SOME COMPLEXITY THEORETIC ASPECTS OF GRAPH ISOMORPHISM AND RELATED PROBLEMS

By

Bireswar Das

THE INSTITUTE OF MATHEMATICAL SCIENCES, CHENNAI.

A thesis submitted to the
Board of Studies in Mathematical Sciences

In partial fulfillment of the requirements

For the Degree of

DOCTOR OF PHILOSOPHY

of

HOMI BHABHA NATIONAL INSTITUTE

April 2010

Homi Bhabha National Institute

Recommendations of the Viva Voce Board

As members of the Viva Voce Board, we recommend that the dissertation prepared by **Bireswar Das** entitled "Some Complexity Theoretic Aspects of Graph Isomorphism and Related Problems" may be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

- **Date :**
Chairman and Convener: V. Arvind (IMSc)

- **Date :**
Member : Samir Datta (CMI)

- **Date :**
Member : Meena Mahajan (IMSc)

- **Date :**
Member : N. S. Narayanaswamy (IITM)

- **Date :**
Member : C. R. Subramanian (IMSc)

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copies of the dissertation to HBNI.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it may be accepted as fulfilling the dissertation requirement.

- **Date :**
Guide : V. Arvind

DECLARATION

I, hereby declare that the investigation presented in the thesis has been carried out by me. The work is original and the work has not been submitted earlier as a whole or in part for a degree/diploma at this or any other Institution or University.

Bireswar Das

To my parents.

ACKNOWLEDGEMENTS

First of all I like to express my deepest gratitude to V. Arvind for supervising my work. It has been a wonderful experience to work with him. I cannot overemphasise his contribution to my research work. I am deeply indebted to him for his advice, encouragement and support.

It has been a pleasure to work with Johannes Köbler. A large part of my research has been done jointly with him. I like to thank him for so many research discussions and his immense help and hospitality during my visit to Humboldt Universität, Berlin.

I am thankful to Kamal Lodaya, Meena Mahajan, R. Ramanujam, Venkatesh Raman and C. R. Subramanian for teaching me theoretical computer science and for the friendly and open research atmosphere that they have nurtured at IMSc.

I was fortunate to enjoy the hospitality of Jacobo Torán during my visit to University of Ulm. I thank him for inviting me to University of Ulm and for the wonderful research experience. I wish to thank Peter Bro Miltersen for his help and support to visit Århus University.

I wish to thank Piyush Kurur, Partha Mukhopadhyay, N. Narayanan, Prajakta Nimbhorkar, Raghavendra Rao, Somnath Sikdar, Srikanth Srinivasan and Fabian Wagner for many helpful discussions and suggestions.

A special thanks to the IMSc football team for making my stay at IMSc enjoyable.

Finally, I thank my parents and sister for their help and patience during the time I have been working on this thesis.

Abstract

We study the complexity of graph isomorphism problem for restricted classes of graphs and investigate the complexity of group theoretic problems related to graph isomorphism. More specifically,

- We classify several problems closely related to the graph isomorphism problem in algorithmic group theory in the classes PZK and SZK (problems with perfect/statistical zero-knowledge proofs respectively). Prior to this, these problems were known to be in AM $\cap$ coAM. As PZK $\subseteq$ SZK $\subseteq$ AM $\cap$ coAM, we have a tighter upper bound for these problems.

- We give a constant round perfect zero knowledge proof for the group isomorphism problem when the groups are given by their multiplication tables. The prover and the verifier in this proof system use only polylogarithmically many random bits. Motivated by this, we study honest verifier statistical zero knowledge (HVSZK) proof where the prover, verifier and the simulator use polylogarithmic randomness and the protocol has polylogarithmically many rounds. We show that any language having such a proof system also has an HVSZK proof where not only the prover, verifier and the simulator use polylogarithmic randomness but also has polylogarithmic message size and only 2 rounds.

- We give a polynomial-time oracle algorithm for Tournament Canonization that accesses oracles for Tournament Isomorphism and Rigid-Tournament Canonization. Extending the Babai-Luks Tournament Canonization algorithm [30], we give an $n^{O(k^2+\log n)}$ algorithm for canonization and isomorphism testing of k-hypertournaments, where n is the number of vertices and k is the size of hyperedges.

- We give an FPT algorithm for the bounded color class hypergraph isomorphism problem which has run-time $b!2^{O(b)}N^{O(1)}$ where b is the size of the largest color class and N is the input size.

- We prove that the isomorphism and canonization problem for k-tree is in the class StUL which is contained in UL. We also prove that the isomorphism problem for k-path is complete for L under disjunctive truth-table reductions computable in uniform AC^0.

Contents

List of Figures

1 Introduction

Two graphs $X_1 = (V_1, E_1)$ and $X_2 = (V_2, E_2)$ are *isomorphic* if there is a bijection from the vertex set V_1 of the graph X_1 to the vertex set V_2 of the graph X_2 that preserves the edge relation. Given two graphs X_1 and X_2 the problem of deciding if the two graphs are isomorphic or not is known as *graph isomorphism problem* or GRAPH-ISO. Graph isomorphism problem holds a unique place in the study of computational complexity. The exact complexity status of GRAPH-ISO is still unresolved. We do not know if GRAPH-ISO has a polynomial-time algorithm. On the other hand GRAPH-ISO is very unlikely to be NP-complete[1]. In practice graph isomorphism is not considered hard. Indeed, there are several algorithms that performs very well in many instances of graphs [84]. A theoretical justification of this fact comes from the result of Babai, Erdös and Selkow [26]. They proved that graph isomorphism can be tested in 'linear time' for random graphs. GRAPH-ISO is in the class $\oplus$P, in fact it is low for the class $\oplus$P [18]. This means that even if we provide GRAPH-ISO as an oracle to the complexity class $\oplus$P it does not help to increase the computational power of the class. It was shown in [18] that GRAPH-ISO is in the class SPP, which might be a much smaller class than $\oplus$P. GRAPH-ISO is not known to be hard for P. We only know that GRAPH-ISO is hard for the complexity class DET with respect to logspace computable reductions [105]. The complexity class DET consists of all problems that are logspace reducible to the problem of computing determinant of a $n \times n$ integer matrix.

Apart from its elusive complexity status, GRAPH-ISO is also remarkable in one more respect. It has generated several new concepts in complexity theory and it has been used as non-trivial example of several ideas in complexity theory. For example graph

[1]If GRAPH-ISO is NP-complete then the polynomial hierarchy collapses to the second level [35].

isomorphism problem was one of the first concrete examples that demonstrated the ideas behind Arthur-Merlin games, interactive protocol, zero knowledge proof, lowness etc.

In the late 70's and early 80's Babai and Luks started applying results from permutation group theory and classification of finite groups to study the graph isomorphism problem. Soon the effectiveness of this process became evident when Luks [80] came up with his celebrated polynomial-time algorithm for bounded degree graphs. The group theoretic analogue of GRAPH-ISO is the *set transporter problem* (see [82]). In fact there are several computational problems in permutation group theory which resemble the graph isomorphism problem e.g., coset intersection, group factorization, group conjugacy in permutation groups etc. These problems often share a similar complexity status as GRAPH-ISO. For example, we do not know if there is a polynomial-time algorithm for solving these problems and they are unlikely to be NP-complete. Moreover, solving any of these problems would solve the GRAPH-ISO problem.

The study of computational aspects of group theory started very early. For example computing a list of primitive and transitive permutation groups of low order started during the nineteenth century [97]. Examples of classical results in computational group theory include Dehn's algorithm [42] for solving word problem for certain groups, Todd-Coxeter algorithm [104] for coset enumeration, Knuth-Bendix term-rewriting procedure [73] etc. Any group can be represented as a permutation group by specifying its action on a set. Computational aspects of permutation groups have been studied extensively as a result of which it is one of the most developed area in computational group theory. Indeed, algorithmic permutation group theory is rich with its inventory of efficient algorithms. A permutation group G is a subgroup of $\mathrm{Sym}(\Omega)$, where $\mathrm{Sym}(\Omega)$ is the group of all permutations of a finite set Ω with n elements. Any subgroup G of $\mathrm{Sym}(\Omega)$ can be specified by a set of generators of size $O(n)$. For example only two generators can generate the whole group $\mathrm{Sym}(\Omega)$. This compactness of representation makes the task of algorithm design even more challenging. Let $G \leq \mathrm{Sym}(\Omega)$ be generated by a set S. The runtime of an algorithm whose input G is given by the generating set S is typically measured in terms of $|S|$ and n, where $n = |\Omega|$. The Schreier-Sims algorithm [99] provided a basic building block in the design of algorithm for permutation groups. With the help of this algorithm membership testing, finding the order of a permutation can be performed in polynomial time.

The main surge in the study of *complexity* of permutation group problems came after Luks's polynomial-time algorithm for bounded degree graphs. Problems like coset intersection, group conjugacy, permutation group isomorphism has been put in the class NP∩coAM [23]. The problem of finding lexicographic leader in double coset was shown to be NP-complete. Several restricted versions of the above problems have polynomial-time algorithms [82].

One of the most beautiful concepts in complexity theory is zero knowledge proofs. Zero knowledge proofs are interactive proof systems which involve two parties, a prover and a verifier. The prover wants to prove the validity of some assertion. In doing so both parties interact by sending messages. If the assertion is really valid the the prover succeeds to convince the verifier with high probability. On the other hand if the assertion is false then the verifier will be able to detect it with high probability. Moreover when the assertion is true, the verifier learns nothing more than what it can know without the interaction with the prover. This, seemingly puzzling, condition is known as the zero knowledge condition. This condition is formalized by requiring a randomised polynomial-time algorithm known as the *simulator*. The simulator can *mimic* the interaction between the prover and verifier. Depending on how well the simulator "mimics" the interaction, zero knowledge protocols can be of different types, e.g., perfect zero knowledge, statistical zero knowledge and computational zero knowledge. Though the zero knowledge condition initially seems strange, Goldwasser, Micali and Racoff exhibited perfect zero knowledge proofs for Quadratic Residuosity, Graph Isomorphism [58].

By exhibiting a statistical zero knowledge proof for a computational problem one can infer several interesting results about‘ the complexity of the problem. For example, a problem possessing statistical zero knowledge proofs is in AM ∩ coAM and thus is unlikely to be NP-complete (see [4, 51, 94]).

Goldreich, Sahai and Vadhan [94, 57] have shown that two natural promise problems, Statistical Difference (SD) and Entropy Difference (ED) are complete for the class SZK of languages possessing statistical zero knowledge proofs. In Chapter 2 we show that several problems in permutation group theory have statistical zero knowledge proofs. We exhibit the zero knowledge proofs by a unified argument, by showing that these problems are polynomial-time many-one reducible to Statistical Difference (SD).

An ingenious valence reduction technique due to Zemlyachanko [117] led to a moderately exponential $exp(c\sqrt{n \log n})$ time algorithm for the general graph isomorphism problem (see [21]). To understand GRAPH-ISO more clearly researchers have restricted the problem to various special classes of graphs. Very efficient algorithms have been designed for restricted classes of graphs. The linear time algorithm for isomorphism of trees is the first nontrivial example of such algorithm (see [3]). Hopcroft and Wong [67] gave a linear algorithm for planar graph isomorphism in 1974. Miller gave a polynomial-time algorithm for testing isomorphism of bounded genus graphs. Babai, Grigoryev and Mount gave a polynomial-time algorithm for bounded eigenvalue multiplicity graphs [28]. Luks's celebrated result for bounded degree graph isomorphism came in 1982 [80]. Apart from time complexity, space complexity and parallelizability of GRAPH-ISO for restricted classes of graphs have also been considered over the years. For example, Lindell gave a logspace algorithm for tree isomorphism [79], Miller and Reif gave an AC^1 algorithm for planar graph isomorphism [89], Grohe and Verbitsky gave a TC^1 algorithm for isomorphism of bounded tree width graphs [60]. In Chapter 6 we study the space complexity of k-tree isomorphism and canonization and show that k-tree isomorphism is in a class StUL which is contained in the class UL.

It is often the case that the input for a computational problem comes with a parameter. For example, the vertex cover problem: Given a graph G and an integer k decide if G has vertex cover of size k. The natural parameter for this problem is the integer k. The vertex cover problem has a simple $O(n^k)$ algorithm. However, this problem has better algorithm with runtime $2^k n^{O(1)}$. If k is "small" this algorithm outperforms the $O(n^k)$ algorithm. The goal of parameterized complexity is to study the feasibility and intractability of parameterized problem. Developed by Downey and Fellows [44, 45] it is a thriving field of research. A *parameterized problem* is a language $L \subseteq \Sigma \times \mathbb{N}$. A parameterized problem L is *fixed parameter tractable* (FPT) if given input (x, k) there is an $f(k)n^{O(1)}$ algorithm to decide if $(x, k) \in L$, where f is some fixed function. Toda presented an FPT-algorithm for chordal graphs where the parameter is the maximum size of the simplicial components [103]. Based on Toda's approach, in Chapter 5 we present an FPT-algorithm for bounded color class hypergraph isomorphism with runtime $b!2^{O(b)}N^{O(1)}$ where the parameter b is the size of the largest color class and N is the input size.

Canonization or computing the canonical form of a graph is a problem closely related to the graph isomorphism problem. Let G be a graph class. A function f is said to be *complete invariant* for the class G if for all graphs X and Y in G, $f(X) = f(Y)$ if and only if $X \cong Y$. A complete invariant f is a *canonizing function* if $f(X) \cong X$. Gurevich proved that given a polynomial-time computable complete invariant, a canonical form can be computed in polynomial time [62].

Several algorithms for graph isomorphism for restricted classes of graphs work actually by computing a canonical form or sometimes a complete invariant. For example the linear-time tree isomorphism algorithm (see [3]) works by computing a complete invariant. This algorithm for tree isomorphism is basically a variant of a more general procedure called Weisfeiler-Lehmann algorithm[2] [111, 112]. The Weisfeiler-Lehmann algorithm has been remarkably successful for solving the graph isomorphism for several classes of graphs. Babai, Erdös and Selkow [26] result on isomorphism of random graphs works by finding the canonical form of the graphs. They showed that one dimensional Weisfeiler-Lehman can produce canonical form for all but $n^{-1/7}$ fraction of graphs on n vertices. Sometimes a process known as individualization followed by Weisfeiler-Lehman method can produce the canonical form of graphs, e.g., Zemlyachenko's $exp(c\sqrt{n \log n})$ algorithm for graph canonization algorithm [117]. Grohe and Verbitsky's [60] TC^1 algorithm for isomorphism of bounded tree width graphs also works with the help of Weisfieler-Lehman algorithm. Spielman's $n^{O(n^{1/3} \log n)}$ algorithm for isomorphism of strongly regular graphs also works by actually computing a canonical form. Babai and Luks's $n^{\log n}$ algorithm for tournament isomorphism is an example where the canonization not only exploits the combinatorial properties of the graph but also heavily uses the structure of the automorphism group of the graph.

In mathematics canonical forms are important objects. Canonical forms are important tools to recognize if two mathematical structures are "similar" under some transformation. For example Jordan canonical form for similar matrices, Hermite normal form for unimodular transformations, Smith normal form for matrices over principal ideal domains etc. It is clear that any polynomial-time computable canonizing function f for graphs would give a polynomial-time algorithm for graph isomorphism: Given two graphs X and Y, computing $f(X)$ and $f(Y)$ would be sufficient as by definition $f(X) = f(Y)$ if and only if $X \cong Y$. It is an interesting open question if the other direction is true, i.e., if canonical forms for graph can be computed in polynomial time with the help of the GRAPH-ISO oracle. In Chapter 4 we study the relative complexity of isomorphism and canonization for tournaments.

[2]Actually this more general procedure is called k-dimensional Weisfeiler-Lehmann algorithm (see [38])

1.1 Overview of the Thesis

Chapter2 – SZK Proofs For Black Box Group Problems. In this chapter we classify several algorithmic problems in group theory in the classes PZK and SZK (problems with perfect/statistical zero-knowledge proofs respectively). Prior to this, these problems were known to be in AM $\cap$ coAM. As PZK $\subseteq$ SZK $\subseteq$ AM $\cap$ coAM, we have a tighter upper bound for these problems. Specifically:

- We show that the permutation group problems Coset Intersection, Double Coset Membership, Group Conjugacy are in PZK. Further, the complements of these problems also have perfect zero knowledge proofs (in the liberal sense). We also show that permutation group isomorphism for solvable groups is in PZK. As an ingredient of this protocol, we design a randomised algorithm for sampling short presentations of solvable permutation groups.

- We show that the complement of all the above problems have concurrent zero knowledge proofs.

- We prove that the above problems for *black-box groups* are in SZK.

- Finally, we also show that some of the problems have SZK protocols with efficient provers in the sense of [86].

The results are reported in [10].

Chapter 3 – Limited Randomness Zero Knowledge. Arvind and Tóran [19] showed that group isomorphism problem where the groups are given by their multiplication tables have public coin, constant round interactive proof where the random bits used by the prover and verifier and the message size is polylogarithmic in the size of the input. In this chapter we note that this problem actually has perfect zero knowledge proofs where the prover and verifier use polylogarithmically many random bits and has constant number of rounds. This motivates us to study statistical zero knowledge proofs where the prover and verifier use polylogarithmic randomness and have polylogarithmically many rounds. We analyze the proofs of Goldreich, Sahai and Vadhan [94, 57] in this setting. They found two complete promise problems SD and ED for the class statistical zero knowledge and proved that any problem having statistical zero knowledge proof also has a honest verifier statistical zero knowledge proof with only two rounds. We show that similar results can be shown for statistical zero knowledge proofs which uses polylogarithmic randomness and has polylogarithmically many rounds. For example every problem having such statistical zero knowledge proof is polynomial-time many one reducible to a problem $\beta\mathrm{SD}^{(c)}$ which has a two round honest verifier statistical zero knowledge protocol where the randomness required is still polylogarithmic. We also prove that the protocol can be designed in such a way that the total length of message passed between the prover and the verifier is also polylogarithmic. We exhibit an honest verifier statistical zero knowledge proof with small randomness and 2 rounds for group isomorphism problem for groups given by their multiplication table as a corollary of this general treatment.

Chapter 4 – Isomorphism and Canonization of Tournaments and Hypertournaments In this chapter we study the relative complexity of tournament isomorphism and tournament canonization. We give a polynomial-time oracle algorithm for Tournament Canonization that accesses oracles for Tournament Isomorphism and Rigid-Tournament Canonization. Extending the Babai-Luks Tournament Canonization algorithm [30], we give an $n^{O(k^2+\log n)}$ algorithm for canonization and isomorphism testing of k-hypertournaments, where n is the number of vertices and k is the size of hyperedges.

These results are reported in [15].

Chapter 5 – An FPT Algorithm for Bounded Color Class Hypergraph Isomorphism. In this chapter we present an FPT algorithm for testing isomorphism of two colored hypergraphs X_1 and X_2 where the parameter is the size of the maximum color class. The run-time of our algorithm is $b!2^{O(b)}N^{O(1)}$ where b is the size of the largest color class and N is the input size. The algorithm uses coset intersection of groups whose orbits are bounded. Based on Luks's algorithm [83], we present an FPT algorithm for finding the coset intersection of groups whose orbits are bounded by parameter b.

The results of this chapter are based on [14].

Chapter 6 – Space Complexity of k-tree Isomorphism and Canonization. K-trees are natural generalization of trees [72]. A subgraph of a k-tree is a partial k-tree. The class of partial k-tree coincides with the class of graphs with treewidth bounded by k. In this chapter we look at the problem of (full) k-tree isomorphism. In this chapter we show that isomorphism testing of k-trees is in the class $\mathrm{StUL}(\log n)$ (strongly unambiguous logspace). This bound follows from a deterministic logspace algorithm that accesses a strongly unambiguous logspace oracle for canonizing k-trees. Further we give a logspace canonization algorithm for k-paths.

These results are reported in [12].

2
SZK Proofs for Black Box Group Problems

2.1 Introduction

In this chapter and the next we study the complexity of several group theoretic problems that are related to graph isomorphism. More specifically we prove that these problems have zero knowledge proof system.

Motivated by cryptography, zero knowledge proof systems were introduced by Goldwasser et al [58]. Zero knowledge protocols are a special kind of interactive proof systems in which the verifier gets no information other than the validity of the assertion claimed by the prover. The notion of zero knowledge is formalized by stipulating the existence of a randomized polynomial time *simulator* for a given protocol. The simulator, given some input of the protocol, outputs strings following a probability distribution indistinguishable from the verifier's view of the interaction between prover and verifier for the given input. The notion of indistinguishability can be further qualified. This leads to different notions of zero knowledge protocols. The protocol is *perfect zero knowledge* if the distribution produced by the simulator is identical to the verifier's view for all inputs. It is *statistical zero knowledge* if the two distributions have negligible statistical difference. Finally, the most liberal notion is *computational indistinguishability* where the simulator's output distribution cannot be distinguished from the verifier's view by polynomial-size circuits.

Natural problems like Quadratic Residuosity and Graph Isomorphism (GRAPH-ISO), their complements, a version of the discrete log problem are all known to have perfect zero-knowledge protocols. Some of these protocols have found cryptographic applications. For example, the Fiat-Shamir-Feige identification scheme is based on the ZK protocol for quadratic residuosity.

As a complexity class SZK is quite intriguing. It is closed under complement and is contained in AM$\cap$coAM. It is open if SZK coincides with AM$\cap$coAM. One approach to studying SZK is to explore for new natural problems that it contains. Goldreich, Vadhan, and Sahai [94, 57], investigating SZK, have shown that two natural promise problems, Statistical Difference (SD) and Entropy Difference (ED) are complete for SZK. We use these to exhibit several natural group-theoretic problems in SZK (and PZK) from the area of black-box groups studied by Babai and Szemerédi [23, 32]. It is shown in those papers that most of these black-box group problems are in NP $\cap$ coAM (some are in AM $\cap$ coAM).

In this chapter we put several group-theoretic problems for *permutation groups* in PZK, and for general *black-box groups* in SZK. The results of this chapter is based on [10][1]. We exhibit the zero knowledge proofs by a unified argument, showing that an appropriately defined group equivalence problem is reducible to Statistical Difference. One problem that requires a different technique is solvable permutation group isomorphism. Except for this problem all the other problems we consider in permutation group setting are harder than graph isomorphism problem and in the context of zero knowledge proof they share similar properties with graph isomorphism problem. In fact, as we will later see, that the properties of graph isomorphism problem that are essential for its zero knowledge proof is 'captured' by the group equivalence problem.

We further prove that the complement of some of the above problems has zero knowledge proofs in liberal sense and also has concurrent zero knowledge proofs.

Additionally, the group-theoretic NP problems that we consider also have zero-knowledge protocols with efficient provers (in the sense of Micciancio and Vadhan [86]). They have efficient proofs of knowledge as they are reducible to $\overline{\mathrm{SD}^{1,1/2}}$.

2.2 Preliminaries

We first recall some of the definitions about interactive protocol, zero knowledge etc. There are few minor variants of these definitions available in the literature. We use the

[1]After the publication of the conference version of [10], it was brought to our attention that related work on some permutation group problems was done in [107, 108].

definitions of interactive protocol, interactive proofs etc. from [106].

Definition 2.2.1 (Interactive Protocol) *An* interactive protocol *is a pair of functions* (P, V)*. On input* x*, the* interaction *between* P *and* V *is the following random process* $(P, V)(x)$*:*

1. *Uniformly choose random coins* r_P *and* r_V *(infinite binary strings) for* P *and* V *respectively.*
2. *Repeat the following for* $i = 1, 2, \cdots$
   (a) *If* i *is odd let* $m_i = P(x, m_1, \cdots, m_{i-1}; r_P)$*.*
   (b) *If* i *is even let* $m_i = V(x, m_1, \cdots, m_{i-1}; r_V)$*.*
   (c) *If* $m_i \in \{\texttt{accept, reject, halt}\}$ *then exit the loop.*

The string m_i *is called the* i*th message. If the last message form* P *is* `accept` *(resp.* `reject`*), we say that* P accepts *(resp.* rejects*), and similarly for* V*. We say that the protocol is* polynomially bounded *if there is a polynomial* $p(.)$ *such that, on common input* x*, at most* $p(|x|)$ *messages are exchanged and each is of length at most* $p(|x|)$ *(with probability* 1 *over the choice of* r_P *and* r_B*).*

Definition 2.2.2 *A* promise problem Π *is a pair* (Π_Y, Π_N) *of two disjoint sets* Π_Y *and* Π_N*. The set* Π_Y *is the set of "yes instances" and* Π_N *is the set of "no instances".*

A language L can be viewed as the promise problem $\Pi_L = (L, \Sigma^* - L)$, where Σ is the alphabet.

Definition 2.2.3 (Interactive Proofs IP) *Let* (P, V) *be an interactive protocol and let* $\Pi = (\Pi_Y, \Pi_N)$ *be a promise problem.* (P, V) *is said to be an* interactive proof system *for* Π *with* completeness error $c : \mathbb{N} \longrightarrow [0, 1]$ *and* soundness error $s : \mathbb{N} \longrightarrow [0, 1]$ *if the following conditions hold:*

1. *(Efficiency)* (P, V) *is polynomially bounded and* V *is polynomial-time computable.*
2. *(Completeness) If* $x \in \Pi_Y$*, then* V *accepts with probability at least* $1 - c(k)$ *in* $(P, V)(x, 1^k)$*.*

3. *(Soundness) If $x \in \Pi_N$, then for any P^*, V rejects with probability at least $1-s(k)$ in $(P^*, V)(x, 1^k)$.*

We require that $c(k)$ and $s(k)$ be computable in time $poly(k)$ and that $1-c(k) > s(k) + 1/poly(k)$. If $c(k) = 0$ for all k then we say that the system has perfect completeness. IP *is the class of all problems possessing interactive proofs.*

Definition 2.2.4 *Let (P, V) be an interactive protocol. Let $m_1, \cdots, m_t$ be all the messages exchanged between P and V on input x and let r be the substring of r_V containing all the random bits that V read during the interaction.*

- *V's* view *on common input x is the random variable $\langle P, V\rangle(x) = (m_1, \cdots, m_t; r)$.*
- *The* number of rounds *is t.*
- *The* message size *is $\sum_{i=1}^{t} |m_i|$.*

Definition 2.2.5 (Statistical Difference) *Let X and Y be two distributions on a finite set S. The* statistical difference *between X and Y is*

$$\text{StatDiff}(X, Y) = \frac{1}{2}\sum_{s\in S} |Pr(X = s) - Pr(Y = s)|.$$

A distribution X on a finite set S is said to be ε-uniform if $\frac{1}{|S|}(1-\varepsilon) \leq Pr[X = s] \leq \frac{1}{|S|}(1+\varepsilon)$. If X is ε-uniform on S then $\text{StatDiff}(X, U_S) \leq \varepsilon/2$, where U_S is the uniform distribution on S.

Definition 2.2.6 *A function $f : \mathbb{N} \to \mathbb{N}$ is* negligible *if for all polynomial $p : \mathbb{N} \to \mathbb{N}$ there exists n_0 such that for all $n > n_0$ $f(n) < \frac{1}{p(n)}$.*

Definition 2.2.7 *An interactive proof system (P, V) for a promise problem $\Pi = (\Pi_Y, \Pi_N)$ is said to be* statistical zero-knowledge SZK *if for every randomized polynomial-time interactive machine V^*, there is a probabilistic polynomial-time algorithm M^* and a negligible function μ such that for all $x \in L$ and $k > 0$ the following two conditions hold:*

1. *The machine M^* is* useful, *i.e., on input $(x, 1^k)$, M^* fails (and outputs* fail*) with probability at most $\frac{1}{2}$.*

2. *Let $m^*(x, 1^k)$ be the random variable describing the distribution of $M^*(x, 1^k)$ conditioned on $M^*(x, 1^k) \neq fail$. Let $\langle P, V\rangle(x, 1^k)$ be the* view *i.e., the messages passed between P and V during the execution of the protocol and V's random bits. Then*
$$\lambda = \text{StatDiff}(m^*(x, 1^k), \langle P, V\rangle(x, 1^k)) \leq \mu(k).$$

When $\lambda = 0$ for all $x \in L$ then the protocol is perfect zero-knowledge PZK.

Definition 2.2.8 *A promise problem $\Pi = (\Pi_Y, \Pi_N)$ is in the class* honest verifier statistical zero knowledge HVSZK *if there is an interactive proof system (P, V) such that there is a negligible function μ and a useful polynomial-time randomized algorithm S such that if $x \in \Pi_Y$ then on input $(x, 1^k)$,*

$$\text{StatDiff}(\overline{S}(x, 1^k), < P, V > (x, 1^k)) \leq \mu(k),$$

where $\overline{S}(x, 1^k)$ is the output distribution of S on input $(x, 1^k)$ conditioned on $S(x, 1^k) \neq fail$.

If $\mu(k) = 0$ for all k then Π is said to be in the class honest verifier perfect zero knowledge HVPZK.

A boolean circuit $X : \{0, 1\}^m \longrightarrow \{0, 1\}^n$ induces a distribution on $\{0, 1\}^n$ by the evaluation $X(x)$ where $x \in \{0, 1\}^m$ is picked uniformly at random. We use X to denote this distribution encoded by circuit X.

Definition 2.2.9 (SD$^{\alpha,\beta}$) *For constants $0 \leq \beta < \alpha \leq 1$, the promise problem* $\text{SD}^{\alpha,\beta} = (\text{SD}_Y^{\alpha,\beta}, \text{SD}_N^{\alpha,\beta})$ *takes as input distributions X and Y given by circuits, and has "yes instances"* $\text{SD}_Y^{\alpha,\beta}$ *and "no instances"* $\text{SD}_N^{\alpha,\beta}$ *defined as follows.*

$$\text{SD}_Y^{\alpha,\beta} = \{(X_1, X_2) \mid \text{StatDiff}(X_1, X_2) \geq \alpha\},$$
$$\text{SD}_N^{\alpha,\beta} = \{(X_1, X_2) \mid \text{StatDiff}(X_1, X_2) \leq \beta\}.$$

The problem $\text{SD}^{2/3,1/3}$ *is called the* statistical difference problem *and will be often denoted as* SD.

The above problem is defined in terms of statistical difference. The *entropy difference problem* is a similar problem defined in terms of entropy difference [57]. We recall the definition of entropy below.

Definition 2.2.10 (Entropy/Shannon Entropy) *Let X be a distribution on a finite set S. The* entropy *of X, denoted* $\mathrm{H}(X)$ *is defined as*

$$\mathrm{H}(X) = \sum_{s \in S} Pr[X = s] \log \frac{1}{Pr[X = s]}.$$

Definition 2.2.11 (ED**)** *The promise problem*

$$\mathrm{ED} = (\mathrm{ED}_Y, \mathrm{ED}_N)$$

takes two distributions X and Y encoded by boolean circuits as input and has "yes instances" ED_Y *and "no instances" ED_N defined as follows:*

$$\begin{aligned} \mathrm{ED}_Y &= \{(X_1, X_2) \mid \ \mathrm{H}(X_1) \geq \mathrm{H}(X_2) + 1\}. \\ \mathrm{ED}_N &= \{(X_1, X_2) \mid \ \mathrm{H}(X_2) \geq \mathrm{H}(X_1) + 1\} \end{aligned}$$

We recall some important results from [94, 56, 57, 106].

Theorem 2.2.12 (Goldreich-Sahai-Vadhan) [94, 57]

(i) $\mathrm{SD}^{2/3,1/3}$ *and* ED *are complete for* SZK.

(ii) SZK *is closed under complement.*

(iii) HVSZK $=$ SZK.

(iv) The complement of the problem $\mathrm{SD}^{1,0}$*, i.e.,* $\overline{\mathrm{SD}^{1,0}}$ *is in* PZK.

We recall some basic group theory. Let G be a finite group and Ω a finite nonempty set. The action of the group G on the set Ω is defined by a map $\alpha : \Omega \times G \longrightarrow \Omega$ such that for all $x \in \Omega$ (i) $\alpha(x, id) = x$, i.e., the identity $id \in G$ fixes each $x \in \Omega$, and (ii) $\alpha(\alpha(x, g_1), g_2) = \alpha(x, g_1 g_2)$ for $g_1, g_2 \in G$,. We write x^g instead of $\alpha(x, g)$ when the group action is clear from the context.

Definition 2.2.13 *Let G be a finite group acting on a finite set Ω. For $x \in \Omega$, its G-*orbit *x^G is the set*

$$\{y | y \in X, y = x^g \text{ for some } g \in G\}.$$

Notice that Ω is partitioned *into orbits. When the group is clear from the context x^G is called the orbit of x.*

We write $H \leq G$ when H is a subgroup of G. The *symmetric group* on a finite set Ω consisting of all permutations on Ω is denoted by $\mathrm{Sym}(\Omega)$. If $\Omega = [n] = \{1, \cdots, n\}$ we write S_n instead of $\mathrm{Sym}([\mathrm{n}])$. A *finite permutation group* G is a subgroup of $\mathrm{Sym}(\Omega)$ for some Ω.

The permutation group *generated* by a subset $S \subseteq \mathrm{Sym}(\Omega)$ is the smallest subgroup of $\mathrm{Sym}(\Omega)$ containing S and is denoted by $\langle S \rangle$. Each element of the group $\langle S \rangle$ is expressible as a product of elements of S.

The subgroup $G^{(i)}$ of $G \leq S_n$ that fixes each of $\{1, \ldots, i\}$ is a *pointwise stabilizer* subgroup. We have a tower of subgroups for G

$$G = G^{(0)} \geq G^{(1)} \geq G^{(2)} \geq \cdots \geq G^{(n-1)} = \{id\}.$$

It is not difficult to see that the index $[G^{(i-1)} : G^{(i)}]$ is at most n. Let R_i be a set of complete and distinct coset representatives of $G^{(i)}$ in $G^{(i-1)}$ for each i. Then $\bigcup_{i=1}^{n-1} R_i$ generates G and is known as a *strong generating set* for G. Given a permutation $\pi \in G$ it is easy to check if $\pi \in G^{(i)}$. It is also easy to check if two permutations $\pi, \sigma \in G^{(i)}$ are in the same coset of $G^{(i+1)}$ in $G^{(i)}$. We just have to check if $\pi^{-1}\sigma \in G^{(i+1)}$. These observations yield a polynomial-time algorithm [98, 99, 53] to compute a strong generating set of a permutation group G which can be used to test in polynomial time if $g \in S_n$ is in the group $\langle S \rangle \leq S_n$. Finally, it is sometimes the case that there is an efficient algorithm for testing membership in a subgroup H of G where $G \leq S_n$ is given by a generating set, but no generating set for H is given. By [98, 99, 53] we can efficiently compute a generating set for H if its index in G is polynomially bounded.

Theorem 2.2.14 (Schreier Generators) *Let $G = \langle S \rangle \leq S_n$, $H \leq G$. Let R be a set of coset representatives of H in G. Then*

$$B = \{r'xr^{-1} \mid r, r' \in R, x \in S\} \cap H$$

generates H. The generators in B are called Schreier generators.

The algorithm in the above theorem computes a set of coset representatives R for the subgroup H of $G = \langle S \rangle$. The algorithm makes $m^2|S|$ tests of membership in H where $m = [G : H]$. The set B of Schreier generators for H can be of size polynomial in m. We can convert it to an $O(n^2)$ size generating set [98, 99, 53].

A subgroup H of G is *normal* in G, denoted $H \triangleright G$, if for all $g \in G$ and $h \in H$, $g^{-1}hg \in H$. The idenity subgroup and G are both trivial normal subgroups of G. A group G is simple if it has no nontrival normal subgroup. The subgroup generated by $\{xyx^{-1}y^{-1} \mid x, y \in G\}$ is the commutator subgroup G' of G. Recall that G' is the unique smallest normal subgroup of G such that G/G' is commutative. The derived series of G is $G \triangleright G' \triangleright G'' \triangleright \cdots$. We say G is *solvable* if this series terminates in $\{id\}$. There are polynomial-time algorithms to compute the derived series and to test solvability for permutation groups G given by generating sets (see e.g. [82]).

A *composition series* of G is a tower of subgroups $\{id\} = G_1 \triangleleft G_2 \triangleleft \cdots \triangleleft G_m = G$ such that G_i/G_{i+1} is *simple* for each i. Recall that G is solvable iff G_i/G_{i+1} is cyclic of prime order for each i in any composition series for G.

2.3 Chapter Overview

1. In Section 2.4 we

    - Define a problem PGE.
    - Prove that PGE polynomial-time many-one reduces to $\overline{\mathrm{SD}^{1,0}}$.

    $$\mathrm{PGE} \leq_m^{\mathrm{P}} \overline{\mathrm{SD}^{1,0}}.$$

    - Define permutation group problems SUBSP-TRANS and CONJ-GROUP. Prove that they are polynomial-time reducible to PGE.

    $$\text{SUBSP-TRANS} \leq_m^{\mathrm{P}} \mathrm{PGE},$$

    $$\text{CONJ-GROUP} \leq_m^{\mathrm{P}} \mathrm{PGE}.$$

    Since several permutation group problems are reducible to either SUBSP-TRANS or CONJ-GROUP the above reductions imply that they are in PZK.

2. In Section 2.5 we prove $\overline{\mathrm{PGE}}$ has perfect zero knowledge proof in liberal sense.

3. In Section 2.6 we prove that the solvable permutation group isomorphism problem is polynomial-time many-one reducible to $\overline{\mathrm{SD}^{1,0}}$.

4. In Section 2.7 we define the black box group problem GE. Prove

$$\mathrm{GE} \leq_m^{\mathrm{P}} \overline{\mathrm{SD}^{1,1/3}}.$$

   We show that several black box group problems are polynomial-time many-one reducible to GE. As a consequence we deduce that they are in SZK.

5. In Section 2.8 we prove that several permutation group problems have concurrent zero knowledge proofs.

6. In Section 2.9 we prove that several black box group problems have SZK proofs with efficient prover.

Figure 2.1 we schematically represent the reductions among several problems. For the reductions among the permutation group problems see [82]. The problem of deciding if two groups given by their multiplication tables are isomorphic or not is denoted as GrIso. We will study this problem in the next chapter.

2.4 Group Problems in PZK

In this section we show that various permutation group problems (not known to be in P) are in PZK. Examples of such problems are Coset Intersection, Double Coset Membership, Conjugate Subgroups etc. We define these problems below (see [82] for more details). These are all problems known to be harder than GRAPH-ISO.

We prove the membership of all these problems in PZK by one general result. We define a generic problem called *Permutation Group Equivalence* PGE and show that PGE is polynomial-time many-one reducible to $\overline{\mathrm{SD}^{1,0}}$. As $\overline{\mathrm{SD}^{1,0}} \in$ PZK it follows that PGE $\in$ PZK. The problem PGE is generic in the sense that all considered problems, except the isomorphism problem, about permutation groups are polynomial-time many-one reducible to PGE. Thus, it follows as a corollary that all these problems are in PZK.

Permutation group isomorphism problem requires a different approach. In fact, in this chapter we show only for solvable groups that this problem is in PZK.

Recall that instances of $\overline{\mathrm{SD}^{1,0}}$ are pairs of circuits (X_0, X_1), where $X_i : \{0,1\}^n \longrightarrow \{0,1\}^m$ defines a probability distribution on $\{0,1\}^m$, for uniformly distributed input in

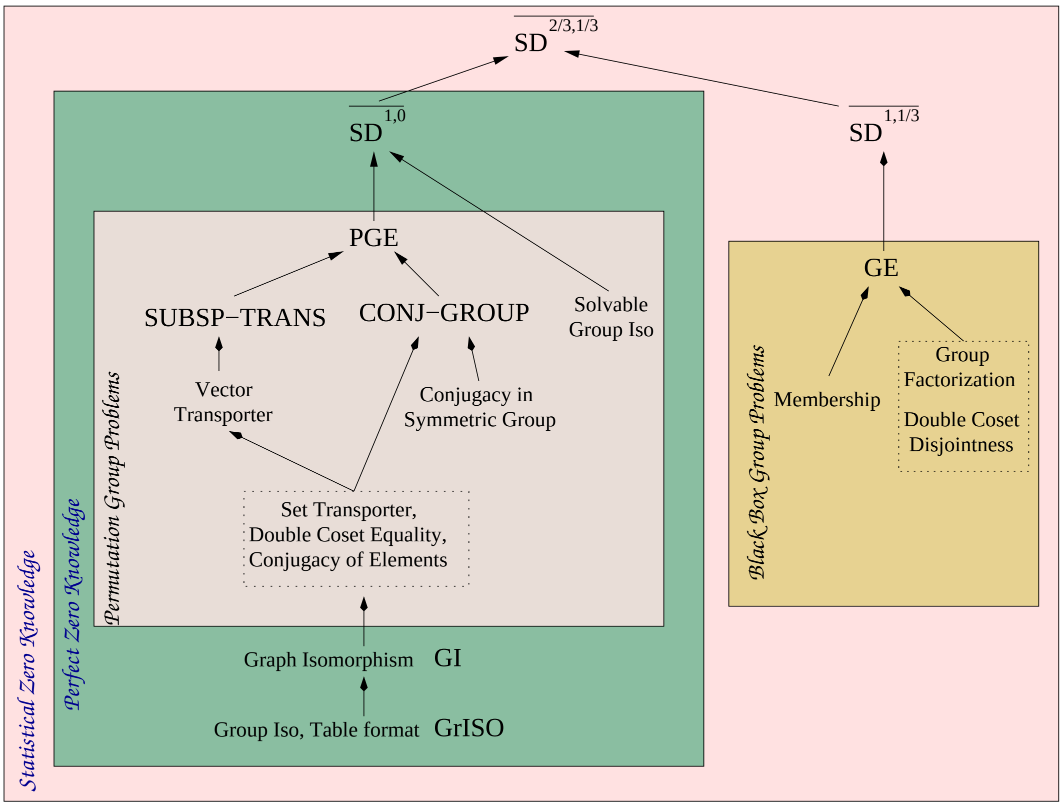
$\overline{SD}^{2/3,1/3}$
$\overline{SD}^{1,0}$
$\overline{SD}^{1,1/3}$
PGE
SUBSP−TRANS
CONJ−GROUP
Solvable Group Iso
Vector Transporter
Conjugacy in Symmetric Group
Set Transporter, Double Coset Equality, Conjugacy of Elements
Graph Isomorphism GI
Group Iso, Table format GrISO
GE
Membership
Group Factorization
Double Coset Disjointness
Statistical Zero Knowledge
Perfect Zero Knowledge
Permutation Group Problems
Black Box Group Problems

Figure 2.1: Schematic Diagram of Reductions.

$\{0,1\}^n$. The promise is that the statistical difference between X_0 and X_1 is 0 or 1 and the problem is to determine which is the case. This is known to be in PZK [106].

Definition 2.4.1 *The* Permutation Group Equivalence Problem PGE *has inputs consisting of 4-tuples* (x, y, T, τ)*. Here* $T \subset S_n$*. Let* $G = \langle T \rangle$ *. Then* $\tau : G \times S \longrightarrow S$ *is a polynomial-time computable group action of* G *on* S*, for some* $S \subseteq \{0,1\}^m$*, where* $m = n^{O(1)}$*. More precisely, given* $g \in G$ *and* $s \in S$*, the image* $s^{\tau(g)}$ *of* s *under* g *(which we denote simply by* s^g*) is polynomial-time computable. Finally,* $x, y \in S$*.*

The PGE *problem is the following* promise problem*: given an input 4-tuple* (x, y, T, τ) *with the promise that* $x, y \in S$ *and that* τ *defines a group action of* $\langle T \rangle$ *on* S*, the problem is to decide if* x *and* y *are in the same* G*-orbit. I.e. is* $x^g = y$ *for some* $g \in G$*?*

Theorem 2.4.2 PGE *is polynomial-time many-one reducible to* $\overline{\mathrm{SD}^{1,0}}$.

Proof. Let (x, y, T, τ) be an input instance of PGE such that $x, y \in S$ and $S \subseteq \{0,1\}^m$. Define two circuits $X_{x,T}, X_{y,T} : \{0,1\}^k \longrightarrow \{0,1\}^m$, where k is polynomial in n to be fixed later. In the sequel we assume that it is possible to uniformly pick a random element from the set $[i]$ for each positive integer i given in unary.[2] The circuit $X_{x,T}$ on input which is a random string $r \in \{0,1\}^k$ will use r to randomly sample an element from the group $G = \langle T \rangle$. This is a polynomial-time procedure based on the Schreier-Sims algorithm for computing a strong generating set $\bigcup_{i=1}^{n-1} R_i$ for G, where R_i is a complete set of distinct coset representatives of $G^{(i)}$ in $G^{(i-1)}$ for each i. Then we can sample from G uniformly at random by picking $x_i \in R_i$ uniformly at random and computing their product $g_r = x_1 x_2 \cdots x_{n-1}$. The circuit $X_{x,T}$ then outputs x^{g_r}. By construction, x^{g_r} is uniformly distributed in the G-orbit of x. Likewise the other circuit $X_{y,T}$ will output a uniformly distributed element of the G-orbit of y. Since G defines a group action on S, the two orbits are either disjoint or identical. In particular, the orbits are identical if and only if $x = y^g$ for some $g \in G$. Thus, the statistical difference between $X_{x,T}$ and $X_{y,T}$ is 0 or 1 depending on whether $x = y^g$ for some $g \in G$ or not. This proves the theorem. ∎

We show that several permutation group problems are reducible to PGE. There is a table of reductions for permutation group problems in Luks' article [82]. It suffices to show that the following two "hardest" problems from that table are reducible to PGE (apart from permutation group isomorphism which we consider in the next section).

