THE UNIVERSITY OF CHICAGO

TESTING ISOMORPHISM OF COMBINATORIAL AND ALGEBRAIC STRUCTURES

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY
PAOLO CODENOTTI

CHICAGO, ILLINOIS
AUGUST 2011

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank my advisor László Babai, and the thesis committee members Yury Makarychev, and Martin Fürer, for their time and efforts. I am particularly grateful to László Babai for helping me throughout my academic career, and for always being a teacher and a mentor, never missing an opportunity to teach me about anything and everything, from linear algebra and group theory, to the pronunciation and spelling of names in different languages. I owe much of my academic achievements to him.

All of the results in this thesis are joint work with László Babai. The results in Chapters 4, 5, and 6 are joint work also with Youming Qiao; and the results in Chapters 4 and 6 also with Joshua A. Grochow.

I was lucky to interact with many great researchers throughout my studies, and I tried to learn something from all of them. Thanks go to Jehoshua Bruck and everyone at Paradise Lab at Caltech, where I visited for two summers as an undergrad, for creating such a welcoming environment where I got started in research. Thanks to Pedro Felzenszwalb, who advised me in my masters thesis on min-convolutions, for giving me engaging problems to work on. Thanks to Julia Chuzhoy for introducing me to research on approximation algorithms, and teaching me much about the area in the process. It was a pleasure to talk and collaborate with Youming Qiao and Joshua A. Grochow. I would like to thank Janos Simon for many interesting discussions, and helpful advice.

Thanks to all the professors, staff and students in the Computer Science department for always being welcoming and helpful, and making these years productive, interesting, and fun.

I am very grateful for my family. My parents have given me great opportunities, and their advice is always useful. My siblings have been my role models: Luca taught me to do what I love, Giulia to be excited about what I do, and Marco to work hard.

Thanks to my wonderful fiancé Lindsey for her support and encouragement.

ABSTRACT

Our main results are algorithms to test isomorphism of hypergraphs and of groups.

We give an algorithm to test isomorphism of hypergraphs of rank k in time $\exp(\tilde{O}(k^2\sqrt{n}))$, where n is the number of vertices (joint work with L. Babai, FOCS 2008). (The rank is the maximum size of hyperedges; the tilde refers to a polylogarithmic factor.) The case of bounded k answered a then 24-year-old question and removes an obstacle to improving the worst-case bound for Graph Isomorphism testing. The best previously known bound, even for $k = 3$, was C^n (Luks 1999).

We consider the problem of testing isomorphism of groups of order n given by Cayley tables. The $n^{\log n}$ bound on the time complexity for the general case has not been improved over the past four decades. We demonstrate that the obstacle to efficient algorithms is the presence of abelian normal subgroups; we show this by giving a polynomial-time isomomrphism test for "semisimple groups," defined as groups without abelian normal subgroups (joint work L. Babai, Y. Qiao, in preparation). This concludes a project started with with L. Babai, J. A. Grochow, Y. Qiao (SODA 2011). That paper gave an $n^{\log\log n}$ algorithm for this class, and gave polynomial-time algorithms assuming boundedness of certain parameters. The key ingredients for this algorithm are: (a) an algorithm to test permutational isomorphism of permutation groups in time polynomial in the order and simply exponential in the degree; (b) the introduction of the "twisted code equivalence problem," a generalization of the classical code equivalence problem by admitting a group action on the alphabet; (c) an algorithm to test twisted code equivalence; (d) a reduction of semisimple group isomorphism to permutational isomorphism of transitive permutation groups under the given time limits via "twisted code equivalence."

The twisted code equivalence problem and the problem of permutational isomorphism of permutation groups are of interest in their own right.

CHAPTER 1

INTRODUCTION AND PRELIMINARIES

1.1 Introduction

Our main results are algorithms to test isomorphism of hypergraphs and of "semisimple groups" (defined as groups with no abelian normal subgroups). For the Hypergraph Isomorphism problem we use group theoretic techniques intertwined with combinatorial partitioning, refinement, and individualization techniques. This allows us to obtain an isomorphism test that runs in time $\exp(\widetilde{O}(k^2\sqrt{n}))$ for hypergraphs of rank k. (The rank is the maximum size of a hyperedge; the tilde is suppressing poly-logarithmic factors). Via a study of the structure of semisimple groups, we reduce Semisimple Group Isomorphism to Permutational Isomorphism of permutation groups, and Twisted Code Equivalence. By solving the latter two problems, of independent interest, we obtain a polynomial time isomorphism test for this class. The key case of Permutational Isomorphism required for our main result is the transitive case. In this case we are able to *list* all permutational isomorphisms in time linear in the order, and simply-exponential in the degree. The Twisted Code Equivalence problem generalizes of the code equivalence problem by allowing a group to act on the alphabet. Codes over alphabets of size 2 are hypergraphs (of arbitrary rank). Our algorithm for Twisted Code Equivalence generalizes Luks's dynamic programming simply-exponential time hypergraph isomorphism test [57].

1.1.1 Hypergraph isomorphism – moderately exponential algorithms

NP problems are defined via a *witness space* $W(x)$ (where x is the input). $|W(x)|$ is typically exponential in $|x|$. Exhaustive search of $W(x)$ solves the problem in time $|W(x)|\text{poly}(|x|)$. To reduce this time to $\exp\left((\ln|W(x)|)^{1-c}\right)\text{poly}(|x|)$ for some constant $c > 0$ usually requires nontrivial insight; we call algorithms running within such a time bound "moderately exponential."[1] Thus for the Graph Isomorphism problem, a moderately exponential algorithm would run in $\exp(O(n^{1-c}))$ where n is not the length of the input but the number of vertices (since the witness space in the definition consists of $n!$ permutations).

Moderately exponential algorithms for Graph Isomorphism were found in the wake of the introduction of group theoretic methods to the Graph Isomorphism problem ([2, 55]); the best existing bounds are of the form $\exp(\widetilde{O}(\sqrt{n}))$ and arise from a combination of Luks's seminal work on divide-and-conquer methods to manipulate permutation groups [55], and a combinatorial trick due to Zemlyachenko [84]. The result appears in [23] (1983); Luks subsequently refined the bound to $\exp\left(O(\sqrt{n \log n})\right)$ (see [21], 1983). This remains the state of the art after a quarter century; reducing the leading $\sqrt{n}$ term to $n^{1/2-c}$ in the exponent is a major open problem. For an important special case, that of strongly regular graphs, which admit an elementary $\exp(\widetilde{O}(\sqrt{n}))$ isomorphism test [3], the exponent of the exponent was reduced to 1/3 by Spielman in 1996 [74].

While isomorphism of explicit hypergraphs is polynomial-time equivalent to Graph Isomorphism, reductions do not preserve moderate exponentiality, so the natural question arises whether isomorphism of hypergraphs, even of hypergraphs of bounded rank (bounded size of hyper-edges) can be tested in moderately exponential time. This question was stated in [23] in 1983, and it was pointed out there that the absence of a moderately exponential bound for 4-uniform hypergraphs is an obstacle to improving the $\widetilde{O}(\sqrt{n})$ bound to $n^{1/2-c}$ in the exponent of the bound for Graph Isomorphism, adding to the significance of the problem.

In an important development, Luks reduced the trivial factorial bound to simply exponential (C^n) for testing isomorphism of hypergraphs of any rank [57] (1999) (again, n is the number of vertices). However, this bound does not qualify as moderately exponential, and Luks reiterates the long-standing open problem of moderately exponential isomorphism test for hypergraphs of bounded rank.

1. We follow the isomorphism literature in this use of the term 'moderately exponential' (cf. [4, 57]). The term is used with another meaning in the area of exact algorithms for NP-hard problems. See Appendix A.1 for more details and a comparison of the two definitions.

We present the solution to this old problem, not only for bounded rank but even for rather large ranks.

Recall that a hypergraph $X = (V, E)$ consists of a set V of vertices and a set E of "hyper-edges;" the hyper-edges are subsets of V. The *rank* of X is the maximum size of its hyper-edges.

As above, we use the soft-Oh notation to suppress polylogarithmic factors, so for a function $f(n)$ we write $\widetilde{O}(f(n))$ to denote the class of functions $f(n)(\log n)^{O(1)}$. So for instance $n! = \exp(\widetilde{O}(n))$. Our main result, which is joint work with László Babai [16] is that isomorphism of hypergraphs of rank k with n vertices can be tested in $\exp\left(\widetilde{O}(k^2\sqrt{n})\right)$. Note that this bound is $\exp(\widetilde{O}(\sqrt{n}))$ for bounded k.

1.1.2 Group isomorphism – bottlenecks and approach

The isomorphism problem for groups asks to determine if two groups, given by their Cayley tables (multiplication tables), are isomorphic. Tarjan is credited for pointing out that if one of the groups is generated by k elements then isomorphism can be decided in time $n^{k+O(1)}$ where n is the order of the groups; indeed one can list all isomorphisms within this time bound (cf. [62]). Since $k \leq \log n$ for all groups, this in particular gives an $n^{\log n+O(1)}$-time algorithm for all groups (log to the base 2) and a polynomial-time algorithm for finite simple groups (because the latter are generated by 2 elements, a consequence of their classification [34]). In spite of considerable attention to the problem over the past quarter century, no general bound with a sublogarithmic exponent has been obtained. While the abelian case is easy ($O(n)$ according to Kavitha [49], improving Savage's $O(n^2)$ [70] and Vikas's $O(n \log n)$ [77]), just one step away from the abelian case lurk what appear to be the most notorious cases: nilpotent groups of class 2. These groups G are defined by the property that the quotient $G/Z(G)$ is abelian, where $Z(G)$ is the center of G. No complete structure theory of such groups is known; recent work in this direction by James Wilson [81, 82] commands attention.

Recently, other special classes of solvable groups have been considered. A group G is an

A-group if all its Sylow subgroups are abelian. A Sylow tower of a group is a normal chain where for every Sylow subgroup, there exists a factor isomorphic to it. Babai and Qiao [27] (building on the works [67] and [53]) present an efficient isomorphism test for A-groups that possess a Sylow tower. We note that a group that is both an A-group and a p-group can only be abelian, thus the result of [27] does not touch p-groups of class 2.

