ring isomorphism?*

7 Derandomization

Babai classified in [7] the graph non-isomorphism problem in AM, a randomized version of NP that can be described in terms of Arthur Merlin protocols. Several authors (e.g. [3, 22, 30]) have studied derandomization of AM to NP under suitable hardness assumptions, thus showing that GI belongs to NP $\cap$ coNP. This derandomization works for the entire class AM. It is natural to ask if the AM protocol for graph non-isomorphism can be unconditionally derandomized.

Problem 17. *Can the* AM *protocol for graph non-isomorphism be derandomized unconditionally? Or under weaker hardness assumptions that those used in [3, 22, 30]?*

We considered in [6] the question of whether the group isomorphism problem (for the case of groups given by multiplication tables) lies in NP $\cap$ coNP. This might be easier to show than for the case of GI since group isomorphism in the *table representation* appears to be an easier problem. Following the same approach as it has been done for the case of GI we showed that group non-isomorphism has an Arthur-Merlin protocol with the property that on input groups of size n, Arthur uses $O(\log^6 n)$ random bits and Merlin uses only $O(\log^2 n)$ nondeterministic bits. For the case of solvable groups we could derandomize this restricted protocol applying two different methods showing that:

- there is a nondeterministic polynomial time algorithm for the group non-isomorphism problem restricted to solvable groups that is incorrect for at most $2^{\log^{O(1)} n}$ inputs of length n, and
- under the assumption EXP $\not\subseteq$ ioPSPACE[1] the group isomorphism problem restricted to solvable groups is in NP $\cap$ coNP.

The restriction to solvable groups comes from the fact that for the derandomization, an easy to compute succinct representation for the groups is needed. This exists for the case of solvable groups, but it is an open question whether it exists for general groups (related to a form of the short presentation conjecture known to be true for almost all finite simple groups).

Problem 18. *Do the above derandomization results hold for the case of general groups?*

As mentioned in [6] the derandomization does work for general groups assuming the short presentation conjecture.

Turning to ring isomorphism in the table representation the above problem is easy to resolve. As the additive group is abelian, rings have succinct representations of the appropriate type of polylogarithmic size in the number of ring elements. Therefore Problem 18 can be answered affirmatively for the case of rings with addition and multiplication tables given explicitly. Thus, one would expect that the following problem is easier than for groups. For rings given in this way we can ask the general question.

Problem 19. *Is the ring isomorphism problem in the table representation in NP $\cap$ coNP?*

[1] A language L is in ioPSPACE if there is a PSPACE machine that is correct on L for infinitely many input lengths.

8 Quantum Computing

In this section we describe the attempts at a quantum algorithmic solution to the graph isomorphism problem and the difficulties in this approach. The generic problem that underlies the discrete log problem, integer factoring and graph isomorphism is the hidden subgroup problem. We explain the hidden subgroup problem and summarize the progress made on it. Then we discuss some interesting connections between quantum polynomial time and counting complexity classes. First, we recall the definition of the hidden subgroup problem.

Definition 9. *The input instance of the* hidden subgroup problem HSP *is a finite group G given by a generator set. Additionally, a function f from G to some finite set X is given as an oracle, such that f is constant and distinct on different right cosets of some subgroup H of G. The problem is to determine a generator set for H.*

The hidden subgroup problem is a generic problem which captures several questions. We explain how it captures graph isomorphism. Let X be a finite graph. Now, letting G be the permutation group S_n we define the "hiding function" $f : S_n \longrightarrow \mathcal{G}_n$ as $f(\pi) = X^\pi$, where X^π is the graph obtained from X by permuting its vertices with the permutation π. It is easy to see that the hidden subgroup is the automorphism group Aut(X) of X. Determining the automorphism group of a graph is polynomial-time equivalent to graph isomorphism.

Shor's quantum algorithms for integer factoring and discrete log are essentially solutions of suitable HSP's where the group G is abelian. Indeed, Shor's technique [33] yields a polynomial-time quantum algorithm for HSP when G is abelian (see e.g. [31]). However, the status of HSP is open for general nonabelian groups, except for some special cases (see, e.g. [18, 19, 32]). In particular, for $G = S_n$, it is not known if HSP has quantum polynomial time algorithms.

For ring isomorphism (in the basis representation) it is easy to formulate the problem of computing the automorphism group of a commutative ring of characteristic d as a hidden subgroup problem, where the group G would be the finite group consisting of matrices of a suitable dimension invertible modulo d. Again, such a matrix group is nonabelian in general (it even contains S_n) which makes the corresponding HSP a hard problem.

The hidden subgroup approach to designing an efficient quantum algorithm for graph isomorphism seems to have limitations: very little progress has been made on the nonabelian hidden subgroup problem. It appears that some new quantum algorithmic techniques are required. But there might

be restricted glaph classes of for which it is posible to test isomorphism in quantum polynomial time with the present techniques.