[2] This assumption is required even for the PZK protocol for GRAPH-ISO.

Definition 2.4.3 *The* Subspace Transporter Problem SUBSP-TRANS *has input consisting of a subgroup G of S_n given by generating set T, a representation $\tau : G \longrightarrow GL(\mathbb{F}_q^m)$, and subspaces $W_1, W_2 \subseteq \mathbb{F}_q^m$ given by spanning sets. The question is whether $W_1^g = W_2$ for some $g \in G$. Here the size q of the finite field is a constant.*

Notice here that by W_1^g we mean the image of the subspace W_1 under the matrix $\tau(g)$.

Definition 2.4.4 *The* Conjugacy of Groups Problem CONJ-GROUP *has inputs consisting of three permutation groups G, H_1, H_2 in S_n, given by generating sets. The question is whether there is a $g \in G$ such that $H_1^g = H_2$ (where $H_1^g = g^{-1}H_1g$).*

Lemma 2.4.5

(a) Let $\mathbb{F}_q$ be a fixed finite field. Given as input $X \subset \mathbb{F}_q^n$, there is a polynomial-time algorithm $\mathcal{A}$ that computes a canonical basis *B of the subspace W spanned by X. The output is canonical in the following sense: if $\mathcal{A}$ is given any spanning set of W as input, the output of $\mathcal{A}$ will be B.*

(b) Given as input $X \subset S_n$, there is a polynomial-time algorithm $\mathcal{A}$ that computes a canonical generating set *B of the subgroup G generated by X. The output is canonical in the sense that $\mathcal{A}$ will output B, given any generating set X' of G as input.*

Proof. First we prove (a). Order the elements of $\mathbb{F}_q$ lexicographically. First, we search for the least i such that there is a vector $(v_1, \ldots, v_n) \in W$ with $v_i = 1$ (Notice that $v_1, \ldots v_{i-1}$ have to be zero for all elements of W). For this we can use a polynomial-time algorithm for testing feasibility of linear equations over $\mathbb{F}_q$. Having found i, we search for the least $v_{i+1} \in \mathbb{F}_q$ such that $v_i = 1$. Since q is a constant we can do this search with a constant number of similar feasibility tests. After finding the least v_{i+1} we fix it and search similarly for the least v_{i+2} and so on. Continuing thus, we can compute the lex least nonzero element $\mathbf{u_1}$ in W. Next, in order to find a basis we look for the least index $j > i$ such that there is a nonzero vector $(v_1, \ldots, v_n) \in W$ with $v_1 = v_2 = \ldots = v_{j-1} = 0$ and $v_j = 1$ again by $O(n)$ feasibility tests. After finding j, we can again pick the lex least nonzero vector $\mathbf{u_2}$ with the property that the jth index is the least nonzero coordinate in $\mathbf{u_2}$. Continuing in this manner, we will clearly obtain a basis $\{\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_k}\}$ of W. By our construction this basis is canonical.

Now we prove (b). The algorithm $\mathcal{A}$ will compute a strong generating set from X for the group G using the Schreier-Sims algorithm [82]. Then using the fact that the lex least element of a coset xH (where $x \in S_n$ and $H \leq S_n$) can be computed in polynomial time [18] we can replace each coset representative in the strong generating set by the lex least coset representative. This generating set is canonical by construction. ■

Theorem 2.4.6 *The problems* SUBSP-TRANS *and* CONJ-GROUP *are polynomial-time many-one reducible to* PGE.

Proof. We first consider SUBSP-TRANS. Let (T, S_1, S_2, π) be an input. Let $G = \langle T \rangle$ and $S_1, S_2 \subset \mathbb{F}_q^m$ be spanning sets of W_1 and W_2 respectively. The representation is given by $\pi : G \longrightarrow GL(\mathbb{F}_q^m)$. The reduction from SUBSP-TRANS to PGE maps (T, S_1, S_2, π) to (x, y, T, τ) where x and y are the canonical bases for W_1 and W_2 respectively, in the sense of lemma 2.4.5. The set S in definition 2.4.1 corresponds to the set of canonical bases of all possible subspaces of $\mathbb{F}_q^m$. The group action τ is the algorithm that given $B \in S$ and $g \in G$, first computes the set of vectors $\pi(g)(B)$. Next, using the algorithm in Lemma 2.4.5, τ computes the canonical basis of subspace spanned by $\pi(g)(B)$.

The reduction is similar for CONJ-GROUP. Let (T, S_1, S_2) be an instance of CONJ-GROUP, where T, S_1 and S_2 generate G, H_1, and H_2 respectively. The reduction maps (T, S_1, S_2) to (x, y, T, τ) where x and y are the canonical strong generating sets for H_1 and H_2 respectively in the sense of Lemma 2.4.5. The set S in definition 2.4.1 is the set of canonical strong generating sets for all subgroups of S_n. The group action τ is the algorithm that given $B \in S$ and $g \in G$, applies the algorithm $\mathcal{A}$ in lemma 2.4.5 to compute the canonical generating set for the subgroup generated by $\{g^{-1}xg \mid x \in B\}$. ■

Since SUBSP-TRANS and CONJ-GROUP are harder than several permutation group problems w.r.t. polynomial time many-one reductions [82] we have the following.

Corollary 2.4.7 *The problems of Set Transporter, Coset Intersection, Double Coset Membership, Double Coset Equality, Conjugacy of Elements, Vector Transporter etc are all in* PZK.

2.5 Group Nonequivalence and PZK in liberal sense

It is not known (to the best of our knowledge) if SD^{1,0} is in PZK. However, we observe here that we need the following liberal definition of PZK, because only such PZK protocols are known for even problems like GRAPH-NONISO and Quadratic Nonresiduosity.

We show that Permutation Group Nonequivalence $\overline{\text{PGE}}$ is in PZK in the liberal sense. Consequently, the complement of problems considered in Corollary 2.4.7 and Theorem 2.4.6 are also in PZK (liberal sense).

Definition 2.5.1 (PZK (liberal sense)) [58] *An interactive protocol* (P, V) *is* perfect zero knowledge *if for every probabilistic polynomial time interactive machine* V^* *there exists an expected polynomial-time algorithm* M^* *such that for every* $x \in L$ *the random variable* $\langle P, V^* \rangle(x)$ *and* $M^*(x)$ *are identically distributed.*

Notice that in this definition [58] of PZK the simulator is required to be an *expected polynomial time* algorithm that always outputs some legal transcript. The definition we used in Section 2.4 is more stringent. It requires the simulator to be worst case polynomial time though it may sometimes fail.

A common generalization of several problems in PZK (liberal sense) is $\overline{\text{PGE}}$.

Theorem 2.5.2 $\overline{\text{PGE}}$ *is in* PZK *in liberal sense.*

The proof of this theorem is similar to [55] and some of the techniques are from [54]. We first describe the PZK protocol.

2.5.1 Protocol

Input (x_0, x_1, T, τ) , where $T \subseteq S_n$ generates the group G, $x_0, x_1 \in S$ and τ is a polynomial time computable group action of G on S.

V1 The verifier chooses b randomly from $\{0, 1\}$, g randomly from $G = \langle T \rangle$. Let $s = x_b^g$. The prover is supposed to tell b given s. The verifier also chooses $g_j^{(i)}$ randomly from G and $b_i \in \{0, 1\}$ for $j = 0, 1$ and $i = 1, 2, \cdots, k$. Calculate $s_0^{(i)} = x_{0 \oplus b_i}^{g_0^{(i)}}$ and $s_1^{(i)} = x_{1 \oplus b_i}^{g_1^{(i)}}$. The verifier sends s and the pairs $(s_0^{(i)}, s_1^{(i)})$ for $i = 1, 2, \cdots k$. The prover uses these pairs to check that the verifier is not cheating. We will later fix k.

P1 The prover chooses a random subset I of $[k]$ and sends it to the verifier.

V2 The verifier supplies $g_j^{(i)}$ and b_i for all $i \in I$ and $j = 0, 1$. For all $i \in \overline{I}$, he sends (d_i, h_i) where $d_i = b \oplus b_i$ and $h_i = g^{-1} g_{b \oplus b_i}^{(i)}$. Thus, for an honest verifier $s_{d_i}^{(i)} = s^{h_i}$. (Note that $x^{g_1 g_2} = (x^{g_1})^{g_2}$)

P2 If $s_0^{(i)} \neq x_{b_i}^{g_0^{(i)}}$ or $s_1^{(i)} \neq x_{1 \oplus b_i}^{g_1^{(i)}}$ for any $i \in I$ or $s_{d_i}^{(i)} \neq s^{h_i}$ for any $i \in \overline{I}$ then *reject* the input. Otherwise send β such that there exists g'' such that $s = x_\beta^{g''}$. If no such β exists, then send $\beta = 0$.

V3 If $b = \beta$ *accept*, else *reject*.

Lemma 2.5.3 *The above protocol is an interactive protocol with completeness error* 0 *and soundness error less than or equal to* $1/2$.

If $(x_0, x_1, G, S) \in \overline{\text{PGE}}$ then x_0 and x_1 are in different orbits. Thus, if s is in the orbit of one of them then it cannot be in the orbit of the other element. Hence, the prover will be able to tell from which distribution s comes. Thus the verifier will accept.

If $(x_0, x_1, G, S) \in \text{PGE}$ then the two x_0 and x_1 are in the same orbit. Thus, if s can come from one of the elements it can come from the other element as well. Thus any prover can at best guess b. We prove this formally as follows.

Let $\mathcal{B}$ be a random variable that takes values randomly and uniformly from $\{0, 1\}$. Let $\mathcal{E}$ be a random variable that takes values randomly and uniformly from $G \times (G \times G \times \{0, 1\})^k$. Let f be a function whose domain is $\{0, 1\} \times G \times (G \times G \times \{0, 1\})^k \times \mathcal{P}([k])$, where $\mathcal{P}([k])$ is the power set of $[k]$. The function on the value $(b, g, \{(g_0^{(i)}, g_1^{(i)}, b_i)\}_{i=1}^k, I)$ produces the value

$$\begin{array}{ll} x_b^g, \{(x_{b_i}^{g_0^{(i)}}, x_{1 \oplus b_i}^{g_0^{(i)}})\}_{i=1}^k, & [(g_0^{(i_1)}, g_1^{(i_1)}, b_{i_1}), (g_0^{(i_2)}, g_1^{(i_2)}, b_{i_2}), \cdots, (g_0^{(i_p)}, g_1^{(i_p)}, b_{i_1})], \\ [(b \oplus b_{j_1}, g^{-1} g_{b \oplus b_{j_1}}^{(j_1)}), & (b \oplus b_{j_2}, g^{-1} g_{b \oplus b_{j_2}}^{(j_2)}), \cdots, (b \oplus b_{j_q}, g^{-1} g_{b \oplus b_{j_q}}^{(j_q)})] \end{array}$$

where $I = \{i_1, i_2, \cdots, i_p\}$ and $\overline{I} = \{j_1, j_2, \cdots, j_q\}$. So, $f(\mathcal{B}, \mathcal{E}, I)$ is a random variable which behaves in the same manner as the verifier V would have done on its random coins $(b, g, \{(g_0^{(i)}, g_1^{(i)}, b_i)\}_{i=1}^k)$ and index set $I \subseteq [k]$ sent by the prover.

Claim 2.5.4

$$Pr[\mathcal{B} = 0 | f(\mathcal{B}, \mathcal{E}, I) = Tr] = Pr[\mathcal{B} = 1 | f(\mathcal{B}, \mathcal{E}, I) = Tr]$$

where

$$\begin{aligned} Tr &= [s, \{(s_0^{(i)}, s_1^{(i)})\}_{i=1}^k, \\ &\quad [(h_0^{(i_1)}, h_1^{(i_1)}, e_{i_1}), (h_0^{(i_2)}, h_1^{(i_2)}, e_{i_2}), \cdots, (h_0^{(i_p)}, h_1^{(i_p)}, e_{i_p})] \\ &\quad [(d_{j_1}, h_{j_1}), (d_{j_2}, h_{j_2}), \cdots, (d_{j_q}, h_{j_q})]] \end{aligned}$$

is in the range of the f, $I = \{i_1, i_2, \cdots, i_p\}$ and $\overline{I} = \{j_1, j_2, \cdots, j_q\}$. (Here the probability is taken over $\mathcal{B}$ and $\mathcal{E}$ and not on the index set I. The claim is true of any index set I).

Proof. Let $A_{0,I} = \{(g, \{(g_0^{(i)}, g_1^{(i)}, b_i)\}_{i=1}^k) | f(0, g, \{(g_0^{(i)}, g_1^{(i)}, b_i)\}_{i=1}^k, I) = Tr\}$
and $A_{1,I} = \{(g, \{(g_0^{(i)}, g_1^{(i)}, b_i)\}_{i=1}^k) | f(1, g, \{(g_0^{(i)}, g_1^{(i)}, b_i)\}_{i=1}^k, I) = Tr\}$. We will prove that $|A_{0,I}| = |A_{1,I}|$. Assuming this,

$$\begin{aligned} Pr[f(\mathcal{B}, \mathcal{E}, I) = Tr | \mathcal{B} = 0] &= Pr[f(0, \mathcal{E}, I) = Tr] \\ &= Pr[\mathcal{E} \in A_{0,I}] \\ &= Pr[\mathcal{E} \in A_{1,I}] \\ &= Pr[f(\mathcal{B}, \mathcal{E}, I) = Tr | \mathcal{B} = 1]. \end{aligned}$$

Now applying Bayes' theorem, we get the claim.

Now we prove that $|A_{0,I}| = |A_{1,I}|$. As x_0 and x_1 are in the same orbit, there exists $\gamma \in G$ such that $x_1^\gamma = x_0$. We note that $g \mapsto \gamma g$, $(g_0^{(i)}, g_1^{(i)}, b_i) \mapsto (g_0^{(i)}, g_1^{(i)}, b_i)$ for all $i \in I$ and for all $j \in \overline{I}$

$$\begin{aligned} (g_0^{(j)}, g_1^{(j)}, 0) &\longmapsto (\gamma g_0^{(j)}, \gamma^{-1} g_1^{(j)}, 1) \\ (g_0^{(j)}, g_1^{(j)}, 1) &\longmapsto (\gamma^{-1} g_0^{(j)}, \gamma g_1^{(j)}, 0) \end{aligned}$$

is a bijection from $A_{0,I}$ to $A_{1,I}$. □

Let $\mathcal{R}$ be a random process that on input some $f(\mathcal{B}, \mathcal{E}, I) = Tr$ produces a number. $\mathcal{R}$ acts as a prover P^*. As the prover tries to find b from Tr, $\mathcal{R}$ does so, too. The verifier accepts if the prover can correctly supply him the b that he chooses in step **V1**. This corresponds to the event $\mathcal{R}(f(\mathcal{B},\mathcal{E},I)) = \mathcal{B}$. Hence, we calculate $Pr[\mathcal{R}(f(\mathcal{B},\mathcal{E},I)) = \mathcal{B}]$.

$$Pr[\mathcal{R}(f(\mathcal{B},\mathcal{E},I)) = \mathcal{B}] = \sum_{Tr} Pr[\mathcal{R}(Tr) = \mathcal{B} | f(\mathcal{B},\mathcal{E},I) = Tr] \cdot Pr[f(\mathcal{B},\mathcal{E},I) = Tr].$$

Now,

$$\begin{aligned} Pr[\mathcal{R}(Tr) = \mathcal{B} | f(\mathcal{B},\mathcal{E},I) = Tr] &= \sum_{b} Pr[\mathcal{R}(Tr) = b \wedge \mathcal{B} = b | f(\mathcal{B},\mathcal{E},I) = Tr] \\ &= \sum_{b} Pr[\mathcal{R}(Tr) = b] Pr[\mathcal{B} = b | f(\mathcal{B},\mathcal{E},I) = Tr] \\ &= 1/2 \sum_{b} Pr[\mathcal{R}(f(\mathcal{B},Tr,I)) = b] \\ &\leq 1/2. \end{aligned}$$

Thus, $Pr[\mathcal{R}(f(\mathcal{B},\mathcal{E},I)) = \mathcal{B}] \leq 1/2$. This proves that the soundness error is $\leq 1/2$ completing Lemma 2.5.3.

2.5.2 Simulator

We will prove that for any probabilistic polynomial time verifier V^* there exists an expected time simulator M_{V^*} that produces the same distribution as the protocol does. Let the input length be n. Let $q(n)$ be a polynomial bound on the running time of V^*.

S1 Choose w randomly from $\{0,1\}^{q(n)}$. Simulate V^* on the (x_0, x_1, G, S) using the random coins w. Let V^* send $(g, \{(s_0^{(i)}, s_1^{(i)}\}_{i=1}^{k})$ to the prover.

S2 Choose a random subset I of $[k]$. Use it as the verifier's next input. Suppose the verifier sends $\{(g_0^{(i)}, g_1^{(i)}, b_i)\}_{i\in I}$ and $\{(d_i, h_i)\}_{i\in\overline{I}}$. The simulator checks if $s_0^{(i)} = x_{b_i}^{g_0^{(i)}}$ or $s_1^{(i)} = x_{1\oplus b_i}^{g_1^{(i)}}$ for all $i \in I$ or $s_{d_i}^{(i)} = s^{h_i}$ for all $i \in \overline{I}$. If any of these conditions is violated then the simulator rejects and produces an output accordingly. Else step **S3**.

S3 Choose a random subset K of $[k]$. Again simulate V^* with the same random coin w. But now as the next input (i.e., corresponding to step **P1**), send K to V^*. Suppose now, V^* outputs $\{(\tilde{g}_0^{(i)}, \tilde{g}_1^{(i)}, \tilde{b}_i)\}_{i\in K}$ and $\{(\tilde{d}_i, \tilde{h}_i)\}_{i\in\overline{K}}$.

S3.1 In this step M_{V^*} tries to find the correct orbit of s.

Let for some $i \in I \cap \overline{K}$, $\tilde{d}_i = 0$ and $s^{\tilde{h}_i} = s_0^{(i)}$. But as $s_0^{(i)} = x_{b_i}^{g_0^{(i)}}$ (because M_{V^*} has passed step **S2**), we must have $s \sim x_{b_i}$. So, M_{V^*} sets $\beta = b_i$. Similarly, if $\tilde{d}_i = 1$ and $s^{\tilde{h}_i} = s_1^{(i)}$, M_{V^*} sets $\beta = \overline{b_i}$. The case for $i \in \overline{I} \cap K$ and analogous conditions can also be dealt with similarly.

If no β is found go to step **S3** otherwise go to step **S4**.

S3.2 In parallel find the correct orbit of s using brute force algorithm. We can assume that the brute force algorithm runs in time $2^{\Lambda(n)}$, where $\Lambda(n)$ is a polynomial. If no correct β could be found set $\beta = 0$.

S4 Output the transcript.

Claim 2.5.5 *The simulator M_{V^*} runs in expected polynomial time.*

Proof. It is sufficient to prove that the expected number of times V^* is called is polynomial. Let w be the random coins V^* uses. Let N_w be the number of subsets I on which the simulator has to go to step **S3**, i.e., these are the index set on which the conditions checked by M_{V^*} on step **S2** are true. Let us call these subsets *good*. Clearly, $0 \leq N_w \leq 2^k$.

Case 1: $N_w \geq 2$.

In this case $Pr[A\ randomly\ choosen\ subset\ K\ is\ good\ and\ K \neq I] = \frac{N_w - 1}{2^k}$. Thus, expected number of times V^* is called is

$$1 + \frac{N_w}{2^k}\left(\frac{N_w - 1}{2^k}\right)^{-1} \leq 3.$$

Case 2: $N_w = 1$.

In this case M_{V^*} moves to step **S3** only when the subset I chosen at step **S1** is good. Thus, the probability of this event is $\frac{1}{2^k}$. The brute force algorithm takes time $2^{\Lambda(n)}$. Thus, expected number of times V^* is called is $\frac{2^{\Lambda(n)}}{2^k}$. So if we choose k to be a polynomial greater than $\Lambda(n)$, V^* will be called at most constant number of times. (Note that the proof of Lemma 2.5.3 does not change for this choice of k)

Case 3: $N_w = 0$.

In this case V^* never reaches step **S3**. Thus, V^* is called only once. □

Claim 2.5.6 *Simulator's output distribution and the verifiers view is identically distributed.*

Proof. Same random coin for the verifier and same index set produces same transcript in both the cases. □

Combining Theorem 2.4.6 and Theorem 2.5.2 we obtain the following.

Corollary 2.5.7 *The complement of the following problems are in* PZK *in liberal sense: Set Transporter, Coset Intersection, Double Coset Membership, Double Coset Equality, Conjugacy of Elements, Vector Transporter.*

2.6 Solvable Permutation Group Isomorphism is in PZK

In this section we consider *permutation group isomorphism* PERM-ISO: given two subgroups $\langle S\rangle, \langle T\rangle \leq S_n$ the problem is to test if $\langle S\rangle$ and $\langle T\rangle$ are isomorphic.

Remark. PERM-ISO is in NP ∩ coAM [82]. It is harder than GRAPH-ISO [82] and seems different in structure from GRAPH-ISO or PGE. Like PGE if we try to formulate PERM-ISO using group action we notice that isomorphisms between groups are not permutations on small domains (unlike PGE). Thus, we do not know how to prove certain complexity-theoretic statements for PERM-ISO that hold for GRAPH-ISO. E.g. we do not know if it is in SPP or even low for PP [75], although GRAPH-ISO is in SPP [18]. Indeed, we do not know if PERM-ISO is in SZK.

However, in this section we show that PERM-ISO for solvable groups is reducible to $\overline{\text{SD}^{1,0}}$ and is hence in PZK. We recall some group theoretic definitions.

Let X be a finite set of symbols and $FG(X)$ be the free group generated by X.

Definition 2.6.1 *A pair (X, R) is a* presentation *of a group G where X is a finite set of symbols and R is a set of words over $X \cup X^{-1}$ where each $w \in R$ defines the relation $w = 1$. The presentation (X, R) defines G in the sense that $G \cong FG(X)/N$, where N is the normal closure in $FG(X)$ of the subgroup generated by R. The size of (X, R) is $|X| + \sum_{w\in R}|w|$. Call (X, R) a* short presentation *of the* finite *group G if the size of (X, R) is $(\log|G|)^{O(1)}$.*

It is an important conjecture [27] that all finite groups have short presentations. It is known to be true for large classes of groups. In particular, it is easy to prove that solvable finite groups have short presentations.

Notice that two groups are isomorphic if and only if they have the same set of presentations. Our reduction of solvable permutation group isomorphism to $\overline{\mathrm{SD}^{1,0}}$ will use this fact. Specifically, to reduce solvable PERM-ISO to $\overline{\mathrm{SD}^{1,0}}$ we give a randomized algorithm $\mathcal{A}$ that takes as input the generating set of a solvable group $G \leq S_n$ and outputs a short presentation for G. We can consider $\mathcal{A}(G)$ as a circuit with random bits as input and a short presentation for G as output. Clearly, if $G \not\cong H$ then the circuits $\mathcal{A}(G)$ and $\mathcal{A}(H)$ will output distributions with disjoint support. On the other hand, if $G \cong H$, the circuits $\mathcal{A}(G)$ and $\mathcal{A}(H)$ will compute identical probability distributions on the short presentations (for G and H).

We describe $\mathcal{A}$ in two phases. In the first phase $\mathcal{A}$ computes a random composition series for the input solvable group $G = \langle T \rangle$ following some distribution. In the second phase, $\mathcal{A}$ will deterministically compute a short presentation for G using this composition series. An ingredient for $\mathcal{A}$ is a polynomial-time sampling procedure from $L \setminus N$ where $L \leq S_n$ and $N \lhd L$ are subgroups given by generating sets. We describe this algorithm.

Lemma 2.6.2 (Sampling Lemma) *Let $L \leq S_n$ and $N \lhd L$, where both L and N are given by generating sets. There is a polynomial-time algorithm that samples from $L \setminus N$ uniformly at random (*with no failure probability*).*

Proof. Let $L = \langle S \rangle$ and $N = \langle T \rangle$. Recall that applying the Schreier-Sims algorithm we can compute a strong generating set for L in polynomial time. More precisely, we can compute distinct coset representatives R_i for $L^{(i)}$ in $L^{(i-1)}$ for $1 \leq i \leq n-1$, where $L^{(i)}$ is the subgroup of L that fixes each of $1, 2, \ldots, i$. Notice that $|R_i| \leq n$ for each i. Thus, we have the tower of subgroups $L = L^{(0)} \geq L^{(1)} \geq \ldots \geq L^{(n-1)} = 1$.

We can use the strong generating set $\bigcup R_i$ to sample uniformly at random from L as explained in proof of Theorem 2.4.2. This sampling procedure can be easily modified to sample uniformly from $L \setminus \{1\}$.

We will build on this idea, using some standard group-theoretic algorithms from [82] to sample uniformly from $L \setminus N$. Since $N \lhd L$ each set $NL^{(i)}$ is a subgroup of L.

Furthermore, for each i

$$\frac{|NL^{(i-1)}|}{|NL^{(i)}|} \leq \frac{|L^{(i-1)}|}{|L^{(i)}|} \leq n - i + 1.$$

Thus, $L = NL^{(0)} \geq NL^{(1)} \geq \ldots \geq NL^{(n-1)} = 1$ is also a subgroup tower with each adjacent pair of subgroups of small index. Furthermore, R_i also forms coset representatives for $NL^{(i)}$ in $NL^{(i-1)}$. However, R_i may not be all distinct coset representatives. Since we have the generating set for $NL^{(i)}$ (the union of T and the generating set for $L^{(i)}$) we can find the distinct coset representatives in polynomial time by using membership tests in $NL^{(i)}$, using the fact that $x, y \in R_i$ are not distinct coset representatives for $NL^{(i)}$ in $NL^{(i-1)}$ iff $xy^{-1} \in NL^{(i)}$. Let $S_i \subseteq R_i$ be the distinct coset representatives for each i. Let $|S_i| = m_i$ for each i. We can ignore the indices i for which S_i has only the identity element.

Now, each element $gN \in L/N$ can be expressed uniquely as

$$gN = (g_1N) \cdots (g_{n-1}N) = g_1 \cdots g_{n-1}N, \text{ where } g_i \in S_i.$$

Partition the nontrivial elements of L/N into sets $V_i = \{g_i \cdots g_{n-1}N \mid g_j \in S_j \text{ and } g_i \neq 1\}$. Clearly, $L/N \setminus \{1N\} = \biguplus_{i=1}^{n-1} V_i$. Furthermore, let $|V_i| = (m_i - 1)\prod_{j=i+i}^{n-1} m_j = N_i$ for each i. We can sample uniformly from V_i by uniformly picking $g_i \in S_i \setminus \{1\}$ and $g_j \in_R S_j$, $j = i+1, \ldots, n-1$. Thus, we can sample uniformly from L/N by first picking i with probability $\frac{N_i}{|L|/|N|-1}$ and then sampling uniformly from V_i. Finally, to sample from $L \setminus N$, notice that after picking the tuple $(g_i, \ldots, g_{n-1})$ while sampling from V_i we can pick $x \in N$ (by first building a strong generating set for N). Clearly, $g = g_i \cdots g_{n-1}x$, is uniformly distributed in $L \setminus N$. ∎

We now describe algorithm $\mathcal{A}$. Suppose S is the input to $\mathcal{A}$, where $G = \langle S \rangle$ is a solvable group. In Phase 1, $\mathcal{A}$ first computes the derived series $\{id\} = G_1 \lhd G_2 \lhd \cdots \lhd G_z = G$ of G (in deterministic polynomial time [82]).

Next, $\mathcal{A}(G)$ refines the derived series for G into a random composition series by inserting a chain of normal subgroups between consecutive groups of the series. It suffices to describe this refinement for $G' \lhd G$, where $G' = G_{z-1}$ and $G = G_z$. We can refine each $G_i \lhd G_{i+1}$ similarly.

Suppose $|G/G'| = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_l^{\alpha_l} = m$, $p_1 < p_2 < \cdots < p_l$. Using standard algorithms from [82] we can compute m in polynomial time. As m is smooth (all $p_i \leq n$) we can also factorize m in polynomial time to find the p_i. We will use the ordering of the p_i.

Let $G' = \langle T \rangle$. Since G/G' is abelian, the p_1-Sylow subgroup of G/G' is L/G' where L is generated by the union of T and $\{g^{m/p_1^{\alpha_1}} \mid g \in S\}$.

Notice that $G' \lhd L \lhd G$. Applying Lemma 2.6.2, $\mathcal{A}$ can sample uniformly an $x \in L \setminus G'$. As $|L/G'| = p_1^{\alpha_1}$, the order of xG' is p_1^t for some $t \neq 0$. This t is easily computed by repeated powering. Clearly, $x^{p_1^{t-1}}G'$ is of order p_1. Let $x_1 = x^{p_1^{t-1}}$ and define $N_1 = \langle T \cup \{x_1\} \rangle$. Clearly, G' is normal in N_1 and $|N_1/G'| = p_1$. Since G/G' is abelian it follows that $G' \lhd N_1 \lhd L \lhd G$.

We now repeat the above process for the pair of groups N_1 and G. Using Lemma 2.6.2 we randomly pick $x \in L \setminus N_1$ find the order p_1^s of xN_1 in G/N_1 and set $x_2 = x^{p_1^{s-1}}$. This will give us the subgroup N_2 generated by N_1 and x_2. Thus, we get the refinement $G' \lhd N_1 \lhd N_2 \lhd L \lhd G$, where $|N_2/N_1| = p_1$. Continuing thus, in α_1 steps we obtain the refinement

$$G' \lhd N_1 \lhd N_2 \lhd \cdots \lhd N_{\alpha_1} = L \lhd G.$$

Now, let M/G' be the p_2-Sylow subgroup of G/G'. We can find a generating set for M as before. Notice that $L \lhd ML \lhd G$. Thus, applying the above process we can randomly refine the series $L \lhd ML$ into a composition series where each adjacent pair of groups has index p_2.

Continuing thus, $\mathcal{A}$ refines $G' \lhd G$ into a random composition series between G and G'. This process can be applied to each pair $G_i \lhd G_{i+1}$ in the derived series to obtain a random composition series for G.

After phase 1, the computed composition series for G is described by a sequence $(x_1, x_2, \cdots, x_m)$ of elements from G, where the composition series is $id \lhd \langle x_1 \rangle \lhd \langle x_1, x_2 \rangle \lhd \cdots \lhd \langle x_1, x_2, \cdots, x_m \rangle = G$.

Observe that if $\phi : G \rightarrow H$ is an isomorphism and if $id = G_0 \lhd G_1 \lhd \cdots \lhd G_{m-1} \lhd G_m = G$ and $id = H_0 \lhd H_1 \lhd \cdots \lhd H_{m-1} \lhd H_m = H$ are the derived series of G and H respectively, then ϕ must isomorphically map G_i to H_i for each i. Furthermore, if $(x_1, x_2, \cdots, x_m)$ describes a composition series for G then $(\phi(x_1), \phi(x_2), \cdots, \phi(x_m))$ describes a composition series for H. Let X_i denote the random variable according to which x_i is picked in the above description for G. Similarly, let Y_i denote the random variable for the group H. It is easy to see that $Pr[X_1 = x_1] = Pr[Y_1 = \phi(x_1)]$. Now,

$$Pr[X_i = x_i\ 1 \leq i \leq m] = Pr[X_1 = x_1] \cdot \prod_{i=2}^{m} Pr[X_i = x_i | X_j = x_j, 1 \leq j < i].$$

Notice that to construct x_{i+1} the algorithm refines $\langle x_1, x_2, \cdots, x_i\rangle \lhd G_j$, where G_j is the appropriate group in the derived series. Now, if the algorithm finds $\phi(x_1), \phi(x_2), \cdots, \phi(x_i)$ as the first i components of the composition series for H, then the next element y_i is obtained by refining $\langle \phi(x_1), \phi(x_2), \cdots, \phi(x_i)\rangle \lhd H_j$, where $\phi : G_j \longrightarrow H_j$ is an isomorphism. Thus, it is easy to see that for $i \geq 2$ also we have

$$Pr[X_i = x_i \mid X_j = x_j\ 1 \leq j < i] \;=\; Pr[Y_i = \phi(x_i) \mid Y_j = \phi(x_j)\ 1 \leq j < i].$$

It follows that $Pr[X_i = x_i\ 1 \leq i \leq m] \;=\; Pr[Y_i = \phi(x_i)\ 1 \leq i \leq m]$.

In the second phase, the algorithm $\mathcal{A}$ computes a short presentation for G from its composition series given by $(x_1, x_2, \cdots, x_m)$. Let $p_1 = |\langle x_1\rangle|$ and for $j > 1$,

$$p_j = |\langle x_1, x_2, \cdots, x_j\rangle| / |\langle x_1, x_2, \cdots, x_{j-1}\rangle|.$$

Let the primes in this order be $p_1, p_2, \cdots, p_m$ (not necessarily distinct). Notice that each $g \in G$ can be uniquely expressed as $g = x_m^{l_m}, x_{m-1}^{l_{m-1}}, \cdots, x_1^{l_1}$, $0 \leq l_i \leq p_i - 1$.

$\mathcal{A}$ will compute the short presentation inductively. The cyclic subgroup $\langle x_1\rangle$ has the representation (X_1, R_1) where $X_1 = \{\alpha_1\}$ and $R_1 = \{\alpha_1^{p_1}\}$. We assume inductively that $\langle x_1, x_2, \cdots, x_i\rangle$ has the presentation (X_i, R_i) where $X_i = \{\alpha_1, \alpha_2, \cdots, \alpha_i\}$. We let $X_{i+1} = X_i \cup \{\alpha_{i+1}\}$. In order to define R_{i+1} we notice that $x_{i+1}\langle x_1, \cdots, x_i\rangle = \langle x_1, \cdots, x_i\rangle x_{i+1}$ and $x_{i+1}^{p_{i+1}} \in \langle x_1, x_2, \cdots, x_i\rangle$. Thus, the new relations (as explained e.g. in [19]) are

$$x_{i+1}^{p_{i+1}} = u_{i+1},\ u_{i+1} \in \langle x_1, x_2, \cdots, x_i\rangle$$
$$\forall j,\ 1 \leq j \leq i\ \ x_j x_{i+1} = x_{i+1} w_{i+1,j},\ w_{i+1,j} \in \langle x_1, x_2, \cdots, x_i\rangle$$

To find u_{i+1} notice that if $x \in \langle x_1, x_2, \cdots, x_i\rangle$ then x belongs to one of the cosets $x_i^j\langle x_1, x_2, \cdots, x_{i-1}\rangle$, $j = 0, \cdots, p_i - 1$. To find the exact coset $\mathcal{A}$ can do membership tests $x_i^{-j}x \in \langle x_1, x_2, \cdots, x_{i-1}\rangle$ for each j. As all the primes p_i are small, this is a polynomial-time step, because membership testing for permutation group can be done in polynomila time (cf. [82]). By repeating the same for $\langle x_1, x_2, \cdots, x_{i-1}\rangle$, $\langle x_1, x_2, \cdots, x_{i-2}\rangle$, $\cdots$, $\langle x_1\rangle$ the algorithm will be able to find $u_{i+1} = x_i^{l_i} \cdots x_1^{l_1}$. The corresponding relation will be $\alpha_{i+1}^{p_{i+1}} = \alpha_i^{l_i} \cdots \alpha_1^{l_1}$. The algorithm can compute $w_{i+1,j}$ and the corresponding relation similarly. Now, R_{i+1} is just R_i union the new relations. The number of relations $T(i)$ for $\langle x_1, x_2, \cdots, x_i\rangle$ follows the recurrence relation $T(i+1) = T(i) + i + 1$, $T(1) = 1$. So, the number of relation is $O(m^2)$. But $m = O(\log |G|)$. Hence the presentation is of polynomial length (more precisely it is $O(m^3)$). Suppose $\phi : G \longrightarrow H$ is an isomorphism and $\langle x_1, \cdots, x_m\rangle$ describes a composition series for G. Then $\langle \phi(x_1), \cdots, \phi(x_m)\rangle$ describes a composition series for H.

We notice that the composition series for G described by $(x_1, \cdots, x_m)$ for G and the composition series for H described by $(\phi(x_1), \cdots, \phi(x_m))$ yield the same presentation. This can be seen by observing that the process of obtaining u_{i+1} and $w_{i+1,j}$ is identical in both the cases. Thus, when $G \cong H$ it follows that the distributions produced by $\mathcal{A}(G)$ and $\mathcal{A}(H)$ will be identical. On the other hand, if $G \not\cong H$, $\mathcal{A}(G)$ and $\mathcal{A}(H)$ will have disjoint support. We have proved the following.

Theorem 2.6.3 *The problem of isomorphism testing of solvable permutation groups is polynomial time many-one reducible to* $\overline{\text{SD}}^{1,0}$ *and is hence in* PZK.

Remark. We can extend this result to show that isomorphism testing for m-recognizable permutation groups is also in PZK (see [96, Section 8.3] for definitions).

2.7 Black Box Group Problems

In this section we consider analogous problems over black-box groups [23, 32]. The black-box group model essentially abstracts away the internal structure of the group into a “black-box” oracle that does the group operations. In order to give uniform zero-knowledge protocols we generalize PGE to black-box groups: the *Group Equivalence Problem* GE. It turns out that the key difference from the results of Section 2.4 is that while permutation groups can be uniformly sampled by a polynomial time algorithm, there is no polynomial-time *uniform* sampling algorithm for black-box groups. However, the following seminal result of Babai for almost uniform sampling from black-box groups suffices to show that the considered black-box group problems are in SZK.

Theorem 2.7.1 [23] *Let c,$C > 0$ be given constants, and let $\varepsilon = N^{-c}$ where N is a given upper bound on the order of the group G. There is a Monte Carlo algorithm which, given any set of generators of G, constructs a sequence of $O(log\ N)$ ε-uniform Erdos-Renyi generators at a cost of $O((log\ N)^5)$ group operations. The probability that the algorithm fails is $\leq N^{-C}$.*

If the algorithm succeeds, it permits the construction of ε-uniformly distributed random elements of G at a cost of $O(log\ N)$ group operation per random element.

As the distribution is ε-uniform, not perfectly uniform and the algorithm has certain error probability, we do not get perfect zero knowledge protocols for GE. However, we have the following.

Theorem 2.7.2 GE *is reducible to* $\overline{\text{SD}^{1,\frac{1}{3}}}$ *(relative to the black box group oracle B).*

Proof. The proof is similar to that of Theorem 2.4.2. We reduce GE to $\overline{\text{SD}^{1,\epsilon_1}}$ for some small ϵ_1. Let (q, x, y, T, τ) where elements of $\{0,1\}^q$ represents group elements, T is the set of generating elements of group G and τ is a polynomial time routine that computes the group action and has access to the group oracle B. The reduction maps (q, x, y, T, τ) to the pair of circuits (X_1, X_2), both having access to the black box group oracle B. The circuit X_1 samples $g \in G$ using Babai's algorithm. If the algorithm fails the circuit sets g to be any fixed element of G. Then it produces x^g. The circuit X_2 is similarly defined for y. As in Theorem 2.4.2, we can argue that if x and y are not in the same G-orbit the statistical difference between the two circuits will be 1. But if they are in the same orbit then we can verify that the statistical difference is less than a chosen small number ϵ_1. We can make ϵ_1 close to the ε specified by Theorem 2.7.1 by repeating Babai's algorithm and thus reducing the error introduced due to failure. As ε is inverse exponential, we can make ϵ_1 less than $\frac{1}{3}$. ■

Theorem 2.7.3 GE *is in* SZK^B *(where* SZK^B *stands for SZK in which both prover and verifier have access to the group oracle B).*

Proof. The complexity class SZK is closes under complementation [94]. Thus, $\overline{\text{SD}^{2/3,1/3}} \in$ SZK. Now suffices to observe that the proof [94] that SD$^{2/3,1/3} \in$ SZK relativizes and that SD$^{1,1/3}$ is trivially reducible to SD$^{2/3,1/3}$. ■

As a corollary we also get that several problems considered in [23] and some generalization of permutation group problems are in SZK^B. This partially answers an open question posed in [23] whether the considered problems are in SZK. However, we do not know if the order verification problem and group isomorphism for black-box groups are in SZK, although they are in AM $\cap$ coAM.

Corollary 2.7.4 *Black box group membership testing, Disjointness of double cosets, Disjointness of subcosets, Group factorization etc are in* SZK^B.

Proof. Let (q, x, T) be an instance of black box group membership testing problem, where q is the length of the strings encoding group elements, T generates the group G. To reduce it to GE we notice that $x \in G$ if and only if some element $t \in T$ and x are in the same G-orbit where the G action is just right multiplication, i.e., $z^g = gz$.

Let (q, s, t, A, B) be an instance of double coset disjointness, where $H = \langle A \rangle$, $K = \langle B \rangle$ and the problem is to decide if HsK and HtK are disjoint. Here we notice that $HsK \cap HtK \neq \phi$ *iff* s and t are in the same $H \times K$-orbit where the action is defined by $z^{(h,k)} = h^{-1}zk$.

Disjointness of double coset and group factorization are equivalent because $Hs \cap Kt \neq \phi$ *iff* $H \cap Kts^{-1} \neq \phi$ *iff* $ts^{-1} \in KH$.

Let (q, x, A, B) be an instance of Group factorization, where $G = \langle A \rangle$, $H = \langle B \rangle$. The problem is to decide if $x \in GH$. We notice that $x \in GH$ *iff* x and the identity element e are in the same $G \times H$-orbit. The group action is defined as $z^{(g,h)} = g^{-1}zh$. ∎

2.8 Concurrent Zero Knowledge

Dwork, Naor and Sahai [46] defined the notion of concurrent zero knowledge in which several execution of the protocol occurs concurrently. It is a much stringent condition to demand that a protocol should be zero knowledge even if there are several execution of the protocol concurrently, which can very well be controlled by a single adversarial verifier.

In a concurrent interaction a verifier V can interact with the prover as follows: The verifier sends a pair (m, s) where m is the message for session s. The prover P acts independently in all sessions. The message passing continues until the verifier sends (end, α), where $\alpha = accept$ or $reject$. So, the transcript looks like

$$[(m_1, s_1), p_1, (m_2, s_2), p_2, \cdots, (m_t, s_t), p_t, (end, \alpha)].$$

The verifiers view $\langle P, V\rangle(x)$ on input x is the transcript along with the random bits used by the verifier.

The protocol is said to be zero knowledge if it remains zero knowledge under concurrent interaction. In [85], Micciancio, Ong, Sahai and Vadhan proved

Theorem 2.8.1 [85] $SD^{1,1/2}$ *has concurrent zero knowledge proof system.*

Combined with Theorems 2.8.1, 2.6.3, 2.4.2 and the fact that concurrent zero knowledge is closed under polynomial time many one reduction we get the following corollary immediately:

Corollary 2.8.2 *The complement of the following problems has concurrent zero knowledge proofs: Set Transporter, Coset Intersection, Double Coset Membership, Double Coset Equality, Conjugacy of Elements, Vector Transporter, Solvable Permutation Group Isomorphism.*

2.9 SZK **Proof with Efficient Provers**

An important question is whether we can design SZK protocols with *efficient* provers for all problems in SZK. A notion of efficient provers, considered useful for problems in SZK∩NP, is where the prover has to be a randomized algorithm that has access to an NP witness for an instance x of a language in SZK∩NP. This question is studied in [86] where it is shown that $\overline{\mathrm{SD}}^{1,1/2}$ has such an SZK protocol. Consequently, any problem polynomial-time many-one reducible to $\overline{\mathrm{SD}}^{1,1/2}$ also has such efficient provers.

As a consequence of Theorem 2.7.2 where we show that Group Equivalence for black-box groups is reducible to $\overline{\mathrm{SD}}^{1,1/3}$ it follows from Corollary 2.7.4 and the above-mentioned result of [86] that all NP problems considered in Section 2.7 have SZK protocols with efficient provers.

Theorem 2.9.1 *Black box group membership testing, Double coset membership, Subcoset intersection, Group factorization etc are in* NP ∩ SZK^B *and have SZK protocols with efficient provers.*

2.10 Concluding Remarks

In this chapter we show that SZK (and PZK) contains a host of natural computational black-box problems (respectively permutation group problems). As complexity classes SZK and PZK are quite intriguing. We do not known anything beyond the containment PZK $\subseteq$ SZK $\subseteq$ AM $\cap$ coAM and the closure of SZK under complement. In this context it is interesting to note that all considered permutation group problems (except solvable group isomorphism) are known to be low for PP: we can put PGE in SPP using the methods of [18, 75]. Could it be that the class PZK (or even SZK) is low for PP? We make a final remark in this context. The SZK-complete problem Entropy Difference (ED) is complete even for "nearly flat" distributions, where "flatness" is a technical measure of closeness to the uniform distribution [106]. If we consider ED with the stronger promise that the two input distributions are *uniform on their support* then we can prove that the problem is low for PP.