While class-2 nilpotent groups have long been recognized as the chief bottleneck in the group isomorphism problem, this intuition has never been formalized. The ultimate formalization would reduce the general case to this case. As a first step, we consider a significant class without a chance of a complete structure theory at the opposite end of the spectrum: groups without abelian normal subgroups. Following [69], we call such groups *semisimple*[2]. We exhibit a polynomial-time isomorphism test for all semisimple groups. This result is joint work with László Babai and Youming Qiao [18]. The project was started together with Joshua A. Grochow [17]. In the first paper, we gave an algorithm to test isomorphism of semisimple groups of order n in time $n^{\log\log n}$, and polynomial-time algorithms assuming boundedness of certain parameters. We also showed that a polynomial-time isomorphism test of semisimple groups gives a permutational isomorphism test with running time simply exponential in the degree and polynomial in the order. We now have the converse reduction: from semisimple group isomorphism to permutational isomorphism of (transitive) permutation groups under the given time limits.

1.1.3 Definitions

Given two objects (e.g. (hyper)graphs or groups) an **isomorphism** between them is a bijection between the underlying sets that preserves the structure. For (hyper)graphs an isomorphism is a bijection between the sets of vertices that preserves the edge relationship. For groups an isomorphism is a bijection between the elements that is a homomorphism (it

2. We note that authors use the term 'semisimple group' in several different meanings (see e. g. [30, 44, 75, 76]). See Appendix A.2 for a more detailed comparison.

preserves the group operation). An **automorphism** is an isomorphism from an object to itself.

Hypergraphs: A graph is denoted by a pair (V, E), where V is the set of **vertices**, and E is a set of unordered pairs of vertices, the **edges**. A **hypergraph** is a pair (V, E), where V is a set of vertices, and E is a set of subsets of V, the **hyper-edges**. The **rank** of a hypergraph is the maximum size of a hyper-edge. A bijection between the vertices of two hypergraphs X and Y is a **hypergraph isomorphism** if it preserves the edge relation, i.e., a set of vertices is a hyper-edge in X if and only if the corresponding set of vertices is a hyper-edge in Y.

Semisimple Groups: We say a group is **semisimple** if it has no non-trivial abelian normal subgroups. A **group isomorphism** of two groups G, H is a bijection π between the group element that is a homomorphism, i.e., $(\forall g_1, g_2 \in G)((g_1 g_2)^\pi = g_1^\pi g_2^\pi)$.

Permutation Groups: Given a set Ω, the **symmetric group**, denoted $\mathrm{Sym}(\Omega)$, is the group of all permutations of Ω. A **permutation group** is a subgroup $G \leq \mathrm{Sym}(\Omega)$. The **domain** of G is Ω and the **degree** of G is $|\Omega|$.

Given two finite sets Ω and Δ, a permutation $\sigma \in \mathrm{Sym}(\Omega)$, and a bijection $\pi \colon \Omega \to \Delta$, the **conjugate** of σ by π is $\sigma^\pi := \pi^{-1}\sigma\pi \in \mathrm{Sym}(\Delta)$. Given a set of permutations $\Sigma \subseteq \mathrm{Sym}(\Omega)$, we define $\Sigma^\pi = \{\sigma^\pi : \sigma \in \Sigma\}$. If $G \leq \mathrm{Sym}(\Omega)$ and $H \leq \mathrm{Sym}(\Delta)$ are permutation groups, then a bijection $\pi \colon \Omega \to \Delta$ is a **permutational isomorphism** from G to H if $G^\pi = H$. We say G and H are **permutationally isomorphic** if there is a permutational isomorphism between them.

Codes: A **code** of length n over a finite alphabet Γ is a subset of Γ^A for some set A with $|A| = n$. An **equivalence** of the codes $\mathcal{A} \subseteq \Gamma^A$ and $\mathcal{B} \subseteq \Gamma^B$ is a bijection $A \to B$ that takes $\mathcal{A}$ to $\mathcal{B}$. If $|\Gamma| = 2$ then the code is a Boolean function or hypergraph, so the code equivalence problem is a generalization of the hypergraph isomorphism problem. We introduce a generalization of this problem, where in addition to permuting the coordinates, we are allowed to permute the symbols by some group G acting on Γ, independently for each coordinate. A **G-twisted equivalence** of two codes $\mathcal{A}$ and $\mathcal{B}$ of length n consists of a permutation π of the coordinates, and a list $(g_i)_{i=1}^n$ of n elements of G, one per coordinate, such that permuting the coordinates according to π, and applying g_i to the ith coordinate maps $\mathcal{A}$ to $\mathcal{B}$.

1.1.4 Results

We state the problems we consider, and our main results.

Problem 1.1.1. Hypergraph Isomorphism

Input: *Two hypergraphs X, Y of rank k over n vertices.*

Output: *$X \cong Y$?*

Theorem 1.1.2. *We can solve* Hypergraph Isomorphism *in time* $\exp(\widetilde{O}(k^2\sqrt{n}))$.

Problem 1.1.3. Semisimple Group Isomorphism

Input: *Two semisimple groups G, H, given by their Cayley tables.*

Output: *$G \cong H$?*

Theorem 1.1.4. *We can solve* Semisimple Group Isomorphism *in polynomial time.*

Our result for Semisimple Group Isomorphism depends on algorithms to test Permutational Isomorphism and Twisted Code Equivalence. Our results for these problems are below.

Problem 1.1.5. PERMUTATIONAL ISOMORPHISM

INPUT: *Two permutation groups G, H of order n and degree k.*

OUTPUT: *Are G and H permutationally isomorphic?*

Theorem 1.1.6. *We can solve* PERMUTATIONAL ISOMORPHISM *in time $c^k \mathrm{poly}(n)$, where c is some absolute constant.*

In the special case of *transitive groups* we can *list* all the isomorphisms in the same time bound. This result will be used in our algorithm for SEMISIMPLE GROUP ISOMORPHISM (cf. Chapter 6).

Theorem 1.1.7. *There exists an absolute constant c such that for all pairs of transitive permutation groups G and H of degree k,*
(a) the number of permutational automorphisms of G is at most $|G|\, c^k$;
(b) we can list *the set of all permutational isomorphisms of G and H in time $|G|\, c^k$.*

The solution of this case involves a detailed group-theoretic study of the structure of transitive groups (see Section 1.2.6). Because we represent permutation groups by generators, there is no inherent need for the factor of $|G|$ in the bounds. The question of a permutational isomorphism test for permutation groups in time simply exponential in the degree remains open.

Because of our use of algorithms for permutation groups (specifically, coset intersection), our bounds on the running time of the algorithm for TWISTED CODE EQUIVALENCE will be more effective when the groups have faithful low-degree permutation representations.

Problem 1.1.8. Twisted Code Equivalence

Input: *Two codes $\mathcal{A}$, $\mathcal{B}$ of length n over an alphabet Γ;*

A group $G \leq \mathrm{Sym}(\Gamma)$ given by a faithful permutation representation of degree d.

Output: *Are $\mathcal{A}$ and $\mathcal{B}$ G-twisted equivalent?*

Theorem 1.1.9. *We can solve* Twisted Code Equivalence *in time $c^{nd}\mathrm{poly}(|\mathcal{A}|, |G|, |\Gamma|)$, where c is some absolute constant.*

Note that in the special case where $G = \{\mathrm{id}\}$, twisted group code equivalence is code equivalence and the theorem gives a running time of $c^n\mathrm{poly}(|\Gamma||\mathcal{A}|)$. In particular, this is c^n for hypergraphs, matching the best known bound for general hypergraph isomorphism [57], although our constant is not as good.

1.1.5 Organization

In Section 1.2 we introduce group theoretic definitions and preliminaries. In Chapter 2 we review techniques that that were developed for Graph Isomorphism, and that we will use in our Hypergraph Isomorphism test, presented in Chapter 3. In Chapter 4 we talk about codes, and give our algorithm for twisted equivalence of codes. In Chapter 5 we give our algorithm for permutational isomorphism. Using the algorithms for twisted code equivalence and permutational isomorphism, we describe our polynomial-time isomorphism test for semisimple groups in Chapter 6. In Chapter 7 we conclude and discuss open problems. All chapters are meant to be readable independently; however, Chapter 3 will be more easily understood after reading Chapter 2, and in Chapter 6 we use Theorem 1.1.9 (proven in Chapter 4) and Theorem 1.1.7 (proven Section 5.2).

1.2 Group Theoretic Preliminaries

We review basic concepts of the theory of groups and permutation groups below; for a more detailed introduction we refer the reader to [69, 80, 55, 71].

1.2.1 Groups

In this thesis groups will always be finite. For groups G and H, we write $H \leq G$ to say that H is a **subgroup** of G. Given two groups G and H, a bijection $\pi\colon G \to H$ is a **group isomorphism** if it is a homomorphism, i. e., for all $g_1, g_2 \in G$, $(g_1 g_2)^\pi = g_1^\pi g_2^\pi$. An isomorphism of G with itself is an **automorphism**. The set of all automorphisms forms a group under composition. We denote the group of all automorphisms of a group G by $\mathrm{Aut}(G)$ and the set of G to H isomorphisms by $\mathrm{ISO}(G, H)$. We say G and H are isomorphic (and write $G \cong H$) if $\mathrm{ISO}(G, H)$ is not empty.

If $H \leq G$ and $g \in G$, then $Hg = \{hg \mid h \in H\}$ is a (right) **coset** of H in G. The set $\mathrm{ISO}(G, H)$ is either empty or a coset of $\mathrm{Aut}(G)$. Motivated by this fact, we define a **subcoset** of G to be either a coset $H\sigma$ of G or the empty set (so this is $H\sigma$ for empty H, but the empty set is *not* a subgroup). We highlight this definition since, although it is common in the isomorphism literature, it is not always used in group theory textbooks.

Definition 1.2.1. A **subcoset** of a group G is either a coset $H\sigma$ for some $H \leq G$ and $\sigma \in G$, or the empty set.

Under this definition, intersection of subcosets is again a subcoset, namely, for $H, K \leq G$, and $g_1, g_2 \in G$, $Hg_1 \cap Kg_2$ is either empty or it is a coset of the subgroup $H \cap K$.

$H \lhd G$ indicates that H is a normal subgroup of G. G is **simple** if $|G| > 1$ and G has no nontrivial normal subgroups.

The **commutator subgroup** of a group G, denoted G', is the subgroup generated by the set $\{[g, h] \mid g, h \in G\}$ of **commutators** $[g, h] = g^{-1}h^{-1}gh$. For $g \in G$, **conjugation by**

g means the $G \to G$ map $x \mapsto x^g = g^{-1}xg$. These maps are called **inner automorphisms** of G; they form the group $\mathrm{Inn}(G) \lhd \mathrm{Aut}(G)$.