Problem 20. *Is there a restricted class of graphs for which the isomorphism problem (not known to be in P) has polynomial time quantum algorithms?*

GI has another connection with the class BQP of problems computable in quantum polynomial time. Fortnow and Rogers [17] have shown that BQP is low for PP, i.e. any problem is BQP is is powerless as oracle for PP. This is in fact the best known upper bound for BQP in terms of complexity classes. In [23] it is shown that graph isomorphism and several other permutation group problems are also low for PP. This was strengthened in [4] where it is shown that the hidden subgroup problem for permutation groups (and hence graph isomorphism) is in a more restricted counting complexity class. However, similar questions are open for ring and (nonabelian) group isomorphism.

Problem 21. *Is the ring isomorphism problem low for* PP*? Is the group isomorphism problem for nonabelian groups low for* PP*?*

References

[1] M. Agrawal and N. Saxena. Automorphisms of Finite Rings and Applications to Complexity of Problems. *Proc. Symp. Theoretical Aspects of Computer Science,* LNCS 3404, 1-17, Springer Verlag, Feb 2005.

[2] E. Allender. Arithmetic Circuits and Counting Complexity Classes. *Quaderni di Matematica* series, Edited by Jan Krajicek, 2004.

[3] V. Arvind and J. Köbler, On resource bounded measure and pseudorandomness, *Proc. 17th FSTT Conference* Lecture Notes in Computer Science 1346 Springer Verlag, 235–249, (1997).

[4] V. Arvind and P. P. Kurur. Graph Isomorphism is in SPP. *Proc. Foundations of Computer Science,* 743-750, 2002.

[5] V. Arvind, P. P. Kurur, and T.C. Vijayaraghavan. Bounded color multiplicity graph isomorphism is in the #L Hierarchy. *In Proceedings of the 20th Conference on Computational Complexity,* 2005, to appear.

[6] V. Arvind and J. Torán. Solvable group isomorphism is almost in NP ∩ coNP. *Proc. 19th IEEE Conference on Computational Complexity,* 91-103, 2004.

[7] L. Babai. Trading group theory for randomness. *Proc. 17th ACM Symposium on Theory of Computing*, 421–429, 1985.

[8] L. Babai. A Las Vegas-NC Algorithm for isomorphism of graphs with bounded multiplicity of eigenvalues. *Proc. Foundations of Computer Science,* 303-312, 1986.

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

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

[11] L. Babai, E. M. Luks, and Á. Seress. Permutation Groups in NC. *Proceedings Symp. Theory of Computing,* 409-420, 1987.

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

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

[14] J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I & II.* ETACS monographs on theoretical computer science. Springer-Verlag, Berlin, 1988 and 1990.

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

[16] S. A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64(1):2–22, 1985.

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

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

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

[20] B. Jenner, J. Köbler, P. McKenzie, J. Torán. Completeness results for graph isomorphism. *J. Comput. Syst. Sciences,* 66(3): 549-566, 2003.

[21] N. Kayal and N. Saxena. On ring isomorphism and automorphism problems. *In Proc. 20th IEEE Conference on Computational Complexity,* June 2005, to appear.

[22] A. Klivans and D. van Melkebeek, Graph Isomorphism has subexponential size provers unless the polynomial time hierarchy collapses. *In Proc. 31st ACM STOC*, 1999, 659–667.

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

[24] Johannes Köbler, Uwe Schöning, and Jacobo Torán. *The Graph Isomorphism Problem: Its Structural Complexity.* Birkhauser, 1993.

[25] E. M. Luks. Isomorphism of Graphs of Bounded Valence can be Tested in Polynomial Time. *Journal of Computer and System Sciences,* 25(1): 42-65, 1982.

[26] E. M. Luks. Parallel algorithms for permutation groups and graph isomorphism. In *Proceedings of the IEEE Foundations of Computer Science*, IEEE Computer Society, 292-302, 1986.

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

[28] R. Mathon. A note on the graph isomorphism counting problem. *Information Processing Letters*, 8:131–132, 1979.

[29] G. L. Miller. Isomorphism of k-Contractible Graphs. A Generalization of Bounded Valence and Bounded Genus. *Information and Control,* 56(1/2): 1-20, 1983.

[30] P.B. Miltersen and N. Vinodchandran, Derandomizing Arthur-Merlin games using hitting sets, in *Proc. 40th IEEE Symposium on Foundations of Computer Science,* 1999, 71–80.

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

[32] M. Rötteler and T. Beth, Polynomial-Time Solution to the Hidden Subgroup Problem for a Class of non-abelian Groups. ArXiv preprint quant-ph/9812070, 1998.

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

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

[35] J. Torán. On the hardness of graph isomorphism, *SIAM J. Comput.* 33(5): 1093–1108, 2004.