3 Limited Randomness Zero Knowledge

3.1 Introduction

Two groups G_1 and G_2 are *isomorphic*, denoted $G_1 \cong G_2$, if there is a bijection $\phi : G_1 \longrightarrow G_2$ such that $\phi(gg') = \phi(g)\phi(g')$ for all $g, g' \in G_1$. If the G_1 and G_2 are two finite permutation groups given by their generating sets then deciding if $G_1 \cong G_2$ is known as *permutation group isomorphism problem* (PERM-ISO). Graph isomorphism problem (GI) is polynomial-time Turing reducible to permutation group isomorphism problem. The problem PERM-ISO is in NP $\cap$ coAM [82]. We considered a special case of PERM-ISO in Chapter 2 when the input permutation groups are solvable. In this chapter we consider the problem of deciding if two finite groups given by their *multiplication tables* are isomorphic. This problem is polynomial-time many-one reducible to the graph isomorphism problem. We give a zero knowledge protocol for this problem where the number of random bits used by the prover and the verifier is only polylogarithmic. The contribution of this chapter is that we give a theory of limited resource zero knowledge proofs that is based on the work of Goldreich, Sahai and Vadhan [94, 57].

Definition 3.1.1 (GrIso**)**
Input *: The multiplication tables of two finite groups G_1 and G_2 of order n.*
Decide *: If G_1 and G_2 are isomorphic.*

It was shown in [19] that GrIso has a constant round, public-coin interactive protocol. The total size of the messages exchanged between the prover and the verifier in the interactive protocol in [19] is only polylogarithmic and the verifier uses only polylogarithmically many random bits. In this chapter we give a PZK protocol for GrIso while maintaining all these properties, namely our PZK protocol has constant number of rounds, the prover and the verifier use only polylogarithmically many random bits and the total message size is polylogarithmic. This motivates us to study problems that have zero knowledge protocols where the use of *randomness is limited*. More precisely, we investigate statistical zero knowledge protocols where the prover, verifier and the simulator use only polylogarithmic randomness.

We define a class βHVSZK which consists of all promise problems having honest verifier statistical zero knowledge protocols where the prover, verifier and the simulator use polylogarithmic randomness and have polylogarithmic numbers of rounds. We analyze the techniques developed by Goldreich, Sahai and Vadhan [94, 57] in this setting. In their seminal work they proved that SZK has complete problems SD and ED. They gave an honest verifier statistical zero knowledge protocol for SD which has only *two rounds*. Since SD is complete for SZK, it implies that every problem in SZK has an honest verifier statistical zero knowledge proof with only two rounds. In this chapter we prove that every problem having an honest verifier statistical zero knowledge proof where the prover, verifier and the simulator use polylogarithmically many random bits and the protocol has polylogarithmically many rounds also has an honest verifier statistical zero knowledge protocol where not only the number of random bits used by the prover and the verifier is polylogarithmic but also the *message size* is only *polylogarithmic* and the *number of rounds* is *two*.

As a result of this we conclude that GrIso has an honest verifier statistical zero knowledge proof where the number of random bits required is polylogarithmic and has 2 rounds.

3.2 Preliminaries

We recall some definitions that we will be using in this chapter.

Definition 3.2.1 (Support) *Let X be a distribution over a finite set $\mathcal{U}$. The* support *of X, denoted* Supp(X) *is the set*

$$\{x \in \mathcal{U} \mid Pr[D = x] > 0\}.$$

Definition 3.2.2 *Let X and Y be two distributions over finite sets $\mathcal{U}_1$ and $\mathcal{U}_2$ respectively. The* tensor product *of X and Y is a distribution, denoted $X \otimes Y$, over the set*

$\mathcal{U}_1 \times \mathcal{U}_2$ *defined as follows*

$$Pr[X \otimes Y = (x, y)] = Pr[X = x].Pr[Y = y].$$

We define $\bigotimes^2 X$ as $X \otimes X$. Inductively $\bigotimes^k X = \bigotimes^{k-1} X \otimes X$. It is easy to check that $\mathrm{H}(X \otimes Y) = \mathrm{H}(X) + \mathrm{H}(Y)$, where H is the Shannon entropy (see Definition 2.2.10).

Definition 3.2.3 *A family $\mathcal{H}$ of function from a finite set D to a finite set R is* 2-universal *if for all $x, y \in D$, $x \neq y$ and for all $a, b \in R$,*

$$Pr_{h \in \mathcal{H}}[h(x) = a \textit{ and } h(y) = b] = \frac{1}{|R|^2}.$$

The set of all affine linear functions from $GF(2)^m$ to $GF(2)^n$ is a 2-universal family of hash functions from $\{0, 1\}^m$ to $\{0, 1\}^m$ and is denoted by $\mathcal{H}_{m,n}$. A function $h \in \mathcal{H}_{m,n}$ is of the form $h(x) = Ax + b$ where A is an $n \times m$ 0-1-matrix and b is a n bit 0-1 vector. Thus, to describe any function in $\mathcal{H}_{m,n}$ we need $(mn + n)$ bits.

3.3 Group Isomorphism

In this section we present a PZK protocol for isomorphism of groups given by their multiplication table. The protocol broadly follows the steps in [19, 20].

Before that we need the concept of *cube generating k-sequences*. Let G be a group and let $C = (g_1, \cdots, g_k)$ be a sequence of k elements of G. The *cube* generated by the sequence C is the set

$$Cube(C) = \{g_1^{\epsilon_1}, \cdots, g_k^{\epsilon_k} \mid \epsilon_i \in \{0, 1\}\}.$$

. The sequence C is called a *cube generating k-sequence* if $G = Cube(C)$. The following theorem is due to Erdös and Rényi [47].

Theorem 3.3.1 *Let G be a finite group with n elements and let k be an integer such that $k \geq \log n + \log\log n + 2\log\frac{1}{\delta} + 5$. Let $g_1, \cdots, g_k$ be sampled uniformly at random from G. Then*

$$Pr[C \textit{ is a cube generating sequence for } G] > 1 - \delta.$$

In the above theorem if we pick $k = 4\log n$ then

$$Pr[C \text{is a cube generating sequence for } G] > 1 - 1/n.$$

Following [19, 20], we now describe a way of embedding the group structure of a finite group G with n elements in a set S as follows. Let $\phi : G \longrightarrow S$ be an injective map. Then the elements of $\phi(G) \subseteq S$ naturally inherits the group structure of G as follows. Let $x, y \in \phi(G)$, then $x.y = \phi(\phi^{-1}(x)\phi^{-1}(y))$. Let $C = (g_1, \cdots, g_k)$ be a cube generating sequence of G. We define $\phi_C : G \longrightarrow \{0,1\}^k$ as follows. $\phi_C(g) = (\epsilon_1, \cdots, \epsilon_k)$ where $(\epsilon_1, \cdots, \epsilon_k)$ is the lex-least element in $\{0,1\}^k$ such that $g = g_1^{\epsilon_1} \cdots g_k^{\epsilon_k}$. Thus, $\phi_C(G) \subseteq \{0,1\}^k$ gets a group structure whose multiplication table is defined via $x.y = \phi_C(\phi_C^{-1}(x)\phi_C^{-1}(y))$. Notice that given C of length $k = 4\log n$, the map ϕ_C can be computed in polynomial time.

Now we are ready to give the PZK protocol (P, V) for group isomorphism of two groups $G_0,\ G_1$ given by their multiplication table. The protocol is similar in flavor to the PZK protocol for graph isomorphism and given in Protocol 1. In Protocol 1 **P** stands for prover and **V** for verifier.

Protocol 1 Protocol for GrIso

1 **P**: Pick $k = 4\log n$ random elements $g_1, \cdots, g_k$ from G_0. If $C = (g_1, \cdots, g_k)$ is not a cube generating sequence, then send 'FAIL' to the verifier. Otherwise do the following. Compute ϕ_C as defined above. Send the multiplication for the group $H = \phi_C(G_0)$.

2 **V**: If P sends 'FAIL' then reject. Otherwise pick b uniformly at random from $\{0,1\}$ and send b to P.

3 **P**: Send an isomorphism π from G_b to H.

4 **V**: Check if π is indeed an isomorphism from G_b to H. If π is an isomorphism then accept otherwise reject.

Notice that H is always isomorphic to G_0. If $G_0 \cong G_1$ then the only way the prover could fail to convince the verifier is that he fails to obtain a cube generating sequence for G_0. The protocol has completeness error at most $1/n$. On the other hand if G_0 is not isomorphic to G_1 then no matter what strategy a prover P^* takes H will not be isomorphic to one of the groups G_0 or G_1. Thus, depending on the verifier's random choice he will be caught at least half the times. Therefore, the protocol has soundness error at most $1/2$.

In Simulator 2 we present a simulator to demonstrate that the interactive protocol is actually a perfect zero knowledge protocol. In Simulator 2, V^* is an arbitrary verifier. Notice that even if the simulator does not get a cube generating sequence it does not fail. It merely simulates V^* on 'FAIL'. The real failure of the simulator may occur only at the last step of Simulator 2.

Simulator 2 Simulator for Protocol 1

I Pick b uniformly at random from $\{0, 1\}$. Pick $k = 4 \log n$ random elements $g_1, \cdots, g_k$ from G_b.

II If $C = (g_1, \cdots, g_k)$ is not a cube generating sequence, then,

- Send 'FAIL' to the verifier.
- Simulate V^* on 'FAIL', let s be the response of the verifier.
- Output $('\mathrm{FAIL}', s)$.

III If a cube generating sequence C is found do the following.

- Compute ϕ_C as defined above. Let H be the group on $\phi_C(G_b)$ defined by the embedding ϕ_C.
- Simulate V^* on H. Let c be the response of the verifier. We may assume that $c \in \{0, 1\}$.
- If $c \neq b$ then the simulator fails. Otherwise it outputs (H, c, ϕ_C).

The statistical difference between the verifier's view and the simulator's output conditioned on the event that the simulator does not fail is zero. Hence the protocol is perfect zero knowledge. The analysis is exactly as in the proof that GRAPH-ISO $\in$ PZK (see [54, Section 4.3.2]).

In the next theorem we list the properties of the protocol and the simulator.

Theorem 3.3.2 *The problem* GrIso *has a* PZK *proof system* (P, V) *with the following properties:*

1. *The prover and the verifier on input* (G_0, G_1) *use polylogarithmic randomness* ($O(\log^2 n)$ *bits, where* $n = |G_0|$*).*

2. *If a randomized polynomial-time algorithm V^* acting as a verifier uses polylogarithmic randomness then the simulator while simulating V^* also uses polylogarithmic randomness.*

3. *The number of round in the protocol is 3,*

4. *The message exchanged is polynomial in the size of the input.*

Proof. In step 1 of Protocol 1 the prover picks $k = 4 \log n$ random elements from G_0. Since G_0 has size n, each element of G_0 is encoded as $O(\log n)$ bit string. Hence, the randomness used by the prover is $O(\log^2 n)$.

To simulate V^* the simulator supplies random coins to V^*. It also uses random bits for its own purpose. The simulator uses $O(\log^2 n)$ bits of randomness in step I for its own purpose. Except for that it needs random bit only to simulate V^*'s random coins. Thus, if V^* uses polylogarithmic randomness the the simulator also uses polylogarithmic random bits.

From the description of Protocol 1 it is clear that the number of rounds is 3 and the message size is proportional to the size of the multiplication tables of G_0 and G_1. ∎

In the following sections we will see how to reduce the message size of such protocol in a generic way. We will actually reduce the message size of an honest verifier statistical zero knowledge protocol where the randomness complexity is polylogarithmic and the number of rounds is also polylogarithmic. More precisely, we show that every language having an honest verifier statistical zero knowledge protocol where the prover, verifier and the simulator use polylogarithmic randomness and has polylogarithmic number of rounds also has an honest verifier statistical zero knowledge proof where the prover, verifier and the simulator use polylogarithmic randomness, the message size is also polylogarithmic and the number of rounds is only constant.

3.4 Limited Randomness Zero Knowledge

Next we define the notion of limited randomness zero knowledge. As we shall see there are several possibilities of defining the notion of limited randomness zero knowledge protocols. Recall that a randomized algorithm $\mathcal{A}$ is said to be *useful* if for every input x, $Pr[\mathcal{A}(x) = fail] \leq 1/2$.

Definition 3.4.1 *Let $R, M, E : \mathbb{N} \longrightarrow \mathbb{N}$ be functions on natural numbers. A promise problem $\Pi = (\Pi_Y, \Pi_N)$ is in the class βHVSZK(R, M, E) if there is an interactive proof system (P, V) such that on input $(x, 1^k)$ and $n = |x| + k$ the followings hold:*

1. *Each of P and V uses at most $O(R(n))$ random bits.*
2. *The total message length is at most $O(M(n))$.*
3. *Number of message exchanged is at most $O(E(n))$.*
4. Zero Knowledge: *There is a negligible function μ and a useful polynomial-time randomized algorithm S such that if $x \in \Pi_Y$ then on input $(x, 1^k)$*
   - *S uses at most $O(R(n))$ random bits.*
   - StatDiff$(\overline{S}(x, 1^k), \langle P, V\rangle(x, 1^k)) \leq \mu(k)$*, where $\overline{S}(x, 1^k)$ is the output distribution of S on input $(x, 1^k)$ conditioned on $S(x, 1^k) \neq fail$.*

*If $\mu(k) = 0$ for all k then Π is said to be in β*HVPZK(R, M, E).

Note that the completeness and soundness conditions follow from the definition of interactive proof systems (see Definition 2.2.3). We work with the above definition of honest verifier zero knowledge protocol in this chapter. We also define three more notions of zero knowledge based on the randomness used by the prover, verifier and the simulator.

Definition 3.4.2 (SZK(R, M, E)**)** *For functions $R, M, E : \mathbb{N} \longrightarrow \mathbb{N}$, a promise problem $\Pi = (\Pi_Y, \Pi_N)$ is in the class* SZK(R, M, E) *if there is an interactive proof system (P, V) such that on input $(x, 1^k)$ and $n = |x| + k$ the followings hold:*

1. *Each of P and V uses at most $O(R(n))$ random bits.*
2. *The total message length is at most $O(M(n))$.*
3. *Number of message exchanged is at most $O(E(n))$.*
4. Zero Knowledge: *For every probabilistic polynomial-time algorithm V^* there is a useful polynomial-time randomized algorithm S^* such that if $x \in \Pi_Y$ then*

   StatDiff$(\overline{S^*}(x, 1^k), \langle P, V\rangle(x, 1^k)) \leq \mu(k)$ *for some negligible function μ (which may depend on V^*), where $\overline{S^*}(x, 1^k)$ is the output distribution of S^* on input $(x, 1^k)$ conditioned on $S^*(x, 1^k) \neq fail$.*

If $\mu(k) = 0$ for all k then Π is said to be in $\mathrm{PZK}(R, M, E)$.

We compare this definition with the previous definition. In this definition we allow S^* to use as much randomness as required provided it is polynomially bounded whereas in Definition 3.4.1 the simulator can use at most $O(R(n))$ random bits. Here we only impose the condition that the prover and the verifier in the interactive protocol (P, V) use at most $O(R(n))$ random bits.

Definition 3.4.3 ($\beta\mathrm{SZK}(R, M, E)$ **Limited)** *For functions $R, M, E : \mathbb{N} \longrightarrow \mathbb{N}$, a promise problem $\Pi = (\Pi_Y, \Pi_N)$ is in the class $\beta\mathrm{SZK}(R, M, E)$ if there is an interactive proof system (P, V) such that on input $(x, 1^k)$ and $n = |x| + k$ the followings hold:*

1. *Each of P and V uses at most $O(R(n))$ random bits.*
2. *The total message length is at most $O(M(n))$.*
3. *Number of message exchanged is at most $O(E(n))$.*
4. Zero Knowledge: *For every probabilistic polynomial-time algorithm V^* there is a useful polynomial-time randomized algorithm S^* such that if $x \in \Pi_Y$ then*
   - StatDiff$(\overline{S^*}(x, 1^k), \langle P, V\rangle(x, 1^k)) \leq \mu(k)$ *for some negligible function μ (which may depend on $V*$), where $\overline{S^*}(x, 1^k)$ is the output distribution of S^* on input $(x, 1^k)$ conditioned on $S^*(x, 1^k) \neq fail$..*
   - *S^* uses at most $O(R(n))$ random bits.*

Here the verifier V^* may use any amount of randomness provided it is polynomially bounded but the simulator has to run only with polylogarithmically many random bits.

Definition 3.4.4 (Weak-$\beta\mathrm{SZK}(R, M, E)$ **Weak Limited)** *For functions $R, M, E : \mathbb{N} \longrightarrow \mathbb{N}$, a promise problem $\Pi = (\Pi_Y, \Pi_N)$ is in the class* Weak-$\beta\mathrm{SZK}(R, M, E)$ *if there is an interactive proof system (P, V) such that on input $(x, 1^k)$ and $n = |x| + k$ the followings hold:*

1. *Each of P and V uses at most $O(R(n))$ random bits.*
2. *The total message length is at most $O(M(n))$.*
3. *Number of message exchanged is at most $O(E(n))$.*

4. Zero Knowledge: *For every probabilistic polynomial-time algorithm V^* that uses at most $O(R(m))$ random bits on inputs of length m there is a useful polynomial-time randomized algorithm S^* such that if $x \in \Pi_Y$ then*

   - StatDiff$(\overline{S^*}(x, 1^k), \langle P, V\rangle(x, 1^k)) \leq \mu(k)$ *for some negligible function μ (which may depend on $V*$), where $\overline{S^*}(x, 1^k)$ is the output distribution of S^* on input $(x, 1^k)$ conditioned on $S^*(x, 1^k) \neq fail$.*
   - *S^* uses at most $O(R(n))$ random bits.*

This definition differs from the previous definition in the way that here the simulator works only for verifier that uses small randomness. The inclusions below follows directly from the definitions.

Theorem 3.4.5 • βSZK$(R, M, E) \subseteq$ Weak-βSZK(R, M, E).

- Weak-βSZK$(R, M, E) \subseteq \beta$HVSZK$(R, M, E)$.
- βSZK$(R, M, E) \subseteq$ SZK(R, M, E).

We would often use $\log^c$, $poly$ to denote functions $\log^c(x)$ and $p(x)$, where $p(x)$ is some polynomial. E.g., by βSZK$(\log^c, poly, 3)$ we mean the class βSZK$(log^c(.), p(.), 3)$ for some polynomial $p(.)$

We notice from Section 3.3 that GrIso is in the class PZK(R, M, E) where R is polylogarithmic, M is polynomial and $E = 3$. Observe that if the we replace V^* in Simulator 2 by the prover V of Protocol 1 then we get the following theorem.

Lemma 3.4.6 GrIso $\in \beta$HVSZK$(\log^2, poly, 3)$.

Next we define a version of SD (see Definition 2.2.9) for which the circuits that encode the distributions have only polylogarithmic many input bits.

Definition 3.4.7 (βSD$^{(c)}$) *For a constant $c > 0$ the promise problem*

$$\beta\text{SD}^{(c)} = (\beta\text{SD}_Y^{(c)}, \beta\text{SD}_N^{(c)})$$

takes two distributions X and Y encoded by boolean circuits as input and has "yes instances" $\beta\text{SD}_Y^{(c)}$ and "no instances" $\beta\text{SD}_N^{(c)}$ defined as follows:

$$\begin{aligned}\beta\text{SD}_Y^{(c)} &= \{(X_1, X_2) \mid \text{StatDiff}(X_1, X_2) \geq 2/3 \textit{ number of inputs to } X_1 \textit{ and} \\ &\qquad X_2 \textit{ is at most } \log^c n \textit{ where } n = \textit{max}\{ \textit{ size of } X_1, X_2\}\}. \\ \beta\text{SD}_N^{(c)} &= \{(X_1, X_2) \mid \text{StatDiff}(X_1, X_2) \leq 1/3 \textit{ number of inputs to } X_1 \textit{ and} \\ &\qquad X_2 \textit{ is at most } \log^c n \textit{ where } n = \textit{max}\{ \textit{ size of } X_1, X_2\}\}.\end{aligned}$$

A problem ED of similar flavour which involves entropy of the distributions instead of statistical difference was defined in [94]. Below we define the restricted version of the problem where the input pair of circuits has polylogarithmic many input bits.

Definition 3.4.8 ($\beta\text{ED}^{(c)}$) *For a constant $c > 0$ the promise problem*

$$\beta\text{ED}^{(c)} = (\beta\text{ED}_Y^{(c)}, \beta\text{ED}_N^{(c)})$$

takes two distributions X and Y encoded by boolean circuits as input and has "yes instances" $\beta\text{ED}_Y^{(c)}$ and "no instances" $\beta\text{ED}_N^{(c)}$ defined as follows:

$$\begin{aligned}\beta\text{ED}_Y^{(c)} &= \{(X_1, X_2) \mid \text{H}(X_1) \geq \text{H}(X_2) + 1 \textit{ number of inputs to } X_1 \textit{ and} \\ &\qquad X_2 \textit{ is at most } \log^c n \textit{ where } n = \textit{max}\{ \textit{ size of } X_1, X_2\}\}. \\ \beta\text{ED}_N^{(c)} &= \{(X_1, X_2) \mid \text{H}(X_2) \geq \text{H}(X_1) + 1 \textit{ number of inputs to } X_1 \textit{ and} \\ &\qquad X_2 \textit{ is at most } \log^c n \textit{ where } n = \textit{max}\{ \textit{ size of } X_1, X_2\}\}.\end{aligned}$$

3.5 Results of this Chapter

In this section we prove the main results of this chapter. We first show that the problem $\beta\text{SD}^{(c)}$ is in the class $\beta\text{HVSZK}(\log^{O(c)}, \log^{O(c)}, 2)$. Next we prove that any language L in the class $\beta\text{HVSZK}(\log^{c_1}, poly, \log^{c_2})$ is polynomial-time many-one reducible to $\beta\text{ED}^{(c)}$ for some constant $c = c_1 + c_2 + O(1)$. Finally we prove that the promise problem $\beta\text{ED}^{(c)}$ is polynomial-time many-one reducible to $\beta\text{SD}^{(c')}$ where $c' = O(c)$. As a consequence we obtain $\beta\text{HVSZK}(\log^{c_1}, poly, \log^{c_2}) \subseteq \beta\text{HVSZK}(\log^{O(c_1+c_2)}, \log^{O(c_1+c_2)}, 2)$ i.e., every problem in $\beta\text{HVSZK}(\log^{c_1}, poly, \log^{c_2})$ has a polylogarithmic randomness honest verifier statistical zero knowledge protocol with *constant* number of rounds where the total size of the messages exchanged is only polylogarithmic.

Our approach is to analyze the techniques in [94, 57] for the limited randomness setting. The proof techniques developed by Goldreich, Sahai and Vadhan in [94, 57] involves extensive manipulations with distributions. The main hurdle to make the proofs in [94, 57] work in limited randomness setting is to make sure that the randomness used does not blow up while manipulating the distributions. We also want to keep the total size of the messages exchanged polylogarithmic. This is another aspect in which our goal differs from that in [94, 57] where the message size is only polynomially bounded.

We design a procedure called *output bit reduction* algorithm. Given a pair of circuits (X_1, X_2) which has polylogarithmically many input bits, this algorithm produces another pair of circuits (Y_1, Y_2) such that StatDiff(Y_1, Y_2) is close to StatDiff(X_1, X_2) but the number of output bits of Y_1 and Y_2 is polylogarithmic. In some sense the message size corresponds to the number of output bits of certain circuits appearing in [94, 57] and the techniques of Goldreich, Sahai and Vadhan for manipulating distributions are often dependent on the number of output bits. Thus, as we will see in more detail, to control the message size it is important to apply output bit reduction algorithm. In a loose sense whenever the randomness requirement or the message size tend to go beyond the polylogarithmic bound we apply the output bit reduction algorithm.

The outline of the rest of the chapter is as follows.

- In Section 3.5.1 we give a βHVSZK$(\log^{O(c)}, \log^{O(c)}, 2)$ protocol for βSD$^{(c)}$. We also present the output bit reduction algorithm that preserves statistical difference.
- In Section 3.5.2 we show that ever problem in βHVSZK$(\log^{c_1}, poly, \log^{c_2})$ is polynomial-time many-one reducible to βED$^{(c)}$ where $c = c_1 + c_2 + O(1)$.
- In Section 3.5.3 we give a polynomial-time many-one reduction from βED$^{(c)}$ to βSD$^{(c')}$ where $c' = O(c)$. We also present the the output bit reduction algorithm that preserves entropy difference.

3.5.1 βHVSZK **Protocol for** βSD$^{(c)}$

We will show that βSD$^{(c)} \in \beta$HVSZK$(\log^{O(c)}, \log^{O(c)}, 2)$. We will eventually use a protocol and the corresponding simulator used by Sahai and Vadhan in [94]. But we cannot directly use their protocol as the prover and verifier in that protocol exchange the outputs of the circuits which in general could be as large as $\Theta(n)$ where n is the size of the circuits. To avoid this problem we will obtain a pair of circuits from the given input circuits that has almost the same statistical difference as the input circuits but has polylogarithmically many output bits.

Theorem 3.5.1 (Output Bit Reduction) *Let X_1 and X_2 be two circuits encoding distributions where X_1 and X_2 have at most $\log^c n$ input bits where $n = max\{\text{Size}(X), \text{Size}(Y)\}$. Let $\text{StatDiff}(X_1, X_2) = \delta$. Then in time polynomial in n we can compute two circuits Y_1 and Y_2 with the following properties.*

- $(1-\epsilon)\delta \leq \text{StatDiff}(Y_1, Y_2) \leq (1-\epsilon)\delta + \epsilon$ *where* $\epsilon = 2^{-10}$.
- Y_1 *and* Y_2 *has at most* $\log^{3c} n$ *input and output bits.*

Proof. The algorithm uses fingerprinting by primes. Let λ, σ be positive numbers to be fixed later. For $i \in \{1, 2\}$ we describe the circuit Y_i.

The circuit Y_i picks a list of $\log^\lambda n$ random $\log^\sigma n$ bit numbers. Then using AKS primality testing [1] it picks the first prime number r appearing in the list. If the list contains no prime number the circuit outputs "fail". Otherwise it samples the distribution encoded by the circuit X_i by first picking a $\log^c n$ bit number s and then evaluating the circuit X_i on s. Let $x = X_i(s)$. The circuit Y_i then outputs $(x \bmod r, r)$.

Notice that the number of output bits of Y_i is $2\log^\sigma n$ and the number of input gates is $\log^c n + \log^{\lambda+\sigma} n$.

Let
$$\text{Supp}(X_1) \cup \text{Supp}(X_2) = \{x_1, \cdots, x_M\}.$$
A crucial observation is that $M \leq 2.2^{\log^c n}$.

We say that a prime p is *good* if

$$x_i - x_j \neq 0 \bmod p \text{ for all } 1 \leq i < j \leq M.$$

The numbers $x_1 - x_j$ can have at most $\log(2.2^{\log^c n}) = \log^c n + 1$ prime factors. So, among all $\log^\sigma n$ bit prime numbers at most $\begin{pmatrix} M \\ 2 \end{pmatrix}(\log^c n + 1)$ primes are not good.

Notice that
$$\begin{pmatrix} M \\ 2 \end{pmatrix}(\log^c n + 1) \leq 2^{2\log^c n + 2}\log^c n.$$

By prime number theorem there are at least $\frac{2^{\log^\sigma n}}{\log^\sigma n}$, $\log^\sigma n$ bit prime numbers[1].

$$Pr[r \text{ is not good } \mid r \text{ is prime}] \leq \frac{2^{3\log^c n}\log^\sigma n}{2^{\log^\sigma n}} \leq \frac{2^{3\log^c n}}{2^{0.9\log^\sigma n}}. \tag{3.1}$$

Let $p_0 = Pr[\text{no primes could be found}] \leq \left(1 - \frac{1}{\log^\sigma n}\right)^{\log^\lambda n}$. Thus we have

$$p_0 \leq \left(\frac{1}{2}\right)^{\log^{\lambda-\sigma} n}. \tag{3.2}$$

Let $\{r_1, \cdots, r_k\}$ be the primes that are *not good* and let $\{r_{k+1}, \cdots, r_K\}$ be the good primes. For $i = 1, 2$ the circuit Y_i outputs “fail” only when no prime is found. Thus, the probability that Y_i fails is p_0. Let p_{ij} be probability that Y_1 outputs $(x_i \bmod r_j, r_j)$ for all i and $j = k+1, \cdots, K$. Similarly, we can define q_{ij} as the probability that Y_2 outputs $(x_i \bmod r_j, r_j)$ for all i and $j = k+1, \cdots, K$. Since r_j is a good prime, for $j = k+1, \cdots, K$ we have for all i and $j = k+1, \cdots, K$,

$$\begin{aligned} p_{ij} &= Pr[X_1 = x_i]Pr[\text{the prime} = r_j] \\ q_{ij} &= Pr[X_2 = x_i]Pr[\text{the prime} = r_j] \end{aligned}$$

For r_j, $j \leq k$ let the possible outputs be $(z_{1j}, r_j), \cdots, (z_{l_j j}, r_j)$. Each z_{sj} is $x_i \bmod r_j$ for possibly more than one x_i. Let p_{ij} (q_{ij}) be the probability that Y_1 (resp. Y_2) outputs (z_{ij}, r_j) for all $j = 1, \cdots, k$ and $i = 1, \cdots, l_j$.

Next we calculate StatDiff(Y_1, Y_2).

$$\text{StatDiff}(Y_1, Y_2) = 1/2\left[(p_0 - p_0) + \sum_{j=i}^{k}\sum_{i=1}^{l_j}|p_{ij} - q_{ij}| + \sum_{i=1}^{M}\sum_{j=k+1}^{K}|p_{ij} - q_{ij}|\right]$$

We notice that

$$\begin{aligned} 1/2\textstyle\sum_{i=1}^{M}\sum_{j=k+1}^{K} & |p_{ij} - q_{ij}| \\ & = 1/2\textstyle\sum_{j=k+1}^{K}\sum_{i=1}^{M}|Pr[X_1 = x_i]Pr[r_j] - Pr[X_2 = x_i]Pr[r_j]| \\ & = \textstyle\sum_{j=k+1}^{K}Pr[r_j]\delta \\ & = (1-P)\delta \end{aligned}$$

[1]If $\pi(x)$ is the number of primes less than x then $0.922 \leq \frac{\pi(x)}{x/\ln x}$ which implies $\pi(x) \geq \frac{x}{\log x}$ (see [113]).

where $P = p_0 + \sum_{j=i}^{k} \sum_{i=1}^{l_j} p_{ij} = \sum_{j=i}^{k} \sum_{i=1}^{l_j} q_{ij} + p_0$.

Now,

$$1/2 \sum_{j=i}^{k} \sum_{i=1}^{l_j} |p_{ij} - q_{ij}| = 1/2(P - p_0) \sum_{j=i}^{k} \sum_{i=1}^{l_j} |p_{ij}/(P - p_0) - q_{ij}/(P - p_0)|$$

Notice that, $\sum_{j=i}^{k} \sum_{i=1}^{l_j} p_{ij} = \sum_{j=i}^{k} \sum_{i=1}^{l_j} q_{ij} = P - p_0$. Thus we have

$$0 \leq 1/2 \sum_{j=i}^{k} \sum_{i=1}^{l_j} |p_{ij} - q_{ij}| \leq (P - p_0).$$

Combining the above we obtain,

$$(1 - P)\delta \leq \text{StatDiff}(Y_1, Y_2) \leq (P - p_0) + (1 - P)\delta \leq P + (1 - P)\delta.$$

Next we approximate the value of P.

$$P = Pr[r \text{ is not good } \mid r \text{ is prime}] + Pr[\text{No prime could be found}]$$

From Equation 3.1 and 3.2 we obtain

$$P \leq \frac{2^{3 \log^c n}}{2^{0.9 \log^\sigma n}} + (\frac{1}{2})^{\log^{\lambda - \sigma} n}$$

If we choose $\sigma = c + 1$ and $\lambda = \sigma + 2 = c + 3$ then $P \leq 2^{-10}$ for $n > 100$. Observe that the choice of λ and σ does not depend upon the input size as long as the input size is more that 100 which we may assume without loss of generality.

With this parameters it is easy to see that the number of input and output bits of Y_1 and Y_2 is at most $\log^{3c} n$. ■

We state the following easy corollary.

Corollary 3.5.2 *Let X_1 and X_2 be two circuits encoding distributions with at most $\log^c n$ input bits where $n = \max\{\text{Size}(X_1), \text{Size}(X_2)\}$. Suppose,* StatDiff$(X_1, X_2) = \delta$. *Let (Y_1, Y_2) be the output of the output bit reduction algorithm of Theorem 3.5.1. Then,*

- StatDiff$(Y_1, Y_2) \leq 0.334$ *when* $\delta \leq 1/3$.
- StatDiff$(Y_1, Y_2) \geq 0.666$ *when* $\delta \geq 2/3$.

Polarization

Sahai and Vadhan [94] gave a polynomial-time process that takes a pair of circuits X_1 and X_2 as input and outputs a new pair of circuits Y_1 and Y_2 that have negligible statistical difference if X_1 and X_2 have "small" statistical difference. On the other hand if X_1 and X_2 have "large" statistical difference then Y_1 and Y_2 will have statistical difference almost close to 1. We state the properties of the algorithm precisely in the next theorem.

Lemma 3.5.3 (Sahai-Vadhan [94]) *Let $\alpha, \beta \in [0,1]$ be two constants such that $\alpha^2 > \beta$. Then there is a polynomial-time algorithm* Polarize$_{\alpha,\beta}$ *that on input $(X_1, X_2, 1^k)$, where X_1 and X_2 are two circuits encoding distribution, outputs a pair of circuits (Y_1, Y_2) such that*

- StatDiff$(X_1, X_2) \geq \alpha \Rightarrow$ StatDiff$(Y_1, Y_2) \geq 1 - \frac{1}{2^k}$ *and*

  StatDiff$(X_1, X_2) \leq \beta \Rightarrow$ StatDiff$(Y_1, Y_2) \leq \frac{1}{2^k}$.
- *The number of input (output) bits of circuits Y_1 and Y_2 is at most $2n(4k)^C k \log_\lambda(4k)$ (resp. $2m(4k)^C k \log_\lambda(4k)$), where n (resp. m) is the maximum number of input (resp. output) bits of the circuits X_1 and X_2, $\lambda = \min\{\alpha^2/\beta, 2\}$ and $C = \max\{\log_{\alpha^2/\beta}, (1/\beta), log_2(1/\beta)\}$.*

Sahai and Vadhan gave a proof system for SD and a corresponding simulator in [94] (see [106, Section 3.1.1]). This proof system is called *basic proof system*. The basic proof system was used along with polarization to give the final HVSZK proof for SD. We will also use their the proof system as a subroutine in the protocol for βSD$^{(c)}$. For the sake of completeness we describe the basic proof system in Protocol 3.

Protocol 3 Basic HVSZK Proof System for SD

Input: (X_1, X_2) with n input and m output gates.

1 **V**: Choose $b \in \{1,2\}$ uniformly at random. Obtain a sample x from X_b by picking r uniformly at random from $\{0,1\}^n$ and letting $x = X_b(r)$. Send x to P.

2 **P**: If $Pr[X_1 = x] > Pr[X_2 = x]$ set $a = 1$ otherwise $a = 2$. Send a to V.

3 **V**: If $a = b$ accept. Otherwise reject.

Simulator 4 exhibits the zero knowledge property of Protocol 3.

```
Simulator 4 Simulator for Protocol 3
Input: $(X_1, X_2)$ with $n$ input and $m$ output gates.
1 Choose $b \in \{1, 2\}$ uniformly at random. Choose $r$ uniformly at random from $\{0, 1\}^n$
    and let $x = X_b(r)$.
2 Let a=b;
3 Output (x,a;b,r)
```

Lemma 3.5.4 ([94]) *Let* $\mathrm{StatDiff}(X_1, x_2) = \delta$*. Then the prover strategy in Protocol 3 makes the verifier accept with probability* $(1 + \delta)/2$ *and no prover can succeed with greater probability.*

The output of the simulator algorithm in Simulator 4 has statistical difference exactly $(1 - \delta)/2$ *from the verifier's view.*

Now present the honest verifier protocol for the problem $\beta\mathrm{SD}^{(c)}$ in Protocol 5. Let $d \geq 1$ be a parameter in Protocol 5.

```
Protocol 5 Protocol for $\beta\mathrm{SD}^{(c)}$
Input: $(X_1, X_2, 1^k)$.
1 Both parties apply output bit reduction algorithm of Theorem 3.5.1 on $(X_1, X_2)$ to
    obtain circuits $(Y_1, Y_2)$.
2 Both parties apply Polarize$_{0.666,0.334}$ of Lemma 3.5.3 on input $(Y_1, Y_2, 1^{log^d k-1})$ to
    obtain circuits $(Z_1, Z_2)$.
3 Both parties execute Protocol 3 on the common input $(Z_1, Z_2)$. V accepts or rejects
    according to Protocol 3.
```

Notice that at the first step of the protocol we use the output bit reduction algorithm. This is done to ensure that the prover and verifier do not exchange message of large size. It is in this respect that the above protocol differs from the protocol for SD in [94]. In Simulator 6 we present the simulator for Protocol 5. Let $d \geq 1$ be the same parameter used in Protocol 5.

Theorem 3.5.5 *For every* $d \geq 1$*, Protocol 5 along with Simulator 6 exhibits that* $\beta\mathrm{SD}^{(c)}$ *is in* $\beta\mathrm{HVSZK}(\log^{O(c+d)}, \log^{O(c+d)}, 2)$ *with completeness error* $1/2^{\log^d k}$ *and soundness*

```
Simulator 6 Simulator for Protocol 5
Input: $(X_1, X_2, 1^k)$.
1 Apply Output bit reduction Theorem 3.5.1 on $(X_1, X_2)$ to obtain circuits $(Y_1, Y_2)$.
2 Apply Polarize$_{0.666,0.334}$ of Lemma 3.5.3 on input $(Y_1, Y_2, 1^{log^d k-1})$ to obtain circuits $(Z_1, Z_2)$.
3 Run the simulator algorithm 4 on $(Z_1, Z_2)$ and output whatever it outputs.
```

error $1/2 + 1/2^{\log^d k}$ *and the simulator deviation in Simulator 6 is* $1/2^{\log^d k}$.

Proof. By Theorem 3.5.1 the number of input and output bits of both Y_1 and Y_2 is at most $\log^{3c} n$ where $n = \max\{\text{Size}(X_1), \text{Size}(X_2)\}$. Notice if $(X_1, X_2) \in \beta\text{SD}_Y^{(c)}$ then by Corollary 3.5.2 StatDiff$(Y_1, Y_2) \geq 0.666$ and if $(X_1, X_2) \in \beta\text{SD}_N^{(c)}$ then StatDiff$(Y_1, Y_2) \leq 0.334$. When the prover, verifier in Protocol 5 or the simulator in Simulator 6 polarize (Y_1, Y_2) to obtain (Z_1, Z_2) it will have the following property:

$$\text{If } (X_1, X_2) \text{ is an “yes” instance then StatDiff}(Z_1, Z_2) \geq 1 - \frac{1}{2^{\log^d k} - 1} \text{ and}$$

$$\text{If } (X_1, X_2) \text{ is a “no” instance then StatDiff}(Z_1, Z_2) \leq \frac{1}{2^{\log^d k} - 1}.$$

The completeness and soundness of the protocol follows from Lemma 3.5.4. Moreover the number of input bits of Z_1 and Z_2 will be at most

$$2(\log^{3c} n)(\log^d k)[4 \log^d k]^4 \log_{1.3}(4 \log^d k),$$

which is less than $2^9 \log^{3c+6d}(n + k)$. We can also check that the number of output bits is at most $2^9 \log^{3c+6d}(n + k)$. The number of random bits used in Protocol 5 and Simulator 6 is same as the number of input bits of Z_1 and Z_2 plus one. Also the message size is just one bit more than the number of output bits of Z_1 and Z_2. This concludes the proof. ∎

Corollary 3.5.6 *The promise problem* $\beta\text{SD}^{(c)}$ *is in* $\beta\text{HVSZK}(\log^{O(c)}, \log^{O(c)}, 2)$.

Proof. In Theorem 3.5.5 if we put $d = 2$ we observe that the completeness error and the soundness error of Protocol 5 are $1/2^{\log^2 k}$ and $1/2 + 1/2^{\log^2 k}$ respectively. The simulator deviation in Simulator 6 is $1/2^{\log^2 k}$ which is a negligible function. ∎

3.5.2 Every problem in $\beta\text{HVSZK}(\log^{c_1}, poly, \log^{c_2})$ reduces to $\beta\text{ED}^{(c)}$

We will prove that every problem in βHVSZK$(\log^{c_1}, poly, \log^{c_2})$ is reducible to βED$^{(c)}$ for some suitable choice of c. We adopt the following assumptions from [106]. Let $\Pi = (\Pi_Y, \Pi_N)$ be a promise problem in βHVSZK$(\log^{c_1}, poly, \log^{c_2})$. Let (P, V) be a zero knowledge proof system for Π and let S be the simulator. We assume without loss of generality that the first message is sent by the prover P and the last message is sent by the verifier V and it is the random bits used by V (The verifier may be assumed to have decided whether to accept or reject just before revealing the its random bits). By definition the simulator simulates the verifier's coin. Thus, revealing random bits by V at the end does not affect the zero knowledge condition.

Let $R(n)$ be a bound on the number of random coins used by the verifier on input x of length n. The last message of V is these $R(n)$ random coins. Let $M(n)$ be a bound on the total length of the messages passed between the prover and the verifier and let $E(n)$ be a bound on the number of messages sent from the verifier to the prover. Notice that the total number of messages exchanged between the prover and the verifier is at most $2E(n)$. We also assume that the protocol has soundness and completeness error less than $\frac{1}{2^{40}}$. Let $D(n) = E(n)\log^{c_1+1} n$. Since by definition the simulator deviation μ is a negligible function we assume that $\mu \leq min\{\frac{1}{D(n)}, \epsilon\}$ where ϵ is a small constant that will be fixed later. Without loss of generality we can assume $R(n) \leq O(\log^{c_1}(n))$, $M(n) \leq O(poly(n))$ and $2E(n) \leq O(log^{c_2}(n))$.

Fortnow [51] pointed out that the simulator distribution gives enough information about the "yes" and "no" instances. The simulator output can be thought of as an interaction between a *virtual prover* and a *virtual verifier*. In the "yes" instances either i) the simulator outputs accepting conversation with high probability or ii) the virtual verifier behaves like the real verifier. On the other hand if both of these conditions are true in the "no" instances then the virtual prover will be able to convince the real verifier with high probability. The goal is to separate the "yes" instances from the "no" instances based on the simulator output. To define the notion of virtual prover we take the simulator distribution. The simulator distribution naturally defines a process which is called simulator based prover in [51] (see [106]).

For $j \leq 2E(n)$, we refer to a tuple of strings $\gamma = (m_1, m_2, \cdots, m_j; r)$ as a (partial) conversation transcript if the even-numbered messages in γ (including an accept or reject message) correspond to what V would have sent given the odd-numbered prover messages specified in γ and the random coins r.

For an input x let $S(x)_i$ and $\langle P, V\rangle(x)_i$ denote the first i messages exchanged. When the input is clear from the context we will sometimes denote these as S_i and $\langle P, V\rangle_i$ respectively.

Definition 3.5.7 *Let γ be a conversation history consisting of the first $2i$ messages and x be an input. The* simulation based prover P_s *outputs "fail" if the simulator S has γ as a prefix of an output with probability 0 otherwise it outputs α with probability $p_\alpha = Pr[S(x)_{2i+1} = (\alpha, \gamma) \mid S(x)_{2i} = \gamma]$.*

To show that SD is complete for SZK, Goldreich, Sahai and Vadhan [94, 57] proved several results bounding the relative entropy of the simulator distribution and the verifier's view. We observe that these results work in the limited randomness setting with a few modifications.

The next lemma measures the closeness between the distributions obtained from the simulator and simulation based prover and real verifier interaction.

Lemma 3.5.8 *[57]*

$$RelEnt(S(x), \langle P_s, V\rangle(x)) = R(n) - \sum_{i=1}^{E(n)} [\mathrm{H}(S(x)_{2i}) - \mathrm{H}(S(x)_{2i-1})].$$

Next we state a bound for $RelEnt(S(x), \langle P_s, V\rangle(x))$ in terms of statistical difference between the verifiers view and simulator distribution. For that we need the following well know fact (see [106]).

Fact 3.5.9 *Let X and Y be two distributions on universe $\mathcal{U}$, let $\mathcal{D} = \mathrm{Supp}(X) \cup \mathrm{Supp}(Y)$. Then*

$$|\mathrm{H}(X) - \mathrm{H}(Y)| \leq \log(|\mathcal{D} - 1|)\delta + \mathrm{H}_2(\delta)$$

where $\delta = \mathrm{StatDiff}(X, Y)$.

A lemma similar to the next lemma appears in [57]. We get the following version by using the limited randomness properties.