For $S \subseteq G$ and $g \in G$ we set $S^g = \{s^g \mid s \in S\}$. The **normalizer** of $S \subseteq G$ in G, $N_G(S)$ is the subgroup $\{g \in G \mid S^g = S\}$. The **centralizer** of S is the subgroup $C_G(S) = \{g \in G \mid (\forall s \in S)(gs = sg)\}$. The notation $\varphi\colon G \hookrightarrow H$ means that φ is an **embedding** (injective hopmorphism) of G into H.

1.2.2 Permutation groups

The **symmetric group** $\mathrm{Sym}(\Omega)$ acting on the set Ω consists of all permutations of Ω; the operation is composition. The **alternating group** $\mathrm{Alt}(\Omega)$ is the (unique) subgroup of index 2 in $\mathrm{Sym}(\Omega)$ consisting of the even permutations. Subgroups of $\mathrm{Sym}(\Omega)$ are called **permutation groups**; we refer to Ω as the **permutation domain**. The **degree** of a permutation group is the size of the permutation domain. If $\Omega = [n] := \{1, \ldots, n\}$ then we write S_n for $\mathrm{Sym}(\Omega)$ and A_n for $\mathrm{Alt}(\Omega)$.

For a group G and a set Ω, a homomorphism $\varphi\colon G \to \mathrm{Sym}(\Omega)$ defines a **G-action** $x \mapsto x^\pi$ on Ω ($x \in \Omega, \pi \in G$). The action is **faithful** if the kernel $\ker(\varphi)$ is trivial. An action on Ω induces an action on the subsets of Ω: for $\Delta \subseteq \Omega$ we write $\Delta^\sigma = \{x^\sigma : x \in \Delta\}$. For a permutation group $H \leq \mathrm{Sym}(\Omega)$ we use the phrase "the action of G on Ω is H" if $\mathrm{Im}(\varphi) = H$.

The **orbit** of $x \in \Omega$ is the set $x^G := \{x^\pi : \pi \in G\}$. The orbits partition the permutation domain. The **length** of an orbit is its size. A permutation group $G \leq \mathrm{Sym}(\Omega)$ is **transitive** if $x^G = \Omega$ for some (and hence any) $x \in \Omega$.

The **stabilzer** of a point $x \in \Omega$ in G is $G_x = \{\sigma \in G \mid x^\sigma = x\}$. Note that $|x^G| = |G : G_x|$ because the right coset $G_x\sigma$ consists of all $\tau \in G$ such that $x^\tau = x^\sigma$.

Given an action $G \to \mathrm{Sym}(\Omega)$ and a set $\Delta \subseteq \Omega$, we define the **(pointwise) stabilizer** of Δ in G to be $G_\Delta = \{\sigma \in G : (\forall x \in \Delta)(x^\sigma = x)\}$. Applying this notation to the induced action on subsets we obtain the notation $G_{\{\Delta\}} = \{\sigma \in G : \Delta^\sigma = \Delta\}$ for the **setwise stabilizer** of Δ in G. A subset $\Delta \subseteq \Omega$ is **G-invariant** (or **G-stable**) if $(\forall \sigma \in G)(\Delta^\sigma = \Delta)$,

i. e., $G_{\{\Delta\}} = G$. If Δ is G-invariant, then G_Δ is normal in G, and the quotient group $G^\Delta := G/G_\Delta$ is the **restriction** of G to Δ.

Given a G-action on Ω, a nonempty set $B \subseteq \Omega$ is a **block of imprimitivity** (or simply "a block") if $(\forall \pi \in G)(B^\pi = B$ or $B \cap B^\pi = \emptyset)$. A transitive action is **primitive** if all blocks are trivial (the singletons and the whole domain Ω), and **imprimitive** otherwise. A **system of imprimitivity** is a partition of Ω consisting of a block and all its images under G. Systems of imprimitivitiy are precisely the G-invariant partitions of Ω. For a minimal nontrivial block B (w.r.t. set containment), the system $\mathfrak{B} = \{B^\pi \mid \pi \in G\}$ is maximal, and $G^B_{\{B\}}$ is primitive.

Given two finite sets Ω and Δ, a permutation $\sigma \in \mathrm{Sym}(\Omega)$, and a bijection $\pi \colon \Omega \to \Delta$, the **conjugate** of σ by π is $\sigma^\pi := \pi^{-1}\sigma\pi \in \mathrm{Sym}(\Delta)$. Given a set of permutations $\Sigma \subseteq \mathrm{Sym}(\Omega)$, we define $\Sigma^\pi = \{\sigma^\pi : \sigma \in \Sigma\}$.

If $G \leq \mathrm{Sym}(\Omega)$ and $H \leq \mathrm{Sym}(\Delta)$ are permutation groups, then a bijection $\pi \colon \Omega \to \Delta$ is a **permutational isomorphism** from G to H if $G^\pi = H$. We denote the set of all G to H permutational isomorphisms by $\mathrm{PISO}(G, H)$. We say G and H are **permutationally isomorphic** if $\mathrm{PISO}(G, H) \neq \emptyset$. Note that $\mathrm{PISO}(G, G) = N_{\mathrm{Sym}(\Omega)}(G)$, the normalizer of G in $\mathrm{Sym}(\Omega)$. Each $\pi \in \mathrm{PISO}(G, H)$ induces an isomorphism $\widehat{\pi} : G \to H$. Let $\widehat{\mathrm{PISO}}(G, H) = \{\widehat{\pi} \mid \pi \in \mathrm{PISO}(G, H)\}$.

1.2.3 Algorithms for permutation groups

Permutation groups will be represented by a list of generators; to **compute a subgroup** means to find generators for the subgroup. To **recognize a subgroup** means to be able to test membership in the subgroup (of elements of the supergroup).

The basic tasks of membership testing, computing the order, finding the normal closure, finding pointwise stabilizers, can be done in polynomial time ([72, 73, 39, 51], cf. [71]) and in fact in NC [24]. It follows that from any list of generators, a minimal (non-redundant) list can be computed in polynomial time. Such a list has $\leq 2n$ elements, where $n = |\Omega|$ is the degree [8]. Many more advanced tasks, such as finding a composition series [56], and finding Sylow subgroups [47, 48] can also be done in polynomial time; composition series can even be found in NC [24].

The following observation will allow us to check whether or not a bijection is a permutational isomorphism.

Proposition 1.2.2. *Given two permutation groups G and H of degree n, and a bijection f of the domains, we can decide whether or not f is a permutational isomorphism of G and H in polynomial time in n.*

Proof. We can check membership of the f-images of the generators of G in H and vice versa [39].

$\square$

1.2.4 Isomorphisms vs. permutational isomorphisms

We examine the connection between (abstract) isomorphisms and permutational isomorphisms. Recall that a permutational isomorphism is a bijection between the domains, while an abstract isomorphism is a bijection between the group elements.

Let $G, H \leq S_n$. A permutational isomorphism $\pi \in \mathrm{PISO}(G, H)$ induces a permutation $\widehat{\pi}$ of the group elements that is an (abstract) isomorphism. Two permutational isomorphisms $\pi, \sigma \in \mathrm{PISO}(G, H)$ induce the same isomorphism if and only if $\pi\sigma^{-1} \in C(G)$ (recall $C(G) = C_{S_n}(G)$ is the centralizer of G in S_n). Therefore $|\mathrm{PISO}(G)| \leq |C(G)| \cdot |\mathrm{ISO}(G, H)|$. Using this it is easy to prove that for transitive groups $|\mathrm{PISO}(G, H)|/|\mathrm{ISO}(G, H)| \leq n$ (cf. [35, page 109]). Indeed for transitive groups, there are at most n permutational isomorphisms corresponding to a specific abstract isomorphism, and we can list them in polynomial time.

Proposition 1.2.3. *Given two transitive permutation groups $G, H \leq S_n$, and an isomorphism $\phi \in \mathrm{ISO}(G, H)$, we can list the (at most n) permutational isomorphisms $\pi \in \mathrm{PISO}(G, H)$ corresponding to ϕ in time $O(n^3)$.*

Proof. Pick a set of generators $S \subseteq G$ for G; then S^{ϕ} is the corresponding set of generators for H. We construct two directed edge-colored graphs $X(G)$, and $X(H)$, over the vertex set $[n]$. For each generator $\sigma \in S$, we add to $X(G)$ edges of color σ corresponding to the cycle decomposition of σ. Similarly for every $\sigma \in S$, we add to $X(H)$ edges of color σ corresponding to the cycle decomposition of σ^{ϕ} ($\in S^{\phi}$). (Graph) Isomorphisms of $X(G)$ and $X(H)$ are permutational isomorphisms between G and H; and we can test isomorphism of these colored graphs in linear time. Recall that isomorphisms of colored graphs preserve the colors by definition.

To test isomorphism of these graphs, pick any vertex x in $X(G)$, and run breadth first search starting at x (because the groups are transitive, the choice will not matter). We assign to every vertex the unique word corresponding to the color of the path to that vertex in the BFS. Now repeat the same process for $X(H)$. If the labelings of the vertices do not yield a bijection, then reject. Otherwise check whether the bijection is indeed an isomorphism of the two graphs. The running time is linear in the size of the graphs.

To list all permutational isomorphisms corresponding to φ, we need to try every possible choice of root vertex x for the BFS. Therefore the total running time will be n times linear in the size of the graphs. The graphs have $O(n^2)$ edges, since there is always a set of at most $2n$ generators. Therefore the total running time is $O(n^3)$.

□

1.2.5 Bounds on primitive groups

The following fact will be central to our strategy for both Hypergraph Isomorphism and Permutational Isomorphism.

Theorem 1.2.4 (cf. [33, Theorem 6.1]). *(a) Let $G \leq S_n$ be a primitive group other than S_n or A_n. If n is sufficiently large then $|G| \leq n^{\sqrt{n}}$. (b) For every $\epsilon > 0$, there exists a $c(\epsilon)$ such that for every n, if $G \leq S_n$ is primitive and $|G| > 2^{n^{\epsilon}}$, then G can be generated by $c(\epsilon)$ generators.*

Proof. As described in Theorem 6.1 and the remarks after it in [33], all primitive groups have order $n^{O(n^{1/3})}$, except the following classes of groups: S_n and A_n in their natural action; $S_\ell^{(2)}$ and $A_\ell^{(2)}$, the induced actions of S_ℓ and A_ℓ resp. on subsets of size 2, where $\binom{\ell}{2} = n$; and subgroups of $S_m \wr S_2$, containing $A_m \times A_m$, where $m^2 = n$. S_n, A_n and $S_\ell^{(2)}$ are generated by two elements, while the subgroups of $S_m \wr S_2$ can be generated by 3 elements.