Lemma 3.5.10 *Let $\delta(x) = \mathrm{StatDiff}(S(x), \langle P, V\rangle(x))$. Then*

$$R(n) - \sum_{i=1}^{E(n)} [\mathrm{H}(S(x)_{2i}) - \mathrm{H}(S(x)_{2i-1})] \leq 2E(n) \left[\delta(x) O(\log^{c_1} n) + \mathrm{H}_2(\delta)\right].$$

Proof. Let $\overline{S}$ be a perfect simulator for (P, V). In that case the simulation based prover is P itself. We apply Lemma 3.5.8 to get

$$\begin{aligned} R(n) + \textstyle\sum_{i=1}^{2E(n)}(-1)^{i+1}\mathrm{H}(\langle P, V\rangle_i) &= R(n) + \textstyle\sum_{i=1}^{2E(n)}(-1)^{i+1}\mathrm{H}(\overline{S}_i) \\ &= RelEnt(\overline{S}, \langle P, V\rangle) \\ &= 0. \end{aligned}$$

So, we have

$$\begin{aligned} R(n) + \textstyle\sum_{i=1}^{2E(n)}(-1)^{i+1}\mathrm{H}(S_i) &\leq R(n) + \textstyle\sum_{i=1}^{2E(n)}(-1)^{i+1}\mathrm{H}(\langle P, V\rangle_i) \\ &\quad + \textstyle\sum_{i=1}^{2E(n)} |\mathrm{H}(S_i) - \mathrm{H}(\langle P, V\rangle_i)| \\ &\leq 0 + \textstyle\sum_{i=1}^{2E(n)} |\mathrm{H}(S_i) - \mathrm{H}(\langle P, V\rangle_i)| \\ &\leq 2E(n)[\delta(x)O(\log^{c_1} n) + \mathrm{H}_2(\delta)]. \end{aligned}$$

Recall that the prover, verifier and the simulator use at most $O(\log^{c_1} n)$ random bits. Therefore, the support of S_i and $\langle P, V\rangle_i$ have size at most $2^{O(\log^{c_1} n)}$. Thus, $|\mathrm{Supp}(S_i) \cup \mathrm{Supp}(\langle P, V\rangle_i)| \leq 2^{O(\log^{c_1} n)}$ The last inequality is obtained by applying Fact 3.5.9. ■

Definition 3.5.11 (Relative Entropy) *Let X and Y be two distributions on a finite set S. The* relative entropy *between X and Y, denoted* RelEnt(X, Y) *is defined as*

$$\mathrm{RelEnt}(X, y) = \sum_{s \in S} Pr[X = s] \log \frac{Pr[X = s]}{Pr[Y = s]}.$$

We define the binary relative entropy *for $p, q \in [0, 1]$ by*

$$\mathrm{RelEnt}_2(p, q) = p \log \frac{p}{q} + (1 - p) \log \frac{1 - p}{1 - q}.$$

Lemma 3.5.12 *[see [4, 106]] Let p denote the probability that S outputs an accepting transcript on input x, let q be the maximum probability, taken over all provers P^*, that V accepts in $(P^*, V)(x)$, and assume that $p \geq q$. Then,*

$$RelEnt(S(x), \langle P_S, V\rangle(x)) \geq RelEnt_2(p, q).$$

We are finally ready to give the reduction from $\Pi = (\Pi_Y, \Pi_N)$ to $\beta\mathrm{ED}^{(c)}$. The first distribution is

$$X = S_2 \otimes S_4 \otimes \cdots \otimes S_{2E(n)}.$$

The second distribution is

$$Y = Y_1 \otimes Y_2 \otimes Y_3,$$

where Y_1, Y_2 and Y_3 is defined as follows. $Y_1 = S_1 \otimes S_3 \otimes \cdots \otimes S_{2E(n)-1}$. The distribution Y_2 just outputs $R(n) - 7$ random bits. Y_3 runs the simulator S for $8\ln(D(n)+2)$ times independently. Recall that $D(n) = E(n)\log^{c_1+1} n$. If the verifier rejects the majority of the transcripts, it outputs $D(n)+2$ random bits. Otherwise it outputs fixed string "$fail$".

Since the simulator uses at most $O(\log^{c_1} n)$ random bits, the number of random bits used by the circuit X is $O(E(n)\log^{c_1} n)$. Similarly, the number of random bits used by the circuit Y is at most

$$O(E(n)\log^{c_1} n) + R(n) - 7 + 8\ln(D(n)+2)O(\log^{c_1} n) + D(n) + 2.$$

Notice that $R(n) \leq O(\log^{c_1} n)$, $E(n) \leq O(\log^{c_2} n)$. Therefore, the random bits used by both the circuits is at most $\log^{c_1+c_2+O(1)} n$. Thus, if we take $c = c_1 + c_2 + O(1)$ then the next two claims show that the promise problem $\Pi = (\Pi_Y, \Pi_N) \in \beta\text{HVSZK}(\log^{c_1}, poly, \log^{c_2})$ is polynomial-time many-one reducible to $\beta\text{ED}^{(c)}$.

Claim 3.5.13 *If* $x \in \Pi_Y$ *then* $\text{H}(X) \geq \text{H}(Y) + 1$.

Proof. By Lemma 3.5.10

$$\text{H}(Y_1) + R(n) - \text{H}(X) \leq 2E(n)\left[\mu O(\log^{c_1} n) + \text{H}_2(\mu)\right].$$

As μ is small we may assume $\text{H}_2(\mu) \leq \sqrt{\mu}$. This is where we use the assumption that $\mu \leq \min\{\frac{1}{D(n)}, \epsilon\}$ where $D(n) = E(n)\log^{c_1+1} n$. We also use the fact that the total size $M(n)$ of the messages is more than the number $E(n)$ of messages sent from verifier to the prover. Thus we have

$$\begin{aligned} \text{H}(Y_1) + R(n) - \text{H}(X) &\leq 2E(n)\left[\tfrac{1}{D(n)}O(\log^{c_1} n) + \sqrt{\tfrac{1}{E(n)M(n)}}\right] \\ &\leq 4. \end{aligned}$$

Next we bound the entropy of Y_3. In "yes" instances S outputs rejecting transcript with probability at most $2^{-40} + \mu \leq 1/4$. By Chernoff bound the probability p that the majority of the transcripts are rejecting is

$$p \leq \exp\left[-2\ .\ 8\ln(D(n)+2)\left(\frac{1}{4}\right)^2\right] \leq \frac{1}{D(n)+2}.$$

Using standard facts about entropy we get

$$\mathrm{H}(Y_3) \leq p[D(n)+2] + (1-p).0 + \mathrm{H}_2(p) \leq 2.$$

Thus we have $\mathrm{H}(Y) \leq \mathrm{H}(X)+1$. ■

Claim 3.5.14 *If* $x \in \Pi_N$ *then* $\mathrm{H}(Y) \geq \mathrm{H}(X)+1$.

Proof. We show that either $\mathrm{H}(Y_1)+\mathrm{H}(Y_2) \geq \mathrm{H}(X)+1$ or $\mathrm{H}(Y_3) \geq \mathrm{H}(X)+1$. Suppose that the simulator outputs accepting conversation with probability $\leq 1/4$. By Chernoff bound the probability $1-p$ that the majority of the transcripts are accepting is

$$1-p \leq \exp\left[-2\ .\ 8\ln(D(n)+2)\left(\frac{1}{4}\right)^2\right] \leq \frac{1}{D(n)+2}.$$

Using standard facts we get

$$\mathrm{H}(Y_3) \geq p[D(n)+2] \geq D(n)+1 \geq \mathrm{H}(X)+1.$$

Recall that X uses at most $O(E(n)\log^{c_1} n)$ random bits which implies that $\mathrm{H}(X) \leq O(E(n)\log^{c_1} n) \leq E(n)\log^{c_1+1} n = D(n)$.

Now suppose that the simulator outputs accepting transcript with probability at least $1/4$. Then by Lemma 3.5.12 the relative entropy between S and $\langle P_s, V\rangle$ is at least $\mathrm{RelEnt}_2(1/4, 2^{-40}) \geq 8$. Thus by Lemma 3.5.8, $R(n) - \mathrm{H}(X) + \mathrm{H}(Y_1) \geq 8$. This gives us

$$\mathrm{H}(Y_1) + \mathrm{H}(Y_2) \geq R(n) - 7 + 8 + \mathrm{H}(X) - R(n) = \mathrm{H}(X)+1.$$

■

Theorem 3.5.15 *Every problem in* $\beta\mathrm{HVSZK}(\log^{c_1}, poly, \log^{c_2})$ *is polynomial-time many-one reducible to* $\beta\mathrm{ED}^{(c)}$ *where* $c = c_1 + c_2 + O(1)$.

3.5.3 $\beta\text{ED}^{(c)}$ reduces to $\beta\text{SD}^{(c')}$

We prove that $\beta\text{ED}^{(c)}$ reduces to $\beta\text{SD}^{(c')}$ where $c' = O(c)$. First we design a new output bit reduction algorithm. In general a circuit of size of size N could have as many as $\Theta(N)$ output bits. This new output bit reduction algorithm, given two distributions encoded by boolean circuits X_1 and X_2 where the number of input bits of X_1 and X_2 is polylogarithmic in $N = \max\{\text{Size}(X_1), \text{Size}(X_2)\}$ computes two new circuits Y_1 and Y_2 such that their entropy difference is close to the entropy difference between X_1 and X_2 but the number of input bits and output bits of Y_1 and Y_2 is polylogarithmic in N. Finally we use the techniques developed by Goldreich and Vadhan [57] on the output bit reduced circuits to get the final circuits which has the required statistical difference.

The output bit reduction for ED also uses the same idea of fingerprinting by primes of suitable size as in Lemma 3.5.1.

Theorem 3.5.16 (Output Bit Reduction for Entropy Difference) *Let X_1 and X_2 be two circuits encoding distributions where X_1 and X_2 have at most $n = \log^c N$ input bits where $N = \max\{\text{Size}(X_1), \text{Size}(X_2)\}$.*

Then in time polynomial in N we can compute two circuits Y_1 and Y_2 with the following properties.

- *If* $\text{H}(X_1) \geq \text{H}(X_2) + 1$ *then* $\text{H}(Y_1) \geq \text{H}(Y_2) + 1$.

  If $\text{H}(X_2) \geq \text{H}(X_1) + 1$ *then* $\text{H}(Y_2) \geq \text{H}(Y_1) + 1$
- *The number of input and output bits of Y_1 and Y_2 is at most $\log^{3c} N$.*

Proof. From the two circuits X_1 and X_2 we first construct two circuits A_1 and A_2 respectively as follows:

$$A_1 = \otimes^8 X_1$$
$$A_2 = \otimes^8 X_2$$

Thus,

$$\text{H}(A_1) \geq \text{H}(A_2) + 8 \quad \text{if } \text{H}(X_1) \geq \text{H}(X_2) + 1 \quad \text{and,} \tag{3.3}$$
$$\text{H}(A_2) \geq \text{H}(A_1) + 8 \quad \text{if } \text{H}(X_2) \geq \text{H}(X_1) + 1. \tag{3.4}$$

The circuits A_1 and A_2 have at most $8n$ input gates.

The circuit description is same as in the proof of Theorem 3.5.1 For $i = 1, 2$ we describe the circuits Y_i below. Let λ and σ be positive numbers which will be fixed later.

The circuit Y_i picks a list of $\log^{\lambda} N$ random $\log^{\sigma} N$ bit numbers. Then using AKS primality testing [1] it picks the first prime number r appearing in the list. If the list contains no prime number the circuit outputs "fail". Otherwise it samples the distribution encoded by the circuit X_i by first picking a $\log^{c} N$ bit number s and then evaluating the circuit X_i on s. Let $x = X_i(s)$. The circuit Y_i then outputs $(x \bmod r, r)$.

Notice that the number of output bits of Y_i is at most $2\log^{\sigma} N$ and the number of input bits is at most $8\log^{c} N + \log^{\lambda+\sigma} N$.

Let

$$\text{Supp}(A_1) \cup \text{Supp}(A_2) = \{x_1, \cdots, x_M\}.$$

Observe that $M \leq 2.2^{8n}$.

We say that a prime p is *good* if

$$x_i - x_j \neq 0 \bmod p \text{ for all } 1 \leq i < j \leq M.$$

The numbers $x_1 - x_j$ can have at most $\log(2^{8n+1}) = 8n + 1$ prime factors. So, among all $\log^{\sigma} N$ bit prime numbers at most $\begin{pmatrix} M \\ 2 \end{pmatrix}(8n+1)$ primes are not good.

Notice that

$$\begin{pmatrix} M \\ 2 \end{pmatrix}(8n+1) \leq 8M^2 n.$$

By prime number theorem (see [113]) the number of $\log^{\sigma} N$ bit prime numbers is approximately $\frac{2^{\log^{\sigma} N}}{\log^{\sigma} N}$.

$$Pr[r \text{ is not good } \mid r \text{ is prime}] \leq \frac{8M^2 n \log^{\sigma} N}{2^{\log^{\sigma} N}}. \tag{3.5}$$

Let $p_0 = Pr[\text{no primes could be found}] \leq \left(1 - \frac{1}{\log^{\sigma} N}\right)^{\log^{\lambda} N}$. Thus we have

$$p_0 \leq \left(\frac{1}{2}\right)^{\log^{\lambda-\sigma} N}. \tag{3.6}$$

Let $\{r_1, \cdots, r_k\}$ be the primes that are *not good* and let $\{r_{k+1}, \cdots, r_K\}$ be the good primes. When no prime is found the circuit outputs "fail". The probability that Y_i fails is

p_0. Let p_{ij} be probability that Y_1 outputs $(x_i \bmod r_j, r_j)$ for all i and $j = k+1, \cdots, K$. Similarly, we can define q_{ij} as the probability that Y_2 outputs $(x_i \bmod r_j, r_j)$ for all i and $j = k+1, \cdots, K$. Then, for all i and $j = k+1, \cdots, K$,

$$\begin{aligned} p_{ij} &= Pr[A_1 = x_i] Pr[\text{the prime} = r_j] \\ q_{ij} &= Pr[A_2 = x_i] Pr[\text{the prime} = r_j] \end{aligned}$$

For $r_j, j \leq k$ let the possible outputs be $(z_{1j}, r_j), \cdots, (z_{l_j j}, r_j)$. Each z_{sj} is $x_i \bmod r_j$ for possibly more than one x_i. Let p_{ij} be the probability that Y_1 outputs (z_{ij}, r_j) for all $j = 1, \cdots, k$ and $i = 1, \cdots, l_j$. Similarly, let q_{ij} be the probability that Y_2 outputs (z_{ij}, r_j) for all $j = 1, \cdots, k$ and $i = 1, \cdots, l_j$.

Next we calculate $\mathrm{H}(Y_1)$ and $\mathrm{H}(Y_2)$ in terms of $\mathrm{H}(A_1)$ and $\mathrm{H}(A_2)$ respectively.

$$\mathrm{H}(Y_1) = p_0 \log \frac{1}{p_0} + \sum_{j=i}^{k} \sum_{i=1}^{l_j} p_{ij} \log \frac{1}{p_{ij}} + \sum_{i=1}^{M} \sum_{j=k+1}^{K} p_{ij} \log \frac{1}{p_{ij}}$$

We notice that

$$\begin{aligned} \textstyle\sum_{i=1}^{M} \sum_{j=k+1}^{K} p_{ij} \log \frac{1}{p_{ij}} &= \textstyle\sum_{j=k+1}^{K} \sum_{i=1}^{M} Pr[A_1 = x_i] Pr[r_j] \log \frac{1}{Pr[A_1 = x_i] Pr[r_j]} \\ &= \textstyle\sum_{j=k+1}^{K} \sum_{i=1}^{M} Pr[A_1 = x_i] Pr[r_j] \log \frac{1}{Pr[A_1 = x_i]} \\ &\quad \textstyle + \sum_{i=1}^{M} Pr[A_1 = x_i] \sum_{j=k+1}^{K} Pr[r_j] \log \frac{1}{Pr[r_j]} \\ &= \textstyle\sum_{j=k+1}^{K} Pr[r_j] \mathrm{H}(A_1) + \sum_{j=k+1}^{K} Pr[r_j] \log \frac{1}{Pr[r_j]} \\ &= \textstyle\mathrm{H}(A_1)(1 - P) + \sum_{j=k+1}^{K} Pr[r_j] \log \frac{1}{Pr[r_j]} \end{aligned}$$

where $P = p_0 + \sum_{j=i}^{k} \sum_{i=1}^{l_j} p_{ij} = p_0 + \sum_{j=i}^{k} \sum_{i=1}^{l_j} q_{ij}$.

Let,

$$\begin{aligned} Z_1 &:= p_0 \log \tfrac{1}{p_0} + \textstyle\sum_{j=i}^{k} \sum_{i=1}^{l_j} p_{ij} \log \frac{1}{p_{ij}} \\ Z_2 &:= p_0 \log \tfrac{1}{p_0} + \textstyle\sum_{j=i}^{k} \sum_{i=1}^{l_j} q_{ij} \log \frac{1}{q_{ij}} \end{aligned}$$

Let D_1 be a distribution on a set $\mathcal{D} = \{a_{ij} \mid j = 1, \cdots, k;\ i = 1, \cdots, l_j\} \cup \{a_0\}$ with $Pr[D_1 = a_0] = p_0/P$ and $Pr[D_1 = a_{ij}] = p_{ij}/P$. Similarly we define D_2 on $\mathcal{D}$ with $Pr[D_2 = a_0] = p_0/P$ and $Pr[D_2 = a_{ij}] = q_{ij}/P$. It is easy to check that $\mathrm{H}(D_1) = \log P + \frac{1}{P} Z_1$. Similarly, $\mathrm{H}(D_2) = \log P + \frac{1}{P} Z_2$. This gives, $Z_1 = P\mathrm{H}(D_1) + P \log \frac{1}{P}$

and $Z_2 = P\mathrm{H}(D_2) + P\log\frac{1}{P}$. Thus, we get

$$\mathrm{H}(Y_1) = \mathrm{H}(A_1)(1-P) + P\mathrm{H}(D_1) + P\log\frac{1}{P} + \sum_{j=k+1}^{K} Pr[r_j]\log\frac{1}{Pr[r_j]}$$

and

$$\mathrm{H}(Y_2) = \mathrm{H}(A_2)(1-P) + P\mathrm{H}(D_2) + P\log\frac{1}{P} + \sum_{j=k+1}^{K} Pr[r_j]\log\frac{1}{Pr[r_j]}.$$

The distributions D_1 and D_1 has supports of size at most $Mk+1$. Hence we have $0 \leq \mathrm{H}(D_1), \mathrm{H}(D_2) \leq \log(Mk+1)$. Finally we get,

$$\mathrm{H}(Y_1) - \mathrm{H}(Y_2) \geq (1-P)[\mathrm{H}(A_1) - \mathrm{H}(A_2)] - P\log(Mk+1) \tag{3.7}$$

$$\mathrm{H}(Y_2) - \mathrm{H}(Y_1) \geq (1-P)[\mathrm{H}(A_2) - \mathrm{H}(A_1)] - P\log(Mk+1) \tag{3.8}$$

From Equation 3.5 and 3.6 we get

$$\begin{aligned} P &= Pr[r \text{ is not good } \mid r \text{ is prime}\}] + Pr[\text{No prime could be found}] \\ &\leq \tfrac{8M^2 n \log^\sigma N}{2^{\log^\sigma N}} + (\tfrac{1}{2})^{\log^{\lambda-\sigma} N} \end{aligned}$$

Using the upper bounds $M < 2^{8n+1}$ and $k < 8M^2 n$ (recall that k is the number of primes that are not good) and $n = \log^c N$ we get

$$\begin{aligned} P &\leq \tfrac{8M^2 n \log^\sigma N}{2^{\log^\sigma N}} + (\tfrac{1}{2})^{\log^{\lambda-\sigma} N} \\ &\leq \tfrac{8.2^{16\log^c(N)+2}\log^c N \log^\sigma N}{2^{\log^\sigma N}} + (\tfrac{1}{2})^{\log^{\lambda-\sigma} N} \\ &\leq \tfrac{2^{17\log^c N}}{2^{\log^\sigma N}} + (\tfrac{1}{2})^{\log^{\lambda-\sigma} N} \\ &\leq (\tfrac{1}{2})^{\log^{\sigma-c-1} N} + (\tfrac{1}{2})^{\log^{\lambda-\sigma} N} \end{aligned}$$

We also have

$$\begin{aligned} \log(Mk+1) &\leq \log(2^{8\log^c(N)+1} 8.2^{16\log^c(N)+2}\log^c N) \\ &\leq \log 2^{25\log^c N} \\ &\leq \log^{c+1} N, \quad \text{for large enough } N. \end{aligned}$$

Thus we get,

$$P \log(Mk+1) \le [(\tfrac{1}{2})^{\log^{\sigma-c-1} N} + (\tfrac{1}{2})^{\log^{\lambda-\sigma} N}] \log^{c+1} N, \text{ for large enough } N.$$

If we choose $\sigma = c+3$ and $\lambda = \sigma + 2 = c+5$ then this quantity is at most $1/8$ for large enough N. Observe that the choice of λ and σ does not depend upon the input size as long as the input size is sufficiently large which we may assume without loss of generality. Thus, from Equations 3.7 and 3.8 we have

$$\mathrm{H}(Y_1) - \mathrm{H}(Y_2) \ \ge\ (1-1/8).8 - 1/8 \ge 1 \text{ when } \mathrm{H}(A_1) - \mathrm{H}(A_2) \ge 8 \quad (3.9)$$
$$\mathrm{H}(Y_2) - \mathrm{H}(Y_1) \ \ge\ (1-1/8).8 - 1/8 \ge 1 \text{ when } \mathrm{H}(A_2) - \mathrm{H}(A_1) \ge 8. \quad (3.10)$$

Using Equations 3.3 and 3.4 we have,

$$\text{If } \mathrm{H}(X_1) \ge \mathrm{H}(X_2) + 1 \text{ then } \mathrm{H}(Y_1) \ge \mathrm{H}(Y_2) + 1.$$
$$\text{If } \mathrm{H}(X_2) \ge \mathrm{H}(X_1) + 1 \text{ then } \mathrm{H}(Y_2) \ge \mathrm{H}(Y_1) + 1.$$

Finally, with the choice of σ and λ it is easy to see that the number of input and output bits of Y_1 and Y_2 is at most $\log^{3c} N$. ■

Let (X_1, X_2) be an instance of $\beta\mathrm{ED}^{(c)}$ with each circuits having at most $n = \log^c N$ input bits where $N = \max\{\mathrm{Size}(X_1), \mathrm{Size}(X_2)\}$. We use output bit reduction algorithm of Theorem 3.5.16 to obtain a pair of circuits (Y_1, Y_2) such that they have at most $m = \log^{3c} N$ input and output bits. Now the technique of Goldreich and Vadhan [57] can be applied directly to these circuits. For the sake of completeness, we reproduce their entire argument. Let $q = 9\rho m^2$. We will fix ρ later. Let $Z_1 = \otimes^q Y_1$ and $Z_2 = \otimes^q Y_2$. These two circuits has at most $m' = 9\rho \log^{9c} N$ input gates and output bits. From Z_1 and Z_2, two circuits A_1 and A_2 can be constructed as follows. Let $\mathcal{H}_{2m',m'}$ be a 2-universal family of hash functions.

A_1:

- Pick r uniformly at random from $\{0,1\}^{m'}$, let $x = Z_1(r)$.
- Pict h uniformly at random from $\mathcal{H}_{2m',m'}$.
- Pick y according to the distribution Z_2.

- Output $(x, h, h(r, y))$.

A_2:

- Pick x according to the distribution Z_1.
- Pict h uniformly at random from $\mathcal{H}_{2m',m'}$.
- Pick z uniformly at random from $\{0,1\}^{m'}$.
- Output (x, h, z).

Notice that the above two circuits A_1 and A_2 use at most $(2m'+3)m'$ random bits. It is easy to verify that $(2m'+3)m' \leq 243\rho^2 \log^{18c} N$.

Theorem 3.5.17 (see [106, Section3.4.4]) *If* $\mathrm{H}(X_1) \geq \mathrm{H}(X_2)+1$ *then* $\text{StatDiff}(A_1, A_2) \geq 1 - O(2^{-\rho})$. *If* $\mathrm{H}(X_2) \geq \mathrm{H}(Y)+1$ *then* $\text{StatDiff}(A_1, A_2) \leq \Omega(2^{-\rho})$.

Let ρ be a sufficiently large constant. From the above theorem it is easy to see that if $(X_1, X_2) \in \beta\text{ED}_Y^{(c)}$ then $(A_1, A_2) \in \beta\text{SD}_Y^{(c')}$ and if $(X_1, X_2) \in \beta\text{ED}_N^{(c)}$ then $(A_1, A_2) \in \beta\text{SD}_N^{(c')}$ where $c' = 19c$ (Notice, for a fixed constant ρ, $243\rho^2 \log^{18c} N \leq \log^{19c} N$). Thus, we obtain the following theorem.

Theorem 3.5.18 *The problem* $\beta\text{ED}^{(c)}$ *reduces to* $\beta\text{SD}^{(c')}$ *where* $c' = 19c$.

3.6 Conclusion

In this chapter we proved that every problem having $\beta\text{HVSZK}(\log^{c_1}, poly, \log^{c_2})$ proof system also has a $\beta\text{HVSZK}(\log^{O(c_1+c_2)}, \log^{O(c_1+c_2)}, 2)$ proof system. As a corollary we obtain the following.

Corollary 3.6.1 *The problem* GrIso *is in* $\beta\text{HVSZK}(\log^c, \log^c, 2)$ *for some constant* c.

Proof. By Lemma 3.4.6 GrIso $\in \beta\text{HVSZK}(\log^2, poly, 3)$. Using the above argument we obtain GrIso $\in \beta\text{HVSZK}(\log^{O(2)}, \log^{O(2)}, 2)$. Hence, the result follows. ∎

Remark 3.6.2 *We note that using result similar to Theorem 3.5.1 directly in the protocol we can design a statistical zero knowledge proof system (zero knowledge against any verifier as in Definition 3.4.2) for* GrIso *with polylogarithmic randomness, small message size and constant i.e.,* GrIso $\in$ SZK$((\log^{c_1}, \log^{c_2}, c)$ *for some constants* c, c_1 *and* c_2.

4 Isomorphism and Canonization of Tournaments and Hypertournaments

4.1 Introduction

Computing canonical forms for graphs (and other combinatorial and algebraic structures) is a fundamental problem. Graph canonization, in particular, is very well-studied for its close connection to Graph Isomorphism GRAPH-ISO. Let $\mathcal{G}$ denote all simple undirected graphs on n vertices. A mapping $f : \mathcal{G} \longrightarrow \mathcal{G}$ is a *canonizing function* for $\mathcal{G}$ if for all $X, X' \in \mathcal{G}$: $f(X) \cong X$ and $f(X) = f(X')$ if and only if $X_1 \cong X_2$. I.e., f assigns a *canonical form* to each isomorphism class of graphs. For instance, we can define $f(X)$ as the lexicographically least graph isomorphic to X. This particular canonizing function is computable in $\mathrm{FP}^{\mathrm{NP}}$ by prefix search, but it is known to be NP-hard to compute [30, 82] for certain graph orderings. A specific ordering of graphs that makes the problem NP-complete is described in [30, Section 3.1].

It is a long-standing open question, whether there is *some* canonizing function for graphs that is polynomial-time computable. No better bound than $\mathrm{FP}^{\mathrm{NP}}$ is known for general graphs (for any canonizing function). It is easy to see that GRAPH-ISO is polynomial-time reducible to graph canonization. It is an intriguing open question if the converse reduction holds in general. However, for natural subclasses of graphs for which graph isomorphism has an efficient algorithm there is usually an accompanying efficient canonization algorithm [30]. Specifically, we do not know of any natural subclass of graphs for which graph isomorphism is in polynomial time and graph canonization is not known to be solvable in polynomial time. However, for hypergraphs with

n vertices there is a $2^{O(n)}$ time isomorphism algorithm but no canonization algorithm is known with the same running time [83]. Very recently, Babai and Codenotti [25] have shown a $2^{\tilde{O}(k^2\sqrt{n})}$ isomorphism testing algorithm for hypergraphs with hyperedges of size bounded by k. Here too it is open if canonization can be done in $2^{o(n)}$ time.

The Results of This Chapter

In this chapter we study the complexity of canonization and isomorphism of tournaments as well as hypertournaments. A central motivation for our study is the question whether T-CANON is polynomial-time reducible to TOUR-ISO. While we are not able to settle this question we prove an interesting weaker result: T-CANON has a polynomial-time oracle algorithm with oracle access to TOUR-ISO and an oracle for canonizing *rigid tournaments*. Rigid tournaments have no nontrivial automorphism. It is open whether a similar result holds for general graphs.

The other result in this chapter is an $n^{O(k^2+\log n)}$ algorithm for canonization and isomorphism of k-hypertournaments which builds on [30] and uses quite different properties of the automorphism groups of hypertournaments.

Our approach is based on the techniques of the seminal paper of Babai and Luks [30]. In the sequel we explain the group-theoretic setting in some detail since we will use their approach and methods.

Group Theoretic Preliminaries

In this chapter we need some more group theoretic concepts (see [114, 82]) along with those discussed in Chapter2.

Definition 4.1.1 *A permutation group $G \leq \mathrm{Sym(V)}$ is said to be* transitive *on V if $v^G = V$ for $v \in V$.*

Let $G \leq \mathrm{Sym(V)}$ be a transitive permutation group. A nonempty subset $B \subseteq V$ of points is called a *G-block* if either $B^g = B$ or $B^g \cap B = \emptyset$, for each $g \in G$. For any transitive group G, clearly the whole set V and the singleton sets $\{u\}$, $u \in V$ are blocks; these are known as the *trivial* blocks of G.

Definition 4.1.2 *A transitive permutation group $G \leq \mathrm{Sym(V)}$ is said to be* primitive *if it does not have any nontrivial blocks. Otherwise G is said to be* imprimitive.

Let $G \leq \mathrm{Sym(V)}$ be transitive. Notice that a subset $B \subseteq V$ is a G-block if and only if B^g is a G-block for every $g \in G$. It can be easily seen that the collection of blocks $\{B^g : g \in G\}$ forms a partition of V. This collection of blocks is called the B *block system*. Notice that the permutation group G acts transitively on the B block system (since every $g \in G$ naturally maps blocks to blocks and the action is obviously transitive). A nontrivial block $B \subset V$ is called a *maximal block* if there is no nontrivial block $B' \subset V$ such that $B \subset B'$. In this case, we say that the B-block system $\{B^g : g \in G\}$ is a *maximal block system*.

For a transitive group $G \leq \mathrm{Sym(V)}$, let B and B' be two G-blocks in V such that $B \subset B'$.Then the collection of blocks $\{B^g : g \in G, B^g \subseteq B'\}$ is actually a partition of B'.

A G-block $B \subset V$ is a *maximal subblock* of a G-block B' if $B \subset B'$ and there is no G-block C such that $B \subset C \subset B'$. Let B and B' be G-blocks. A chain $B = B_0 \subset \cdots \subset B_t = B'$ is a *maximal chain* of G-blocks between B and B' if for all i, B_i is a maximal subblock of B_{i+1}.

Let B and B' be two G-blocks such that $B \subset B'$. The B-*block system of* B' is the collection

$$\{B^g : g \in G \text{ and } B^g \subset B'\},$$

which forms a partition of B'. Hence $|B|$ divides $|B'|$.

A *structure tree* of G is a rooted tree whose nodes are labeled by G-blocks such that:

1. The root is labeled V.

2. The leaves are labeled with singleton sets $\{v\}$, $v \in V$.

3. For each internal node labeled by B', the labels of its children constitute a B block system of B', where $B \subset B'$ is a maximal block contained in B'.

There is a natural action of G on each level of a structure tree: $g \in G$ maps a node r to r' in a level iff there is a G-block B such that the labels of r and r' are B and B^g respectively. Furthermore, the action of G on the children of each node in the tree is primitive.

If $G \leq \mathrm{Sym(V)}$ has orbits $V_1, \ldots, V_r$, the *structure forest* is a collection of structure trees $T_1, \ldots, T_r$, where T_i is the structure tree of the transitive action of G on V_i.

For a permutation group $G \leq \mathrm{Sym(V)}$ and a subset of points $\Delta \subset V$, the *set stabilizer* subgroup for Δ is

$$G_\Delta = \{g \in G \mid \Delta^g = \Delta\}.$$

The Babai-Luks canonization procedure

We are now ready to describe the Babai-Luks machinery from [30] and recall some of their results in a form that is useful to us.

Let $X = (V, E)$ be a graph. Let $g \in \mathrm{Sym(V)}$ be a permutation. The action of g on X produces the graph $X^g = (V, E^g)$ where $E^g = \{(u^g, v^g) \mid (u, v) \in E\}$. The graph X^g is sometimes denoted as $g(X)$. Let $G \leq \mathrm{Sym(V)}$. Let $\mathcal{G}$ be any class of graphs (directed or undirected) each of which is defined on vertex set V. We say that the class of graphs $\mathcal{G}$ is *closed under G-isomorphisms* if for every permutation $g \in G$ and graph $X \in \mathcal{G}$ we have $X^g \in \mathcal{G}$.

Let $\mathcal{G}$ be any class of graphs closed under G-isomorphisms. For $X_1, X_2 \in \mathcal{G}$ with vertex set V, we say X_1 is *G-isomorphic* to X_2, denoted by $X_1 \cong_G X_2$ if $X_2 = X_1^g$ for some $g \in G$.

Call $\mathrm{CF}_G : \mathcal{G} \to \mathcal{G}$ a *canonizing function* w.r.t to G, if $\mathrm{CF}_G(X) \cong_G X$, for every $X \in \mathcal{G}$, and $X_1 \cong_G X_2$ if and only if $\mathrm{CF}_G(X_1) = \mathrm{CF}_G(X_2)$, for $X_1, X_2 \in \mathcal{G}$. When the group G is $\mathrm{Sym(V)}$, we write $\mathrm{CF}(X)$ instead of $\mathrm{CF}_G(X)$.

Given a canonizing function CF_G we define a canonizing function $\mathrm{CF}_{G\sigma}$ with respect to a coset $G\sigma$ of G as $\mathrm{CF}_{G\sigma}(X) = \mathrm{CF}_{\sigma^{-1}G\sigma}(X^\sigma)$. Notice that if $\mathcal{G}$ is closed under G isomorphisms then $\mathcal{G}^\sigma$ is closed under $\sigma^{-1}G\sigma$ isomorphisms. We will sometimes denote $\mathrm{CF}_{G\sigma}(X)$ as $\mathrm{CF}(X, G\sigma)$.

Next, we define the *canonical labeling coset* $\mathrm{CL}(X, G\sigma)$ as $\{\tau \in G\sigma \mid X^\tau = \mathrm{CF}_{G\sigma}(X)\}$. It is easy to see that $\mathrm{CL}(X, G\sigma) = (G \cap \mathrm{Aut}(X))\pi = \mathrm{Aut}_G(X)\pi$ for any $\pi \in \mathrm{CL}(X, G\sigma)$, where $\mathrm{Aut}_G(X) = \mathrm{Aut}(X) \cap G$. I.e.

$$\mathrm{Aut}_G(X) = \{g \in G \mid X^g = X\}.$$

We also notice that $\mathrm{CL}(X, G\sigma) = \sigma\mathrm{CL}(X^\sigma, \sigma^{-1}G\sigma)$.

Babai and Luks [30] gave a canonizing algorithm that exploits the group structure of G. The algorithm is recursive and works by a divide-and-conquer strategy on the group G. The divide-and-conquer is based on a structure forest of G. We now briefly describe the algorithm of Babai-Luks that computes the canonical labeling coset CL$(X, G\sigma)$ of a graph X with respect to a coset $G\sigma$.

The first step is to bijectively encode n-vertex graph X as a binary string x. To be specific, we can choose the length n^2 encoding obtained from the adjacency matrix as the concatenation of its rows. Thus, for $1 \leq i, j \leq n$ we have $x(i, j) = 1$ if and only if $(i, j) \in E(X)$. Then permutations $\sigma \in \mathrm{Sym(V)}$ acts on such strings x naturally by sending it to x^σ, where for all $1 \leq i, j \leq n$

$$x^\sigma(i^\sigma, j^\sigma) = x(i, j).$$

Let $m = n^2$. We can think of the given permutation group $G = \langle g_1, \cdots, g_k \rangle$ as a subgroup of S_m acting on the m-bit binary strings as described above, where (i, j) indexes into the m-bit binary string x. The natural ordering on $\{1, \cdots, m\}$ induces an ordering on the subsets of $\{1, \cdots, m\}$. Thus, we can talk about the first G-orbit $A_1 \subseteq [m]$ according to this ordering. Let $A_2 = [m] \setminus A_1$. The algorithm computes CL$(x, G\sigma)$ recursively as CL$(x|_{A_2}, \mathrm{CL}(x|_{A_1}, G\sigma))$, where $x|_{A_i}$ is the substring of x induced by A_i. Thus, in general we need to compute CL$(x|_A, G\sigma)$, where $A \subseteq [m]$ is a *G-stable subset*: each element of G maps A to A. Let CL$_A(x, G\sigma) = \mathrm{CL}(x|_A, G\sigma)$.

If $|A| = 1$ we define CL$_A(x, G\sigma) = G\sigma$. The nontrivial case is when G acts transitively on A. Then, w.r.t. the natural ordering on $[m]$ we can compute the first maximal G-block system on A in polynomial time. Let H be the set stabilizer of this block system. Let $\{\tau_i\}_{i=1}^k$ be a set of coset representatives of H in G. Let G$\sigma = \cup_{i=1}^k H\sigma_i$ where $\sigma_i = \sigma\tau_i$. Recursively compute CL$_A(x, H\sigma_i) = H_i\rho_i$ for all i. Then sort the cosets so that $x^{\rho_1} = x^{\rho_2} = \cdots = x^{\rho_s} < x^{\rho_{s+1}} \leq \cdots \leq x^{\rho_k}$. Output CL$_A(x, G\sigma) = \langle H_1, \{\rho_i\rho_1^{-1}\}_{i=1}^s \rangle \rho_1$.

This, in a nutshell, is the Babai-Luks canonization algorithm (we explain the algorithm with more details in Section 4.3). Clearly, we can recover the canonical labeling coset for the given graph X from the corresponding coset CL$(x, G\sigma)$ for its string encoding x. A detailed analysis of the algorithm can be found in [30].

Definition 4.1.3 *A finite group G is in the class Γ_d if every* nonabelian *composition factor of the group G embeds in the permutation group S_d.*

In other words, all nonabelian composition factors of a group $G \in \Gamma_d$ can be seen as subgroups of S_d. A crucial result of [30] is that the above string canonization algorithm for computing $\mathrm{CL}(x, G\sigma)$ runs in polynomial time if $G \in \Gamma_d$ for a constant d. We state this result in the form of a theorem useful to us for graphs.

Theorem 4.1.4 Babai-Luks Theorem: *If $G \leq S_m$ is a permutation group in the class Γ_d (i.e. all nonabelian composition factors of G are subgroups of S_d), then a canonical labeling coset of a binary string $x \in \{0,1\}^m$ w.r.t a coset $G\sigma$ of S_m, can be found in time m^c, where c depends only on d.*

Theorem 4.1.4 crucially uses the fact that primitive subgroups of S_m in Γ_d are of size bounded by $m^{O(d)}$ [90, 115, 24, 78].

Iterative Canonization

A *finite relational structure* X is a tuple $(D, R_1, \cdots, R_l)$ where D is a finite set called domain and $R_1, \cdots, R_l$ are relations on D with arity $a_1, \cdots, a_l$ respectively. The finite relational structure $X = (D, R_1, \cdots, R_l)$ has bounded arity if for all i, $a_i \leq c$ where c is some constant.

Let $X = (D, R_1, \cdots, R_l)$ be a finite relational structure where R_i has arity a_i and $g \in \mathrm{Sym(D)}$ be a permutation. We can define the relational structure $X^g = (D, R_1^g, \cdots, R_l^g)$ where $R_i^g = \{(d_1^g, \cdots, d_{a_i}^g) \mid (d_1, \cdots, d_{a_i}) \in R_i\}$. A relational structure $Y = (D, S_1, \cdots, S_l)$ is said to be isomorphic if to X if there is a permutation $g \in \mathrm{Sym(D)}$ such that $X^g = Y$. Once we have the notion of isomorphism and group action on relational structures we can easily define canonical form and canonical labeling coset as we did before for graphs.

Let $X = (D, R_1, R_2)$ be a finite relational structure with domain D and two relations R_1 and R_2 with arity a_1 and a_2 respectively. Let $X_1 = (D, R_1)$ and $X_2 = (D, R_2)$ be two relational structures derived from X. Let $\mathrm{CL}_1(X_1, G\sigma) = H_1\rho_1$ be the canonical labeling coset of X_1 w.r.t. some coset $G\sigma$ and some canonical form CF_1. Further $\mathrm{CL}_2(X_2, H_2\rho_1) = H_2\rho_2$ be the canonical labeling coset of X_2 w.r.t $H_1\rho_1$ and some canonical form CF_2. Then we can define X^{ρ_2} to be the canonical form $\mathrm{CF}(X, G\sigma)$ of X with respect to $G\sigma$. The canonical labeling coset $\mathrm{CL}(X, G\sigma)$ of X w.r.t this canonical form will be $H_2\rho_2$.

Lemma 4.1.5 *Let $X = (D, R_1, R_2)$ be a finite relational structure. Then the above process defines a correct canonical form.*

Proof. Let $X = (D, R_1, R_2)$ and $Y = (D, S_1, S_2)$ be two relational structures where R_1 and S_1 have arity a_1 and R_2 and S_2 have arity a_2. Let X and Y be G isomorphic via an isomorphism $\pi \in G$. We also assume the following.

$$\begin{array}{ll} \mathrm{CL}_1(X_1, G\sigma) = H_1\rho_1 & \mathrm{CL}_1(Y_1, G\sigma) = L_1\xi_1 \\ \mathrm{CL}_2(X_2, H_1\rho_1) = H_2\rho_2 & \mathrm{CL}_2(Y_2, L_1\xi_1) = L_2\xi_2 \end{array} \tag{4.1}$$

where $X_1 = (D, R_1)$, $X_2 = (D, R_2)$, $Y_1 = (D, S_1)$ and $Y_2 = (D, S_2)$. Notice that π is an isomorphism between X_1 and Y_1 and between X_2 and Y_2. We need to prove that $X^{\rho_2} = Y^{\xi_2}$. The relational structures $X_2^{\rho_1}$ and $Y_2^{\xi_1}$ are isomorphic via isomorphism $\rho_1^{-1}\pi\xi_1$. Observe that $\rho_1^{-1}H_1\rho_1 = \mathrm{Aut}(X_1^{\rho_1}) \cap \sigma^{-1}G\sigma$ and $\xi_1^{-1}L_1\xi_1 = \mathrm{Aut}(Y_1^{\xi_1}) \cap \sigma^{-1}G\sigma$. But $X_1^{\rho_1} = Y_1^{\xi_1}$. Hence, $\rho_1^{-1}H_1\rho_1 = \xi_1^{-1}L_1\xi_1$. Notice that $\rho_1^{-1}\pi\xi_1 \in \mathrm{Aut}(X_1^{\rho_1}) \cap \sigma^{-1}G\sigma$. This implies $\mathrm{CF}_2(X_2^{\rho_1}, \rho_1^{-1}H_1\rho_1) = \mathrm{CF}_2(Y_2^{\xi_1}, \xi_1^{-1}L_1\xi_1)$. Hence $R_2^{\rho_2} = S_2^{\xi_2}$. As, $\mathrm{CL}_2(X_2, H_1\rho_1) = \rho_1\mathrm{CL}_2(X_2^{\rho_1}, \rho_1^{-1}H_1\rho_1)$ and $\mathrm{CL}_2(Y_2, L_1\xi_1) = \xi_1\mathrm{CL}_2(Y_2^{\xi_1}, \xi_1^{-1}L_1\xi_1)$ we have $\rho_2 = \rho_1\gamma$ and $\xi_2 = \xi_1\delta$ where $\gamma \in \mathrm{CL}_2(X_2^{\rho_1}, \rho_1^{-1}H_1\rho_1)$ and $\delta \in \mathrm{CL}_2(Y_2^{\xi_1}, \xi_1^{-1}L_1\xi_1)$. But as $R_1^{\rho_1} = S_1^{\xi_1}$ and $\gamma, \delta \in \mathrm{Aut}(X_1^{\rho_1}) = \mathrm{Aut}(Y_1^{\xi_1})$ we will have $R_1^{\rho_2} = S_1^{\xi_2}$. This proves that $X^{\rho_2} = Y^{\xi_2}$. ■

Remark 4.1.6 1. *Theorem 4.1.4 yields an $n^{O(\log n)}$ algorithm for Tournament Canonization,* T-CANON, *and Tournament Isomorphism* TOUR-ISO *[30]. The algorithm exploits the fact that automorphism groups of tournaments are solvable and hence in Γ_d for $d = 1$.*

2. *We note that Theorem 4.1.4 is applicable to any finite relational structure $\mathcal{K} = ([n], R_1, R_2, \ldots, R_\ell)$ with relations R_i of bounded arity. Such structures can be easily encoded as binary strings of length $n^{O(1)}$, as described above for graphs. Thus, Theorem 4.1.4 can be applied to canonize such relational structures in polynomial time w.r.t. cosets $G\sigma$ where $G \in \Gamma_d$.*

4.2 Gadget construction for Tournaments

We first recall the definition of tournaments.