□

The above fact is a consequence of the classification of finite simple groups (cf. [33]). For part (a), a slightly weaker $\exp(4\sqrt{n}\ln^2 n)$ upper bound, sufficient for our applications, has an elementary combinatorial proof [5, 6, 66]. However, for the Permutational Isomorphism problem, we will need part (b) of the theorem above, the proof of which requires the classification of finite simple groups.

In our algorithms we will distinguish between two cases: when the primitive groups involved are symmetric or alternating, and when they are not. If the groups are symmetric or alternating, we will be able to make use of the structure. When the groups are not the symmetric or alternating groups, then they will be "small," so we will be able to enumerate them.

We shall also need the following related but elementary estimate.

Fact 1.2.5. Let $G = A_n$ or S_n and let $H < G$ be transitive but not A_n or S_n. If n is sufficiently large then $|G : H| \geq 2^n/\sqrt{2n}$.

Bounding the order of the permutational automorphism group of non-alternating or symmetric primitive groups will be the base case for our algorithm for Permutational Isomorphism. The following fact is easy to prove from Theorem 1.2.4 and Proposition 1.2.3.

Lemma 1.2.6. *If $G \leq S_n$ is a primitive group other than S_n or A_n, then (a) $|\mathrm{PAut}(G)| \leq \exp(\widetilde{O}(\sqrt{n}))$; moreover (b) for every $H \leq S_n$, we can list $\mathrm{PISO}(G, H)$ in time $O(\exp(\widetilde{O}(\sqrt{n}))$.*

Luks's polynomial time isomorphism test for bounded degree graphs [55] motivated the following definition.

Definition 1.2.7. A group G is a Γ_k**-group** if all composition factors of G are subgroups of S_k.

The observation was that if X is a graph of degree d, and e is any edge in the graph, then $\mathrm{Aut}(X)_e$ (the stabilizer of the edge) is a Γ_{d-1}-group. The following bound, motivated by Luks's approach, follows from a more precise classification (cf. [15]).

Theorem 1.2.8 (Babai, Cameron, Pálfy [15])**.** *If $G \leq S_n$ is a Γ_k group, then $|G| \leq n^{c_k}$ for some constant c_k depending only on k.*

This theorem can be used to obtain a simpler polynomial time algorithm for graphs of bounded valence.

1.2.6 Structure trees

Structure trees underlie Luks's divide and conquer approach (cf. Section 2.5.2).

Definition 1.2.9 (invariant tree)**.** An **invariant tree** for a transitive group $G \leq \mathrm{Sym}(\Omega)$ is a rooted tree whose set of leaves is Ω and to which the G-action extends. (Such extension is necessarily unique.)

Definition 1.2.10 (structure tree)**.** An invariant tree for $G \leq \mathrm{Sym}(\Omega)$ is a **structure tree** if every internal node has at least 2 children, and for every internal node u of the tree, the action of the stabilizer G_u on the set of children of u is primitive.

An invariant tree corresponds to a hierarchy of block systems. A structure tree corresponds to a maximal such hierarchy. We label every node v in a tree by the subset $B(v)$ of leaves in the subtree rooted at v. The k^{th} **layer** of a structure tree is the set of all nodes at distance k from the root. The **depth** of a structure tree is d if the leaves are on the d^{th} layer. Note that in a structure tree all leaves must be on the same layer. Indeed for all nodes u, v on the same layer of a structure tree, we have $G_u^{B(u)} \cong G_v^{B(v)}$. This follows from the fact that every node corresponds to a block B in a system of imprimitivity, and the blocks corresponding to nodes on the same layer are conjugates of B. This in particular also implies that a layer of the tree is completely determined by just one block on that layer.

We note that structure trees do always exist, and can be constructed efficiently. Indeed in our algorithm for Permutational Isomorphism of transitive groups, we will need to list all structure trees of a transitive permutation group. The following lemma allows us to do so, and bounds the cost of such an operation.

Lemma 1.2.11. *Let $G \leq \mathrm{Sym}(\Omega)$ be a transitive permutation group of degree n. Then (a) G has at most $n^{2\log n}$ structure trees; (b) we can list all structure trees of G in time $O(n^{2\log n}\mathrm{poly}(n))$.*

Proof. (a) Two minimal blocks intersect either trivially or in exactly one point. Therefore the number of minimal blocks is at most $\binom{n}{2}$, since a pair of points uniquely defines one. Every layer of a structure tree is completely determined by just one block on that layer. Therefore a structure tree of depth ℓ is completely determined by a nested sequence of blocks $\{x \in \Omega\} = B_\ell \subseteq B_{\ell-1} \subseteq \cdots \subseteq B_0 = \Omega$ that cannot be futher refined. Since these must be blocks of imprimitivity, we have $|B_i| \geq 2|B_{i+1}|$, so $\ell \leq \log n$. At step i we have at most $\binom{n}{2}$ choices, so the total number of such sequences is at most $n^{2\log n}$.

(b) To construct all structure trees, we will try every possible sequence of subsets B_i as above. To find all the minimal blocks for one layer, we first introduce some notation. An *orbital* of G is an orbit of the induced G-action pairs. For every orbital Σ_i of G, we construct a digraph $X_i = (\Omega, E_i)$, where for $x, y \in \Omega$, $(x, y) \in E_i$ if $(x, y) \in \Sigma_i$. Now fix a point $x \in \Omega$. For every orbital, look at the connected component containing x. These form a set system of at most $n - 1$ sets. Within this set system, take the minimal ones. These are the minimal blocks containing x. The running time is polynomial for each layer.

□

1.2.7 Products and diagonals

Given groups $G_1, \ldots, G_r$, we write $\prod_{i=1}^r G_i$ for the direct (Cartesian) product $G_1 \times \cdots \times G_r$. We write $\pi_j \colon \prod_{i=1}^r G_i \to G_j$ for the projection map onto the j-th factor. A **subdirect product** of $G_1, \ldots, G_r$ is a subgroup $H \leq \prod_{i=1}^r G_i$ such that $\pi_j(H) = G_j$ for each j.

A particularly important example of subdirect product is the diagonal subgroup of a direct product of isomorphic groups.

Definition 1.2.12 (diagonal)**.** Let $V_1, \ldots, V_r$ be isomorphic groups, $(\forall i)(V_i \cong T)$. A **diagonal** of $(V_1, \ldots, V_r)$ is an embedding $\phi : T \hookrightarrow \prod_{i=1}^r V_i$ such that $\mathrm{Im}(\phi)$ is a subdirect product of the V_i. We use $\mathrm{diag}(V_1 \times \cdots \times V_r)$ to denote the image of some diagonal of $(V_1, \ldots, V_r)$.

More generally, assume we have a system of groups $(V_{1,1}, \ldots, V_{1,k_1}), \ldots, (V_{r,1}, \ldots, V_{r,k_r})$, where for every $i \leq r$, and every $j \leq k_i$, we have $V_{i,j} \cong T_i$. Then a **diagonal product** (or product of diagonals) of the system $(V_{i,j})$ is an embedding $\phi_i \times \cdots \times \phi_r : T_1 \times \cdots \times T_r \hookrightarrow \prod_{j=1}^{k_1} V_{1,j} \times \cdots \times \prod_{j=1}^{k_r} V_{r,j}$, where each ϕ_i is a diagonal of $(V_{i,j})_{j=1}^{k_i}$. For every i, we call the set of factors $(V_{i,j})_{j=1}^{k_i}$ a **block** of the diagonal product.

The **standard diagonal** of T^k is the map $\Delta : t \to (t, \ldots, t)$. Similarly, the **standard diagonal product** of $\prod_{i=1}^r T_i^{k_i}$ is the map $\Delta = \Delta_1 \times \cdots \times \Delta_r$, where for every i, Δ_i is the standard diagonal for $T_i^{k_i}$.

Definition 1.2.13 (permutational diagonal)**.** Let $V_i \leq \mathrm{Sym}(\Omega_i)$ $(i \in [r])$ be permutation groups, permutationally isomorphic to $T \leq \mathrm{Sym}(\Xi)$. A **permutational diagonal** of $\prod_{i=1}^r V_i$ is list of permutational isomorphisms $\varphi_i \in \mathrm{PISO}(T, V_i)$ (so $\varphi_i : \Xi \to \Omega_i$). Let $f_{ij} = \varphi_i^{-1}\varphi_j$. Note that $f_{ij} \in \mathrm{PISO}(V_i, V_j)$. We refer to the f_{ij} as the bijections defininig this diagonal.

We use $\mathrm{pdiag}(V_1 \times \cdots \times V_r)$ to denote some permutational diagonal of $\prod_{i=1}^r V_i$. We can generalize to a permutational diagonal to a system of permutation groups as in Definition 1.2.12.

For example, given $G \leq \mathrm{Sym}(\Omega)$, consider the induced action of G^k on Ω^k. The standard diagonal T of G^k acting on $\{(\omega, \ldots, \omega) \mid \omega \in \Omega\}$ is a permutational diagonal. Note that any permutational diagonal induces a diagonal.

Fact 1.2.14. Let $G \leq H_1 \times \cdots \times H_m$ be a subdirect product, where each H_i is a simple group. Then G is a direct product of diagonals. In other words, after some rearrangement of the factors into blocks, and the selection of isomorphisms between factors in each block, we may write:

$$G = \mathrm{diag}(H_1 \times \cdots \times H_{i_1}) \times \mathrm{diag}(H_{i_1+1} \times \cdots \times H_{i_2}) \times \cdots \times \mathrm{diag}(H_{i_r+1} \times \cdots \times H_m).$$

Fact 1.2.15. Let $G \leq \mathrm{Alt}(\Omega_1) \times \cdots \times \mathrm{Alt}(\Omega_m)$, where $(\forall i)(|\Omega_i| \geq 5, \neq 6)$ be a subdirect product of alternating groups. Then G is a direct product of permutational diagonals. In other words, after some rearrangement of the factors into blocks, and the selection of permutational isomorphisms between factors in each block, we may write:

$$G = \mathrm{pdiag}(\mathrm{Alt}(\Omega_1) \times \cdots \times \mathrm{Alt}(\Omega_{i_1})) \times \cdots \times \mathrm{pdiag}(\mathrm{Alt}(\Omega_{i_r+1}) \times \cdots \times \mathrm{Alt}(\Omega_m)).$$

This is a consequence of the fact that $\mathrm{Aut}(A_n) = S_n$ for all $n \geq 5, n \neq 6$; (cf. e. g., Lemma 2.1 in [14]).

Given a group K with an action on another group H given by $\theta\colon K \to \mathrm{Aut}(H)$, the **semidirect product** $H \rtimes_\theta K$ is a group with underlying set $H \times K = \{(h, k) \colon h \in H, k \in K\}$ and multiplication defined by: $(h_1, k_1)(h_2, k_2) = (h_1 h_2^{\theta(k_1^{-1})}, k_1 k_2)$. When the action θ is understood, we write simply $H \rtimes K$.

If $\theta\colon K \to \mathrm{Sym}(A)$ is a permutation action of K on the set A, the **wreath product** $H \wr_\theta K$ is defined as $H^A \rtimes_{\overline{\theta}} K$, where $\overline{\theta}\colon K \to \mathrm{Aut}(H^A)$ is the action of K on H^A by permuting the factors. That is, $(h_1, \ldots, h_n)^{\overline{\theta}(k)} = (h_{1^{\theta(k)}}, \ldots, h_{n^{\theta(k)}})$, where we have assumed $A = [n]$. If $K \leq \mathrm{Sym}(A)$ is a permutation group, we write simply $H \wr K = H^A \rtimes K$.

CHAPTER 2
GRAPH ISOMORPHISM: HISTORY AND TECHNIQUES

This chapter is not intended to be an exhaustive review of the literature on the Graph Isomorphism problem. We give a brief general overview, and focus on those techniques that we use in our algorithm for Hypergraph Isomorphism. We refer the reader to [68, 9, 31] for more complete overviews.

2.1 History

Early algorithms were discovered for geometric classes of graphs: planar graphs in linear time (Hopcroft and Wong, 1974 [43], building on Hopcroft and Tarjan, 1972 [42]); graphs of bounded genus in polynomial time (Lichtenstein [54], Filotti and Mayer [36], Miller [60]; 1980). The first techniques for general graph isomorphism were partition, refinement and individualization heuristics (see Section 2.4). These techniques are at the heart of most practical Graph Isomorphism algorithms (see e.g. McKay's Nauty [59]), and they provably work in some cases. They solve Graph Isomorphism in linear time for almost all graphs (Babai, Erdős, Selkow [19]; Babai, Kučera [22], 1979). An important class of graphs is that of *strongly regular* graphs. In 1980 Babai showed that the partition-refine heuristics work in time $\exp(\widetilde{O}(\sqrt{n}))$ for strongly regular graphs [3]; and in 1996 Spielman proved that they give an $\exp(\widetilde{O}(n^{1/3}))$ for this class of graphs in a highly non-trivial work, showing the power of these techniques [74]. However, in general these techniques alone fail badly. In 1991 Cai Fürer and Immerman produced examples of non-isomorphic classes of graphs that require exponential time to be distinguished by these techniques [31].

In 1979 Babai introduced group theoretic techniques to the Graph Isomorphism problem [2]. The result in this first paper was that isomorphism of graphs with bounded color classes is in polynomial time. The Cai-Fürer-Immerman graphs have color class size 4, hence already this first application beats the most powerful combinatorial techniques (although Cai Fürer and Immerman's result appeared more than a decade later, and so this was not known at the time). Luks developed techniques that utilize the group theory to greater depth. The crowning achievement of the group theoretic approach is Luks's polynomial time isomorphism test for bounded degree graphs [55]. This later led to the best known algorithm for graph isomorphism. Babai noticed that combining a combinatorial trick due to Zemlyachencho [84] and Luks's bounded degree algorithm gave a moderately exponential algorithm for the general graph isomorphism problem [21]. Later Luks improved the running time to $n^{\sqrt{n \log n}}$ ([21], see also [23]). Other algorithms have been developed using the group theoretic techniques, for different classes of graphs. Isomorphism of graphs with bounded eigenvalue multiplicity can be tested in polynomial time [20], and isomorphism of tournaments can be tested in quasipolynomial time [23].

2.2 Definitions

Definition 2.2.1 (Graph Isomorphism)**.** Let $X = (V, E)$, $Y = (U, F)$ be graphs. An **isomorphism** between X and Y is a bijection $\phi : V \to U$ that preserves edges:

$$(v, w) \in E \iff (v^\phi, w^\phi) \in F.$$

We denote the set of all X to Y isomorphisms by $\mathrm{ISO}(X, Y)$. An isomorphism from X to itself is an **automorphism**. The set of automorphisms of X, denoted $\mathrm{Aut}(X)$ forms a group under composition. Let X and Y be graphs over the same vertex set V. Then $\mathrm{ISO}(X, Y)$ is either empty or a coset of $\mathrm{Aut}(X)$ in $\mathrm{Sym}(V)$.

2.2.1 Colorings

Definition 2.2.2 (Graph Coloring)**.** For a graph $X = (V, E)$, and a set $\mathcal{C}$ of colors, a **vertex-coloring** of X is a function $c : V \to \mathcal{C}$; an **edge-coloring** of X is a function $c : E \to \mathcal{C}$.

We use the term 'coloring' to mean either a vertex-coloring or an edge-coloring. Sometimes a vertex-coloring is called a **labeling** and the colors are called **labels**. A **color class** is a set of vertices (or edges) that receive the same color. A vertex-coloring (edge-coloring respectively) partitions the vertices (edges respectively) into color classes. A coloring c' is a **refinement** of a coloring c if the partition into color classes induced by c' is a refinement of the partition induced by c.

An **invariant coloring** is a coloring that is preserved by all automorphisms. More precisely a vertex-coloring is invariant if $(\forall \phi \in \mathrm{Aut}(X))(\forall v \in V)(c(v^\phi) = c(v))$, and similarly, an edge coloring is invariant if $(\forall \phi \in \mathrm{Aut}(X))(\forall e \in E)(c(e^\phi) = c(e))$. For example, coloring the vertices by degrees is an invariant vertex-coloring. Coloring the edges by the number of triangles they appear in is an invariant edge-coloring.

A **colored graph** is a triple $X = (V, E, c)$, where (V, E) is a graph, and c is a coloring. Isomorphisms of colored graphs preserve the colors by definition. More precisely, given two vertex-colored graphs $X = (V, E; c)$, $Y = (U, F; d)$, with $c, d : V \to \mathcal{C}$, an isomorphism between X and Y is a bijection $\phi : V \to U$ such that $(\forall v \in V)(d(v^\phi) = c(v))$ and $(v, w) \in E \iff (v^\phi, w^\phi) \in F$. (The definition is analogous for edge-colored graphs.)

2.2.2 Invariants, canonical labelings, and canonical forms

Throughout this discussion, $\mathcal{X}$ will be a class for which isomorphism is defined. We will think of $\mathcal{X}$ as the class of (hyper)graphs, groups, a subclass of these (e.g. bipartite graphs), or a generalization (e.g. directed graphs).

Definition 2.2.3 (Invariant)**.** An **invariant** for a class $\mathcal{X}$ is a function $f : \mathcal{X} \to \Omega$, for some set Ω such that $(\forall X, Y \in \mathcal{X})(X \cong Y \Rightarrow f(X) = f(Y))$.

Invariants can provide proof of non-isomorphism and are useful to reduce the search space for isomorphisms in general.

Some examples of graph invariants are: the number of edges, the multiset of degrees, the multiset of eigenvalues, the characteristic polynomial. Planarity, connectivity, and all commonly considered graph properties are also invariants. To phrase it in the language of the definition, any graph property is a function f from the set of all graphs to $\{0,1\}$, where a graph X has the property if and only if $f(X) = 1$.

Some invariants for groups are: the order, the multiset of orders of the elements, the multiset of (isomorphism types of the) composition factors.

Definition 2.2.4 (Complete Invariant)**.** A **complete invariant** for a class $\mathcal{X}$ is a function $f : \mathcal{X} \to \Omega$, where Ω is some set of labels, such that $(\forall X, Y \in \mathcal{X})(X \cong Y \iff f(X) = f(Y))$.

Definition 2.2.5 (Canonical Form)**.** A **canonical form** for a class $\mathcal{X}$ is a complete invariant $f : \mathcal{X} \to \mathcal{X}$.

Definition 2.2.6 (Canonical Labeling)**.** A **canonical labeling** procedure for a class $\mathcal{X}$ of (hyper)graphs is a complete invariant $f : \mathcal{X} \to \mathcal{K}$, where $\mathcal{K}$ is the set of labelings.

Recall that a labeling is a vertex-coloring. In other words, a canonical labeling procedure assigns a labeling to each graph that uniquely identifies its isomorphism type.

2.3 Computational Complexity

The Graph Isomorphism problem is of particular interest in computational complexity because it is a natural problem that is easily seen to be in NP, however it is not known to be in P and it is not believed to be NP-complete. In fact, Graph Isomorphism is in NP $\cap$ coAM ([40], cf. [26]), and therefore it is not NP-complete unless the polynomial time hierarchy collapses ([29], cf. [26]). Even before the introduction of interactive proofs allowed for these results to be discovered, Graph Isomorphism was not believed to be NP-complete, since the decision version is as hard as the counting version, while for NP-complete problems the counting versions seem to be much harder. Indeed counting isomorphisms of two graphs is polynomial-time equivalent to Graph Isomorphism (Babai, Mathon [1, 58]).

2.3.1 Reductions

Problem 2.3.1 (Graph Isomorphism)**.** *Given two graphs, are they isomorphic?*

We will ask the same question for special classes of graphs. For example, Bipartite Graph Isomorphism asks whether two bipartite graphs are isomorphic, Vertex-Colored Graph Isomorphism asks whether two vertex-colored graphs are isomorphic (recall that these isomorphisms must preserve the colors by definition). As always in complexity theory, when we cannot prove separation results, we try to find reductions between problems. A problem is GI-hard if Graph Isomorphism can be reduced to it in polynomial time, and GI-complete if it is polynomial-time equivalent to Graph Isomorphism. The following theorems show that many problems are polynomial time equivalent to Graph Isomorphism. The proofs consist mainly of constructions using combinatorial gadgets.