Definition 4.2.1 (tournament) *A directed graph $T = (V, A)$ is a* tournament *if for each pair of distinct vertices $u, v \in V$, exactly one of (u, v) or (v, u) is in A.*

Let $T = (V, A)$ be a tournament. We say the the vertex v is an *in-neighbor* (*out-neighbor*) of a vertex u if $(v, u) \in A$ (resp. $(u, v) \in A$). The *in-degree*(*out-degree*) of a vertex v is the number of in-neighbors (resp. out-neighbors) of v. A tournament $T = (V, A)$ is called *regular* if each vertex of T has the same in-degree. If $T = (V, A)$ is a regular tournament with n vertices then the in-degree and out-degree of a vertex v is $(n - 1)/2$ and n must be an odd number.

In this section, we explain some polynomial-time reductions concerning TOUR-ISO that are useful for our algorithm presented in Theorem 4.3.9. A key technique here is "fixing" nodes in a tournament. A node v in a graph X is a *fixpoint* if $v^\pi = v$ for every $\pi \in \text{Aut}(X)$. By the *fixing* of v in X we mean a construction that modifies X to another graph X' using a gadget so that v is forced to be fixed in X'. We will describe a gadget construction for fixing several nodes in a tournament so that the resulting graph is again a tournament. We use it to show that a colored version of Tournament Isomorphism is polynomial-time many-one reducible to TOUR-ISO.

Let $T_1 = (V_1, A_1)$ and $T_1 = (V_1, A_1)$ be two tournaments whose vertices are colored. The color-tournament isomorphism problem is to decide if T_1 and T_2 are isomorphic via an isomorphism ϕ that preserves the vertex color. I.e. v and $\phi(v)$ have the same color for each vertex $v \in V_1$. As a consequence, we derive some observations related to tournament isomorphism and automorphism (Theorem 4.2.3) useful for canonization. Let $u_1, u_2, \cdots, u_l$ be the nodes of a tournament $T = (V, A)$ that we want to fix. The gadget we use is shown in Figure 4.1. Call the resulting tournament T'.

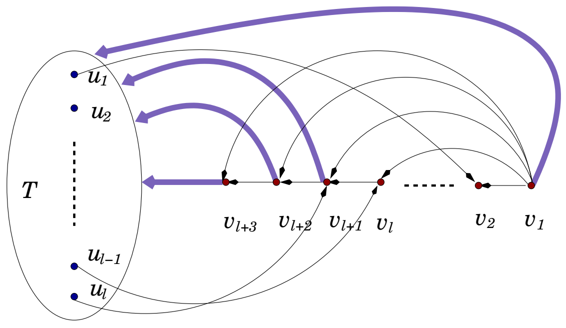

Figure 4.1: Vertex Fixing.

Here, $v_1, v_2, \cdots, v_{l+3}$ are $l + 3$ new vertices used in the gadget. Notice that v_1 is the unique vertex that beats all other vertices of T'. For $2 \leq j \leq l + 1$, v_j beats v_k for $k > j$, and beats all the vertices of T except u_{j-1}. Vertex v_{l+2} beats v_{l+3}, and both v_{l+2}

and v_{l+3} beat all vertices of T. The thick edge between v_1 and T indicates that v_1 beats all the vertices of T. All thick edges have similar meaning.

More precisely, $T' = (V', A')$ where $V' = V \cup \{v_1, \cdots, v_{l+3}\}$ and we can write the edge set as

$$A' = A \cup A_1 \cup A_2 \cup A_3,$$

$$\text{where } A_1 = \{(v_j, u_k) \mid 2 \leq j \leq l, k \neq j\} \cup \{(u_j, v_{j+1}) \mid j = 1, \cdots, l\},$$

$$A_2 = \{(v_1, u_k), (v_{l+2}, u_k), (v_{l+3}, u_k) \mid 1 \leq k \leq l\}, \text{ and}$$

$$A_3 = \{(v_j, v_k) \mid j < k\}.$$

Lemma 4.2.2 *Any automorphism of T' fixes $\{u_1, u_2, \cdots, u_l\}$.*

Proof. Notice that $v_1, v_2, v_3, \cdots, v_l$ are the unique vertices of in-degree $0, 2, 3, \cdots, l$, respectively. Hence they are fixed by any automorphism of T'. Also, v_{l+1} and v_{l+2} are the only vertices of in-degree $l + 1$. But, the directed edge (v_{l+1}, v_{l+2}) forces the fixing of these two vertices by all automorphisms. As v_{i+1} has a unique incoming edge from u_i, $1 \leq i \leq l$, each of $u_1, u_2, \cdots, u_l$ is fixed by all automorphisms of T'. ∎

Search and decision for GRAPH-ISO are known to be polynomial-time equivalent to computing a generating set for the automorphism group Aut(X) of a graph X. We show similar results for tournaments. In fact, we give a general approach to proving this equivalence for any class of graphs and apply it to tournaments.

For a class of graphs $\mathcal{G}$, let GRAPH-ISO$_{\mathcal{G}}$ denote the decision problem:

$$\text{GRAPH-ISO}_{\mathcal{G}} = \{\langle X_1, X_2 \rangle \in \mathcal{G} \times \mathcal{G} \mid X_1, X_2 \text{ are isomorphic}\}.$$

Two vertex-colored graphs[1] $X_1, X_2 \in \mathcal{G}$ are said to be isomorphic if there is a color preserving graph isomorphism between them. Let C-GRAPH-ISO$_{\mathcal{G}}$ be the corresponding decision problem. The graph automorphism problem is: GA$_{\mathcal{G}} = \{X \in \mathcal{G} \mid X$ has a nontrivial automorphism$\}$. For $X \in \mathcal{G}$, let AUT$_{\mathcal{G}}$ be the problem of computing a generating set for the automorphism group of X. The following theorem is easy to prove using standard techniques from [76].

[1] In this chapter, vertex and edge colorings are simply labels without any constraints like proper vertex/edge colorings etc.

Theorem 4.2.3 *Let $\mathcal{G}$ be any class of graphs. If* C-GRAPH-ISO$_\mathcal{G}$ *is polynomial-time many-one reducible to* GRAPH-ISO$_\mathcal{G}$ *then*

1. GA$_\mathcal{G}$ *is polynomial-time Turing reducible to* GRAPH-ISO$_\mathcal{G}$.

2. *Search version of* GRAPH-ISO$_\mathcal{G}$ *is polynomial-time Turing reducible to decision version of* GRAPH-ISO$_\mathcal{G}$.

3. AUT$_\mathcal{G}$ *is polynomial-time Turing reducible to* GRAPH-ISO$_\mathcal{G}$.

Proof. 1. Let $X \in \mathcal{G}$ be the given graph. In order to check if there is an automorphism of Aut(X) that maps vertex u to vertex v, we construct two colored graphs X_u and X_v from X as follows: X_u is a copy of X in which u is colored red and all other vertices are colored blue, and X_v is a copy of X in which v is colored red and all other vertices are colored blue. Clearly, $(X_u, X_v) \in$ C-GRAPH-ISO$_\mathcal{G}$ iff some automorphism of X maps u to v. As C-GRAPH-ISO$_\mathcal{G}$ is reducible to GRAPH-ISO$_\mathcal{G}$, we can test whether some automorphism of X maps u to v via a polynomial-time many-one (or Turing) reduction to GRAPH-ISO$_\mathcal{G}$. Note that X has a nontrivial automorphism iff for some pair of distinct vertices u and v, X_u and X_v are isomorphic. Putting it together gives us a polynomial-time Turing reduction from GA$_\mathcal{G}$ to GRAPH-ISO$_\mathcal{G}$.

2. It is easy to see that the reduction of search version to decision version can be done using the same idea as described in the proof of part 1.

3. Let X be an instance of AUT$_\mathcal{G}$. The goal is to compute a generating set for Aut(X) in polynomial time using C-GRAPH-ISO$_\mathcal{G}$ as oracle. This construction follows the standard way of building a strong generating set for Aut(X) by collecting all coset representatives for a pointwise stabilizer tower (see, e.g. [82]). We give a brief outline. Let $G_0 =$ Aut(X) and $V(X) = [n]$. We will construct a tower of subgroups $G_0 \geq G_1 \geq \cdots \geq G_{n-1} = \{1\}$ by their generating sets, where $G_i = \{g \in G_0 \mid j^g = j, 1 \leq j \leq i\}$. Suppose S is a generating set for G_i and $\{g_1, g_2, \cdots, g_l\}$ are coset representatives of G_i in G_{i-1}. Then $S \cup \{g_1, g_2, \cdots, g_\ell\}$ is a generating set for G_{i-1}. Furthermore, $\ell \leq n-i+1$ as the distinct coset representatives must map i to different points. Thus, the set of all coset representatives gives a generating set for G_0 of size $O(n^2)$. Now we describe how to compute the coset representatives of G_i in G_{i-1}. Notice that finding the coset representatives is equivalent to testing if $G_{i-1} \cap$ Aut(X) has an automorphism that maps i to k for a $k \geq i+1$. We can reduce this to C-GRAPH-ISO$_\mathcal{G}$ as follows: Take two copies X_1 and X_2 of X. Pick vertex $k \geq i+1$ in X_2. For $1 \leq j \leq i-1$, in both X_1 and X_2

color the vertex j using color j. Next, color both $i \in V(X_1)$ and $k \in V(X_2)$ using color i. The remaining vertices of both X_1 and X_2 are colored 0. Clearly, $X_1 \cong X_2$ iff k is in the orbit of i. We can test this with a query to GRAPH-ISO$_{\mathcal{G}}$, and by part 2 we can find the actual isomorphism. This gives us a distinct coset representative corresponding to k. Continuing thus, we can find all coset representatives. Clearly, this is a polynomial-time oracle procedure with queries to GRAPH-ISO$_{\mathcal{G}}$. ■

We now show C-TOUR-ISO $\leq_m^P$ TOUR-ISO, implying that tournaments satisfy the conditions of Theorem 4.2.3.

Theorem 4.2.4 *Color tournament isomorphism problem is polynomial time many-one reducible to tournament isomorphism problem.*

Proof. Let T_1, T_2 be tournaments with vertices colored using l distinct colors $\{c_i\}_{i=1}^{l}$. Let C_i denote the set of vertices colored with c_i. Our reduction transforms T_1 and T_2 into uncolored tournaments T_1'' and T_2'' such that $T_1 \cong T_2$ if and only if $T_1'' \cong T_2''$. The construction for T_1 is depicted in Figure 4.2 (to avoid clutter, we do not show all the edges).

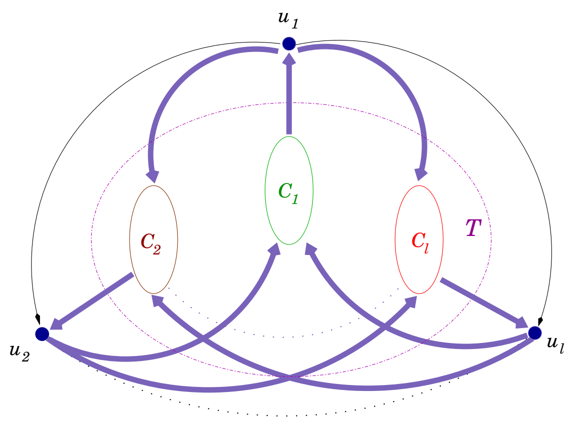

Figure 4.2: Colored to Uncolored Tournament.

We first construct a tournament T_1' from T_1 by adding l new vertices $u_1, \cdots, u_l$ and new edges. We describe the new edges in T_1'. Vertex u_i beats the vertices in each color class C_j with $j \neq i$, and u_i is beaten by all vertices in color class C_i. For $1 \leq i < j \leq l$, vertex u_i beats vertex u_j in the tournament T_1'. Likewise, the tournament T_2' is obtained from T_2 by introducing new vertices $v_1, v_2, \cdots, v_l$. The vertex v_i beats the vertices in each color class C_j with $j \neq i$, and u_i is beaten by all vertices in color class C_i. For $1 \leq i < j \leq l$, vertex v_i beats vertex v_j.

Now, using the gadget of Lemma 4.2.2, we fix $u_1, u_2, \cdots, u_l$ in T_1'. Call the resulting tournament T_1''. Similarly, T_2'' is obtained from T_2' by fixing vertices $v_1, v_2, \cdots, v_l$ using the gadget of Lemma 4.2.2.

By the construction in Lemma 4.2.2, any isomorphism from T_1'' to T_2'' is forced to map u_i to v_i for each i. Hence, this isomorphism will induce a color-preserving isomorphism from T_1 to T_2. Conversely, any color-preserving isomorphism from T_1 to T_2 can be extended to an isomorphism from T_1'' to T_2'' that maps u_i to v_i for each i. It follows that the colored tournaments T_1 and T_2 are isomorphic if and only if $T_1'' \cong T_2''$. ■

4.3 Canonical Labeling of Tournaments

We first recall an important fact about tournaments and the tournament canonization of Babai and Luks. This fact is stated in [30] for instance, but it seems folklore. We also recall the easy proof.

Lemma 4.3.1 *The automorphism group of a tournament has an odd number of elements.*

Proof. Let $T = (V, A)$ be a tournament. If $|\text{Aut}(T)|$ is even then by Cauchy's theorem $\text{Aut}(T)$ has a permutation π of order 2. Let $\pi = C_1 \cdots C_l$ be the decomposition of π as a product of disjoint 2-cycles. Suppose $C_1 = (u, v)$. Then the permutation π maps (u, v) to (v, u). But either $(u, v) \in A$ or $(v, u) \in A$. Hence π cannot be a an automorphism and $|\text{Aut}(T)|$ must be odd. ■

A celebrated group theory result is the Feit-Thompson odd-order theorem whose statement we recall below.

Theorem 4.3.2 ([49]) *Every odd order group is solvable.*

The following corollary is an easy consequence of Lemma 4.3.1 and Theorem 4.3.2.

Corollary 4.3.3 *The automorphism group of a tournament is solvable.*

As mentioned in Remark 4.1.6, an $n^{O(\log n)}$ algorithm for tournament canonization is described based on their string canonization result stated in Theorem 4.1.4.

Theorem 4.3.4 [30, Theorem 4.1] *There is an $n^{O(\log n)}$ algorithm for* T-CANON*, the Tournament Canonization problem.*

We sketch the proof of Theorem 4.3.4 as we need the main ideas:

Let $T = (V, A)$ be a tournament with $|V| = n$. The tournament T is *regular* if every vertex has the same outdegree. It is easy to observe that T is regular implies n is odd and every vertex has both indegree and outdegree equal to $\frac{n-1}{2}$.

If the input tournament T is not *regular*, the algorithm will partition V as $V = \cup_{i=1}^{k} V_i$, where $V_i \subset V$ is the subset of vertices with out-degree i, where V_i is empty if there are no vertices of outdegree i and k is the maximum outdegree in T. Let T_i be the tournament induced by V_i. Using Theorem 4.1.4 the algorithm recursively computes $\mathrm{CL}(T_i, \mathrm{Sym(V_i)}) = \mathrm{H_i\rho_i}$, for all i, where $H_i = \mathrm{Aut}(T_i)$. Then, we set

$$\mathrm{CL}(T, \mathrm{Sym(V)}) = \mathrm{CL(T, H_1\rho_1 \times H_2\rho_2 \times \cdots \times H_k\rho_k)} = \mathrm{CL(T, H\rho)},$$

where $H \leq \mathrm{Sym(V)}$ is the product group $H_1 \times H_2 \times \cdots \times H_k$ and $\rho \in \mathrm{Sym(V)}$ is the k-tuple $(\rho_1, \rho_2, \cdots, \rho_k)$, where $\rho_i \in \mathrm{Sym(V_i)}$.

Importantly, as each H_i is solvable, H is also a solvable group. Thus, $\mathrm{CL}(T, H\rho)$ can be computed in polynomial time by Theorem 4.1.4.

If $t(n)$ is the running time bound, then for this stage of computation it satisfies the recurrence relation: $t(n) = \sum_{i=1}^{k} t(n_i) + n^{O(1)}$, where $n_i = |V_i|$.

The more difficult case is when T is a *regular* tournament. For each $v \in V$, the algorithm will canonize the tournament with v as the first vertex. Among the canonical forms thus obtained, the algorithm will pick the lexicographically least. The algorithm proceeds as follows for a $v \in V$: Put $V' = V \setminus \{v\}$ and let T' be the tournament induced by V'. We have the partition $V' = V_1' \cup V_2'$, where V_1' is the set of $(n-1)/2$ vertices that beat v and V_2' is the set of $(n-1)/2$ vertices beaten by v. Let the tournaments induced by V_1' and V_2' be T_1' and T_2', respectively. Next, the algorithm recursively computes $\mathrm{CL}(T_i', \mathrm{Sym(V_i')}) = \mathrm{H_i\rho_i}$ for $i = 1, 2$. Again using Theorem 4.1.4, the algorithm will compute $\mathrm{CL}(T, \mathrm{Sym(V')}) = \mathrm{CL(T, H_1\rho_1 \times H_2\rho_2)}$ to get the canonical labeling coset with v as the first vertex. As mentioned, the algorithm repeats this process for all the vertices $v \in V$. From among these n cosets, we compute $\mathrm{CL}(T, \mathrm{Sym(V)})$ as the union of those cosets that give rise to the lex-least canonical labeling. It can be shown that this union must paste into a coset of the form Aut$(T)\sigma$ [30]. For this case, the recurrence relation for the time bound $t(n)$ satisfies the recurrence relation

$$t(n) = n(2t(\frac{n-1}{2}) + n^{O(1)}).$$

Solving the two recurrence relations for $t(n)$ yields the running time bound $n^{O(\log n)}$ [30].

Wreath Product

Here we recall the notion of wreath product of two groups that will naturally appear in the proof of our result.

Definition 4.3.5 *Let $X \leq \mathrm{Sym(V)}$ and $Y \leq \mathrm{Sym}([\ell])$ be two groups. The* wreath product *of X and Y, denoted $X \wr Y$, is a group with elements $\{(x_1, \cdots, x_\ell, y) \mid x_1, \cdots, x_\ell \in X$ and $y \in Y\}$ and the binary operation*

$$(x_1, \cdots, x_\ell, y)(x'_1, \cdots, x'_\ell, y') = (x_1 x'_{1^y}, \cdots, x_\ell x'_{\ell^y}, yy').$$

As a set $X \wr Y$ is same as the direct product $X^\ell \times Y$, but it differs in the group operation in the sense that the element y from Y permutes the indices of X^ℓ. We have to be careful about the indices. Notice that $(x_1, \cdots, x_\ell, y)(x'_{1^\sigma}, \cdots, x'_{\ell^\sigma}, y') = (x_1 x'_{1^{y\sigma}}, \cdots, x_\ell x'_{\ell^{y\sigma}}, yy')$ and *not* $(x_1 x'_{1^{\sigma y}}, \cdots, x_\ell x'_{\ell^{\sigma y}}, yy')$. The inverse of an element $(x_1, \cdots, x_\ell, y) \in X \wr Y$ is $(x_{1^\sigma}^{-1}, \cdots, x_{\ell^\sigma}^{-1}, y^{-1})$ where $\sigma = y^{-1}$. The wreath product $X \wr Y$ defines a natural action on the set $V \times [\ell]$ as follows: Let $(a, i) \in V \times [\ell]$ and $(x_1, \cdots, x_\ell, y) \in X \wr Y$ then

$$(a, i)^{(x_1, \cdots, x_\ell, y)} = (a^{x_i}, i^y).$$

It is easy to check that the above definition is indeed a group action. Moreover, this action embeds $X \wr Y$ in $\mathrm{Sym(V} \times [\ell])$. Thus if $G = (V \times [\ell], E)$ is a (di)graph then for all elements $\pi \in X \wr Y$ we can talk about the graph G^π in the usual way.

Fact 4.3.6 *Let H be a group and N be a normal subgroup of H. Then H is solvable if and only if both H/N and N are solvable.*

Let $X \leq \mathrm{Sym(V)}$ and $Y \leq \mathrm{Sym}([\ell])$ be solvable groups. Notice that $X^{\langle \ell \rangle} := X^\ell \times \{id\}$ is a normal subgroup of $X \wr Y$, where id is the identity element of Y. Notice also that $X \wr Y / X^{\langle \ell \rangle} \cong H$. Thus by Fact 4.3.6 we get the following lemma.

Lemma 4.3.7 *If $X \leq \mathrm{Sym(V)}$ and $Y \leq \mathrm{Sym}([\ell])$ are solvable groups the $X \wr Y$ is also solvable.*

The result

We are now ready to describe our result in this section. As mentioned in the introduction, we are motivated by the problem whether Graph Canonization is polynomial-time Turing reducible to Graph Isomorphism.

Since the general problem seems difficult to approach, we raise the question for the more restricted case of tournaments: can tournament canonization T-CANON be polynomial-time Turing reduced to Tournament Isomorphism TOUR-ISO? Even for this restricted problem we do not know the answer. However, we make some progress on the problem by giving a polynomial-time oracle algorithm for T-CANON that accesses oracle TOUR-ISO with an additional oracle for canonizing *rigid* tournaments. Thus, canonizing rigid tournaments seems to be the bottleneck in reducing T-CANON to TOUR-ISO. Let RT-CANON denote the functional oracle for computing the canonical form of a rigid tournament. Since rigid tournaments have trivial automorphism groups, notice that the canonical form trivially gives the canonical labeling coset as well.

We believe this weaker result is interesting and throws some light on the original problem. One interpretation of our result is that rigid tournaments are the hardest instances of canonization. We do not know if a similar result holds for general graphs. We make crucial use of Theorem 4.1.4 and the fact that the automorphism groups of tournaments is solvable.

We start with a definition.

Definition 4.3.8 *A tournament is called* vertex transitive *if for all pair of vertices u, v there is an automorphism π of T such that $u^\pi = v$.*

Naturally a vertex transitive tournament is regular.

Theorem 4.3.9 *There is a polynomial-time oracle algorithm for* T-CANON *that accesses oracles for* TOUR-ISO *and* RT-CANON.

Proof. Let $T = (V, A)$ be the input tournament to be canonized. Denote by T-CANON(T) the function computing the canonical labeling coset CL$(T, \mathrm{Sym(V)})$ of T, where V is the vertex set of T. It has the following recursive description:

T-CANON(T):

1. **Orbit computing:** With oracle queries to TOUR-ISO and using the vertex fixing technique of Theorem 4.2.3 we can compute a polynomial size generating set for Aut(T). Then, we can compute in polynomial time the partition of V into Aut(T)-orbits in polynomial time using standard permutation group techniques [82, 76].

2. **If orbits are singletons:** This happens precisely when T is a rigid tournament. In this case we query the RT-CANON oracle to obtain a canonical form for T. Notice that in this case the canonical labeling coset CL$(T, \mathrm{Sym(V)})$ is a singleton since Aut(T) has only one element.

3. **Single orbit:** If V has only one orbit w.r.t. Aut(T) then the tournament is *vertex-transitive*. As T is vertex-transitive it follows that T is regular.

   In this case, we can take *any one vertex* v of T and make it the first vertex of the canonical form. Let T_v denote the tournament induced by $V' = V \setminus \{v\}$. We recursively find the canonical labeling coset of T_v with respect to $\mathrm{Sym(V)}'$, where $V' = V \setminus \{v\}$. A crucial point is that it suffices to compute this for any one vertex v. The reason, as we will prove in Claim 4.3.10, is that doing this for any other vertex u will give rise to the same canonical form since there is an automorphism that maps u to v. (We observe that this is not true in the case of regular tournaments that are not vertex transitive. Indeed, we recall from our proof sketch of Theorem 4.3.4 that n recursive calls are made in the regular case. This step makes a crucial difference to the running time we obtain.)

   The vertex v defines the partition $V' = V_1 \cup V_2$, where V_1 is the set of all vertices that beat v in the tournament T and V_2 is the set of all vertices beaten by v in T. As T is regular, $|V_1| = |V_2| = (n-1)/2$. Suppose the tournaments induced by V_1 and V_2 are T_1 and T_2 respectively. Recursively compute $H_1\rho_1 :=$ T-CANON(T_1) and $H_2\rho_2 :=$ T-CANON(T_2).

   Now, applying Theorem 4.1.4, we compute T-CANON(T_v) = CL$(T_v, H_1\rho_1 \times H_2\rho_2)$. The algorithm of Theorem 4.1.4 runs in polynomial time as both H_1 and

H_2 are solvable groups, being automorphism groups of tournaments. This gives the canonical ordering for T_v. Placing v as the overall first vertex gives the canonical ordering for T. Let T' denote the resulting tournament (which is the canonical form for T). Finally, the canonical labeling coset is easy to compute from T and T' with queries to TOUR-ISO by applying Theorem 4.2.3.

4. **Nonrigid with more than one orbit:** This is the general case when there are more than one orbit and T is not rigid. Let $O_1, O_2, \cdots, O_\ell$ be the orbits of T (computed using queries to TOUR-ISO as explained in the first step).

   [**Case (a)**] Let T_i be the tournament induced by O_i, for $1 \leq i \leq \ell$. We first consider the case that yields an easy recursive step. Suppose not all T_i are isomorphic to each other (which we can easily find with queries to TOUR-ISO). Then we partition the set of orbits O_j into k collections $S_1, S_2, \cdots, S_k$, where for each S_i

   $$O_j, O_m \in S_i \text{ iff } T_j \text{ isomorphic to } T_m.$$

   We assume that $\text{CF}(T') < \text{CF}(T'')$ for all $T' \in S_i$ and $T'' \in S_j$ for all $i < j$ where we compare two graphs lexicographically. Note that $\text{CF}(T')$ and $\text{CF}(T'')$ can be computed recursively.

   Now, for $1 \leq i \leq k$, let $\hat{T}_i$ denote the tournament induced by the union of all the orbits in S_i.

   For each i, the algorithm recursively computes the canonical labeling coset $H_i\rho_i :=$ T-CANON$(\hat{T}_i)$.

   Then, by applying Theorem 4.1.4 the algorithm computes the overall canonical labeling coset as $\text{CL}(T, H\rho)$ — where $H = H_1 \times H_2 \times \cdots \times H_k$ and $\rho = (\rho_1, \cdots, \rho_k)$. This can be computed in polynomial time by Theorem 4.1.4 because each H_i is solvable, being the automorphism group of a tournament.

   [**Case (b)**] We are now in the case when the tournaments T_i induced by the orbit O_i of T are all isomorphic, for $1 \leq i \leq \ell$. This is the more interesting case:

   Since T_i are induced by orbits they are all regular tournaments. Hence $|O_i|$ is odd for each i. Furthermore, all O_i are of same size since T_i are all isomorphic. Thus, $|O_i| = t$ for each i, where t is an odd positive integer. Rename the vertices in O_i arbitrarily (say in lex order) by $\{(a, i) \mid a \in [t]\} = [t] \times \{i\}$ so that the tournament

T is on vertex set $[t] \times [\ell]$ and T_i is on vertex set $[t] \times \{i\}$. For $1 \leq i \leq \ell$, let T'_i denote the tournament obtained from T_i by renaming each vertex $(a, i) \in [t] \times \{i\}$ by a so that $V(T'_i) = [t]$.

From the ℓ orbits of T, we will construct a new tournament $\mathcal{T}$ with vertex set $[\ell]$. For $1 \leq i \leq \ell$, the vertex i of $\mathcal{T}$ represent orbit O_i. We still have to define the edges of $\mathcal{T}$. To that end, let X_{ij} denote the directed bipartite graph between orbits O_i and O_j (whose edges are the original tournament edges). As O_i and O_j are orbits of Aut(T) and $|O_i| = |O_j|$, the directed bipartite graph X_{ij} has the following property: there is a positive integer α_{ij} such that, in the graph X_{ij}, the indegree of each vertex in O_i is α_{ij} and the outdegree of each vertex in O_j is α_{ij}. Since, for each i, $|O_i| = t$ and t is odd, $t - \alpha_{ij} \neq \alpha_{ij}$. The edges of $\mathcal{T}$ are now defined as follows: for $1 \leq i \neq j \leq \ell$, (i, j) is an edge in $\mathcal{T}$ if $t - \alpha_{ij} > \alpha_{ij}$, otherwise (j, i) is an edge in $\mathcal{T}$. We call $\mathcal{T}$ the *shrunk* tournament.

The idea here is to obtain the canonical form of T by relabeling the orbits and inside each orbit by relabeling the vertices. In other words we will pick the canonical form from the set $\{T^\theta \mid \theta \in \mathrm{Sym}([\mathrm{t}]) \wr \mathrm{Sym}([\ell])\}$. (Notice that since T is a tournament on $[t] \times [\ell]$, the action of $\mathrm{Sym}([\mathrm{t}]) \wr \mathrm{Sym}([\ell])$ is well defined on T).

We first recursively compute the canonical labeling cosets $H_i\sigma_i$ for each tournament T'_i, $1 \leq i \leq \ell$. Note that $H_i \leq \mathrm{Sym}([\mathrm{t}])$ and $\sigma_i \in \mathrm{Sym}([\mathrm{t}])$. As all the tournaments T'_i are isomorphic they will have the same canonical form $\hat{T}$. We can easily verify that $\sigma_i^{-1}H_i\sigma_i =$ Aut$(\hat{T})$. Next we recursively compute the canonical labeling coset $H\rho :=$ T-CANON$(\mathcal{T})$ of the shrunk tournament. Notice that $H\rho$ is a coset of $\mathrm{Sym}([\ell])$. If $\hat{\mathcal{T}}$ is the canonical form of $\mathcal{T}$ then we have $\rho^{-1}H\rho =$ Aut$(\hat{\mathcal{T}})$. Finally, we compute the canonical form of T^ξ with respect to Aut$(\hat{T}) \wr$ Aut$(\hat{\mathcal{T}})$ using Babai-Luks algorithm (Theorem 4.1.4), where $\xi = (\sigma_1, \cdots, \sigma_\ell, \rho)$. By Lemma 4.3.7 Aut$(\hat{T}) \wr$ Aut$(\hat{\mathcal{T}})$ is solvable and hence Babai-Luks algorithm will run in polynomial time. Let the output of Babai-Luks algorithm be the coset $G\pi$. Then the recursive step returns $\xi^{-1}G\pi$. (More precisely, it returns the canonical labeling coset by returning $\xi^{-1}G\xi$ as the group and $\xi^{-1}\pi$ as the coset representative.)

This completes the description of the tournament canonizing algorithm. It is easy to see from the above description that the running time is polynomially bounded. We prove the correctness of the algorithm through a series of claims about the different steps of the algorithm.

We establish the correctness of Step 3 in the next claim.

Claim 4.3.10 *Let u, v be two vertices in a vertex transitive tournament $T = (V, A)$. Then the canonical form computed in Step 3 by fixing vertex u is same as the canonical form computed by fixing vertex v.*

Proof of Claim Let π be an automorphism of the vertex transitive tournament T such that $\pi(u) = v$. Let $V_u = V \setminus \{u\}$ and $V_v = V \setminus \{v\}$. Let T_u and T_v be the tournaments induced by V_u and V_v respectively. Clearly, π is an isomorphism between T_u and T_v. Let V_1' be the set of vertices beaten by u and V_2' be the set of vertices that beat u in the tournament T_u. As defined in the algorithm, V_1 and V_2 are the vertex sets that are beaten by v and beat v respectively in T_v.

Let T_1', and T_2' be the sub-tournaments of T_u induced by V_1' and V_2' respectively. Similarly, T_1 and T_2 are the sub-tournaments of T_v induced by V_1, and V_2 respectively. Notice that π is an isomorphism between T_1' and T_1 and also between T_2' and T_2.

Suppose we run the algorithm by fixing u instead of v in step 3. It will recursively compute CL$(T_1', \mathrm{Sym}(\mathrm{V}_1')) = \mathrm{H}_1'\rho_1'$ and CL$(T_2', \mathrm{Sym}(\mathrm{V}_2')) = \mathrm{H}_2'\rho_2'$, where $H_1' =$ Aut(T_1') and $H_2' =$ Aut(T_2'). Next it will compute CL$(T_u, H_1'\rho_1' \times H_2'\rho_2')$. Similarly, if the algorithm fixes v then it computes CL$(T_v, H_1\rho_1 \times H_2\rho_2)$ where $H_1 =$ Aut(T_1) and $H_2 =$ Aut(T_2).

Since T_1' and T_1 are isomorphic (π is an isomorphism), inductively it follows that the algorithm computes the same canon for them which is $\rho_1(T_1) = \rho_1'(T_1') = \overline{T}_1$. Furthermore, it is easy to see that the automorphism group of $\overline{T}_1$ is Aut$(\overline{T}_1) = \rho_1^{-1}H_1\rho_1 = \rho_1'^{-1}H_1'\rho_1'$.

Similarly, T_2' and T_2 are isomorphic (again via π). Thus, inductively it follows that the algorithm computes the same canon for them which is $\rho_2(T_2) = \rho_2'(T_2') = \overline{T}_2$, and the automorphism group of $\overline{T}_2$ is Aut$(\overline{T}_2) = \rho_2^{-1}H_2\rho_2 = \rho_2'^{-1}H_2'\rho_2'$.

Let $H = H_1 \times H_2$ and $\rho = (\rho_1, \rho_2)$. Similarly, let $H' = H_1' \times H_2'$ and $\rho' = (\rho_1', \rho_2')$. It follows from the above that $\rho^{-1}H\rho = \rho'^{-1}H'\rho' =$ Aut$(\overline{T}_1) \times$ Aut$(\overline{T}_2)$.

We need to prove that we obtain the same tournament as canonical form for both T_u and T_v. That is, we need to show that

$$\text{CF}(T_u, H'\rho') = \text{CF}(T_v, H\rho).$$

By definition of canonical form with respect to cosets we have

$$\begin{aligned}\text{CF}(T_u, H'\rho') &= \text{CF}(T_u^{\rho'}, \rho'^{-1}H'\rho') \text{ and}\\ \text{CF}(T_v, H\rho) &= \text{CF}(T_v^{\rho}, \rho^{-1}H\rho).\end{aligned}$$

As explained $\rho^{-1}H\rho = \rho'^{-1}H'\rho' = \text{Aut}(\overline{T}_1) \times \text{Aut}(\overline{T}_2)$. Thus if we show that some permutation τ in $\text{Aut}(\overline{T}_1) \times \text{Aut}(\overline{T}_2)$ maps $T_u^{\rho'}$ to T_v^{ρ} then the above equality will follow from Theorem 4.1.4.

But, $T_u^{\rho'}$ and T_v^{ρ} are isomorphic via isomorphism $\rho_1'^{-1}\pi\rho$. Furthermore, it is easy to check that $\rho_1'^{-1}\pi\rho \in \text{Aut}(\overline{T}_1) \times \text{Aut}(\overline{T}_2)$. This completes the proof of this claim. ■

Claim 4.3.11 *The algorithm correctly computes the canonical form of T in Step 4 Case(a).*

Proof of Claim Let T' be an isomorphic copy of T. Let π be an isomorphism form T to T'. Let $O_1, \cdots, O_\ell$ and $O_1', \cdots, O_\ell'$ be the orbits of T and T' respectively. Let $\hat{T}_i$ be the tournament defined in Step 4 Case(a), $1 \leq i \leq k$. Similarly we define $\hat{T}_i'$ for T', $1 \leq i \leq k$. Clearly, for each i the tournaments $\hat{T}_i$ and $\hat{T}_i'$ are also isomorphic via isomorphism π. Let T-CANON$(\hat{T}_i) = \text{Aut}(\hat{T}_i)\rho_i$ and T-CANON$(\hat{T}_i') = \text{Aut}(\hat{T}_i')\rho_i'$. Let $\text{CF}(\hat{T}_i) = \text{CF}(\hat{T}_i') = X_i$.

Let H denote the group $\text{Aut}(\hat{T}_1) \times \text{Aut}(\hat{T}_2) \times \cdots \times \text{Aut}(\hat{T}_k)$ and $\rho = (\rho_1, \cdots, \rho_k)$. Similarly, let H' denote the group $\text{Aut}(\hat{T}_1') \times \text{Aut}(\hat{T}_2') \times \cdots \times \text{Aut}(\hat{T}_k')$ and $\rho' = (\rho_1', \cdots, \rho_k')$.

We need to show that $\text{CF}(T, H\rho)$ and $\text{CF}(T', H'\rho')$ are identical. By definition of canonical form with respect to a coset we have

$$\begin{aligned}\text{CF}(T, H\rho) &= \text{CF}(T^{\rho}, \rho^{-1}H\rho) \text{ and}\\ \text{CF}(T', H'\rho') &= \text{CF}(T'^{\rho'}, \rho'^{-1}H'\rho').\end{aligned}$$

Now, it is easy to see that the groups $\rho^{-1}H\rho$ and $\rho'^{-1}H'\rho'$ are identical: in fact, it is the product $\text{Aut}(X_1) \times \cdots \times \text{Aut}(X_k)$ of the automorphism groups of the tournament X_i, $1 \leq i \leq k$.

Thus, it will follow from Theorem 4.1.4 that $\text{CF}(T, H\rho) = \text{CF}(T', H'\rho')$ if we show that there is a permutation $\sigma \in \text{Aut}(X_1) \times \cdots \times \text{Aut}(X_k)$ such that σ is an isomorphism from the tournament T^{ρ} to $T'^{\rho'}$. Indeed, it is easy to see that $\rho^{-1}\pi\rho'$ is an isomorphism from T^{ρ} to $T'^{\rho'}$, and the permutation $\rho^{-1}\pi\rho'$ is in $\text{Aut}(X_1) \times \cdots \times \text{Aut}(X_k)$. ■

Claim 4.3.12 *The algorithm correctly computes the canonical form of T in Step 4 Case(b).*

Proof of Claim Let T_1 and T_2 be two tournaments isomorphic via isomorphism π such that Step 4 Case(b) is applied to both of them for canonization. Inductively assume that the algorithm correctly works for smaller tournaments. We will show that the algorithm produces the same canonical form for both T_1 and T_2.

Suppose T_1 has the orbits $O_{11}, \cdots, O_{1\ell}$ under the action of Aut(T_1). Similarly, suppose T_2 has orbits $O_{21}, \cdots, O_{2\ell}$. Let $|O_{1i}| = |O_{2j}| = t$ for all i and j. We assume that the the tournaments T_1 and T_2 have same vertex set $[t] \times [\ell]$ and for $s = 1, 2$, $O_{si} = [t] \times \{i\}$. Let $T_{11}, \cdots, T_{1\ell}$ and $T_{21}, \cdots, T_{2\ell}$ be the tournaments induced by the orbits $O_{11}, \cdots, O_{1\ell}$ and $O_{21}, \cdots, O_{2\ell}$ respectively. For $s = 1, 2$ and $i = 1, \cdots, \ell$ let T'_{si} be the tournament obtained from T_{si} by renaming each vertex (a, i) by a. Since we are in Step 4 Case(b), all the tournaments $T'_{11}, \cdots, T'_{1\ell}, T'_{21}, \cdots, T'_{2\ell}$ will be isomorphic to each other. The isomorphism π will map orbits to orbits. Let $\hat{\pi} \in \mathrm{Sym}([\ell])$ be such that $\pi(O_{1i}) = O_{2i^{\hat{\pi}}}$. Let $\pi'_i = \pi \mid_{O_{1i}}$. Hence T_{1i} will be isomorphic to $T_{2i^{\hat{\pi}}}$ via π'_i. Let $\pi_i : [t] \longrightarrow [t]$ be the isomorphism from T'_{1i} to $T'_{2i^{\hat{\pi}}}$ which corresponds to π'_i (i.e., if $\pi'_i(a, i) = (b, i^{\hat{\pi}})$ then $\pi_i(a) = b$). Let $H_{si}\sigma_{si}$ be the canonical labelling coset of T'_{si} for $s = 1, 2$ and $i = 1, \cdots, \ell$. Since these tournaments are isomorphic they will have same canonical form $\hat{T}$. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be the shrunk tournaments obtained from T_1 and T_2 respectively. Clearly $\mathcal{T}_1$ and $\mathcal{T}_2$ are isomorphic via isomorphism $\hat{\pi}$. Let $\hat{\mathcal{T}}$ be the canonical form of $\mathcal{T}_1$ and $\mathcal{T}_2$. Also assume that $H_1\rho_1$ and $H_1\rho_1$ are the canonical labeling cosets of $\mathcal{T}_1$ and $\mathcal{T}_2$ returned by the recursive step of the algorithm. Notice that $\rho_1^{-1}H_1\rho_1 = \rho_2^{-1}H_2\rho_2 =$ Aut$(\hat{\mathcal{T}})$. While canonizing T_1 the algorithm canonizes $T_1^{\xi_1}$ with respect to Aut$(\hat{T}) \wr$ Aut$(\hat{\mathcal{T}})$ and while canonizing T_2 it canonizes $T_2^{\xi_2}$ with respect to the same group Aut$(\hat{T}) \wr$ Aut$(\hat{\mathcal{T}})$ where $\xi_1 = (\sigma_{11}, \cdots, \sigma_{1\ell}, \rho_1)$ and $\xi_2 = (\sigma_{21}, \cdots, \sigma_{2\ell}, \rho_2)$. Hence it is enough to prove that $T_1^{\xi_1}$ and $T_2^{\xi_2}$ are isomorphic via Aut$(\hat{T}) \wr$ Aut$(\hat{\mathcal{T}})$ isomorphism. Observe that $T_1^{\xi_1}$ and $T_2^{\xi_2}$ are isomorphic via $\gamma := \xi_1^{-1}(\pi_1, \cdots, \pi_\ell, \hat{\pi})\xi_2$. So, it is enough to prove that $\gamma \in$ Aut$(\hat{T}) \wr$ Aut$(\hat{\mathcal{T}})$. But

$$\gamma = (\sigma^{-1}_{11^{\rho_1^{-1}}}\pi_{1^{\rho_1^{-1}}}\sigma_{21^{\rho_1^{-1}\hat{\pi}}}, \cdots, \sigma^{-1}_{1\ell^{\rho_1^{-1}}}\pi_{\ell^{\rho_1^{-1}}}\sigma_{2\ell^{\rho_1^{-1}\hat{\pi}}}, \rho_1^{-1}\hat{\pi}\rho_2).$$

It is easy to see that $\rho_1^{-1}\hat{\pi}\rho_2 \in$ Aut$(\hat{\mathcal{T}})$. Let $i^{\rho_1^{-1}} = j$, then $\mu := \sigma^{-1}_{1\ell^{\rho_1^{-1}}}\pi_{\ell^{\rho_1^{-1}}}\sigma_{2\ell^{\rho_1^{-1}\hat{\pi}}} = \sigma_{1j}^{-1}\pi_j\sigma_{2j^{\hat{\pi}}}$. The tournaments T'_{1j} and $T'_{2j^{\hat{\pi}}}$ are isomorphic via isomorphism π_j (by definition of π_j's and $\hat{\pi}$). The tournaments T'_{1j} and $T'_{2j^{\hat{\pi}}}$ are mapped to their canonical form $\hat{T}$ by the isomorphisms σ_{1j} and $\sigma_{2j^{\hat{\pi}}}$ respectively. This gives $\mu \in \mathrm{Aut}(\hat{T})$. Thus, $\gamma \in \mathrm{Aut}(\hat{T}) \wr \mathrm{Aut}(\hat{\mathcal{T}})$. ■

The overall correctness of the algorithm follows from the proofs of the above claims.

We now analyze the running time. Let $T(n)$ bound the running time. In Step 1, we compute the orbits in polynomial time with queries to the TOUR-ISO oracle. If the tournament is rigid then we canonize it with a single query to RT-CANON. The remaining steps involve recursive calls. The recurrence relation for $T(n)$ in Step 3 is $T(n) = 2T((n-1)/2) + n^{O(1)}$, and in Step 4 Case(b) it is given by $T(n) = \ell T(n/\ell) + T(n/t) + n^{O(1)}$ for $\ell > 1$ and $t > 1$ because we need to compute the canonical labeling coset for ℓ tournaments induced by n/ℓ-sized orbits and the shrunk tournament of size $\ell = n/t$. For Step 4 Case (a), the recurrence is $T(n) = \sum_{i=1}^{k} T(n_i) + n^{O(1)}$. It follows by induction that $T(n) = n^{O(1)}$. In each step the application of Theorem 4.1.4 takes polynomial time because the permutation group used is always solvable. ■

As mentioned in the introduction, it seems unlikely that Theorem 4.3.9 can be shown for general graphs using the same methods. This is because our reduction crucially uses the fact that the automorphism group of tournaments are solvable, enabling us to repeatedly use the algorithm of Theorem 4.1.4 with a polynomial time bound. In case of general graphs, it is unlikely that in the intermediate stages of recursion we will have groups in Γ_d to effectively apply Theorem 4.1.4.

4.4 Hypertournament Isomorphism and Canonization

In this section we study isomorphism and canonization of hypertournaments using the method of Babai and Luks.

Hypertournaments are a generalization of tournaments and have been studied by graph theorists (see e.g. [63]). We recall the definition.

Definition 4.4.1 (k-Hypertournament) *Given positive integers n and k, a k-hypertournament T on n vertices is a pair (V, A) where V is a set of n* vertices *and A is a set of k-tuples of vertices called* arcs *so that for each subset $S \in \binom{V}{k}$, A contains* exactly one *of the (k! many) k-tuples whose entries belong to S.*

In general, *Hypergraph Isomorphism* (HGI) is easily seen to be polynomial-time many-one equivalent to Graph Isomorphism: given a hypergraph X with n vertices and

m hyperedges we can represent it uniquely as a bipartite graph Y with n vertices on one side and m vertices on the other (see Section 5.1 for more details). Thus, known complexity-theoretic upper bounds for GRAPH-ISO like NP $\cap$ coAM [31] apply to HGI.