Theorem 2.3.2 (Booth, Colbourn [28]; Miller [61])**.** *The following are polynomial-time equivalent:*

- Graph Isomorphism*,*
- Bipartite Graph Isomorphism*,*
- Isomorphism of Vertex-Colored graphs*,*
- Directed Graph Isomorphism*,*
- Isomorphism of Systems of Binary Relations *(directed graphs with colored vertices and edges).*

Theorem 2.3.3 (Babai, Mathon [1, 58])**.** *The following are polynomial-time equivalent:*

- Graph Isomorphism*,*
- *given X, find generators of* $\mathrm{Aut}(X)$*,*
- *given X, compute* $|\mathrm{Aut}(X)|$*,*
- *given X, find the orbits of* $\mathrm{Aut}(X)$*,*
- *given X, Y, compute* $|\mathrm{ISO}(X, Y)|$*.*

In a precise sense, Graph Isomorphism is universal among isomorphism problems of explicit structures. First let us define what is meant by explicit structures.

Definition 2.3.4. A **relational structure** is a $t+1$-tuple $(\Omega; R_1, \dots, R_t)$, where Ω is a set, and $\forall i = 1 \to t, R_i \subseteq \Omega^{k_i}$ is an ordered k_i-tuple. The **arity** of the relational structure is $\max_i\{k_i\}$.

Graphs are relational structures of arity 2; k-hypergraphs are relational structures of arity k. Algebraic structures are also relational structures, for example, groups are relational structures of arity 3.

An isomorphism of two relational structures, $(\Omega; R_1, \dots, R_t)$ and $(\Delta, R'_1, \dots, R'_t)$ is a bijection $f : \Omega \to \Delta$ that preserves the relations, i. e., $(\forall i)(S \in R_i \iff f(S) \in R'_i)$.

Theorem 2.3.5 (Miller [61]). *Isomorphism of explicit relational structures polynomial-time reduces to* Graph Isomorphism.

Here explicit means that the relations have to be listed. For example, groups given by Cayley tables are explicit relational structures, but permutation groups given by sets of generators are not.

Observation 2.3.6. Hypergraph Isomorphism *Karp-reduces to* Graph Isomorphism.

This reduction does not preserve the number of vertices, and hence an algorithm with running time $O(\exp(n^c))$ for Graph Isomorphism does not imply an algorithm with the same running time for Hypergraph Isomorphism.

Let us give a proof of a more precise version of observation above in the case of 4-uniform hypergraphs.

Observation 2.3.7 (cf. Babai, Luks [23]). *Isomorphism of 4-uniform hypergraphs with n vertices Karp reduces to isomorphism of graphs with $O(n^2)$ vertices.*

Proof. Let $X = (V, E)$, $Y = (W, F)$ be 4-uniform hypergraphs. We will construct edge-colored graphs $X' = (V', E')$ and $Y' = (W', F')$ such that $X \cong Y \iff X' \cong Y'$. The construction is the same for the two hypergraphs, so let us focus on X'. The vertex set V' is the set of (unordered) pairs of vertices in V. We add red edges between pairs that share a node: $\{u, v\}, \{u, w\}$. We add black edges between two pairs $\{u, v\}, \{w, x\}$ if and only if $\{u, v, w, x\} \in E$. It is easy to see that this construction preserves isomorphisms. To remove the edge colors, we use a standard technique that reduces isomorphism of edge-colored graphs to Graph Isomorphism. This reduction increases the number of vertices by a constant factor when the number of colors is bounded (in our case the number of colors is 2).

□

As was noted already in 1983 by Babai and Luks [23], this simple construction implies that if we could solve Graph Isomorphism in time $\exp(C^n)$, then we could solve 4-uniform Hypergraph Isomorphism in time $\exp(C^{2n})$, and the lack of a better than simply exponential algorithm for 4-uniform Hypergraph Isomorphism (and in fact, even for 3-uniform hypergraphs) was an obstacle to improving the best known algorithm for Graph Isomorphism. Our algorithm to test isomorphism of hypergraphs remove this obstacle.

2.3.2 Relationship to coset intersection

A problem that is closely related to Graph Isomorphism is the Coset Intersection problem.

Problem 2.3.8. Coset Intersection

Input: *generators for two permutation groups $G, H \leq S_n$,*

two permutations $\pi, \sigma \in S_n$.

Output: *$G\pi \cap H\sigma = \emptyset$?*

A coset $G\pi$ is given by a coset representative $\pi' \in G\pi$ and a list of generators of the group G. The following observation is of Luks gives the connection to Graph Isomorphism.

Theorem 2.3.9 (Luks [55]). Graph Isomorphism *Karp-reduces to* Coset Intersec-

TION.

Proof. Let $X = (V, E)$ and $Y = (W, F)$ be graphs over n vertices. W.l.o.g $V = W = [n]$. Let $L \subseteq S_{\binom{n}{2}}$ be the coset that preserves the edges, that is $L = \{\sigma \in S_{\binom{n}{2}} : E^\sigma = F\}$. Note that L is a coset of $\mathrm{Sym}(E) \times \mathrm{Sym}([\binom{n}{2}] \setminus E)$, which is easy to compute. Then $\mathrm{ISO}(X, Y) = L \cap S_n^{(2)}$, where $S_n^{(2)}$ is the induced action of S_n on pairs.

$\square$

We will make use of coset intersection subroutines in our algorithms to test Hypergraph Isomorphism and Twisted Code Equivalence. We use the following result.

Theorem 2.3.10 (Babai 1983 [7], see also [11, 21]). *Given generators for $G, H \leq S_n$ and $\pi, \sigma \in S_n$, the subcoset $G\pi \cap H\sigma$ can be found in time* $\exp(\widetilde{O}(\sqrt{n}))$.

In fact, Luks's simply exponential algorithm for the same problem [57] would suffice for the Twisted Code Equivalence problem. For Hypergraph Isomorphism instead we will need the full force of the above result.

2.4 Partition and Refinement Techniques

Throughout this section n will denote the number of vertices of the graph, and m the number of edges.

2.4.1 Naive refinement

In this subsection we describe an effective heuristic for graph isomorphism (cf. Read and Corneil [68]). This heuristic is at the heart of most practical graph isomorphism algorithms (see e.g. McKay's Nauty [59]). For almost all graphs this technique provably works in polynomial time, giving some theoretical explanation for its success [19, 22].

A procedure to obtain an invariant vertex-coloring where each vertex gets a unique color is a canonical labeling procedure (see Section 2.2.2 for definitions). The goal of the refinement technique is to get closer to such a coloring. Given a vertex-colored graph $X = (V, E; c)$ with color set $\mathcal{C}$, one "refinement step" computes a new vertex-coloring c' that is a refinement of c. The new color of a node is defined by the multiset of colors of the neighbors, more precisely,

$$c'(v) := (c(v), \{c(u) \mid (u, v) \in E\}) \qquad \forall v \in V.$$

Note that any isomorphism that preserves c must preserve c'. Hence if c was invariant, then so is c'. The set of potential colors of c' may be large, but at most n colors actually appear, so we can number those colors that do appear, and assume $c' : V \to [n]$. This trick allows us to recursively apply the refinement step until we reach a 'stable coloring,' that is, any further refinement does not change it. The number of refinement steps that can occur is at most n, since if $c \neq c'$, at least one color class must be partitioned. Therefore the overall running time is $O(n(n + m))$.

This is a general procedure that works for any initial vertex-coloring of a graph. Often it is used as a subroutine in isomorphism algorithms. The Naive Refinement algorithm starts by coloring each vertex according to its degree, and recursively refines the coloring until a stable coloring is reached. Note that this algorithm does not even being to partition if the graph is regular. Therefore different techniques are needed for these graphs. However, this very simple application can be shown to canonically label almost all graphs in just one refinement step.

Theorem 2.4.1 (Babai-Erdős-Selkow [19]). *Apart from a $n^{-1/7}$ fraction of graphs, the Naive Refinement procedure finds a canonical labeling in one refinement step.*

The heart of the proof is an estimate on the top degrees of vertices in a random graph.

Theorem 2.4.2 (Babai-Erdős-Selkow [19]). *For every constant c, for almost all graphs, the highest $c \log n$ degrees are distinct.*

Using a more complicated argument, Naive Refinement was shown to canonically label all but an exponentially small fraction of graphs in just three refinement steps.

Theorem 2.4.3 (Babai-Kučera [22]). *Apart from a c^{-n} fraction of graphs, the Naive Refinement procedure finds a canonical labeling in at most 3 rounds.*

2.4.2 Weisfeiler-Lehman process

In 1968 Weisfeiler and Lehman introduced a generalization of naive refinement that looks at colorings of ordered pairs of vertices [78] (cf. [79]). We call this process the Weisfeiler-Lehman process (W-L for short). Note that for any non-trivial graph (not the empty graph or the complete graph), we have a non-trivial partition of the pairs into diagonal pairs, edges, and non-edges.

We can refine the coloring of the pairs as before, by looking at "neighbors." A neighbor is a pair that shares a vertex. Given a coloring $c : V \times V \to \mathcal{C}$, we create the refined coloring $c' : V \times V \to \mathcal{C}$ according to the multiset of colors of the neighbors:

$$c'(u, v) = \{c(w, v) \mid w \in V\} \cup \{c(u, w) \mid w \in V\}$$

Note that, as for naive refinement, we are be increasing the potential number of colors, but there are at most n^2 distinct colors at any time, so by re-numbering at every step, we can assume our set of colors always has size at most n^2. Each refinement step takes time $O(n^2 m)$, and there are at most n^2 refinement steps, so overall the running time is $O(n^4 m)$. In fact, one refinement step can be modeled by a matrix multiplication, and by applying repeated squaring, and a data management trick, we can reach a stable coloring in $O(\log n)$ rounds, so the overall cost is $O(\log n)$ matrix multiplications.

A graph is **strongly regular** if the number of common neighbors of two vertices depends only on whether the two vertices are adjacent or not. In other words, a strongly regular graph with parameters (d, λ, μ) is a regular graph of degree d, where the vertices of every edge have λ common neighbors, and the vertices of every non-edge share μ common neighbors. Strongly regular graphs are already stable under W-L, so again, we need different techniques for these graphs.

One can obtain more powerful refinement techniques by looking at ordered k-tuples of vertices. Babai introduced the term k-dimensional Weisfeiler-Lehman process (k-dim W-L for short) for these procedures [25]. The most powerful version of k-dim W-L starts by coloring the k-tuples by their isomorphism type, and refines by looking at the colors of the "neighbors," i.e. k-tuples that share any vertices, distinguishing based on the number of common vertices. The $(k+1)$-dim W-L is more powerful the the k-dim W-L (every graph that is canonically labeled by the k-dim W-L is also canonically labeled by the $k+1$-dim W-L). Note moreover that the n-dim W-L always canonically labels graphs over n vertices. The running time, however increases exponentially in k: one step of refinement of the k-dim W-L takes time $O(n^k + m)$.

There was hope to use low-dimensional W-L to solve Graph Isomorphism efficiently (in quasi-polynomial time or even polynomial time). However, these hopes were shattered in 1991, when Cai Fürer and Immerman exhibited an example of a pair of classes of graphs which k-dim W-L will distinguish only for $k = \Omega(n)$.

Theorem 2.4.4 (Cai-Fürer-Immerman [31]). *For every n, there exist non-isomorphic graphs G_n and H_n such that if k-dim W-L distinguishes G_n from H_n then $k = \Omega(n)$.*

Because the k-dim W-L process encompasses many combinatorial approaches to the Graph Isomorphism problem, this theorem proved that all these approaches will fail to yield efficient algorithms for Graph Isomorphism testing in general.

Some of these combinatorial techniques do turn out to be useful as part of other algorithms. Indeed we will make use of colorings and refinements in our algorithm for Hypergraph Isomorphism. One combinatorial technique we will make use of, which can be used to start a refinement process, is to assign a unique color to one of the vertices; in this case we say that we **individualize** that vertex (see Figure 2.4.2). Note that this gets us closer to a coloring into singletons. However this comes at a cost. For isomorphism testing, we must try assigning the unique color to every vertex in the other graph. Therefore individualization

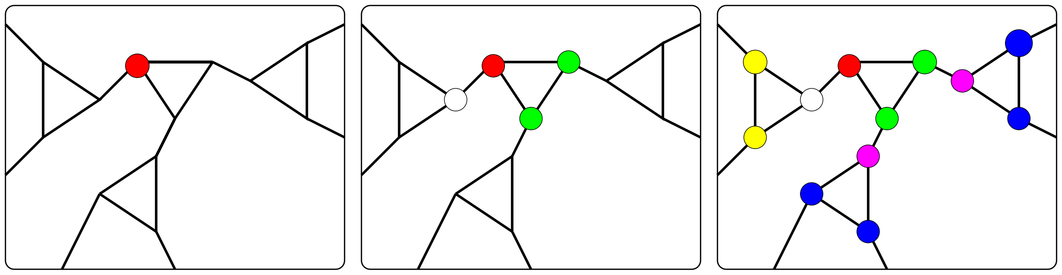
(a) Individualization (b) First refinement step (c) Second refinement step

Figure 2.1: (a) Individualizing one vertex corresponds to giving it a unique color (in this case red). (b) Individualization is usually followed by refinement (in this case naive refinement). (c) Hopefully individualizing one vertex results in a lot of refinement.

comes at a multiplicative cost of n.

A basic combinatorial technique involves combining individualization with a low (1 or 2)-dimensional W-L process [79]. (Figure 2.4.2 depicts an example of individualization followed by naive refinement.) It is not hard to see that ℓ individualization followed by k-dim W-L are at most as powerful as $(\ell + k)$-dim W-L, hence in general the Cai-Fürer-Immerman examples [31] work against this technique as well.

For the special class of strongly regular graphs, a good source of hard examples for Graph Isomorphism because of their high degree of regularity, simple individualization and refinement has been shown to give algorithms that match, and even do better than the group theoretic techniques for general graph isomorphism.

In 1979, using combinatorial arguments, Babai was able to show that $\widetilde{O}(\sqrt{n})$ individualizations suffice to decide isomorphism of strongly regular graphs. (Recall that the tilde stands for poly-logartithmic factors.)

Theorem 2.4.5 (Babai [3])**.** *Isomorphism of strongly regular graphs can be decided in time* $\exp(4\sqrt{n}\ln^2 n)\,\mathrm{poly}(n)$.

This result was improved by Spielman in 1996.

Theorem 2.4.6 (Spielman [74])**.** *Isomorphism of strongly regular graphs can be decided in time* $\exp(\widetilde{O}(n^{1/3}))$.

Spielman in fact proves that $\widetilde{O}(n^{1/3})$ individualizations suffice to decide isomorphism of two strongly regular graphs. The proof is based on Neumaier's claw bound [64], which implies that low-degree strongly regular graphs have small second largest eigenvalue (i.e. are good expanders), unless they have a specific structure. Spielman is then able to use the fact that graphs with small second largest eigenvalue "look random" (in a precise sense, using the theory of expanders). For the structured case, Spielman used results of Miller [62] and Cameron [32], as well as some new individualization and refinement algorithms. For graphs of high degree (roughly $\Omega(n^{2/3})$), Spielman used a more precise version of Babai's result (Theorem 2.4.5 cf. [3]).

2.5 Group Theoretic Techniques

2.5.1 Tower of groups

The first result proven using group theoretic techniques was the following.

Theorem 2.5.1 (Babai, 1979 [2]). *Isomorphism of two vertex-colored graphs with bounded color classes can be decided in Las Vegas polynomial time*

The algorithm was derandomized by Furst, Hopcroft and Luks [39].

This paper introduced the "tower of groups" approach. The situation is the following. Let $X = (V, E, c)$ be a colored graph with color classes $C_1, \ldots, C_r$ of size bounded by a constant b: $(\forall i \le r)(|C_i| \le b)$. The main idea is summarized in the following claim.

Claim 2.5.2 (Babai, 1979 [2]). *Given a vertex-colored graph X with bounded multiplicity of colors (for each color class C, $|C| \le b$), there exists a chain (tower) of groups $G_0 \ge G_1 \ge \cdots \ge G_t \cong \mathrm{Aut}(X)$, such that:*

- $G_0 = \prod \mathrm{Sym}(C_i)$;
- *Each G_i is recognizable;*
- $|G_{i-1} : G_i| \le f(b)$ *is bounded by a function depending only on b.*

Recall that a subgroup $H \leq G$ is recognizable if given some $g \in G$, we can test membership of g in H in polynomial time.

In the application to vertex-colored graphs with bounded color classes, the idea is to construct graphs X_i over the same (colored) vertex set as X, starting from X_0 the empty graph (no edges), up to $X_t = X$. Each group G_i is the automorphism group of X_i. Let $t = \binom{r}{2} + r$ be the number of unordered pairs of colors (including diagonal pairs), and order the pairs of colors arbitrarily. To construct X_i for $i \geq 1$, add the edges between the color classes in the ith pair to X_{i-1}. It can be shown that Claim 2.5.2 does indeed hold for this chain.

Given such a chain of subgroups, the idea was to construct coset representatives for G_i in G_{i-1} by randomly sampling elements of G_{i-1}, and checking if the new element lies in a new coset (which can be done since G_i is recognizable). Since the number of cosets is bounded by $f(b)$, with high probability if we choose about $f(b) \log f(b)$ elements, then we picked at least one element from each coset. We are sweeping under the rug details about how to do this with high probability for all levels of the tower simultaneously. Since G_0 is a well understood group, we can generate elements of G_0 distributed uniformly at random. With some additional work it can be shown by induction that this allows us to generate elements uniformly at random from each G_i. Furst Hopcroft and Luks observed that in fact one can apply pick elements deterministically, and determine when all coset representatives have been found [39], which yields a derandomized version of Theorem 2.5.1.

Another application of this method is to test isomorphism of graphs with bounded eigenvalue multiplicity.

Theorem 2.5.3 (Babai, Grigoriev and Mount [20]). *Isomorphism of two graphs with bounded eigenvalue multiplicity can be decided in Las Vegas polynomial time.*

This result can also be derandomized with the observation from Furst, Hopcroft, and Luks [39]

2.5.2 Divide and conquer

Luks introduced two divide and conquer strategies for isomorphism problems, which were central to his polynomial-time isomorphism test of graphs of bounded degree [55]. Originally Luks developed these strategies for the different (but very much related) problem of string isomorphism. We present these divide and conquer ideas for bipartite graphs, since this is similar to our application for hypergraphs.

Let X and Y be bipartite graphs over the same set of vertices $V = V_1 \dot\cup V_2$. Let $E(X)$ and $E(Y)$ denote the set of edges of X and Y, respectively. Isomorphisms of X and Y preserve the parts by definition, that is, $\mathrm{ISO}(X, Y) \leq \mathrm{Sym}(V_1) \times \mathrm{Sym}(V_2)$. For the purposes of recursion, we generalize the problem to isomorphisms within some subset $L \subseteq \mathrm{Sym}(V_1) \times \mathrm{Sym}(V_2)$. Define

$$\mathrm{ISO}(L; X, Y) := \{\tau \in L \mid (\forall (v_1, v_2) \in V_1 \times V_2)((v_1, v_2) \in E(X) \iff (v_1, v_2)^\tau \in E(Y))\}.$$

Note that $\mathrm{ISO}(L; X, Y) = \mathrm{ISO}(X, Y) \cap L$. In fact we need a further generalization of this problem. Let $B \subseteq V_1 \times V_2$. Then

$$\mathrm{ISO}_B(L; X, Y) := \{\tau \in L \mid (\forall (v_1, v_2) \in B)((v_1, v_2) \in E(X) \iff (v_1, v_2)^\tau \in E(Y))\}.$$

Since our graphs will be called X and Y throughout, we omit them and write $\mathrm{ISO}(L)$ and $\mathrm{ISO}_B(L)$.