However, consider an instance of HGI: (X_1, X_2), with n vertices and m hyperedges each. The reduction to GRAPH-ISO maps it to a pair of graphs (Y_1, Y_2) with vertex sets of size $m + n$. The best known isomorphism testing algorithm due to Luks and Zemlyachenko (see [30]) which has running time $c^{\sqrt{|V| \lg |V|}}$ ($|V|$ is the size of the vertex set V) will take time $c^{\sqrt{(m+n) \lg(m+n)}}$ when combined with the above reduction and applied to HGI. In [83] a different, dynamic-programming based algorithm with running time $2^{O(n)}$ is developed.

We study the analogous question for hypertournaments in this section. The main motivation is to see if k-hypertournaments have enough structure like tournaments so that the Babai-Luks method and Theorem 4.1.4 in particular can be applied to obtain an efficient algorithm. We note here that it is not known if HYPER-TOUR-ISO is polynomial-time reducible to TOUR-ISO. We consider k-Hypertournament Isomorphism (HYPER-TOUR-ISO$_k$) and give an $n^{O(k^2+\log n)}$ algorithm for the problem for k-hypertournaments, for each k. In fact, we actually give an $n^{O(k^2+\log n)}$ algorithm for the corresponding canonization problem. We first establish some observations about automorphisms of k-hypertournaments. The next lemma generalizes the fact that usual tournaments have automorphism groups of odd order.

Lemma 4.4.2 *For $k \geq 2$, the automorphism group* Aut(T) *of a k-hypertournament T has the following property: for any prime factor p of k it holds that p does not divide the size of* Aut(T).

Proof. Let $T = (V, A)$. For $k = 2$, T is a usual tournament and in this case it is a well-known fact that Aut(T) has odd cardinality.

Suppose $k > 2$ and p is any prime factor of k. Suppose p divides Aut(T). Let $\pi \in$ Aut(T) be an order p element (which must exist by Cauchy's theorem for finite groups). Since $\pi \in \mathrm{Sym(V)}$, we can write it as a product of disjoint p-cycles, $\pi = C_1 C_2 \cdots C_\ell$, where the remaining $n - p\ell$ elements of V are fixed by π. Let $k/p = t$. If $k \leq p\ell$ then let $S = \cup_{i=1}^{t} C_i$. Notice that π maps S to S. Now, suppose $e \in A$ is the unique hyperedge of the tournament on the k element set S. Then $e^\pi \neq e$, since π *reorders* the sequence defining hyperedge e. Thus, e^π is not a hyperedge of T, contradicting $\pi \in$ Aut(T).

In the other case, if $k > p\ell$, choose S' as any subset of size $k - p\ell$ of the $n - p\ell$ points fixed by π, and let $S = S' \cup C_1 \cup \cdots \cup C_\ell$. Again, let $e \in A$ be the hyperedge defined by this subset S. Again, notice that e^π is not a hyperedge of T, since π will reorder the sequence defining e. This contradicts the assumption that there is an order p element $\pi \in \text{Aut}(T)$. ∎

Recall that a *section* of a group G is a quotient group of some subgroup of G. A section that is a simple group is a *simple section*. Simple sections of G are precisely the composition factors of subgroups of G.

An easy corollary of the above lemma is the following. Recall that the alternating group A_k is the subgroup of S_k consisting of all even permutations (permutations that can be written as a product of an even number of transpositions).

Corollary 4.4.3 *For $k > 2$, the automorphism group* $\text{Aut}(T)$ *of a k-hypertournament T does not have the alternating group A_k as section.*

Proof. Suppose H/K is a section of G. I.e. H is some subgroup of G and K is some normal subgroup of H. Since $|H|$ divides $|G|$, it follows that $|H/K|$ also divides $|G|$. Thus, the order of a section of G must divide $|G|$.

Now, suppose to the contrary that A_k is a section of $\text{Aut}(T)$, where T is a k-hypertournament for some $k > 2$. Then $|A_k|$ divides $|\text{Aut}(T)|$. As $|A_k| = (k!)/2$, it follows that $k!$ divides $2|\text{Aut}(T)|$. It implies that k must divide $|\text{Aut}(T)|$, as 2 is a factor of $(k-1)!$. Hence, each prime factor of k divides $|\text{Aut}(T)|$, which contradicts Lemma 4.4.2. ∎

Definition 4.4.4 [78] *A finite group G is said to be a group* not involving *the alternating group A_k if A_k does not occur as a section of the group G.*

Denote by $\mathcal{C}_k$ the class of finite groups G not involving A_k. The class $\mathcal{C}_k$ is known to be closed under taking subgroups, quotients and extensions [78].

Notice that by Corollary 4.4.3, the automorphism group $\text{Aut}(T)$ of any k-hypertournament T does not involve A_k and hence $\text{Aut}(T) \in \mathcal{C}_k$. This property is crucial for our canonization algorithm. We recall a celebrated result about *primitive permutation groups* not involving A_k. We state the theorem from [78], which is a strengthening of the original result due to Babai et al [24].

Theorem 4.4.5 [24, 78] *Let k be a positive integer and $G \leq S_n$ be a* primitive group *in $\mathcal{C}_k$ (i.e. G is a group not involving A_k), then $|G|$ is bounded by $n^{O(k)}$.*

Let $T = (V, A)$ be a k-hypertournament with n vertices. We define the i-*degree* of a vertex $v \in V$ as the number d_i of hyperedges in which v occurs at the ith position, $1 \leq i \leq k$. Thus, to each $v \in V$ we can associate its *degree vector* $(d_1, d_1, \ldots, d_k)$, which is a k-tuple of nonnegative integers such that $0 \leq d_i \leq \binom{n-1}{k-1}$.

We say T is a *regular* k-hypertournament if every vertex $v \in V$ has the same degree vector $(d_1, d_2, \ldots, d_k)$.

Proposition 4.4.6 *Let T be an n-vertex regular k-hypertournament where the degree vector of each $v \in V$ is $(d_1, d_2, \ldots, d_k)$.*

1. $n \sum_{i=1}^{k} d_i = k\binom{n}{k}$.
2. $d_i = \frac{1}{n}\binom{n}{k}$ *for each i.*

Proof. To see the first part, notice that $n \sum_{i=1}^{k} d_i$ adds up the terms of the degree vector $(d_1, d_2, \ldots, d_k)$ for all the n vertices in V. That amounts to counting k times each of the $\binom{n}{k}$ hyperedges of T. To see the second part, notice that in the $\binom{n}{k}$ hyperedges every vertex $v \in V$ occurs as the ith vertex in exactly d_i hyperedges. Thus, $nd_i = \binom{n}{k}$. ∎

An *edge-colored* k-hypertournament is a k-hypertournament $T = (V, A)$ such that its edges are colored with r colors for some positive integer r. More precisely, the edge-coloring is given by a mapping $c : A \to \{1, 2, \cdots, r\}$ that assigns to each edge one of r different colors.

Two edge-colored k-hypertournaments T_1 and T_2 are isomorphic if there is a bijection $\psi : V(T_1) \to V(T_2)$ such that $e \subset V(T_1)$ is a hyperedge in T_1 if and only if $\psi(e)$ is a hyperedge in T_2 and both e and $\psi(e)$ have the same color. Canonical forms for k-hypertournaments are defined in the usual manner under the action of the permutation group $\mathrm{Sym(V)}$.

We will actually consider the more general problem of isomorphism and canonization of an edge-colored k-hypertournament $T = (V, A)$. An important step in our algorithm will be the application of the Babai-Luks algorithm Theorem 4.1.4 to k-hypertournaments. Let $T = (V, A)$ be an edge-colored k-hypertournament and $G \leq \mathrm{Sym(V)}$ be any permutation group.

For each i, $1 \leq i \leq r$, we encode the i^{th} color class $c^{-1}(i) \subset A$ as a binary string b_i of length $k!\binom{n}{k}$. The string b_i is indexed by the $k!\binom{n}{k}$ different k-sequences S of vertices from V, where $b_i[S] = 1$ if and only if $S \in A_i$. Hence the entire hypergraph T gets encoded as the binary string $b_1 b_2 \cdots b_r$ of length $r.k!\binom{n}{k} = m$. We consider this string to be indexed by (i, S) where $1 \leq i \leq r$ and S is an ordered k-sequence of vertices.

The subgroup $G \leq \mathrm{Sym(V)}$ has a natural action on the ordered k-sequences S under which $S = \langle v_1, v_2, \cdots, v_k \rangle$ is mapped to $S^\sigma = \langle v_1^\sigma, v_2^\sigma, \cdots, v_k^\sigma \rangle$ by $\sigma \in \mathrm{Sym(V)}$. Under this action, σ maps the index (i, S) to (i, S^σ) for each color i and k-sequence S.

Therefore, we can consider G as a subgroup of S_m acting on length-m binary strings. Putting it together, we get an instance of string canonization to which Theorem 4.1.4 is applicable. Notice that if $G \in \mathcal{C}_k$, then by Theorem 4.4.5 the string canonization algorithm of Theorem 4.1.4 will run in time $m^{O(k)}$. Hence, we have the following as an immediate corollary of Theorems 4.1.4 and 4.4.5.

Corollary 4.4.7 *Given a k-hypertournament $T = (V, A)$ with edges colored by r colors, a subgroup $G \leq \mathrm{Sym(V)}$ such that $G \in \mathcal{C}_k$, and a $\sigma \in \mathrm{Sym(V)}$ the edge-colored k-hypertournament T can be canonized under $G\sigma$ action in time $(r.k!\binom{n}{k})^{O(k)}$. In particular, the canonical labeling coset $\mathrm{CL}(T, G\sigma)$ can be computed in $n^{O(k^2)}$ time.*

The main question then is how do we reduce k-hypertournament canonization to a case where we have in place a permutation group from $\mathcal{C}_k$ so we can invoke Theorem 4.1.4. We are now ready to prove the main result of this section.

Theorem 4.4.8 *There is an $n^{O(k^2+\log n)}$ time algorithm for canonizing edge-colored k-hypertournaments. As a consequence, isomorphism testing for k-hypertournaments is in $n^{O(k^2+\log n)}$ time.*

Proof. We first give a description of the canonization algorithm, which is recursive. Let $T = (V, A)$ be the input edge-colored k-hypertournament.

Algorithm Description.

Case 0 $\{k = 2\}$: If $k = 2$ then we can invoke the Babai-Luks canonizing algorithm (Theorem 4.3.4) that runs in time $n^{O(\log n)}$.

Case 1 $\{$*Vertex partitioning by degree vectors*$\}$: The algorithm carries out this step for $k > 2$:

If T is not regular, partition V as, $V = V_1 \cup V_2 \cup \cdots \cup V_m$, where V_i ($1 \leq i \leq m$) is the set of all vertices having the same degree vector, where the degree vectors are sorted in lexicographic order. For $1 \leq i \leq m$, let T_i be the k-hypertournament induced by V_i. We recursively compute $\mathrm{CL}(T_i, \mathrm{Sym(V_i)})$ for all i. Let $\mathrm{CL}(T_i, \mathrm{Sym(V_i)}) = \mathrm{H_i}\rho_\mathrm{i}$ where $H_i =$ Aut(T_i) and $\rho_i \in \mathrm{CL}(T_i, \mathrm{Sym(V_i)})$.

Since H_i are the automorphism groups of edge-colored k-hypertournaments, notice that $H_i \in \mathcal{C}_k$ by Corollary 4.4.3. Let H denote the group product $H_1 \times \cdots \times H_m$ and $\rho = (\rho_1, \cdots, \rho_m)$. Then we set $\mathrm{CL}(T, \mathrm{Sym(V)}) = \mathrm{CL(T, H}\rho)$. Notice that by Corollary 4.4.7, we can compute $\mathrm{CL}(T, H\rho)$ in time $n^{O(k^2)}$.

Repeated application of this phase eventually reduces the original k-hypertournament into an *ordered set* of regular k-hypertournaments, and it suffices to canonize each regular k-hypertournament in this list. In the next phase we explain the canonization of regular k-hypertournaments.

Case 2 {*Regular k-hypertournament phase*}: Suppose $T = (V, A)$ is a regular k-hypertournament. If $k = 2$ then we invoke Case 0.

Suppose $k > 2$. The algorithm will make $n = |V|$ recursive calls, one for each vertex $v \in V$. In the call corresponding to v, the algorithm places v as the first vertex in the canonical ordering and recurses on smaller hypertournaments. Finally, the algorithm picks the ordering that is lexicographically least among these n orderings.

We now describe the recursive call that places v as the first vertex of the ordering. Using v, the algorithm will decompose the k-hypertournament T into an edge-colored $(k-1)$-hypertournament T' on $n-1$ vertices and a k-hypertournament T'' on $n-1$ vertices.

The edge-colored $(k-1)$-hypertournament $T' = (V', A')$ is defined as follows: $V' = V \setminus \{v\}$. For $1 \leq i \leq k$, the set of hyperedges A_i (colored i) consists of all $\frac{1}{k}\binom{n-1}{k-1}$ sequences of length $k-1$ obtained by taking each of the $\frac{1}{k}\binom{n-1}{k-1}$ hyperedges of T containing v at the ith place and dropping v from the sequence. The set of all hyperedges is the *disjoint* union $A' = \cup_{i=1}^{k} A_i$. Notice that T' is a $(k-1)$-hypertournament that is edge-colored using k colors.

Next we define $T'' = (V', A'')$ with vertex set V'. Let A'' denote all hyperedges of T not containing v. Then $T'' = (V', A'')$ is a k-hypertournament on $n-1$ vertices.

First, the algorithm will recursively canonize the edge-colored $(k-1)$-hypertournament T'. Let $\mathrm{CL}(T', \mathrm{Sym(V')}) = \mathrm{G}\rho$. By Corollary 4.4.3, $G \in \mathcal{C}_{k-1}$. Now, the algorithm invokes Corollary 4.4.7 and applies the Babai-Luks algorithm to directly canonize T'' w.r.t. the coset $G\rho$ in time $n^{O(k^2)}$. Suppose that algorithm returns the coset CL$(T'', G\rho) = H_v\tau_v$ for vertex v. From the set of vertices V we can pick the subset S such that for each $v \in S$ the canonical labeling coset $H_v\tau_v$ gives the lexicographically least tournament. Clearly, these cosets $H_v\tau_v$, $v \in S$ must paste together into a single coset Aut$(T)\tau$, which is CL$(T, \mathrm{Sym(V)})$, where τ can be chosen as τ_v for some fixed $v \in S$, and a generating set for Aut(T) is the union of generating sets for $H_u, u \in S$ and the set $\{\tau_u\tau_w^{-1} \mid u, w \in S\}$.

Correctness.

We argue the correctness of the above canonization algorithm by induction on k and n. As the base case, notice that the algorithm correctly works for $k = 2$ and for all n as that is the edge-colored tournament canonization algorithm [30] (see Theorem 4.3.4).

As induction hypothesis, suppose the canonization algorithm works correctly for all edge-colored ℓ-hypertournaments for $\ell < k$. Further, suppose the algorithm works correctly for all edge-colored k-hypertournaments with fewer than n vertices.

For the induction step let $T = (V, A)$ be an edge-colored k-hypertournament on n vertices with $k > 2$. The algorithm will either apply the steps in Case 1 or in Case 2. We need to argue correctness for both cases.

Case 1. Suppose Case 1 is applied to the k-hypertournament $T = (V, A)$ on n vertices. This is the case when T is not regular and the vertex set is partitioned as $V = \cup_{i=1}^{m} V_i$, where each V_i is a vertex subset of all vertices with the same degree vector and the index i is the sorted order of the degree vectors. By induction hypothesis, for the k-hypertournaments T_i induced by V_i the algorithm correctly computes CL$(T_i, \mathrm{Sym(V_i)}) = \mathrm{H_i\rho_i}$, where $H_i =$ Aut(T_i) and $\rho_i \in$ CL$(T_i, \mathrm{Sym(V_i)})$. Thereafter, the algorithm invokes the Babai-Luks string canonization algorithm on T encoded as a string and the product of cosets $H_1\rho_1 \times \cdots \times H_m\rho_m$. Thus the correctness of the induction step for this Case 1 follows from Theorem 4.1.4.

Case 2. Next, suppose T is a regular k-hypertournament, which means that Case 2 is applied to it. Let $T_1 = (V, A_1)$ and $T_2 = (V, A_2)$ be two isomorphic k-hypertournaments on n vertices to which Case 2 of the algorithm is applied. We need to show that the algorithm produces the same canonical form for both T_1 and T_2. Let $\psi : V \to V$ be an isomorphism from T_1 to T_2.

Recall that the algorithm works as follows in Case 2: Let the input be T_1 and $v \in V$ be a vertex. The algorithm computes an edge-colored $(k-1)$-hypertournament T_1' on $V \setminus \{v\}$ and the remainder k-hypertournament T_1'' on $V \setminus \{v\}$. Recursively, the algorithm canonizes T_1' and using the resulting canonizing coset $G_1\rho_1$ the algorithm canonizes T_1'' (by applying Theorem 4.1.4). Let $T_1^{(v)}$ denote the k-hypertournament obtained by the algorithm by picking $v \in V$ as the first vertex. The algorithm carries out these steps for each $v \in V$ and picks the lexicographically least among them as the canonical form.

In order to show that T_1 and T_2 result in the same canonical form, we claim it suffices to argue that $T_1^{(v)}$ and $T_2^{(\psi(v))}$ are the same k-hypertournaments where $T_2^{(\psi(v))}$ is the canonical form obtained by the algorithm by picking $\psi(v)$ as the first vertex while canonizing T_2. For, if the multisets $\{\{T_1^{(v)} \mid v \in V\}\}$ and $\{\{T_2^{(\psi(v))} \mid v \in V\}\}$ are the same, then clearly the lexicographically least elements in the sets are the same (with multiplicity).

Thus, it suffices to show that $T_1^{(v)} = T_2^{(\psi(v))}$. Since $\psi : V \to V$ is an isomorphism from T_1 to T_2, notice that ψ will also be an (edge-color preserving) isomorphism between the $(k-1)$-hypertournaments T_1' and T_2' where T_2' is the edge-colored $(k-1)$-hypertournament obtained by the algorithm when canonizing T_2 by placing $\psi(v)$ as the first vertex. Hence, by induction hypothesis the algorithm will compute the same canonical forms for T_1' and T_2'. More precisely, $\sigma_1(T_1') = \sigma_2(T_2')$ for any $\sigma_1 \in G_1\rho_1$ and $\sigma_2 \in G_2\rho_2$ where $G_2\rho_2$ is the canonical labeling coset obtained by the algorithm while canonizing T_2'. Therefore, canonizing T_1'' using coset $G_1\rho_1$ and canonizing T_2'' using coset $G_2\rho_2$ will also result in the same k-hypertournaments by Lemma 4.1.5. Putting it together it follows that $T_1^{(v)} = T_2^{(\psi(v))}$. This completes the correctness proof.

Running Time.

To analyze the running time, let $t(n,k)$ denote the running time taken by the algorithm for n-vertex k-hypertournaments. We claim that the following two recurrences hold for Cases 1 and 2 respectively:

$$t(n,k) = \begin{cases} \sum_{i=1}^{m} t(|V_i|, k) + n^{O(k^2)} & \text{Case 1 and } k > 2 \\ n(t(n-1, k-1) + n^{O(k^2)}) & \text{Case 2 and } k > 2 \end{cases}$$

First, notice that $t(n, 2) = n^{O(\log n)}$ using [30] for the base case $k = 2$. For Cases 1 and 2 the recurrence relation follows directly from the algorithm description; the additive term of $n^{O(k^2)}$ in the two cases is a consequence of Theorem 4.4.7.

The claimed time bound $t(n, k) = n^{O(k^2+\log n)}$ is the solution for this recurrence. ■

Remark 4.4.9 *Finally, analogous to Theorem 4.3.9, we note that it is possible to canonize k-hypertournaments in polynomial time with queries to k-hypertournament isomorphism and rigid k-hypertournament canonization. The details of this reduction are quite similar to the proof of Theorem 4.3.9.*

5 An FPT Algorithm for Bounded Color Class Hypergraph Isomorphism

5.1 Introduction

We recall that a *hypergraph* is an ordered pair $X = (V, E)$ where V is the vertex set and $E \subseteq 2^V$ is the edge set. Two hypergraphs $X_1 = (V_1, E_1)$ and $X_2 = (V_2, E_2)$ are said to be *isomorphic*, denoted $X_1 \cong X_2$, if there is a bijection $\phi : V_1 \longrightarrow V_2$ such that $e = \{u_1, \cdots, u_l\} \in E_1$ if and only if $\phi(e) = \{\phi(u_1), \cdots, \phi(u_l)\} \in E_2$. Given two hypergraphs X_1 and X_2 the hypergraph isomorphism problem (HGI) is to decide if $X_1 \cong X_2$. Graph isomorphism (GI) is obviously polynomial-time reducible to HGI. Conversely, HGI is also known to be polynomial-time reducible to GI: Given two hypergraphs $X_1 = (V_1, E_1)$ and $X_2 = (V_2, E_2)$ as instance for HGI, the reduced instance of GI consists of two corresponding bipartite graphs Y_1 and Y_2 defined as follows. The graph Y_1 has vertex set $V_1 \uplus E_1$ and edge set $E(Y_1) = \{\{v, e\} \mid v \in V_1, e \in E_1 \text{ and } v \in e\}$.[1] Similarly Y_2 has vertex set $V_2 \uplus E_2$ and edge set $E(Y_2) = \{\{v, e\} \mid v \in V_2, e \in E_2 \text{ and } v \in e\}$. Clearly, the hypergraphs X_1 and X_2 are isomorphic if and only if $Y_1 \cong Y_2$.

We now define bounded color class hypergraph isomorphism problem (BCHGI), which is the main problem of interest for this chapter.

Definition 5.1.1 (BCHGI) **Input**: *Two hypergraphs $X_1 = (V, E_1)$ and $X_2 = (V, E_2)$ where $V = C_1 \uplus \cdots \uplus C_m$ and for each i, $|C_i| \leq b$.*

[1] Given two set C and D the disjoint union of C and D is denoted as $C \uplus D$.

Decide: *If there is an isomorphism ϕ form X_1 to X_2 such that $v \in C_i$ if and only if $\phi(v) \in C_i$.*

The bounded color class graph isomorphism graph isomorphism problem (BCGI) is the analogous problem where instead of hypergraphs we have graphs. Frust, Hopcroft and Luks gave a polynomial-time algorithm for BCGI [53] with running time $2^{O(b^2)}n^{O(1)}$, where n is the number of vertices and b is the size of the largest color class. Although HGI is polynomial time many-one reducible to GI, the reduction we described above does not impose any bound on the size of the color classes of the bipartite graphs Y_i. More specifically, in the bipartite graph Y_i, the vertex partition corresponding to the edges of the hypergraph X_i does not get partitioned into classes of size bounded by any function of b since the hyperedges are of *nonconstant* size, in general. Thus, the polynomial-time algorithm for BCGI cannot be applied to the bounded color class hypergraph isomorphism problem (BCHGI) to get a polynomial-time algorithm.

However, an algorithm for BCHGI with running time of the form $n^{O(b)}$ was shown in [17], where n is the number of vertices and b bounds the color classes. This algorithm basically applies Luks' seminal result [80] showing that the set stabilizer problem with respect to a class of group Γ_b can be solved in time $n^{O(b)}$. Recall that a finite group G is said to be in the class Γ_b if every *nonabelian* composition factor of G is isomorphic to some subgroup of the symmetric group S_b.

Parameterized Complexity and Isomorphism Testing

Parameterized Complexity is a fundamental strategy for coping with intractability. Pioneered by Downey and Fellows in [44], it is a flourishing area of research (see, e.g. the monographs [45, 50]). Fixed parameter tractability provides a notion of feasible computation less restrictive than polynomial time. It provides a theoretical basis for the design of new algorithms that are efficient and practically useful for small parameter values. We quickly recall the rudiments of this theory relevant for the present chapter. More details (especially on the levels of the W-hierarchy) will be given in the next section (see also [45, 50]).

Computational problems often have inputs consisting of two or more parts where some of these parts typically take only small values. For example, an input instance of the vertex cover problem is (G, k), and the task is to determine if the graph G has a vertex cover of size k. A similar example is the k-clique problem where again an input instance is a pair (G, k) and the problem is to test if the graph G has a clique of size k. For such problems an exhaustive search will take time $O(n^k)$, where n is the number of vertices in G. However, a finer classification is possible. The vertex cover problem has an algorithm running in time $2^k n^{O(1)}$ (even in time $O(1.2738^k + kn)$ [41]), whereas no algorithm is known for the k-clique problem of running time $O(n^{o(k)})$. Thus, if the parameter k is such that $k \ll n$, then we have a faster algorithm for the k-vertex cover problem than is known for the k-clique problem.

In their seminal work, Downey and Fellows [44, 45] also developed a theory of intractability for parameterized problems as a tool to classify parameterized problems according to their computational hardness. The W-hierarchy consists of the levels $W[t]$, $t \geq 1$, together with the two classes W[SAT] and W[P] and we have the inclusions

$$\text{FPT} \subseteq \mathsf{W}[1] \subseteq \mathsf{W}[2] \subseteq \cdots \subseteq \mathsf{W}[\text{SAT}] \subseteq \mathsf{W}[\text{P}].$$

Since Graph Isomorphism has no known polynomial time algorithm for general class of graphs, one approach is to study isomorphism testing for special classes of graphs. Often these special classes are defined by bounding some graph parameter. For example, by bounding the maximum degree, or treewidth, or genus of the graph by some constant b we get degree-b graphs, or treewidth-b graphs, or genus-b graphs respectively. Thus, these natural graph parameters give rise to parameterized versions of the graph isomorphism problem. Interestingly, the best known isomorphism testing algorithms for graphs with bounded degree [80], bounded treewidth graphs [34], and for bounded genus graphs [87] all have worst-case running time bound $n^{O(b)}$ where b is the parameter and n is the number of vertices.

It is an interesting open question if GI has FPT algorithm with respect to any of the aforementioned parameters. We know that GI is unlikely to be hard for NP [4]. It would be interesting to know if these parameterized versions of GI are W[1]-hard or fixed parameter tractable.

Though there are several open problems regarding the parametrized complexity of graph isomorphism problem with respect to natural parameters, on the positive side we note that some FPT algorithms have been obtained for graph isomorphism with parameters tree-distance width [116], maximum size of the simplicial components [103], apart from the result of Furst et al [53] on bounded color classes.

The results

In this chapter we present an FPT algorithm for bounded color class hypergraph isomorphism that runs in time $b!2^{O(b)}N^{O(1)}$ where b is the size of the largest color class and N is the sum of number of vertices and edges.

We will use as subroutine some FPT algorithms for certain parameterized permutation group theory problems. In Section 5.3 we define permutation groups with a *color class bound*. Following Luks's method [83], we design an FPT algorithm for coset intersection of permutation groups where the parameter is the color class bound. We also give an FPT algorithm for set transporter problem. (These problems are formally defined in Section 5.3). While the parametrized complexity of permutation group problems, for different parameters, can be interesting in its own right, it could be applicable to graph isomorphism. For example, an FPT algorithm for the set transporter problem with respect to groups in Γ_b, with b as parameter, will result in an FPT algorithm for isomorphism testing for graphs of maximum degree b, which is a major open problem. The parameter b is a different natural parameterization for permutation group problems.

5.2 Preliminaries

While designing the FPT algorithm for bounded color class hypergraph isomorphism we will have to handle multihypergraphs: these are hypergraphs where the edge set is a multiset (each edge occurs with a multiplicity). Two multihypergraphs $X_1 = (V_1, E_1)$ and $X_2 = (V_2, E_2)$ are isomorphic via an isomorphism ϕ if $e \in E_1$ has the same multiplicity as $\phi(e)$ in E_2.

Apart form the basic permutation group theory introduced in the preliminary section of Chapter 2 we need the following two definitions for this chapter.

Definition 5.2.1 *For a permutation $\pi \in \mathrm{Sym}(\Omega)$ and $C \subseteq \Omega$ we define $C^\pi = \{x^\pi \mid x \in C\}$. For a permutation group $G \leq \mathrm{Sym}(\Omega)$, a subset $C \subseteq \Omega$ is called G-*stable *if $C^\pi = C$ for all $\pi \in G$.*

Definition 5.2.2 *For a group $G \leq \mathrm{Sym}(\Omega)$ and a set $C \subseteq \Omega$, the* stabilizer subgroup *of G that* stabilizes *C, denoted G_C, is defined as*

$$G_C = \{\pi \in G \mid C^\pi = C\}.$$

5.3 Permutation Group Problems

We say that a group $G \leq \mathrm{Sym}(\Omega)$ has *color class bound* b if each G-orbit is of size bounded by b. Equivalently, Ω can be partitioned into a disjoint union of color classes $C_1 \uplus \cdots \uplus C_m$, where each *color class* C_i is G-stable and $|C_i| \leq b$. In this section we describe FPT algorithms for some permutation group problems where the parameter is the color class bound for the input groups. Given $G = \langle S \rangle$ by generating set S, since the orbits of G can be computed in polynomial time we can test if G has color class bound b in polynomial time.

The basic idea in solving these group-theoretic problems parameterized by the color class bound is to tackle one color class at a time. In some sense, we will restrict the problem to a particular color class and solve the problem for that color class and then proceed to the next color class.

5.3.1 Set Transporter

First we define the set transporter problem for general permutation groups.

Definition 5.3.1 (Set Transporter)

Input : *A group* $G \leq \mathrm{Sym}(\Omega)$ *given by generating set,* $z \in \mathrm{Sym}(\Omega)$ *and two subsets* $\Pi_1, \Pi_2 \subseteq \Omega$.

Output : $(Gz)_{\Pi_1 \to \Pi_2} = \{x \in Gz \mid \Pi_1^x = \Pi_2\}$.

For general permutation groups, it is known that Graph Isomorphism is polynomial-time reducible to Set Transporter. However, we are interested in Set Transporter parameterized by the color class bound.

Definition 5.3.2 (Bounded Color Class Set Transporter BC-TRANS)

Input : *A group* $G \leq \mathrm{Sym}(\Omega)$ *given by a generating set where* $\Omega = C_1 \uplus \cdots \uplus C_m$ *and for each* i*,* C_i *is* G*-stable,* $z \in \mathrm{Sym}(\Omega)$ *and two subsets* $\Pi_1, \Pi_2 \subseteq \Omega$.

Parameter : $b =$ *max* $\{|C_1|, \cdots, |C_m|\}$.

Output : $(Gz)_{\Pi_1 \to \Pi_2} = \{x \in Gz \mid \Pi_1^x = \Pi_2\}$.

The FPT algorithm for solving BC-TRANS works by restricting the problem to a particular color class, solving it, and then proceeding to the next color class. The restricted version of the problem is defined as follows.

Definition 5.3.3 (Restricted BC-TRANS)

Input : *A group* $G \le \mathrm{Sym(C \uplus D)}$ *given by a generating set where* C *and* D *are* G*-stable,* $z \in \mathrm{Sym(C \uplus D)}$ *and two subsets* $\Pi_1, \Pi_2 \subseteq C$*.*

Parameter : $b = |C|$.

Output : $(Gz)_{\Pi_1 \to \Pi_2} = \{x \in Gz \mid \Pi_1^x = \Pi_2\}$.

Theorem 5.3.4 *Let* $G \le \mathrm{Sym(C \uplus U)}$ *be a group given by a generating set and* C *be* G*-stable with* $b = |C|$*. Let* $z \in \mathrm{Sym(C \uplus U)}$ *be a permutation and* Π_1, Π_2 *be two subsets of* C*. Then* $(Gz)_{\Pi_1 \longrightarrow \Pi_2}$ *can be computed in time* $2^{O(b)}n^{O(1)}$ *where* $n = |C| + |U|$*.*

Proof. Let G_{Π_1} be the stabilizer subgroup of G that stabilizes Π_1, i.e., $G_{\Pi_1} = \{x \in G \mid \Pi_1^x = \Pi_1\}$. Since C is G-stable the set Π_1 can only move to a subset of size $|\Pi_1|$ contained in C. Thus,

$$[G_{\Pi_1} : G] \le \binom{b}{|\Pi_1|} \le 2^b.$$

Also note that given $x \in G$, it only takes $O(n)$ time to check if $x \in G_{\Pi_1}$.

As discussed in the priliminary section of Chapter 2, a set of coset representatives $\{\rho_1, \cdots, \rho_t\}$ of G_{Π_1} in G can be computed in time $2^{O(b)}n^{O(1)}$. By Theorem 2.2.14 a set of generators can also be computed in time $2^{O(b)}n^{O(1)}$. Using Schreier-Sims methology, the number of generators can be reduced to n^2 by computing a strong generating set for G_{Π_1} also in time $2^{O(b)}n^{O(1)}$.

We write

$$Gz = G_{\Pi_1}\rho_1 z \uplus \cdots \uplus G_{\Pi_1}\rho_t z.$$

The algorithm next picks the coset $G_{\Pi_1}\rho_i z$ that sends Π_1 to Π_2 if there is any such coset[2] and outputs that coset otherwise it outputs the empty set. ■

Remark 5.3.5 *The group* G_{Π_1} *that is output by the algorithm stabilizes any* G*-stable set* F*.*

Theorem 5.3.6 *Let* $G \le \mathrm{Sym}(\Omega)$ *be a group given by a generating set. Suppose* $\Omega = C_1 \uplus \cdots \uplus C_m$ *such that each* C_i *is* G*-stable. Let* z *be a permutation in* $\mathrm{Sym}(\Omega)$ *and* Π_1 *and* Π_2 *be two subsets of* Ω*. Then* $(Gz)_{\Pi_1 \longrightarrow \Pi_2}$ *can be computed in time* $2^{O(b)}n^{O(1)}$ *where* $b =$ *max*$\{|C_1|, \cdots, |C_m|\}$*.*

[2]There can be at most one such coset.

Proof. Let $\Pi_1^{(i)} = C_i \cap \Pi_1$ and $\Pi_2^{(i)} = C_i \cap \Pi_2$. Let $G_0 = G$ and $z_0 = z$. For $i = 1, \cdots, m$ compute

$$G_i z_i = (G_{i-1} z_{i-1})_{\Pi_1^{(i)} \longrightarrow \Pi_2^{(i)}}$$

using the algorithm of Theorem 5.3.4. By Remark 5.3.5 the subgroups G_i computed by the algorithm will stabilize the sets C_j for each i and j. Thus, by Theorem 5.3.4 computing $G_i z_i$ from $G_{i-1} z_{i-1}$ will take time $2^{O(b)} n^{O(1)}$. Hence, the overall running time is also $2^{O(b)} n^{O(1)}$. Notice that $\Pi_1 = \Pi_1^{(1)} \uplus \cdots \uplus \Pi_1^{(m)}$ and $\Pi_2 = \Pi_2^{(1)} \uplus \cdots \uplus \Pi_2^{(m)}$. Furthermore, for each i and $x \in G_m z_m$ we have $(\Pi_1^{(i)})^x = \Pi_2^{(i)}$. Hence it is easy to see that $G_m z_m = (Gz)_{\Pi_1 \longrightarrow \Pi_2}$. ■

Definition 5.3.7 (Bounded Color Class Set Stabilizer BC-STAB)

Input : *A group* $G \leq \mathrm{Sym}(\Omega)$ *given by a generating set where* $\Omega = C_1 \uplus C_2 \cdots \uplus C_m$ *and for each* i, C_i *is* G*-stable,* $z \in \mathrm{Sym}(\Omega)$ *and a subset* $\Pi \subseteq \Omega$*.*

Parameter : $b =$ *max* $\{|C_1|, \cdots, |C_m|\}$.

Output : $(Gz)_\Pi = \{x \in Gz \mid \Pi^x = \Pi\}$.

If we set $\Pi_1 = \Pi$ and $\Pi_2 = \Pi$ in Definition 5.3.2 of BC-TRANS, we get BC-STAB problem. Thus, by Theorem 5.3.6 this problem can also be solved in time $2^{O(b)} poly(n)$ where $n = |\Omega|$.

Corollary 5.3.8 *The problem BC-STAB can be solved in time* $2^{O(b)} poly(n)$.

5.3.2 Coset Intersection

The coset intersection problem in its full generality is defined as follows.

Definition 5.3.9 (Coset Intersection)

Input : *Two groups* $G, H \leq \mathrm{Sym}(\Omega)$ *be given by generating sets and* $x, y \in \mathrm{Sym}(\Omega)$.

Output : $Gx \cap Hy$.

Graph Isomorphims is known to be polynomial-time reducible to Coset Intersection [82] since Coset Intersection is polynomial time many-one equivalent to Set Transporter. However, the known polynomial-time reduction between these problems increases the parameter value; it is not a paramterized reduction [45]. Therefore, we cannot invoke the algorithm for BC-TRANS to given an FPT algorithm for the parametrized version of the coset intersection problem which we will define. In this section we present an FPT algorithm for BC-INTER. This algorithm will be used to solve the bounded color class hypergraph isomorphism problem.

Definition 5.3.10 (BC-INTER)

Input : *Let $\Omega = C_1 \uplus \cdots \uplus C_m$. Let $G, H \leq \mathrm{Sym}(\Omega)$ be given by their generating sets such that for each i C_i is both G-stable and H-stable, $x, y \in \mathrm{Sym}(\Omega)$. Let $|C_i| \leq b$ for all i.*

Parameter : *$b = max\{|C_1|, \cdots, |C_m|\}$.*

Output : *$Gx \cap Hy$.*

Let $L = G \times H \leq \mathrm{Sym}(\Omega) \times \mathrm{Sym}(\Omega) = \mathcal{G}$. Let $z = (x, y) \in \mathcal{G}$. Let $\Pi = \{(a, a) \mid a \in \Omega\}$. Notice that $(Lz)_\Pi = \{x \in Lz \mid \Pi^x = \Pi\}$ projected to the first or second coordinate is $Gx \cap Hy$. Since this is an instance of BC-STAB we could use Corollary 5.3.8 to compute $(Lz)_\Pi$. Unfortunately, such an algorithm for would take time $2^{O(b^2)} n^{O(1)}$, since the group L has color class size bound b^2 (and not b). Nevertheless, using a different algorithm we can still solve the problem in $2^{O(b)} n^{O(1)}$ time.

Lemma 5.3.11 *$(Lz)_\Pi$ can be computed in time $2^{O(b)} n^{O(1)}$.*

Proof. The main claim required to prove the lemma is stated below.

Claim 5.3.12 *Let $\Omega_1 = C \uplus U$ and $\Omega_2 = D \uplus V$. Let $L \leq \mathrm{Sym}(\Omega_1) \times \mathrm{Sym}(\Omega_2)$ be given by generating set such that C and D are L-stable. Let $z \in \mathrm{Sym}(\Omega_1 \times \Omega_2)$. Let $\Pi \subseteq C \times D$. Then $(Lz)_\Pi$ can be computed in time $2^{O(b)} n^{O(1)}$ where $n = |\Omega_1| + |\Omega_2|$ and $b = \max\{|C|, |D|\}$.*

Before proving the above claim, we apply it to prove the lemma. We will use the following two consequences of Claim 5.3.12.

Claim 5.3.13 *Let $\Omega_1 = C_1 \uplus \cdots \uplus C_l$ and $\Omega_2 = D \uplus V$. Let $L \leq \mathrm{Sym}(\Omega_1) \times \mathrm{Sym}(\Omega_2)$ be a permutation group given by a generating set, such that D and each C_i are L-stable sets. Let $z \in \mathrm{Sym}(\Omega_1 \times \Omega_2)$ and $\Pi \subseteq \Omega_1 \times D$. Then $(Lz)_\Pi$ can be computed in time $2^{O(b)} n^{O(1)}$ where $b = \max\{|D|, |C_1|, \cdots, |C_l|\}$ and $n = |\Omega_1| + |\Omega_2|$.*

Proof. We define the sets $\Pi_i = \Pi \cap (C_i \times D)$. We will repeatedly apply Claim 5.3.12. To start off we first compute $L_1 z_1 = (Lz)_{\Pi_1}$. Then we apply Claim 5.3.12 to compute $L_i z_i = (L_{i-1} z_{i-1})_{\Pi_i}$ from $L_{i-1} z_{i-1}$ for $i = 2, \cdots, m$. By Claim 5.3.12 it follows that the time taken for this computation is $2^{O(b)} n^{O(1)}$. We claim that for all i, $L_i z_i = (Lz)_{\Pi_1 \uplus \cdots \uplus \Pi_i}$. This follows form the fact that $((Lz)_{\Pi_1 \uplus \cdots \uplus \Pi_{i-1}})_{\Pi_i} = (Lz)_{\Pi_1 \uplus \cdots \uplus \Pi_i}$. Thus at the end of the computation we have $L_m z_m = (Lz)_{\Pi_1 \uplus \cdots \uplus \Pi_m} = (Lz)_{\Pi}$. ■

Claim 5.3.14 *Let $\Omega_1 = C_1 \uplus \cdots \uplus C_l$ and $\Omega_2 = D_1 \uplus \cdots \uplus D_m$. Let $L \leq \mathrm{Sym}(\Omega_1) \times \mathrm{Sym}(\Omega_2)$ be a permutation group give by a generating set such that each C_i and each D_j is an L-stable subset. Let $z \in \mathrm{Sym}(\Omega_1 \times \Omega_2)$ and $\Pi \subseteq \Omega_1 \times \Omega_2$. Then $(Lz)_\Pi$ can be computed in time $2^{O(b)} n^{O(1)}$ where $b = \max\{|C_i|, |D_j|\}_{i,j}$ and $n = |\Omega_1| + |\Omega_2|$.*

Proof. Define $\Pi_j = (C_1 \uplus \cdots \uplus C_l) \times D_j$ for $j = 1, \cdots, m$. We can successively compute $L_j z_j = (L_{j-1} z_{j-1})_{\Pi_j}$ for $j = 1, \cdots, m$, by Claim 5.3.13. Finally, we observe that $L_m z_m = (Lz)_\Pi$. ■

The proof of the lemma now follows easily by setting $\Omega_1 = \Omega_2 = C_1 \uplus \cdots \uplus C_m$ in Claim 5.3.14.

To prove Claim 5.3.12 we use ideas from [83]. In [83], the author defines a version of set transporter problem and gives an algorithm that can be easily modified to be applied in the parametrized setting. The parametrized version of the problem is defined below.

Definition 5.3.15

Input : *Let $\Omega_1 = C \uplus U$ and $\Omega_2 = D \uplus V$. Let $L \leq \mathrm{Sym}(\Omega_1) \times \mathrm{Sym}(\Omega_2)$ be given by generating set such that C and D are L-stable. Let $z \in \mathrm{Sym}(\Omega_1) \times \mathrm{Sym}(\Omega_2)$. Let $\Pi \in C \times D$ and $\Theta \subseteq \Phi \times \Psi \subseteq C \times D$. We also assume that $L_\Theta = L$.*

Parameter : $b = \max\{|C|, |D|\}$.

Output : $(Lz)_\Pi[\Theta] = \{x \in Lz \mid (\Pi \cap \Theta)^x = \Pi \cap \Theta^x\}$.

We can observe from [83] that $(Lz)_\Pi[\Theta]$ can be computed in time $2^{O(b)} n^{O(1)}$ where $n = |\Omega_1| + |\Omega_2|$. For the sake of completeness we now give a proof. We modify Luks's proof slightly to suit the current parametrized setting.

We assume without loss of generality that $|\Phi|$ and $|\Psi|$ are powers of 2. If not, we can add some more points on Φ and Ψ and let L act trivially on these points. This will increase the size of the input only by a factor of 4. These extra points can be removed easily from the algorithm's output.

Since $L_\Theta = L$ we have $\Theta^x = \Theta^z$ for all $x \in Lz$. If $(Lz)_\Pi[\Theta]$ is not empty then for $x, y \in (Lz)_\Pi[\Theta]$ we have $(\Pi \cap \Theta)^x = \Pi \cap \Theta^z = (\Pi \cap \Theta)^y$ and hence $(Lz)_\Pi[\Theta]$ is coset of $L_{\Pi\cap\Theta}$. For simplicity we will denote $(Lz)_\Pi[\Theta]$ by $(Lz)[\Theta]$.

If $|\Pi \cap \Theta| \neq |\Pi \cap \Theta^z|$ then $(Lz)[\Theta]$ is empty.

Let $|\Pi \cap \Theta| = |\Pi \cap \Theta^z| = 1$. Let $\Pi \cap \Theta = \{u\}$ and $\Pi \cap \Theta^z = \{v\}$. Let L_u be the stabilizer of the point u which can be computed using Schreier-Sim method. Then we can express L as the disjoint union of cosets as

$$L = L_u x_1 \uplus \cdots \uplus L_u x_k$$

and

$$Lz = L_u x_1 z \uplus \cdots \uplus L_u x_k z$$

We pick the subcoset $L_u x_i z$ (if there is any) that maps u to v.

Next we assume $|\Pi \cap \Theta| = |\Pi \cap \Theta^z| > 1$. If $|\Phi| > 1$ we pick $\Phi_1 \subset \Phi$ such that $|\Phi_1| = |\Phi|/2$ and let $\Theta_1 = \Phi_1 \times \Psi$. Otherwise if $|\Psi_i| > 1$ then we pick $\Psi_1 \subset \Psi$ such that $|\Psi_1| = |\Psi|/2$ and let $\Theta_1 = \Phi \times \Psi_1$. In both the cases we let $\Theta_2 = \Theta \setminus \Theta_1$.

Let $M = L_{\Theta_1}$. Notice that $[L : M] \leq \begin{pmatrix} |\Phi| \\ \frac{|\Phi|}{2} \end{pmatrix} \leq 2^{O(b)}$, when we choose to divide the set Φ. If we divide the set Ψ then $[L : M] \leq \begin{pmatrix} |\Psi| \\ \frac{|\Psi|}{2} \end{pmatrix}$ which is still less than $2^{O(b)}$.

We write L as the disjoint union of the cosets of M

$$L = My_1 \uplus \cdots \uplus My_t$$

and

$$Lz = My_1 z \uplus \cdots \uplus My_t z.$$

As discussed in the preliminary section of Chapter 2, this decomposition of Lz can be computed in time $2^{O(b)} n^{O(1)}$ using Schreier-Sims methodology. Since M stabilizes Θ_1, $(My_i z)[\Theta_1]$ will be a coset.

We use the following relation to setup the recursive calls

$$(My_i z)[\Theta] = ((My_i z)[\Theta_1])[\Theta_2]$$

Finally we paste the answers to the subproblems $(My_1z)[\Theta]$ to get $(Lz)[\Theta]$.

$$(Lz)[\Theta] = \cup_{i=1}^{t}(My_iz)[\Theta]$$

Clearly the running time is $2^{O(b)}n^{O(1)}$.

■

Putting it together, we have shown the following.

Theorem 5.3.16 *There is a $2^{O(b)}n^{O(1)}$ time bounded FPT algorithm for* BC-INTER, *where $n = |\Omega|$.*

5.4 FPT Algorithm for Colored Hypergraph Automorphism and Isomorphism

We will give an FPT algorithm for finding the automorphism group of a hypergraph (i.e., a set of generators for the automorphism group) which has running time $(b!)2^{O(b)}N^{O(1)}$ where b is the bound on the size of the color classes and N is the sum of number of vertices and edges.

Our algorithm is a dynamic programming algorithm. The subproblems of this dynamic programming algorithm will involve multi-hypergraphs. For this reason we begin with a multihypergraph $X = (V, E)$ where V is partitioned into color classes $C_1, \cdots, C_m$. We assume that $|C_i| \leq b$ for all i.

We first partition the edges into different sets of edges that we call blocks. We define these more formally.

Definition 5.4.1 *We say two edges $e_1, e_2, \in E$ are i-*equivalent *and write $e_1 \equiv_i e_2$ if*

$$e_1 \cap C_j = e_2 \cap C_j,\ \ 1 \leq j \leq i.$$

*For each i this defines an equivalence relation. Thus, i-equivalence partitions E into equivalence classes that we term as (i)-*blocks, *$1 \leq i \leq m$. Notice that i-equivalence is a refinement of j-equivalence for $i > j$. Thus, if e_1 and e_2 are in the same (i)-block then they are in the same (j)-block for all $j < i$.*

The algorithm proceeds in stages.

Stage 1. The first stage corresponds to the last color class m. Let A and A' be two (m)-blocks. The (m)-blocks A and A' induce two multihypergraphs Y and Y' on the vertex set C_m respectively. The algorithm will find the set of isomorphisms $ISO_m(Y, Y')$ between these two multihypergraphs. Notice that for all $e_1, e_2 \in A$, $e_1 \cap C_m = e_2 \cap C_m = a$, for some $a \subseteq C_m$. Similarly, for all $e'_1, e'_2 \in A'$, $e'_1 \cap C_m = e'_2 \cap C_m = a'$. Thus the multihypergraph Y induced by A on C_m has only one edge a with multiplicity $|A|$. Similarly, Y' has one edge with multiplicity $|A'|$. Recall that A and A' may themselves be multisets. Hence, $|A|$ and $|A'|$ are their sizes as multisets.

Clearly, $ISO_m(Y, Y') = \phi$ if $|A| \neq |A'|$. Otherwise, $ISO_m(Y, Y') \subseteq \mathrm{Sym(C_m)}$ will be the coset that maps a to a'. Notice that this is a set transporter problem and can be solved in time $2^{O(b)}$ by Theorem 5.3.6. For all pairs of (m)-blocks A and A', the dynamic programming algorithm stores the set $ISO_m(Y, Y')$ in a table where Y and Y' are the multihypergraph induced by A and A' respectively.

Stage $m - i + 1$. The $m - i + 1$th stage of the algorithm handles (i)-blocks. Let A and A' be two (i)-blocks. Let Y and Y' be the multihypergraphs induced by A and A', respectively, on the vertex set $C_i \cup \cdots \cup C_m$. Thus, $V(Y) = C_i \cup \cdots \cup C_m$ and $E(Y) = \{\{e \cap C_i \cup \cdots \cup C_m \mid e \in A\}\}$. The multihypergraph $Y' = (V(Y'), E(Y'))$ is similarly defined.

In this stage the algorithm will compute the coset $ISO_i(Y, Y')$ of all isomorphisms between Y and Y' and store the result in the (A, A') entry of the dynamic progamming table. We explain how the computation is done.

Clearly $ISO_i(Y, Y')$ is empty if the size of the multiedge sets $E(Y)$ and $E(Y')$ of these two multihypergraphs differ. Suppose $E(Y)$ and $E(Y')$ has same size as multisets. For all $e_1, e_2 \in A$ let $e_1 \cap C_i = e_2 \cap C_i = a$, and for all $e'_1, e'_2 \in A'$ let $e'_1 \cap C_i = e'_2 \cap C_i = a'$. Let $S \subseteq \mathrm{Sym(C_i)}$ be the set consisting of permutations that map a to a'. If S is empty then so is $ISO_i(Y, Y')$. Suppose S is nonempty. Let ρ be a permutation on $C_{i+1} \cup \cdots \cup C_m$ that maps Y to Y' isomorphically *when restricted* to color classes $C_{i+1}, \cdots, C_m$. Let $\rho' \in S$ be any permutation. Then, because A and A' are (i)-blocks we have $(\rho, \rho') \in ISO_i(Y, Y')$, where (ρ, ρ') denotes the permutation obtained by extending ρ with ρ' on the color class C_i.

The (i)-block A can be partitioned into $(i + 1)$-blocks $A_1, \cdots, A_k$. Similarly, A'

can be partitioned into $(i+1)$-blocks $A'_1, \cdots, A'_{k'}$. For each j, the $(i+1)$-block A_j induces a multihypergraph Z_j on the color classes $C_{i+1}, \cdots, C_m$. More specifically, $V(Z_j) = C_{i+1} \cup \cdots \cup C_m$ and $E(Z_j) = \{\{e \cap V(Z_j) \mid e \in A\}\}$. Similarly, let $Z'_{j'}$ be the multihypergraph induced by the $(i+1)$-block $A'_{j'}$.

If $k \neq k'$ then $ISO_i(Y, Y') = \emptyset$. Suppose $k = k'$. For each s and t the algorithm has already computed in Stage $m-i$ the coset $ISO_{i+1}(Z_s, Z'_t)$ and stored it in the table. We can use these cosets to efficiently compute $ISO_i(Y, Y')$. Let $a_s = e \cap C_{i+1}$, $e \in A_s$ and $a'_t = e' \cap C_{i+1}$, $e' \in A'_t$. For each $\rho \in \mathrm{Sym(C_{i+1})}$ the algorithm will do the following.

If $\{a_1, \cdots, a_k\}^\rho \neq \{a'_1, \cdots, a'_k\}$ then discard the ρ and move to the next permutation in $\mathrm{Sym(C_{i+1})}$. Otherwise let $a_j^\rho = a'_{\pi(j)}$ where the permutation $\pi : [k] \longrightarrow [k]$ is the corresponding mapping of indices from the $(i+1)$-blocks $A_1, \cdots, A_k$ to $A'_1, \cdots, A'_k$. Let the coset of isomorphisms $ISO_{i+1}(Z_j, Z'_{\pi(j)})$ be denoted by

$$H_{j\pi(j)}\sigma_{j\pi(j)} = ISO_{i+1}(Z_j, Z'_{\pi(j)}).$$

Now, applying Theorem 5.3.16 the algorithm can compute the intersection

$$H_\rho\sigma_\rho = \cap_{j=1}^{k} H_{j\pi(j)}\sigma_{j\pi(j)}.$$

As $k \leq 2^b$, notice that the algorithm of Theorem 5.3.16 will take time bounded by $2^{O(b)}N^{O(1)}$. Next the algorithm will paste these cosets for all the different $\rho \in \mathrm{Sym(C_{i+1})}$ to form a single coset

$$H\sigma = \cup_{\rho \in \mathrm{Sym(C_{i+1})}} H_\rho\sigma_\rho.$$

Consider $S \times H\sigma$. It is clear that S is a coset and $S \times H\sigma$ contains $ISO_i(Y, Y')$. Crucially, since A and A' are both (i)-blocks it follows that $ISO_i(Y, Y') = S \times H\sigma$. Thus, the algorithm has computed $ISO_i(Y, Y')$ and will store it in the table corresponding to the entry (A, A').

We analyze the running time at this stage. The number of blocks at any stage is bounded by the number of edges of X. Thus, the $(m-i+1)$th stage takes time bounded by $b!2^{O(b)}N^{O(1)}$, where the $b!$ factor is because we cycle through all the $\rho \in \mathrm{Sym(C_{i+1})}$.

Finally, in the mth stage the algorithm will process all the pairs of (1)-blocks in the same way as described above. In order to compute $\mathrm{Aut}(X)$ the algorithm does the following.

Let $A_1, \cdots, A_k$ be the (1)-blocks and let $Y_1, \cdots, Y_k$ respectively be the multihypergraphs induced on $C_1 \cup \cdots \cup C_m$. Let $a_j = e \cap C_1$ where $e \in A_j$. Notice that a permutation $\rho \in \mathrm{Sym(C_1)}$ may or may not induce a permutation on the set $\{a_1, \cdots, a_k\}$ depending on whether $\{a_1, \cdots, a_k\}^\rho = \{a_1, \cdots, a_k\}$ or not. If ρ induces a permutation, let $\pi \in S_k$ be the permutation, i.e., $a_j^\rho = a_{\pi(j)}$. Let $H_{j\pi(j)}\sigma_{j\pi(j)} = ISO_1(Y_j, Y_{\pi(j)})$. The algorithm then computes

$$H_\rho \sigma_\rho = \cap_{j=1}^{k} H_{j\pi(j)}\sigma_{j\pi(j)}.$$

Finally, the algorithm pastes these $H_\rho\sigma_\rho$ to get the automorphism group

$$\mathrm{Aut}(X) = \cup_{\rho \in \mathrm{Sym(C_1)}} H_\rho \sigma_\rho.$$

It is easy to check that the algorithm takes time $b!2^{O(b)}N^{O(1)}$ time to compute Aut(X).

Theorem 5.4.2 *Let $X = (V, E)$ be a colored hypergraph with $V = C_1 \uplus \cdots \uplus C_m$ where C_i is colored with color i and $|C_i| \leq b$ for all i. Let N be the sum of number of vertices and edges of X. Given X as input there is an algorithm that computes* Aut(X) *in time $b!2^{O(b)}N^{O(1)}$.*

Next we indicate how the above algorithm can be easily modified to give isomorphisms algorithm for colored hypergraphs with same running time. Let $X = (V, E)$ and $X' = (V', E')$ be two colored hypergraphs. Without loss of generality we assume $V = V' = C_1 \uplus \cdots \uplus C_m$ where C_i is the ith color class. As before we partition the edges of both the hypergraphs into (i)-block for all i. Again we compute $ISO_i(Y, Y')$ where Y and Y' are multihypergraphs induced by (i)-block pair (A, A'), with the only difference that the block A comes form hypergraph X and A' form X'. In the final step we compute the set of isomorphism $ISO(X, X')$ from X to X'. Thus we have

Theorem 5.4.3 *Let $X = (V, E)$ and $X' = (V, E')$ be two colored hypergraphs with $V = C_1 \uplus \cdots \uplus C_m$ where C_i is colored with color i and $|C_i| \leq b$ for all i. Let N be the sum of number of vertices and edges of X. Given X and X' as input there is an algorithm that computes the set $ISO(X, X')$ of isomorphism form X to X' in time $b!2^{O(b)}N^{O(1)}$.*

6 Space Complexity of K-tree Isomorphism and Canonization

6.1 Introduction

It often turns out that NP-hard graph problems when restricted to the class of *partial k-trees* for constant k have efficient polynomial-time algorithms [33, 95]. Partial k-trees are also known as the class of graphs of treewidth k. For constant k, in general, they are known as *bounded treewidth graphs* (formal definitions are given in Section 6.3). Bodlaender [34] gave the first polynomial-time algorithm for testing the isomorphism of partial k-trees. Bodlaender's algorithm, based on dynamic programming, runs in time $O(n^{k+4.5})$.

Our interest is in a *complexity-theoretic* characterization of Graph Isomorphism for partial k-trees using space bounded complexity classes. We explain our motivation for studying the space complexity of k-tree isomorphism. On the one hand, we have Lindell's result [79, 69] that tree canonization is complete for deterministic logspace,[1] which tightly characterizes the complexity of both isomorphism and canonization of trees. What about partial k-tree isomorphism? The recent TC^1 upper bound for isomorphism of partial k-trees by Grohe and Verbitsky [60] raises the question about a tight complexity-theoretic classification of the problem. It is tempting to conjecture that partial k-tree isomorphism should also be complete for deterministic logspace, just like ordinary tree isomorphism. However, the best known complexity bound for even rec-

[1]Provided that the tree is given in the pointer notation; using the parenthesis notation the problem is NC^1-complete [37, 69].

ognizing partial k-trees is LOGCFL (the class of decision problems that are logspace many-one reducible to CFLs) [110].

The TC^1 bound of [60] suggests that we can put the problem in a natural complexity class contained in TC^1 like LOGCFL or DET, or perhaps somewhere in the logspace counting hierarchy. The logspace counting classes, introduced in the seminal paper [7], contain many natural problems sitting in NC^2 and have been used to characterize most natural problems in NC^2 satisfactorily from a complexity-theoretic viewpoint. A comprehensive study can be found in Allender's survey article [5].

The content of this chapter is based on [12]. In this chapter we show that *full* k-tree canonization is in FL$^{\mathrm{NL}}$. Recall that the *canonization problem* for graphs is to produce a *canonical form* $\text{CF}(X)$ for a given graph X such that $\text{CF}(X)$ is isomorphic to X and $\text{CF}(X_1) = \text{CF}(X_2)$ for any pair of isomorphic graphs X_1 and X_2. It is easy to see that Graph Isomorphism is AC^0 reducible to Graph Canonization. However, in general it is not known if the two problems are even polynomial-time equivalent.

Interestingly, the NL oracle required for k-tree canonization is a language computed by an NL machine M that is *strongly unambiguous*: for any two configurations x and y of machine M there is at most one computation path from x to y. The class of languages accepted by such NL machines is denoted $\mathrm{StUSPACE}(\log n)$ (StUL for short) by Allender and Lange [6]. As shown in [6], StUL is in fact contained in DSPACE$(\log^2 n/\log\log n)$, improving the DSPACE$(\log^2 n)$ bound given by Savitch's theorem. Furthermore, the complexity class StUL is closed under complementation and even closed under logspace Turing reductions [36, Corollary 15]. Thus, it follows that k-tree isomorphism is in StUL. The class StUL is not known to be contained in L. In fact, it contains the class ULIN of unambiguous linear languages [36] which is not known to be in L.

Recently, based on similar approach it has been shown that k-tree isomorphism is in L [74]. In Section 6.5 we briefly sketch their algorithm.

We note that parallel algorithms are known for k-tree isomorphism. For example, in [59] a processor efficient AC^2 algorithm was given for k-tree isomorphism. Since $\mathrm{StUL} \subseteq \mathrm{UL} \subseteq$ NL $\subseteq \mathrm{AC}^1$, our upper bound is tighter than previously known bounds from a complexity-theoretic perspective.

We also look into the problem of canonizing k-paths, a special case of k-trees, and give a logspace canonization algorithm for k-paths. We show that k-path isomorphism is complete for L.

6.2 Preliminaries

In this chapter, for a graph X, let $V(X)$ denote its vertex set and $E(X)$ denote its edge set. The set $\{w \in V(X) \mid \{v, w\} \in E(X)\}$ of all *neighbors* of $v \in V(X)$ is denoted by $N(v)$. For a subset $U \subseteq V(X)$, we use $X[U]$ to denote the *induced subgraph* of X, where $V(X[U]) = U$ and $E(X[U]) = \{e \in E(X) \mid e \subseteq U\}$.

Next we recall some complexity classes defined by circuits and some space bounded classes.

A language A is in the complexity class NC^i (resp. AC^i) if there is a uniform family of circuits $\{C_n\}_n$ of depth $O(\log^i n)$ and size $poly(n)$ with internal AND, OR and NOT gates with bounded (resp. unbounded) fan-in that accepts A. TC^i is the extension of AC^i where we additionally allow unbounded $MAJORITY$ gates.

The complexity class L consists of all languages A accepted by a deterministic $O(\log n)$ space bounded Turing machine. NL is defined in the same way by using non-deterministic machines. FL is the class of all functions computable by a deterministic $O(\log n)$ space bounded Turing machine.

A nondeterministic Turing machine M is called *unambiguous*, if for any input x, it has at most one accepting computation path.

M is said to be *reach-unambiguous* if it is unambiguous and for any input x, there is at most one computation path from the starting configuration to any other configuration.

M is said to be *strongly unambiguous* if it is unambiguous and for any pair of configurations u and v of M there is at most one computation path from u to v.

A *mangrove* is a directed acyclic graph such that there is at most one directed path from i to j for any two nodes i and j in the graph. In other words, a directed graph is a mangrove if and only if for any node u the subgraph induced by u and all nodes reachable from u is a rooted directed tree.

Note that an unambiguous machine M is strongly unambiguous if and only if its configuration graph is a mangrove.

A language A is in the class UL (RUL, StUL) if there is an $O(\log n)$ space bounded unambiguous (reach-unambiguous, strongly unambiguous, respectively) Turing machine accepting A. It is well known that

$$\mathrm{NC}^1 \subseteq \mathrm{L} \subseteq \mathrm{StUL} \subseteq \mathrm{RUL} \subseteq \mathrm{UL} \subseteq \mathrm{NL} \subseteq \mathrm{AC}^1 \subseteq \mathrm{TC}^1 \subseteq \mathrm{NC}^2.$$

The following result of Allender and Lange [6] shows that Savitch's $\log^2 n$ space deterministic simulation of NL can be improved for StUL and RUL.

Theorem 6.2.1 [6] $\mathrm{StUL} \subseteq \mathrm{RUL} \subseteq$ DSPACE$(\log^2 n/\log\log n)$.

6.3 k-Tree Canonization

We define the classes of k-trees and partial k-trees in the following definition (see [65]).

Definition 6.3.1 *The class of* k-trees *is inductively defined as follows.*

- *A clique with k vertices (k-clique for short) is a k-tree.*
- *Given a k-tree X' with n vertices, a k-tree X with $n+1$ vertices can be constructed by introducing a new vertex v and picking a k-clique C in X' and then joining v to each vertex u in C. Thus, $V(X) = V(X') \cup \{v\}$, $E(X) = E(X') \cup \{\{u, v\} \mid u \in C\}$.*

A partial k-tree *is a subgraph of a k-tree.*

Before we go into the k-tree canonization we notice that the following characterization of k-trees gives a logspace algorithm for recognizing k-trees.

Definition 6.3.2 [72] *Let $X = (V, E)$ be a graph. A subset S of V is called a* vertex separator *for two nonadjacent vertices $u, v \in V$, if in the subgraph of X induced by the vertex set $V - S$ the two vertices u, v are in different connected components. A vertex separator S for u, v is called* minimal*, if no proper subset of S is a vertex separator for u and v. A subset $S \subseteq V$ is a* minimal vertex separator *if S is a minimal vertex separator for some pair of vertices $u, v \in V$.*

Lemma 6.3.3 [40] *A graph X with $n > k$ vertices is a k-tree if and only if*

- *every pair of nonadjacent vertices u and v has a k-clique as a minimal vertex separator and*
- *$E(X)$ contains exactly $\binom{k}{2} + k(n-k)$ edges.*

It is easy to see that the two conditions of Lemma 6.3.3 can be checked in logspace. Hence, from now on we can assume that the input graph X is a k-tree. Further we assume that $V(X) = \{1, \ldots, n\}$.

Our algorithm for k-tree canonization works by reducing the problem to the problem of canonizing certain labeled trees that encode essential information about k-trees. Our initial goal is to define this labeled tree. For this we use the concept of the layer decomposition of a k-tree with respect to a base B. This concept was introduced in [71] for testing isomorphism in hookup classes. Subsequently, it was used in [39, 59] for the design of efficient k-tree isomorphism algorithms.

Definition 6.3.4 (cf. [71, 59]) *Let $X = (V, E)$ be a k-tree and let $B \subseteq V$ be a k-clique in X. Then the B-*decomposition *of G is the sequence of sets $B(0), \ldots, B(p)$ such that $B(0) = B$ and $p = \max\{i \geq 0 \mid B(i) \neq \emptyset\}$, where $B(i+1)$ is inductively defined by*

$$B(i+1) = \{v \in V - B[i] \mid N(v) \cap B[i] \text{ is a } k\text{-clique}\}.$$

Here, $B[i]$ denotes the union $B[i] = B(0) \cup \cdots \cup B(i)$.

The set $B(i)$ is called the ith layer of the B-decomposition of X. Intuitively, the layers of the B-decomposition indicate the order in which vertices could be added to X when we choose B as the initial k-clique. More precisely, starting with the k-tree $X_0 = X[B]$, $X_{i+1} = X[B[i+1]]$ can be constructed from $X_i = X[B[i]]$ by adding the vertices in $B(i+1)$ to X_i. Recall that v can be added to X_i if and only if the set $N(v) \cap B[i]$ of v's neighbors in $B[i]$, henceforth denoted by $N_i(v)$, induces a k-clique in X_i. In [71], this set $N_i(v)$ is called the *support* of $v \in B(i)$).

If this process is successful, i.e., if each vertices of X is covered by some layer $B(i)$, then B is called a *base* of X (cf. [71]).

We first show that any k-clique B in X can be used as a base for constructing X (see Lemma 6.3.8).

Definition 6.3.5 *A vertex v of a k-tree X is called* simplicial, *if $N(v)$ induces a k-clique in X.*

Claim 6.3.6 *Any k-tree X with $n \geq k+2$ vertices has at least two nonadjacent simplicial vertices.*

Proof. The proof is by induction on n. If $n = k + 2$, then X is obtained from a $(k + 1)$-clique X' by choosing a k-clique C in X' and introducing a new node v which is joined to each vertex in C. Let u be the unique vertex in X' not covered by C. Then u and v are two nonadjacent simplicial vertices in X. For the inductive step assume that X has $n > k + 2$ vertices. Then X has been obtained from a k-tree X' by introducing a new node v and joining it to each vertex in a k-clique C of X'. Clearly, v is simplicial in X. Further, by the induction hypothesis, X' has two nonadjacent simplicial vertices u_1 and u_2. Since u_1 and u_2 are nonadjacent, at least one of them does not belong to C and therefore it is also simplicial in X. ■

Claim 6.3.7 *Let B be a k-clique of a k-tree X with $n \geq k + 1$ vertices. Then X has a simplicial vertex $v \notin B$.*

Proof. If $n = k + 1$, then the only vertex not in B is simplicial. If $n > k + 1$, then Claim 6.3.6 guarantees the existence of two nonadjacent simplicial vertices that cannot be both in B. ■

Lemma 6.3.8 *For any k-tree $X = (V, E)$ and any k-clique B, the B-decomposition forms a partition of V.*

Proof. The proof is by induction on n. The base case $n = k$ is clear. For the inductive step assume that $n \geq k + 1$ and let $B(0), \ldots, B(p)$ be the B-decomposition of X. By Claim 6.3.7, X has a simplicial vertex v not in B. It is easy to prove that $X - v$ is a k-tree and hence, by the induction hypothesis, the B-decomposition $B'(0), \ldots, B'(p')$ of $X - v$ forms a partition of $V - \{v\}$. Now let $i \geq 0$ be the minimum integer such that $N(v) \subseteq B'[i]$. Then it follows that $B(i + 1) = B'(i + 1) \cup \{v\}$ and $B(j) = B'(j)$ for all $j \neq i + 1$, implying that $V = B[p]$. ■

The following properties of the B-decomposition have been proved in [71].

Proposition 6.3.9 *If B is a base for a k-tree $X = (V, E)$, then the B-decomposition $B(0), \ldots, B(p)$ has the following properties.*

1. *Any two vertices in $B(i)$, $i \geq 1$, are nonadjacent. Hence, $N_{i-1}(v) = N_i(v)$ for any vertex $v \in B(i)$.*

2. *Any vertex $v \in B(i)$, $i \geq 2$, has a unique neighbor $f(v) \in B(i-1)$, called the* father *of v w.r.t. B.*

In order to efficiently compute information on the B-decomposition of a k-tree X we use a directed graph $D(X, B)$ which is defined as follows (whenever X and B are clear from the context we simply write D instead of $D(X, B)$). D has the vertex set

$$V(D) = \{B\} \cup \{(C, v) \mid v \notin C \text{ and } C \cup \{v\} \text{ is a } (k+1)\text{-clique in } X\}$$

and the vertices of D are joined by the directed edges in the set

$$\begin{aligned} E(D) = \{(B, (B, v)) \mid (B, v) \in V(D)\} \cup \\ \{((C, v), (C', v')) \mid v \in C', v' \notin C, |C \cap C'| = k-1\}. \end{aligned}$$

This means that in D we provide a transition from (C, v) to (C', v') if C' can be obtained from C by replacing some vertex $u \in C$ by v, i.e., $C' = (C - \{u\}) \cup \{v\}$. Our next aim is to show that D is a mangrove (see Lemma 6.3.12).

Claim 6.3.10 *For any vertex $v \in B(i)$, $i \geq 1$, D has a directed path of length i from B to $(N_{i-1}(v), v)$.*

Proof. We prove the claim by induction on i. The base case $i = 1$ is clear. For the inductive step assume that $v \in B(i)$, $i \geq 2$ and let $f(v) \in B(i-1)$ be the father of v. By the induction hypothesis it follows that D has a directed path of length $i-1$ from B to $(N_{i-2}(f(v)), f(v))$. Clearly, $f(v) \in N_{i-1}(v)$ and $v \notin N_{i-2}(f(v))$. Further, since $f(v)$ is the only vertex in $N_{i-1}(v)$ belonging to $B(i-1)$, the remaining $k-1$ vertices belong to $B[i-2]$ and, as they are also neighbors of $f(v)$, they belong to the support $N_{i-2}(f(v))$ of $f(v)$. This shows that D has an edge from $(N_{i-2}(f(v)), f(v))$ to $(N_{i-1}(v), v)$. ∎

Claim 6.3.11 *If D has a directed path $B, (C_1, v_1), \ldots, (C_{i-1}, v_{i-1}), (C, v)$ of length $i \geq 1$ from B to some vertex (C, v), then $v \in B(i)$ and $C = N_{i-1}(v) \subseteq B \cup \{v_1, \ldots, v_{i-1}\}$.*

Proof. Again the proof is by induction on i. If $E(D)$ contains the edge $(B, (B, v))$, then clearly $v \in B(1)$ via the support $N_0(v) = B$.

For the inductive step let us assume that $B, (C_1, v_1), \dots, (C_{i-1}, v_{i-1}), (C, v)$ is a directed path of length $i \geq 2$ from B to (C, v). Then by the induction hypothesis it follows that $v_{i-1} \in B(i-1)$ via the support $C_{i-1} = N_{i-2}(v_{i-1}) \subseteq B \cup \{v_1, \dots, v_{i-2}\}$. As $N_{i-2}(v_{i-1}) = N_{i-1}(v_{i-1})$ by part 1 of Proposition 6.3.9, this implies that C_{i-1} contains all neighbors of v_{i-1} in $B[i-1]$. Since v is a neighbor of v_{i-1} that does not belong to C_{i-1}, v cannot be in $B[i-1]$. As $((C_{i-1}, v_{i-1}), (C, v)) \in E(D)$, it follows that C is obtained from C_{i-1} by replacing some vertex u in C_{i-1} by v_{i-1}, i.e., $C = (C_{i-1} - \{u\}) \cup \{v_{i-1}\} \subseteq B \cup \{v_1, \dots, v_{i-1}\}$. Hence, all vertices in C belong to $B[i-1]$ and are adjacent to v, implying that $v \in B(i)$ via the support $N_{i-1}(v) = C$ (notice that $C \subsetneq N_{i-1}(v)$ would imply $v \notin B[p]$). ■

Lemma 6.3.12 *For any k-clique B in a k-tree X, the graph $D(X, B)$ is a mangrove.*

Proof. We first show that $D = D(X, B)$ does not have different paths from B to the same node (C, v).

By Claim 6.3.11, all paths from B to (C, v) have the same length. In order to derive a contradiction let i be minimal such that there are two different paths

$$B, (C_1, v_1), \dots, (C_{i-1}, v_{i-1}), (C, v) \text{ and,}$$

$$B, (C'_1, v'_1), \dots, (C'_{i-1}, v'_{i-1}), (C, v)$$

of length i from B to some node (C, v). Then v_{i-1} and v'_{i-1} must be different, since otherwise Claim 6.3.10 implies that $C_{i-1} = N_{i-2}(v_{i-1}) = N_{i-2}(v'_{i-1}) = C'_{i-1}$, contradicting the minimality of the path length i. But now Claim 6.3.11 implies that v_{i-1} and v'_{i-1} both belong to $B(i-1)$ as well as to the support C of v, contradicting part 2 of Proposition 6.3.9.

To complete the proof suppose there are different directed paths between two nodes (C, v) and (C', v') in $D(X, B)$. Then we would also have different directed paths between the two nodes C and (C', v') in $D(X, C)$, contradicting the argument above. ■

Now let $T = T(X, B)$ be the subgraph of $D(X, B)$ induced by the vertices reachable from B. Then Lemma 6.3.12 implies that T is a directed rooted tree with root B.

In fact, from Claims 6.3.10 and 6.3.11 it is immediate that by projecting the first component out from the nodes $(C, v) \in V(T)$ we get exactly the tree $T(X)$ defined in

[71]. There, the following labeling with respect to a bijection $\theta : B \to \{1, 2, \ldots, k\}$ has been defined.

Let (C, v) be a node in T. W.l.o.g. suppose that $C = \{v_1, \ldots, v_k\}$ where $C \cap B = \{v_1, \ldots, v_m\}$ for some $m \leq k$. Notice that by part 1 of Proposition 6.3.9, the $k - m$ vertices in $C - B$ belong to $k - m$ different layers $B(i_1), \ldots, B(i_{k-m})$. Then vertex (C, v) is labeled by the set $\{\theta(v_1), \ldots, \theta(v_m), k + i_1, \ldots, k + i_{k-m}\}$.

We denote the tree T together with this labeling by $T(X, B, \theta)$. The following theorem is due to [71].

Theorem 6.3.13 *Let X and X' be two k-trees, let B be a base for X and let $\theta : B \longrightarrow \{1, \ldots, k\}$ be a bijection. Then X and X' are isomorphic if and only if there exists a base B' for X' and a bijection $\theta' : B' \to \{1, \ldots, k\}$ such that the two labeled trees $T(X, B, \theta)$ and $T(X', B', \theta')$ are isomorphic.*

The proof of Theorem 6.3.13 crucially hinges on the fact that each isomorphic copy T' of the labeled tree $T(X, B, \theta)$ provides enough information to reconstruct X from T' up to isomorphism. To see why, for $i \geq 1$ let B_i be the set of vertices of T' that have distance i from the root of T' and let p be the maximum distance of any vertex in T' from the root. Then starting with a k-clique X_0 we can successively add in parallel all the vertices $v \in B_i$ to X_{i-1} for $i = 1, \ldots, p$. The crucial observation is that the labeling $\{\theta(v_1), \ldots, \theta(v_m), k + i_1, \ldots, k + i_{k-m}\}$ of the node v in T' tells us to which vertices in X_{i-1} vertex v should be connected (recall that Claim 6.3.11 guarantees that all vertices in the support of a node either belong to the base or lie on the path from the root to that vertex in the corresponding tree).

To canonize k-trees we use Lindell's [79] deterministic logspace canonization algorithm for trees which can be made to work for any labeled tree by constructing gadgets for labels. More precisely, consider the algorithm A that on input a k-tree X computes the canon of all labeled trees $T(X, B, \theta)$ for all k-cliques B in X and all bijections $\theta : B \to \{1, \ldots, k\}$ and picks the lexicographically least among them. Then Theorem 6.3.13 implies that

- if two k-trees X and H are isomorphic then any tree of the form $T(X, B, \theta)$ is isomorphic to some tree of the form $T(H, B', \theta')$ and
- if X and H are non-isomorphic then no tree of the form $T(X, B, \theta)$ is isomorphic to some tree of the form $T(H, B', \theta')$.

Hence, A outputs the same tree T for both k-trees X and H if and only if X and H are isomorphic, implying that A computes a complete invariant for k-trees.

Furthermore, as explained above, the output tree T of A on input X provides enough information to reconstruct X from T in logspace up to isomorphism. The combination of A with this reconstruction procedure thus yields the desired canonization algorithm A' for k-trees. It remains to show that A can be implemented in $\mathrm{FL}^{\mathrm{StUL}}$. In the next lemma we show that the labeled trees $T(X, B, \theta)$ can be computed in logspace relative to some oracle in StUL. The following claim provides this oracle.

Claim 6.3.14 *The problem of deciding whether a vertex (C, v) of D has distance i from B is in StUL.*

Proof. The algorithm tries to guess a path of length i from B to (C, v) in the tree $T = T(X, B)$. For that, starting with vertex B, it iteratively guesses a next node (C', v') and checks if T provides an edge from the actual node to that node. If after i steps the algorithm reaches (C, v) then it accepts, otherwise it rejects. Clearly, the algorithm runs in logspace since it has to store only two nodes of T and some counters. Since $D(X, B)$ is a mangrove by Claim 6.3.12, it is easy to see that the configuration graph is also a mangrove. ∎

Lemma 6.3.15 *On input a k-tree X, a k-clique B and a bijection $\theta : B \to \{1, \ldots, k\}$, the labeled tree $T(X, B, \theta)$ can be computed in logspace relative to some oracle in StUL.*

Proof. The algorithm for generating $T = T(X, B, \theta)$ first outputs $V(T)$ by checking for each node (C, v) in $V(D)$ whether it is reachable from B by using the StUL oracle of Claim 6.3.14. If so, it computes the label of (C, v) by recomputing the layer numbers of all the vertices in C (again using the StUL oracle). Finally, for each distinct pair of nodes in $V(T)$ it checks whether D provides a directed edge between them. ∎

This shows that the algorithm A described above can indeed be implemented in logspace relative to some oracle in StUL. Hence, we can state our main result.

Theorem 6.3.16 *For each fixed k there is a canonizing algorithm for k-trees that runs in $\mathrm{FL}^{\mathrm{StUL}}$.*

As StUL is closed under logspace Turing reductions [36, Corollary 15], we immediately get the following complexity upper bound for testing isomorphism for k-trees.

Corollary 6.3.17 *The isomorphism problem for k-trees is in* StUL*.*

Let X be a k-tree. Let $\phi : V(X) \longrightarrow C$ be a coloring of X, where C is a set of colors. Let B be a k-clique of X and $\theta : B \longrightarrow \{1, \cdots, k\}$ be a bijection. We define a rooted labeled tree $T = T(X, \phi, B, \theta)$ as follows. The vertices and edges of $T(X, \phi, B, \theta)$ are same as of $T(X, B, \theta)$. The only difference between $T(X, B, \theta)$ and $T(X, \phi, B, \theta)$ is in the labelling. The root of $T(X, \phi, B, \theta)$ is labeled by the set $\{(\theta(v), \phi(v)) \mid v \in B\}$. Let (C, v) be a node in T. Let $C = \{v_1, \ldots, v_k\}$ where $C \cap B = \{v_1, \ldots, v_m\}$ for some $m \leq k$. Suppose the $k - m$ vertices in $C - B$ belong to $k - m$ different layers $B(i_1), \ldots, B(i_{k-m})$. Then vertex (C, v) is labeled by the tuple

$$(\{(\theta(v_1), \phi(v_1)), \cdots, (\theta(v_m), \phi(v_m)), i_1, \cdots, i_{k-m}\}, \phi(v)).$$

Theorem 6.3.18 *Let $X = (V, E)$ and $X' = (V', E')$ be two k-trees. Let $\phi : V \longrightarrow C$ and $\phi' : V' \longrightarrow C$ be two colorings of the graphs X and X' respectively. Let B be a k-clique of X and $\theta : B \longrightarrow \{1, \cdots, k\}$ be a bijection. Then X and X' are isomorphic as colored graphs if and only if there is a k-clique B' and bijection $\theta' : B' \longrightarrow \{1, \cdots, k\}$ such that $T(X, \phi, B, \theta)$ is isomorphic to $T(X', \phi', B', \theta')$.*

Proof. If X and X' are isomorphic as colored graphs via the isomorphism

$$\tau : V(X) \longrightarrow V(X')$$

then it is easy to check that $T(X, \phi, B, \theta)$ is isomorphic to $T(X', \phi', \tau(B), \theta')$ where $\theta' : \tau(B) \longrightarrow \{1, \cdots, k\}$ is the bijection which satisfies $\theta'(v') = \theta(\tau^{-1}(v'))$ for all $v' \in \tau(B)$.

For the other direction suppose $T = T(X, \phi, B, \theta)$ and $T' = T(X', \phi', B', \theta')$ are isomorphic for some k-clique B' and bijection $\theta' : B' \longrightarrow \{1, \cdots, k\}$ via the isomorphism $\overline{\tau}$. We define a map $\tau : V(X) \longrightarrow V(X')$ as follows. Let (C, v) be a vertex of $T(X, \phi, B, \theta)$. Let $\overline{\tau}((C, v)) = (C', v')$. Then define $\tau(v) = v'$. Let $v \in B$. The root of T will have $(\theta(v), \phi(v))$ in its label. As T and T' are isomorphic their their roots get same color. So, there will be some vertex $v' \in B'$ such that $(\theta'(v'), \phi'(v')) = (\theta(v), \phi(v))$. Then define $\tau(v) = v'$. We will prove by induction on the number n of the vertices of X that if T and T' are isomorphic then τ is an isomorphism between X and X' that preserves color.

The base case ($n = k$) is easy. So, we assume $n > k$. As the height of the tree corresponds to the number of layers 6.3.10 and since $T = T(X, \phi, B, \theta)$ is isomorphic to $T' = T(X', \phi', B', \theta')$, X and X' will have same number p of layers with respect to B and B' respectively. Let $B(0), \cdots, B(p)$ and $B'(0), \cdots, B'(p)$ be the layers of X and X' respectively. Let $v \in B(p)$ be a simplicial vertex in X. Let (C, v) be the node containing v in T. Let $\overline{\tau}((C, v)) = (C', v')$. It is clear that $T(X - \{v\}, \phi \mid_{V-\{v\}}, B, \theta) = T(X, \phi, B, \theta) - \{(C, v)\}$ will be isomorphic to $T(X' - \{v'\}, \phi' \mid_{V'-\{v'\}}, B', \theta') = T(X', \phi', B', \theta') - \{(C', v')\}$ via $\overline{\tau}$ restricted to the nodes of $T(X - \{v\}, \phi \mid_{V-\{v\}}, B, \theta)$. So, by induction hypothesis $\tau \mid_{V-\{v\}}$ will be an isomorphism between $X - \{v\}$ and $X' - \{v'\}$. Now, $\tau(v) = v'$ by the definition of τ (remember, $\overline{\tau}((C, v)) = (C', v')$). Notice that $\phi(v) = \phi(v')$. Thus τ preserves color, as by induction hypothesis $\tau \mid_{V-\{v\}}$ preserves color.

Let $\{v, w\} \in E$. By induction hypothesis, it is enough to prove $\{\tau(w), \tau(v)\} \in E'$. Suppose $w \in B(0) = B$. Then as v is simplicial $w \in C$. Hence (C, v) gets $(\theta(w), \phi(w))$ in its label. The node (C', v') can get this label only from $\tau(w)$. Thus, $\tau(w) \in C'$ and $\{\tau(w), \tau(v)\} = \{\tau(w), v'\}\} \in E(X')$.

Let $w \in B(i)$, where $i > 0$. Let $u \in B(p-1)$ be the father of v in the B-decomposition of X. Let (C_u, u) be the node in T corresponding to u. Similarly, let $u' \in B'(p-1)$ be the father of v' in the B'-decomposition of X' and $(C_{u'}, u)$ be the node in T' corresponding to u'. As $\overline{\tau}((C, v)) = (C', v')$, the isomorphism $\overline{\tau}$ must also map the only neighbor (C_u, u) of (C, v) to the only neighbor $(C_{u'}, u')$ of (C', v'), i.e., $\overline{\tau}((C_u, u)) = (C_{u'}, u')$. Thus, $\tau(u) = u'$. If $w = u$ then, by the above reasoning $\{\tau(v), \tau(w)\} = \{v', u'\} \in E'$. If $w \neq u$ then $w \in B(i)$ where $0 < i < p-1$. Let (C_w, w) be the node of T that corresponds to w. Let $\overline{\tau}((C_w, w)) = (C_{w'}, w')$. So, $\tau(w) = w'$. As $\overline{\tau}$ is an isomorphism between T and T', the distance of (C_w, w) from the root of T must be same as the distance of $(C_{w'}, w')$ from the root of T'. Thus by Claim 6.3.10 and 6.3.11 $w' \in B'(i)$. Notice that as $\{w, v\}, \{u, v\} \in E$ and v is simplicial, $\{u, w\} \in E$. Hence by induction hypothesis $\{\tau(u), \tau(w)\} = \{u', w'\} \in E'$. As v has the neighbor w in ith layer (C, v) gets i in its label. Thus, the image (C', v') of (C, v) under $\overline{\tau}$ must also get i in its label. So there must be some $w'' \in B'(i)$ which is adjacent to v'. But w'' must be adjacent to u' also, because v' is simplicial and u' is adjacent to v'. By Proposition 6.3.9, the vertex u' cannot have two neighbors w' and

w'' in the same layer i. So we must have $w' = w''$. Hence $\{\tau(v), \tau(w)\} = \{v', w'\} = \{v', w''\} \in E'$.

■

Theorem 6.3.19 *Let X be a k-tree and $\phi : V(X) \longrightarrow C$ be a coloring of the vertices of X. Then the canonical form of the colored graph X can be computed in* $\mathrm{FL}^{\mathrm{StUL}}$.

Proof. The algorithm A to canonize a colored k-tree again uses Lindell's deterministic logspace algorithm to canonize labeled trees, $T(X, \phi, B, \theta)$ for all k-cliques B of X and all bijections $\theta : B \longrightarrow \{1, \cdots, k\}$ and take the least among them. As a consequence of Theorem 6.3.18, this will give complete invariant. But form the complete invariant the canon of the colored graph X can be computed in logspace. The only question is how to produce $T(X, \phi, B, \theta)$ form the colored graph X. But this problem is again reduces to deciding the layer number of a vertex v in X in the B-decomposition of X. The layer number of v can be found in StUL as in Claim 6.3.14. The labeled tree $T(X, \phi, B, \theta)$ can be found in a similar way as in Lemma 6.3.15. ■

An *automorphism* of a graph $X = (V, E)$ (possibly colored) is a bijection $\xi : V \longrightarrow V$ that is an isomorphism for X to itself (in case of colored graphs the isomorphism is color preserving). An automorphism ξ is nontrivial if for some vertex v of X, $\xi(v) \neq v$. If X has no nontrivial automorphism then the graph is called *rigid*. It is easy to see that the set of all automorphisms of a graph X forms a group under composition. We denote this group by $Aut(X)$. Next we list some problems related to automorphism.

Definition 6.3.20 (GA**)** Input: *A graph $X = (V, E)$.*

Problem: *Decide if X has a nontrivial automorphism.*

Definition 6.3.21 (SAuto**)** Input: *A graph $X = (V, E)$ and a list of pairs of vertices $(u_1, v_1), \cdots, (u_l, v_l)$.*

Problem: *Decide if X has an automorphism ξ such that $\xi(u_i) = v_i$ for $i = 1, \cdots, l$.*

Definition 6.3.22 (AUTO**)** Input: *A graph $X = (V, E)$.*

Problem: *Compute a generating set for the automorphism group $Aut(X)$ of X.*

Definition 6.3.23 (SAUTO**)** Input: *A graph $X = (V, E)$ and a list of pairs of vertices $(u_1, v_1), \cdots, (u_l, v_l)$.*

Problem: *Compute an automorphism ξ of X such that $\xi(u_i) = v_i$ for $i = 1, \cdots, l$, if there is any such automorphism, otherwise output "does not exist".*

Let $X = (V = \{1, \cdots, n\}, E)$ be a graph. Observer that GA $\leq_T^L$ SAuto, since a logspace machine can query the oracle for SAuto with (X, i, j) for all $1 \leq i < j \leq n$ and accept if any of the query is answered yes.

Lemma 6.3.24 SAuto $\leq_T^L$ AUTO.

Proof. Let $X = (V, E)$ be a graph. We first show SAuto $\leq_T^L$ AUTO. Let

$$(X, ((u_1, v_1), \ldots, (u_l, v_l)))$$

be an instance of SAuto. We form a colored graph Y by taking two disjoint copies of X. In the first copy we color vertex u_i using color c_i, and in the second copy we color vertex v_i using color c_i for $i = 1, \ldots, l$, where the colors $c_1, \ldots, c_l$ are distinct. We can now query AUTO for input Y to get a generating set for Aut(Y). By construction, $(X, ((u_1, v_1), \ldots, (u_l, v_l)))$ is a yes instance of SAuto if and only if there is a generator mapping u_i to v_i for $i = 1, \ldots, l$. A logspace machine can easily check for this by examining the generators.

■

Lemma 6.3.25 AUTO $\leq_T^L$ SAUTO.

Proof. Let $X = (V = \{1, \cdots, n\}, E)$ be a graph. Let $G = \text{Aut}(X)$ be the automorphism group of X. Let $G^{(i)}$ denote the subgroup of G which fixes the vertices $1, \cdots, i$, i.e., each element g in $G^{(i)}$ satisfies the property $j^g = j$ for all $j = 1, \cdots, i$. Clearly $G^{(i)}$ is a subgroup of $G^{(i-1)}$. Thus we get a tower of subgroups

$$< id >= G^{(n-1)} \leq \cdots \leq G^{(i)} \leq G^{(i-1)} \leq \cdots \leq G^{(1)} \leq G^{(0)} = G$$

where id is the identity permutation. A logspace machine starts with $G^{(n-1)}$ which is identity. Suppose the machine has output a generating set $S^{(i)}$ for $G^{(i)}$. We describe how the machine will augment $S^{(i)}$ with the coset representatives of the cosets of $G^{(i)}$

in $G^{(i-1)}$. Each coset of $G^{(i)}$ in $G^{(i-1)}$ corresponds to a vertex j where i is moved all the permutation in that coset. Let $g_1G^{(i)}, \cdots, g_mG^{(i)}$ be the cosets of $G^{(i)}$ in $G^{(i-1)}$. Notice $G^{(i-1)} = G^{(i)} \cup g_1G^{(i)} \cup \cdots \cup g_mG^{(i)}$. To find the coset representatives the oracle for SAUTO can be queried with the instance $(X, (1,1), \cdots, (i-1, i-1), (i,j))$ for all $j > i$. Let for $j = j_1, \cdots, j_m$ the oracle outputs the permutations $\tau_1, \cdots, \tau_m$. (For some query the oracle may output "does not exists"). The generating set for $G^{(i-1)}$ will be $S^{(i)} \cup \{\tau_1, \cdots, \tau_m\}$. Whenever a query is answered by a permutation τ, the permutation τ is written on the output tape by the machine. The machine can thus compute Aut(X) by outputting the coset representatives of $G^{(i)}$ in $G^{(i-1)}$ for $i = n-1, \cdots, 1$. ■

The set of all isomorphisms form X to CF(X) is called *Canonical Labelling Coset* CL. It can be checked that the CL is of the form Aut$(X)\xi$, where ξ is some isomorphism from X to CF(X). So, the canonical labeling coset CL of a graph X can be computed easily if some member ξ of CL and the automorphism group Aut(X) is known.

Definition 6.3.26 (CL-Coset**)** Input: *A graph* $X = (V, E)$.

Problem: *Compute the canonical labeling coset* CL *of* X.

Definition 6.3.27 (COLOR-CP**)** Input: *A graph* $X = (V, E)$ *be a colored graph.*

Problem: *Compute an isomorphism from* X *to* CF(X)*, i.e., compute a member of the canonical labeling coset of the colored graph* X.

Remark 6.3.28 *Notice that* COLOR-CP *is neither a language nor a (single valued) function (same input might have different output). It is basically a search problem. We subsequently assume that any oracle for* COLOR-CP *actually computes a* function*, i.e., an input has unique output.*

Next we show that a logspace machine can solve SAUTO if it has access to an oracle for COLOR-CP.

Lemma 6.3.29 SAUTO $\leq_T^L$ COLOR-CP.

Proof. Let $(X, ((u_1, v_1), \cdots, (u_l, v_l)))$ be a instance of SAUTO where $X = (V, E)$ is a graph and $u_1, \cdots, u_l, v_1, \cdots, v_l \in V$. Let X_1 be the graph X with $u_1, \cdots, u_l$ colored with colors $c_1, \cdots, c_l$ respectively. Similarly, let X_2 be the graph X with $v_1, \cdots, v_l$ colored with colors $c_1, \cdots, c_l$ respectively. A logspace machine queries the oracle for

COLOR-CP with X_1 and X_2 and checks if the two canonical forms are the same. Notice that it requires several queries to do this because the result of a query cannot be stored. If they two canonical forms are same then the machine proceeds to compute the required automorphism. Let ξ_1 be an isomorphism from the colored graph X_1 to its canonical form $\mathrm{CF}(X_1)$ and let ξ_2 be an isomorphism form X_2 to $\mathrm{CF}(X_2)$. Then it is clear that $\xi = \xi_1\xi_2^{-1}$ maps u_i to v_i for $i = 1, \cdots, l$. The logspace machine finds the image of all vertex under the map ξ. To know where a vertex u goes under the permutation ξ it first queries the oracle for COLOR-CP with X_1 and finds the placement $v' = \xi(u)$. Next it queries COLOR-CP with X_2 and finds the preimage $v = \xi_2^{-1}(v')$ of v'. It can then output the information that u is mapped to v under ξ and proceeds to find the image of the next vertex under ξ. ■

As AUTO $\leq_T^L$ SAUTO and SAUTO $\leq_T^L$ COLOR-CP we get the following.

Lemma 6.3.30 AUTO $\leq_T^L$ COLOR-CP.

If the canonical placement and the automorphism group of a graph can be computed the it readily gives a solution to CL-Coset. From Lemma 6.3.30 we an deduce the lemma.

Lemma 6.3.31 CL-Coset $\leq_T^L$ COLOR-CP.

Our algorithm A for canonizing colored k-trees can be easily modified to give an algorithm which gives the canonical placement as follows. While canonizing a color k-tree X the algorithm has to keep track of the vertices of X and the corresponding vertices of the labelled tree $T(X, \phi, B, \theta)$. Also, Lindell's deterministic algorithm for tree canonization can be easily modified to give a logspace algorithm for canonical placement of trees. In fact Lindell's algorithm implicitly computes the canonical placement of a tree. Hence by the above lemma we get the theorem.

Theorem 6.3.32 *For any fixed k, the problem* CL-Coset *for k-trees is in* StUL.

6.4 k-Path Canonization

A k-path is a special type of k-tree. The subgraphs of k-paths are called partial k-paths. They coincide with the graphs of *pathwidth* at most k [92]. In [61] a polynomial time algorithm for subgraph isomorphism for bounded pathwidth graphs was given. Here we look at the space complexity of the canonization problem for k-paths.

Definition 6.4.1 *An* interval graph *is a graph whose vertices can be put in one to one correspondence with a set of open intervals on the real line such that two vertices are adjacent if and only if the corresponding intervals have a nonempty intersection.*

Definition 6.4.2 [72] *A k-*path *is a k-tree which is an interval graph.*

An alternative constructive definition of k-paths is given in [61]. The idea is to restrict the choice of the k-clique used as support for adding a new vertex depending on the support of the previously added vertex. The restriction can be best described by maintaining the notion of *current clique*.

Initially the starting clique is the current clique. When a new vertex is added it is joined to each vertex in the current clique. After adding the new vertex the current clique may remain the same (in that case the new vertex added becomes simplicial) or it may change by dropping a vertex and adding the new vertex in the current clique. Clearly, when a vertex is dropped it cannot come back in the current clique.

The difference between the definition of k-tree and the constructive definition of k-path is that for k-trees a new vertex can be joined to any k-clique when expanding a k-tree, whereas for k-paths a new vertex can only be added to the current clique of a k-path.

From this constructive definition of k-paths the following characterization of k-paths in the terminology of Section 6.3 can be obtained. Recall that a *caterpillar* is a rooted tree in which each node has at most one child that is not a leaf.

Lemma 6.4.3 *A k-tree X is a k-*path *if and only if for some base B of X, the tree $T(X, B)$ is a* caterpillar.

Proof Sketch. Assume that $X = (V, E)$ is a k-path and let C_i, $i = k, \dots, n-1$, be the current k-clique that has been used as support for adding vertex v_{i+1} to $X_i = X[\{v_1, \dots, v_i\}]$, where $C_k = \{v_1, \dots, v_k\}$ is the initial k-clique. Notice that $C_i \neq C_{i+1}$ implies $C_j \neq C_i$ for all $j > i$. Now it is easy to verify that $T = T(X, C_1)$ is a caterpillar with vertices $C_k, (C_k, v_{k+1}), \dots, (C_{n-1}, v_n)$ containing for each $j \geq k$ with $C_j = C_k$ the edge $(C_k, (C_k, v_{j+1}))$ and for each pair i, j with $C_i \neq C_{i+1} = C_j$ the edge $((C_i, v_{i+1}), (C_j, v_{j+1}))$.

For the other direction assume that $T = T(X, B)$ is a caterpillar and let $B(0), \dots, B(p)$ be the B-decomposition of X. We call $v \in V - B$ a leaf node if $(N_{i-1}(v), v)$ is a leaf in T. Now we can order the vertices of X in such a way that all the vertices in $B(i)$ precede the vertices in $B(i+1)$ and within each layer $B(i)$, $i > 0$, the leaf nodes come first. Let $v_1, \ldots, v_n$ be such an ordering. Then it is easy to verify that we can construct X from the initial k-tree $X_k = X[B] = X[\{v_1, \ldots, v_k\}]$ by successively adding the vertices v_{i+1} to $X_i = X[\{v_1, \ldots, v_i\}]$ using $N_i(v_{i+1})$ as the current clique. ■

Let $X = (V, E)$ be a k-path. Let $\phi : V \longrightarrow C$ be a coloring of its vertices, where C is a set of colors. To canonize the given colored k-path X we use a similar approach as the one that we used in Section 6.3 for colored k-trees. In fact, the only difference is that now our algorithm A additionally checks for each base B whether $T(X, B)$ is a caterpillar. Notice that this can easily be done in logspace as follows.

Starting with the root B as the current node, the algorithm verifies that the current node has at most one child (C', v') in $T(X, B)$ that is not a leaf and then proceeds with (C', v') as the next current node (if the current node has two or more non leaf children, the algorithm detects that $T(X, B)$ is not a caterpillar).

As soon as the algorithm reaches a node that has only leaves as children it decides that $T(X, B)$ is a caterpillar and starts to compute the canons of the labeled trees $T(X, \phi, B, \theta)$ for all bijections $\theta : B \longrightarrow \{1, \ldots, k\}$ as explained in Section 6.3.

Since for a caterpillar $T(X, B)$ the oracle described in Claim 6.3.14 is clearly decidable in logspace, the algorithm A works in logspace. The output of the algorithm A will be a canonical invariant, but as mentioned in Section 6.3, a canonical form of X can be computed in logspace form this canonical invariant. Thus, we have the following theorem.

Theorem 6.4.4 *Let $X = (V, E)$ be a k-path for some fixed k. Let $\phi : V \longrightarrow C$ be a coloring of its vertices. Then the canonical form of the colored graph X can be computed in* L.

As a consequence of the above theorem we get the result.

Corollary 6.4.5 *For each fixed k there is a logspace canonizing algorithm for k-paths. Hence, the isomorphism problem for k-paths is in* L.

Here also we note that the above canonization algorithm can be modified easily to give a logspace algorithm for COLOR-CP for k-paths. Hence by Lemma 6.3.31 we deduce that $CLCoset$ for k-paths is in FL.

Corollary 6.4.6 *For any fixed k, computing the canonical labelling coset of a k-path is in* FL.

We know that the problem GA logspace disjunctively reduces to the problem of deciding if two colored graphs are isomorphic. Hence, as another corollary of Theorem 6.4.4 we get

Corollary 6.4.7 *Given a k-path X deciding if X has nontrivial automorphism, i.e., the* GA *problem for k-paths is in* L.

Next we will prove that k-path isomorphism is complete for L under disjunctive truth-table reductions computable in uniform AC^0, which we denote in the sequel by $\leq_d^{\mathrm{AC}^0}$. To do this we will give a $\leq_d^{\mathrm{AC}^0}$ reduction from the following L-complete problem [48] to k-path isomorphism.

Definition 6.4.8 (Order between Vertices(ORD)) Input: *A directed path $X = (V = \{v_1, \cdots, v_n\}, E)$ given by its adjacency list and two vertices $v_i, v_j \in V$.*

Problem: *Decide if $v_i < v_j$ in the total order induced on V by the directed edges.*

Our reduction is essentially based on the reduction of ORD to tree isomorphism problem shown in [69].

Let (X, v_i, v_j) be an input instance of ORD. Without loss of generality we assume that v_1 and v_n are the vertices with indegree zero and outdegree zero respectively. We assume that v_i and v_j are different from v_1 and v_n.

Now, consider a new *undirected* graph X' obtained from X by coloring the vertex v_i red, vertex v_j green and the last vertex v_n is colored blue, and by dropping the edge directions. Define $\binom{n}{2}$ pairs of undirected graphs $X_{p,q}$ for $1 \leq p < q < n$, where each $X_{p,q}$ has vertex set $\{v_1, \cdots, v_n\}$ and edge set $\{\{v_m, v_{m+1}\} \mid 1 \leq m \leq n-1\}$. Furthermore, in $X_{p,q}$ the node v_p is colored red, v_q is colored green and the last vertex v_n is colored blue. Clearly, (X, v_i, v_j) is a yes instance of ORD if and only if the colored graph X' is isomorphic to $X_{p,q}$ for some $1 \leq p < q < n$.

Next we construct k-paths corresponding to the graphs X' and $X_{p,q}$. First we show how to construct a k-path $H = (V', E')$ corresponding to X'. It will have the following vertex set

$$V' = V \cup \{1, \cdots, k-1\} \cup \{a_l \mid l = 1, 2\} \cup \{b_l \mid l = 1, \cdots, 4\} \cup \{c_l \mid l = 1, \cdots, 8\}$$

We will define the graph using current cliques. Before that we define the following sets.

$L_1 = \{v_{l_1}\},\ v_{l_1} = v_1$
for $m \leq k,\ L_m = \{v_{l_1}, \cdots, v_{l_m}\}$ where $d(v_1, v_{l_m}) = m$ and $(v_{l_p}, v_{l_{p+1}}) \in E$
for $m > k,\ L_m = \{v_{l_1}, \cdots, v_{l_k}\}$ where $d(v_1, v_{l_k}) = m$ and $(v_{l_p}, v_{l_{p+1}}) \in E$

The sequence of current cliques in H' is the following:

$$\{1, \cdots, k-1\} \cup L_1, \{2, \cdots, k-1\} \cup L_2, \cdots, \{k-1\} \cup L_{k-1}, L_k, \cdots, L_n.$$

Let C_{m_1} be the clique in the above sequence where v_i appears for the first time. Also let C_{m_2} be the clique where v_j appears for the first time. We attach the vertices a_1 and a_2 to the clique C_{m_1} and $b_1, \cdots, b_4$ to the current clique C_{m_2}. We attach $c_1, \cdots, c_8$ to the clique L_n. This completes the description of the k-path H'. Notice that to compute H' it is not necessary to compute the set $L_1, \cdots, L_n$ or the current clique sequence. Given any two vertex u and v in V it is sufficient to know if the distance between them is less than k in the directed graph X. This computation can be done in AC^0.

Next we construct a k-path $H_{p,q} = (V', E_{p,q})$ which corresponds to the graph $X_{p,q}$. The vertex set V' is same as the vertex set of H'. We will define the graph using current cliques.

Let

$R_1 = \{v_1\}$
for $m \leq k,\ R_m = \{v_1, \cdots, v_m\}$
for $m > k,\ R_m = \{v_{m-k+1}, \cdots, v_m\}$.

The sequence of current cliques in $H_{p,q}$ is the following

$$\{1, \cdots, k-1\} \cup R_1, \{2, \cdots, k-1\} \cup R_2, \cdots, \{k-1\} \cup R_{k-1}, R_k, \cdots, R_n.$$

We attach the vertices a_1 and a_2 to the clique R_p and the vertices $b_1, \cdots, b_4$ to the clique R_q. We attach $c_1, \cdots, c_8$ to the clique R_n. Again we notice that $H_{p,q}$ can be computed in AC^0.

It is easy to see that if X' and $X_{p,q}$ are isomorphic then so are H' and $H_{p,q}$. For the other direction we will use Theorem 6.3.13. Let $B = \{1, \cdots, k-1\} \cup \{v_1\}$. The tree $T = T(H, B, \theta)$ is a caterpillar where $\theta : B \longrightarrow \{1, \cdots, k\}$ is defined as follows: $\theta(i) = i$ for $i = 1, \cdots, k-1$ and $\theta(v_1) = k$. The caterpillar will have two legs at the node corresponding to the vertex v_i, four legs at the node corresponding to the vertex v_j and eight legs at node corresponding to v_n. Let $T_{p,q} = T(H_{p,q}, B', \theta')$ for some B' and θ'. For T and $T_{p,q}$ to be isomorphic B' must be same as B. This caterpillar will have two, four and eight legs at nodes corresponding to the vertices v_p, v_q and v_n respectively. Now it is easy to see that if H' and $H_{p,q}$ are isomorphic then X' and $X_{p,q}$ are also isomorphic.

Theorem 6.4.9 *For each k, k-path isomorphism is complete for* L *under $\leq_d^{\mathrm{AC}^0}$ reduction.*

We have seen in Corollary 6.4.7 that the GA problem for k-path is in L. Here we will prove that the problem is in fact complete for L under $\leq_d^{AC^0}$ reduction. We will do this by showing that the ORD problem AC^0 disjunctively reduces to the k-path automorphism problem. Let (X, v_i, v_j) be an instance of ORD. Again, we will use the idea of the colored graphs X' and $X_{p,q}$. Let $H_{p,q}$ be the colored graph obtained by fusing the vertex v_1 of the graphs X' and $X_{p,q}$. Notice that $H_{p,q}$ has a nontrivial automorphism if and only if (X, v_i, v_j) is an "yes" instance of ORD. Next we will construct a k-path $C_{p,q}$ which will represent the graph $H_{p,q}$. We will construct $C_{p,q}$ using the idea of current cliques. We assume without loss of generality that v_1 has indegree 0, v_n has out degree 0 and the distances from v_1 to v_i and v_j are more that $k+1$. We also assume that the distance between v_i and v_j is more than 2. As before we define the following sets

$$\begin{aligned}
&R_1 = \{v_1\}\\
&\text{for } m \leq k,\ R_m = \{v_1, \cdots, v_m\}\\
&\text{for } m > k,\ R_m = \{v_{m-k+1}, \cdots, v_m\}.
\end{aligned}$$

and the sets

$$\begin{aligned}
&L_1 = \{v_{l_1}\},\ v_{l_1} = v_1\\
&\text{for } m \leq k,\ L_m = \{v_{l_1}, \cdots, v_{l_m}\} \text{ where } d(v_1, v_{l_m}) = m \text{ and } (v_{l_p}, v_{l_{p+1}}) \in E\\
&\text{for } m > k,\ L_m = \{v_{l_1}, \cdots, v_{l_k}\} \text{ where } d(v_1, v_{l_k}) = m \text{ and } (v_{l_p}, v_{l_{p+1}}) \in E
\end{aligned}$$

The vertex set of $C_{p,q}$ is

$$V = \{w_i \mid i = 1, \cdots, n\} \cup \{v_i \mid i = i, \cdots, n\} \cup \{-k-1, \cdots, 2k+1\}.$$

For integers a ,b where $a < b$ let $[a, b]$ denote the interval $\{i \mid i = a \text{ to } b\}$. The sequence of current cliques is as follows.

$$
\begin{aligned}
&L_n, \cdots, L_{k+1}, L_k,\\
&L_{k-1} \cup \{-k-1\}, L_{k-2} \cup [-k-1, -k], \cdots, L_1 \cup [-k-1, -3],\\
&[-k-1, -2], \cdots, [k+2, 2k+1],\\
&[k+3, 2k+1] \cup R_1, [k+4, 2k+1] \cup R_2, [2k, 2k+1] \cup\ R_{k-2}, \{2k+1\} \cup R_{k-1},\\
&R_k, \cdots R_n.
\end{aligned}
$$

Let L_{i_1} be the first set in the sequence where v_i appears and let L_{i_2} be the first set where v_j appears. Notice that L_{i_2-1} and L_{i_2+1} are the two sets just before and after the set L_{i_2} in the sequence. We connect vertex u_1 to all the vertices in the set L_{i_i}, we connect the vertices u_2 and u_3 to the cliques L_{i_2-1} and L_{i_2+1} respectively. Similarly we connect the vertices w_1, w_2 and w_3 to all the vertices of the cliques R_p, R_{q-1} and R_{q+1} respectively.

We have to prove that $C_{p,q}$ has a nontrivial automorphism if and only if $H_{p,q}$ has a nontrivial automorphism for some p, q where $p < p + 1 < q$. For that we use the next theorem. The proof of the theorem is similar to Theorem 6.3.13.

Theorem 6.4.10 *Let X be a k-tree. Let B be a clique in X and $\theta : B \longrightarrow \{1, \cdots, k\}$ be a bijection. Then X has a nontrivial automorphism iff there is a clique B' and bijection $\theta' : B' \longrightarrow \{1, \cdots, k\}$ such that $B' \neq B$ or $B = B'$ but $\theta(v) \neq \theta'(v)$ for some $v \in B = B'$ and $T(X, B, \theta)$ is isomorphic to $T(X, B', \theta')$.*

Suppose $C_{p,q}$ has a nontrivial automorphism. To apply Theorem 6.4.10 we choose B to be $\{1, \cdots, k\}$ and θ to be identity. Notice that for any other clique B', $T(C_{p,q}, B', \theta')$ (for any θ') will have different height. This forces B' to be same as B. It is not hard to prove that the only choice of θ' that can give nontrivial automorphism is $\theta'(i) = k+1-i$. But then this will force $H_{p,q}$ to have nontrivial automorphism. The other direction is similar. If $H_{p,q}$ has a nontrivial automorphism then so does $C_{p,q}$. This can be seen again using Theorem 6.4.10 by taking $B = B' = \{1, \cdots, k\}$, $\theta(i) = i$ and $\theta'(i) = k + 1 - i$. As before the graph $C_{p,q}$ can be computed in AC^0.

Theorem 6.4.11 *The $Auto$ problem for k-paths is complete for L under $\leq_d^{\mathrm{AC}^0}$ reduction.*

6.5 Remarks

Building on the ideas described in this chapter, Köbler and Kuhnert recently gave a logspace algorithm for k-tree isomorphism [74]. Their result along with Theorem 6.4.9 implies that the k-tree isomorphism is hard of logspace. In this section we mention the salient points of their algorithm. See [74] for details.

In Section 6.3 (page 115) we defined a mangrove $D(X, B)$ and from that we defined a tree $T(X, B)$ in page 116. In [74], instead of doing that they define the following graph.

Definition 6.5.1 ([74]) *Let $X = (V, E)$ be a k-tree. The* tree representation $T(X)$ *of X is defined by*

$$V(T(X)) = \{M \subseteq V \mid M \textit{ is a } k\textit{-clique or a } (k+1)\textit{-clique}\}$$

$$E(T(X)) = \{\{M_1, M_2\} \subseteq V \mid M_1 \subsetneq M_2\}$$

They prove that the tree representation is actually a tree (and thus the name tree representation) and it can be computed in logspace.

Theorem 6.5.2 ([74]) *Let X be a k-tree. Then*

- *The tree representation $T(X)$ of X is a tree.*
- *The tree representation $T(X)$ can be computed in* FL.

Later they define a scheme for labelling the nodes of the tree representation which is similar to $T(X, B, \theta)$ in Section 6.3 (page 117) which preserves isomorphism. The labelling scheme is shown to be computable in logspace. This gives the logspace algorithm for k-tree isomorphism.

Bibliography

[1] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Annals of Mathematics. Second Series*, 160(2):781–793, 2004.

[2] M. Agrawal and N. Saxena. Automorphisms of Finite Rings and Applications to Complexity of Problems. In *STACS*, pages 1–17, 2005.

[3] A. Aho, J. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA, 1974.

[4] W. Aiello and J. Håstad. Statistical zero-knowledge languages can be recognized in two rounds. *Journal of Computer and System Sciences*, 42(3):327–345, June 1991.

[5] E. Allender. Arithmetic circuits and counting complexity classes. In J. Krajíćek, editor, *Complexity of Computations and Proofs*, *Quaderni di Matematica*, volume 13, pages 33–72, Seconda Universita di Napoli, 2004.

[6] E. Allender and K.-J. Lange. RUSPACE$(\log n) \subseteq DSPACE(\log^2 n/ \log\log n)$. *Theory of Computing Systems*, 31(5):539–550, 1998. formerly Mathematical Systems Theory.

[7] E. Allender and M. Ogihara. Relationships among PL, #l, and the determinant. *RAIRO: R. A. I. R. O. Informatique Théorique et Applications*, 30(1):1–21, 1996.

[8] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Appl. Math.*, 23:11–24, 1989.

[9] S. Arnborg and A. Proskurowski. Canonical representations of partial 2-trees and partial 3-trees. *BIT*, 32:197–214, 1992.

[10] V. Arvind and B. Das. SZK Proofs for Black-Box Group Problems. *Theory Comput. Syst (Conference version appeared in CSR'06)*, 43(2):100–117, 2008.

[11] V. Arvind and B. Das. Limited Randomness Statistical Zero Knowledge. *In Preparation*, 2009.

[12] V. Arvind, B. Das, and J. Köbler. The Space Complexity of k-Tree Isomorphism. In Takeshi Tokuyama, editor, *Algorithms and Computation, 18th International Symposium, ISAAC 2007, Sendai, Japan, December 17-19, 2007, Proceedings*, volume 4835 of *Lecture Notes in Computer Science*, pages 822–833. Springer, 2007.

[13] V. Arvind, B. Das, and J. Köbler. A Logspace Algorithm for Partial 2-Tree Canonization. In Edward A. Hirsch, Alexander A. Razborov, Alexei L. Semenov, and Anatol Slissenko, editors, *Computer Science - Theory and Applications, Third International Computer Science Symposium in Russia, CSR 2008, Moscow, Russia, June 7-12, 2008, Proceedings*, volume 5010 of *Lecture Notes in Computer Science*, pages 40–51. Springer, 2008.

[14] V. Arvind, B. Das, J. Köbler, and S. Toda. An FPT Algorithm for Bounded Color Class Hypergraph Isomorphism. *Manuscript*, 2009.

[15] V. Arvind, B. Das, and P. Mukhopadhyay. On Isomorphism and Canonization of Tournaments and Hypertournaments. In Tetsuo Asano, editor, *Algorithms and Computation, 17th International Symposium, ISAAC 2006, Kolkata, India, December 18-20, 2006, Proceedings*, volume 4288 of *Lecture Notes in Computer Science*, pages 449–459. Springer, 2006.

[16] V. Arvind, B. Das, and P. Mukhopadhyay. The Complexity of Black-Box Ring Problems. In Danny Z. Chen and D. T. Lee, editors, *Computing and Combinatorics, 12th Annual International Conference, COCOON 2006, Taipei, Taiwan, August 15-18, 2006, Proceedings*, volume 4112 of *Lecture Notes in Computer Science*, pages 126–135. Springer, 2006.

[17] V. Arvind and J. Köbler. On Hypergraph and Graph Isomorphism with Bounded Color Classes. In Bruno Durand and Wolfgang Thomas, editors, *STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science, Marseille, France, February 23-25, 2006, Proceedings*, volume 3884 of *Lecture Notes in Computer Science*, pages 384–395. Springer, 2006.

[18] V. Arvind and P. Kurur. Graph isomorphism is in spp. In *FOCS*, pages 743–750, 2002.

[19] V. Arvind and J. Torán. Solvable group isomorphism is (almost) in NP intersect coNP. In *Annual IEEE Conference on Computational Complexity (formerly Annual Conference on Structure in Complexity Theory)*, volume 19, 2004.

[20] V. Arvind and J. Torán. The complexity of quasigroup isomorphism and the minimum generating set problem. In *ISAAC*, pages 233–242, 2006.

[21] L. Babai. Moderately exponential bound for graph isomorphism. In Ferenc Gécseg, editor, *Proceedings of the 1981 International FCT-Conference on Fundamentals of Computation Theory*, volume 117 of *LNCS*, pages 34–50, Szeged, Hungary, August 1981. Springer.

[22] L. Babai. A Las Vegas-NC Algorithm for Isomorphism of Graphs with Bounded Multiplicity of Eigenvalues. In *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science, FOCS'86 (Toronto, Canada, October 27-29, 1986)*, pages 303–312. IEEE, IEEE, 1986.

[23] L. Babai. Bounded round interactive proofs in finite groups. *SIAM J. Discrete Math.*, 5(1):88–111, 1992, February.

[24] L. Babai, P. J. Cameron, and P. P. Pálfy. On the order of primitive groups with restricted nonabelian composition factors. *Journal of Algebra*, 79:161–168, 1982.

[25] L. Babai and P. Codenotti. Isomorphism of hypergraphs of low rank in moderately exponential time. In *Proceedings 39th Ann. IEEE Symp. on Theory of Computing*, pages 667–676. IEEE Comp. Soc. Press, 2008.

[26] L. Babai, P. Erdös, and S. M. Selkow. Random graph isomorphism. *SIAM Journal on Computing*, 9(3):628–635, August 1980.

[27] L. Babai, A. J. Goodman, W. M. Kantor, E. M. Luks, and P. P. P'alfy. Short presentations for finite groups. *Journal of Algebra*, 194(1):79–112, 1997.

[28] L. Babai, D. Y. Grigoriev, and D. M. Mount. Isomorphism of graphs with bounded eigenvalue multiplicity. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, STOC'82 (San Francisco, CA, May 5-7, 1982)*, pages 310–324, New York, 1982. ACM, ACM Press.

[29] L. Babai and L. Kučera. Canonical labelling of graphs in linear average time. In *20th Annual Symposium on Foundations of Computer Science*, pages 39–46, San Juan, Puerto Rico, 29–31 October 1979. IEEE.

[30] L. Babai and E. M. Luks. Canonical labeling of graphs. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, STOC'83 (Boston, MA, May 25-27, 1983)*, pages 171–183, New York, 1983. ACM, ACM Press.

[31] 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(2):254–276, April 1988.

[32] L. Babai and E. Szemerédi. On the complexity of matrix group problems I. In *25th Annual Symposium on Foundations of Computer Science*, pages 229–240, Singer Island, Florida, 24–26 October 1984. IEEE.

[33] H. L. Bodlaender. Dynamic programming on graphs with bounded treewidth. In T. Lepistö and A. Salomaa, editors, *Automata, Languages and Programming, 15th International Colloquium (ICALP)*, volume 317 of *Lecture Notes in Computer Science*, pages 105–118, Tampere, Finland, 11–15 July 1988. Springer-Verlag.

[34] H. L. Bodlaender. Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *Journal of Algorithms*, 11(4):631–643, December 1990.

[35] R. B. Boppana, J. Håstad, and S. Zachos. Does co-NP have short interactive proofs? *Inf. Process. Lett.*, 25(2):127–132, 1987, May.

[36] G. Buntrock, B. Jenner, K.-J. Lange, and P. Rossmanith. Unambiguity and fewness for logarithmic space. In Lothar Budach, editor, *Proceedings of Fundamentals of Computation Theory (FCT '91)*, volume 529 of *LNCS*, pages 168–179, Berlin, Germany, September 1991. Springer.

[37] S. R. Buss. Alogtime algorithms for tree isomorphism, comparison, and canonization. In Georg Gottlob, Alexander Leitsch, and Daniele Mundici, editors, *Kurt Gödel Colloquium*, volume 1289 of *Lecture Notes in Computer Science*, pages 18–33. Springer, 1997.

[38] Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.

[39] N. Chandrasekharan. Isomorphism testing of k-trees is in NC, for fixed k. *Information Processing Letters*, 34(6):283–287, 28 May 1990.

[40] N. Chandrasekharan and S. S. Iyengar. NC algorithms for recognizing chordal graphs and k trees. *IEEE Trans. Computers*, 37(10):1178–1183, 1988.

[41] J. Chen, I. A. Kanj, and G. Xia. Improved parameterized upper bounds for vertex cover. In R. Kralovic and P. Urzyczyn, editors, *Mathematical Foundations of Computer Science MFCS*, volume 4162 of *LNCS*, pages 238–249. Springer, 2006.

[42] M. Dehn. Über undedliche diskontinuierlich gruppen. *Math. Ann.*, 71:116–144, 1911.

[43] R. Diestel. *Graph theory*. Springer-Verlag, New York, 2 edition, 2000.

[44] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4):873–921, August 1995.

[45] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.

[46] C. Dwork, M. Naor, and A. Sahai. Concurrent zero-knowledge. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing, STOC'98 (Dallas, Texas, May 23-26, 1998)*, pages 409–418, New York, 1998. ACM Press.

[47] P. Erdös and Rényi. Probabilistic methods in group theory. *Journal d'Analyse Mathématique*, 14(1):127–138, 1965.

[48] K. Etessami. Counting quantifiers, successor relations, and logarithmic space. *Journal of Computer and System Sciences*, 54(3):400–411, 1997.

[49] W. Feit and J. Thompson. Solvability of groups of odd order. *Pacific Journal of Mathematics*, 13:775–1029, 1963.

[50] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag, 2006.

[51] L. Fortnow. The complexity of perfect zero-knowledge. *ADVCR: Advances in Computing Research*, 5, 1989.

[52] M. Fürer. Graph Isomorphism Testing without Numerics for Graphs of Bounded Eigenvalue Multiplicity. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'95 (San Francisco, California, January 22-24, 1995)*, pages 624–631, Philadelphia, PA, 1995. ACM SIGACT, SIAM, Society for Industrial and Applied Mathematics.

[53] M. Furst, J. Hopcroft, and E. Luks. Polynomial-time algorithms for permutation groups. In *Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science, FOCS'80 (Syracuse, NY, October 13-15, 1980)*, pages 36–41. IEEE, IEEE, 1980.

[54] O. Goldreich. *Foundations of cryptography*. Cambridge University Press, 2001.

[55] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM, JACM*, 38(3):691–729, July 1991.

[56] O. Goldreich, A. Sahai, and S. Vadhan. Honest-verifier statistical zero-knowledge equals general statistical zero-knowledge. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC-98)*, pages 399–408, New York, May 23–26 1998. ACM Press.

[57] O. Goldreich and S. Vadhan. Comparing entropies in statistical zero knowledge with applications to the structure of SZK. In *Proceedings of the 14th Annual IEEE Conference on Computational Complexity (CCC-99)*, pages 54–75, Los Alamitos, May 4–6 1999. IEEE Computer Society.

[58] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive systems. *SIAM Journal of Computing*, 18(1):186–208, 1989.

[59] J. G. Del Greco, C. N. Sekharan, and R. Sridhar. Fast parallel reordering and isomorphism testing of k-trees. *Algorithmica*, 32(1):61–72, 2002.

[60] M. Grohe and O. Verbitsky. Testing graph isomorphism in parallel by playing a game. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener,

editors, *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part I*, volume 4051 of *Lecture Notes in Computer Science*, pages 3–14. Springer, 2006.

[61] A. Gupta, N. Nishimura, A. Proskurowski, and P. Ragde. Embeddings of k - connected graphs of pathwidth k. *Discrete Applied Mathematics*, 145(2):242–265, 2005.

[62] Y. Gurevich. From invariants to canonization. *Bulletin of the EATCS*, 63, 1997.

[63] G. Gutin and A. Yeo. Hamiltonian paths and cycles in hypertournaments. *Journal of Graph Theory*, 25(4):277–286, 1997.

[64] M. Hall, Jr. *The Theory of Groups*. The Macmillan Company, New York, NY, USA, 1959.

[65] F. Harary and E. M. Palmer. On acyclic simplicial complexes. *Mathematica*, 15:115–122, 1968.

[66] C. M. Hoffmann. *Group-Theoretic Algorithms and Graph Isomorphism*, volume 136. Springer, Berlin, Heidelberg, 1982.

[67] J. E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs. In *Proceedings of the 6th Annual ACM Symposium on Theory of Computing, STOC'74 (Seattle, WA, April 30 - May 2, 1974)*, pages 172–184, New York, 1974. ACM, ACM Press.

[68] A. Jakoby and M. Liskiewicz. Paths problems in symmetric logarithmic space. In *29th International Colloquium on Automata, Languages,and Programming, ICALP'02*, volume 2380 of *Lecture Notes in Computer Science*, pages 269–280, 2002.

[69] B. Jenner, J. Köbler, P. McKenzie, and J. Torán. Completeness results for graph isomorphism. *J. Comput. Syst. Sci.*, 66(3):549–566, 2003.

[70] N. Kayal and N. Saxena. On the ring isomorphism and automorphism problems. In *IEEE Conference on Computational Complexity*, pages 2–12, 2005.

[71] M. M. Klawe, D. G. Corneil, and A. Proskurowski. Isomorphism testing in hookup classes. *SIAM J. Algebraic Discrete Methods*, 3:260–274, 1982.

[72] T. Kloks. *Treewidth — computations and approximations*, volume 842 of *LNCS*. Springer-Verlag, Berlin-Heidelberg-New York-London-Paris-Tokyo-Hong Kong-Barcelona-Budapest, 1994.

[73] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–267. Pergamon, New York, 1970.

[74] J. Köbler and S. Kuhnert. The isomorphism problem for k-trees is complete for logspace. *ECCC*, TR09-053, 2009.

[75] J. Köbler, U. Schöning, and J. Torán. Graph Isomorphism is Low for PP. In *STACS*, pages 401–411, 1992.

[76] J. Köbler, U. Schöning, and J. Torán. *The graph isomorphism problem*. Progress in Theoretical Computer Science. Birkhäuser Verlag, Boston-Basel-Berlin-Stuttgart, 1993.

[77] H. W. Lenstra, Jr. Algorithms in algebraic number theory. *Bulletin of the American Mathematical Society*, 26:211–244, 1992.

[78] M. Liebeck and A. Shalev. Simple groups, permutation groups and probability. *Journal of Amer. Soc.*, 12:497–520, 1999.

[79] S. Lindell. A logspace algorithm for tree canonization. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, STOC'92 (Victoria, British Columbia, Canada, May 4-6, 1992)*, pages 400–404, New York, 1992. ACM SIGACT, ACM Press.

[80] E. M. Luks. Isomorphism of graphs of bounded valance can be tested in polynomial time. *Journal of Computer and System Sciences*, 25(1):42–65, August 1982.

[81] E. M. Luks. Lectures on Polynomial-Time Computation in Groups. *Technical Report NU-CCS-90-16*, 1990.

[82] E. M. Luks. Permutation groups and polynomial-time computation, in groups and computation. *DIMACS series in Discrete Mathematics and Theoretical Computer Science*, 11:139–175, 1993.

[83] E. M. Luks. Hypergraph isomorphism and structural equivalence of Boolean functions. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing, STOC'99 (Atlanta, Georgia, May 1-4, 1999)*, pages 652–658, New York, 1999. ACM Press.

[84] B. D. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.

[85] D. Micciancio, S. J. Ong, A. Sahai, and S. Vadhan. Concurrent zero knowledge without complexity assumptions. *Electronic Colloquium on Computational Complexity (ECCC)*, (093), 2005.

[86] D. Micciancio and S. P. Vadhan. Statistical zero-knowledge proofs with efficient provers: Lattice problems and more. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 282–298. Springer, 2003.

[87] G. L. Miller. Isomorphism testing for graphs of bounded genus. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing, STOC'80 (Los Angeles, CA, April 28-30, 1980)*, pages 225–235, New York, 1980. ACM, ACM Press.

[88] G. L. Miller. Isomorphism of k-contractible graphs. A generalization of bounded valence and bounded genus. *Information and Control*, 56(1/2):1–20, January/February 1983.

[89] G. L. Miller and J. H. Reif. Parallel tree contraction, part 2: Further applications. *SIAM J. Comput.*, 20(6):1128–1147, 1991, December.

[90] P. P. Pálfy. A polynomial bound for the orders of primitive solvable groups. *J. Algebra*, pages 127–137, 1982.

[91] E. Petrank and G. Tardos. On the knowledge complexity of $\mathcal{NP}$. In *37th Annual Symposium on Foundations of Computer Science*, pages 494–503, Burlington, Vermont, 14–16 October 1996. IEEE.

[92] A. Proskurowski. Maximal graphs of path-width k or searching a partial k-caterpillar. *Technical Report UO-CIS-TR-89-17*, 1989.

[93] O. Reingold. Undirected ST-connectivity in log-space. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC 2005), 2005*, pages 376–385. ACM, 2005.

[94] A. Sahai and S. Vadhan. A complete promise problem for statistical zero-knowledge. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS-97)*, pages 448–457, Los Alamitos, October 20–22 1997. IEEE Computer Society Press.

[95] P. Scheffler and D. Seese. A combinatorial and logical approach to linear-time computability. In *European Conference on Computer Algebra (EUROCAL'87)*, volume 378, pages 379–380. Springer-Verlag, 1987.

[96] A. Seress. *Permutation Group Algorithms*. Cambridge Univ. Press, August 2003.

[97] M. W. Short. *The Primitive Soluble Permutation Groups of Degree less than* 256, volume 1519 of *Lecture Notes in Math.* Springer-Verlag, Berlin, Heidelberg, New York, 1992.

[98] C. C. Sims. Computational methods in the study of permutation groups. In *Computational problems in abstract algebra*, pages 169–183, Oxford, 1970. (Oxford, 1967), Pergamon Press.

[99] C. C. Sims. Some group theoretic algorithms. In A. Dold and B. Eckmann, editors, *Topics in algebra*, volume 697 of *Lecture Notes in Math.*, pages 108–124, Berlin, Heidelberg, New York, 1978. (Canberra, 1978), Springer-Verlag.

[100] D. A. Spielman. Faster isomorphism testing of strongly regular graphs. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing, STOC'96 (Philadelphia, Pennsylvania, May 22-24, 1996)*, pages 576–584, New York, 1996. ACM Press.

[101] T. Thierauf and F. Wagner. The isomorphism problem for planar 3-connected graphs is in unambiguous logspace. *Technical Report: TR07-068, Electronic Colloquium on Computational Complexity (ECCC)*, 2007.

[102] T. Thierauf and F. Wagner. The isomorphism problem for planar 3-connected graphs is in unambiguous logspace. In Susanne Albers and Pascal Weil, editors, *25th Annual Symposium on Theoretical Aspects of Computer Science, STACS'08*, volume 08001 of *Dagstuhl Seminar Proceedings*, pages 633–644, 2008.

[103] S. Toda. Computing automorphism groups of chordal graphs whose simplicial components are of small size. *IEICE Transactions*, 89-D(8):2388–2401, 2006.

[104] J. A. Todd and H. S. M. Coxeter. A practical method for enumerating cosets of a finite abstract group. *Proc. Edinburgh Math. Soc.*, 5:26–34, 1936.

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

[106] S. Vadhan. A study of statistical zero-knowledge proofs. *PhD Thesis, http://www.eecs.harvard.edu/˜salil/papers/phdthesis-abs.html*, 1997.

[107] O. Verbitsky. On the double coset membership problem for permutation groups. In *Algebraic structures and their applications, Proceedings of the 3rd international algebraic conference held in framework of the Ukrainian mathematical congress (Kiev, 2001)*, pages 351–363, Institute of Mathematics, Ukrainian Academy of Sciences (2002), 2001.

[108] O. Verbitsky. Zero-knowledge proofs of the conjugacy for permutation groups. *Visn. L'viv. Univ., Ser. Mekh.-Mat. (Bulletin of the Lviv University, Series in Mechanics and Mathematics)*, pages 195–2005, 2003.

[109] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, 2nd edition, 2003.

[110] E. Wanke. Bounded tree-width and LOGCFL. *Journal of Algorithms*, 16(3):470–491, May 1994.

[111] B. Weisfeiler. On construction and identification of graphs. *Lecture Notes in Mathematics*, 558, 1976.

[112] B. Weisfeiler and A. A. Lehman. A reduction of graph to a canonical form and algebra arising during this reduction (in russian). *Nauchno-Technicheskaya Informatsia, Seriya 2*, 9:12–16, 1968.

[113] E. W. Weisstein. Prime number theorem. *From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/PrimeNumberTheorem.html*.

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

[115] T. R. Wolf. Solvable and nilpotent subgroups of $GL(n, q^m)$. *Can. J. Math.*, pages 1097–1111, 1982.

[116] K. Yamazaki, H. L. Bodlaender, B. de Fluiter, and D. M. Thilikos. Isomorphism for graphs of bounded distance width. *Algorithmica*, 24(2):105–127, 1999.

[117] V. Zemlyachenko, N. Kornienko, and R. Tyshkevich. Graph Isomorphism Problem (Russian). In *The Theory of Computation I, Notes Sci. Sem LOMI*, 118, 1982.