The following two identities, first observed by Luks [55] are easy to prove from the definitions above.

$$\mathrm{ISO}_B(L_1 \cup L_2) = \mathrm{ISO}_B(L_1) \cup \mathrm{ISO}_B(L_2). \tag{2.1}$$

$$\mathrm{ISO}_{B_1 \cup B_2} = \mathrm{ISO}_{B_2}(\mathrm{ISO}_{B_1}(L)). \tag{2.2}$$

Now we consider the case where L is a subcoset $G\sigma$, for $G \leq \mathrm{Sym}(V_1) \times \mathrm{Sym}(V_2)$, and $\sigma \in \mathrm{Sym}(V_1) \times \mathrm{Sym}(V_2)$. In this case if B is G-invariant ($B^G = B$), then $\mathrm{ISO}_B(G\sigma)$ is a

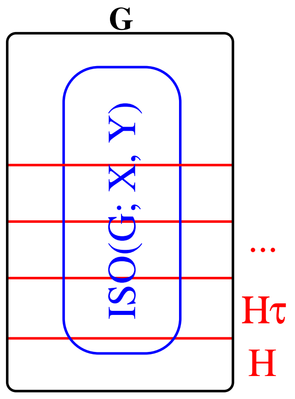

Figure 2.2: The partition of G into cosets also partitions $\mathrm{ISO}(G; X, Y)$.

subcoset of $\mathrm{Sym}(V_1) \times \mathrm{Sym}(V_2)$. Now suppose $H \leq G$, and let R be a set of (right) coset representatives. Then $G = \dot\cup_{\tau \in R} H\tau$, and $G\sigma = \dot\cup_{\tau \in R} H\tau$ (see Figure 2.5.2). Therefore, by Equation (2.1), we have

$$\mathrm{ISO}_B(G\sigma) = \cup_{\tau \in R} \mathrm{ISO}_B(H\tau\sigma). \tag{2.3}$$

If $B = B_1 \dot\cup \ldots \dot\cup B_k$ and the B_i are G-invariant (for example, the B_i are G-orbits), then Equation (2.2) gives:

$$\mathrm{ISO}_B(G\sigma) = \mathrm{ISO}_{B_1}(\mathrm{ISO}_{B_2}(\ldots \mathrm{ISO}_{B_k}(G\sigma)\ldots)). \tag{2.4}$$

These are the equations we will use for divide and conquer. Let G_i be the projection of G to V_i. Recall that we are interested in finding $\mathrm{ISO}_B(G; X, Y)$. We now describe the two divide-and-conquer ideas. The first allows us to reduce to the case where each G_i acts transitively. We will apply the second in some cases when one of the G_i has an imprimitive action.

First, if the G-action on one of the parts is not transitive, then we can apply Equation (2.4) and reduce to the case where G acts transitively on each part. Note that to be precise, in this case B_i will be the set of all pairs $(v_1, v_2) \in B$ such that v_1 is in the ith orbit. Let n be the number of vertices, and $b = |B|$, and let $T(n, b)$ be the running time for n and b. Let $b_i = |B_i|$, so $b = b_1 + \cdots + b_k$. Then when we apply this reduction we have

$T(n,b) \le \sum_{i=1}^{k} T(n,b_i)$.

We can also apply the divide-and-conquer ideas when G is transitive on each of the parts. Let $B_1, \ldots, B_k$ be a minimal system of imprimitivity (maximal blocks) of the G-action on one of the two parts. (Again, technically $B_i = \{(v_1, v_2) \mid v_1 \text{ is in the } i\text{th block}\}$.) Let $\varphi : G \to S_k$ be the primitive action of G on the blocks, and let $K = \ker(\varphi) \lhd G$. Then we can apply Equation (2.3), and break the problem into $|G : K|$ subproblems of the form $\mathrm{ISO}_B(K\tau)$. In K the blocks are invariant, hence we can apply Equation (2.4). To analyze the running time, let P be the primitive action of G on the blocks. The recurrence for the running time is

$$T(n,b) \le |P|\, k\, T(n, b/k),$$

since we use Equation (2.3) to reduce to $|P|$ problems of the form $\mathrm{ISO}_B(K\tau)$, and by Equation 2.4, each of these breaks into k instances of size b/k.

This recurrence will have a good solution if we have a bound of the form $|P| \le k^s$ for the class of groups under consideration. If this is the case, then

$$T(n,b) \le k^{s+1} T(n, b/k) \Rightarrow T(n,b) \le n^{s+1}.$$

Luks's isomorphism test for bounded degree graphs inspired the Babai-Cameron-Pálfy bound on the order of Γ_k groups (Definition 1.2.7 and Theorem 1.2.8 cf. [15]). Because the groups involved in isomorphism of graphs of maximum degree d are Γ_d groups, this gives a polynomial bound on the running time for the recurrence above in this case. This is not a complete description of the isomorphism test for bounded degree graphs, just a description of the divide and conquer strategies that were introduced in this test.

In general, we can apply this recurrence whenever the primitive G-action on the blocks is small enough. Indeed part (a) of Theorem 1.2.4 gives a moderately exponential bound of $k^{\sqrt{k}}$ on the order of primitive groups that are not the alternating or symmetric groups. Therefore this allows us to deal with all cases other than the alternating or symmetric groups in their natural actions. Dealing with the symmetric and alternating groups is going to be the key step in our hypergraph isomorphism test.

The divide-and-conquer strategies can be seen as descending in a structure forest of G, hence they are also sometimes called "Luks descent."

CHAPTER 3
HYPERGRAPH ISOMORPHISM

3.1 Main Result

We restate our main result.

Theorem 3.1.1. *Isomorphism of hypergraphs of rank k with n vertices can be tested in* $\exp\left(\widetilde{O}(k^2\sqrt{n})\right)$.

A key ingredient of our procedure is a moderately exponential algorithm for the Coset Intersection problem: given two cosets of the symmetric group, determine their intersection. The intimate connection of this problem to Graph Isomorphism was first highlighted in Luks's seminal paper [55]; an $n^{O(\sqrt{n})}$ algorithm was found by the first author:

Theorem 3.1.2. *([11]) The Coset Intersection Problem can be solved in $n^{O(\sqrt{n})}$ time. In fact, it can be solved in time $n^{O(1)}m^{O(\sqrt{m})}$ time where m is the length of the longest orbit of the two groups.*

The $n^{O(\sqrt{n})}$ result was first described in [7] (1983). It was included with a sketch of the proof in [21]. The proof of the main structural group theoretic lemma was given in full in [14]. While the outline of the coset intersection algorithm given in [21] omits key details, a full proof is now available [11].

Other ingredients include the moderately exponential Graph Isomorphism test [23] and a combinatorial lemma (Lemma 3.2.9).

3.2 Overall Strategy

For the purposes of recursion, we shall need to consider the more general G-isomorphism problem, i. e., the problem of finding all isomorphisms between two objects X, Y on the same underlying set (vertex set) that belong to a given permutation group G. If $\mathrm{ISO}(X, Y)$ denotes the set of isomorphisms of X and Y then the set of their G-isomorphisms is $\mathrm{ISO}(G; X, Y) = G \cap \mathrm{ISO}(X, Y)$.

The overall scheme of our procedure is gradual approximation. In each round we consider an invariant. If the invariant fails to yield isomorphism rejection, we bring Y "closer" to X in the sense that X and Y now look identical with respect to the invariant. We reduce G in the process by making it respect the invariant. A simple illustration of this principle is to compare the degrees of the vertices of X and Y; if this comparison does not yield isomorphism rejection then we move each vertex of Y to a vertex of X of the same degree and reduce G to its subgroup that preserves the degree of each vertex of X.

We describe this idea on a slightly more formal level. While searching for irregularities in X, we may discover a proper subgroup $H \leq G$ such that $\mathrm{Aut}(G; X) \leq H$. Repeating the same process for Y either we refute G-isomorphism of X and Y or find a subgroup $K \leq G$ such that $\mathrm{Aut}(G; Y) \leq K$ and $K = \sigma H \sigma^{-1}$ is a conjugate of H for some $\sigma \in G$ which is computed along the way. Now we replace Y by Y^σ and G by H.

The goal is to reach a point when G becomes a subgroup of the automorphism group $\mathrm{Aut}(X)$ (so $\mathrm{Aut}(G; X) = G$); once this is the case, we conclude that either $X = Y$ or X and Y are not G-isomorphic.

Another aspect of our procedure is that it is recursive. If we find any irregularity in X, we either reject Y (if the same irregularity is not found in Y), or we split X as well as Y into "more regular" parts, determine the isomorphisms of the parts, and paste the results together.

An illustration of how this works is the case of invariant coloring of edges. (For instance, for graphs, we may color edges according to the number of triangles containing them; isomorphisms preserve this coloring.) Given the sets of isomorphisms of the corresponding pairs of edge-color-classes of the two hypergraphs, we need to take the intersection of these sets. Noting that the set of isomorphisms of X and Y is a coset, we see that this last step is an instance of Coset Intersection. This idea permits us to work with highly regular structures only.

Finally, following Luks's divide-and-conquer idea for permutation groups [55] (see Section 2.5.2), we need to delve into the group structure to some depth. As in [11], the bottleneck arises from transitive groups with a large alternating or symmetric group action on a set of blocks of imprimitivity ("giant action"); handling this case constitutes the principal technical contribution of this paper. For the divide-and-conquer to succeed, we need to switch from hypergraphs to k-partite k-hypergraphs; this is analogous to moving to bipartite graphs from graphs.

3.2.1 Reduction to k-partite k-hypergraphs

A **k-partite set** Ω is an ordered family of k sets $\Omega = (\Omega_1, \dots, \Omega_k)$. We think of the Ω_i as being disjoint (replacing, as usual, Ω_i by $\Omega_i \times \{i\}$). Let $\mu(\Omega) = \dot{\bigcup} \Omega_i$ be the set of "points" in Ω, and let $\pi(\Omega) = \prod \Omega_i$. We shall think of the elements of $\pi(\Omega)$ as the **transversals** of Ω, i. e., those subsets of $\mu(\Omega)$ which intersect each Ω_i in exactly one element. We say that the k-partite set $\Psi = (\Psi_1, \dots, \Psi_k)$ is a $s$ubset of Ω ($\Psi \subseteq \Omega$) if for each i, $\Psi_i \subseteq \Omega_i$.

If Ω and Ψ are k-partite sets then by a function $\sigma : \Omega \to \Psi$ we mean a function $\sigma : \mu(\Omega) \to \mu(\Psi)$ that respects the parts: $\Omega_i^\sigma \subseteq \Psi_i$. We say that a function $\sigma : \Omega \to \Psi$ is a bijection between Ω and Ψ if it is a bijection between $\mu(\Omega)$ and $\mu(\Psi)$.

For any $I \subseteq [k]$, let $\rho_I(\Omega) = (\Omega_i : i \in I)$, the $|I|$-partite set that is the restriction of Ω to I. We call the elements of $\pi(\rho_I(\Omega))$ the I-partial transversals or I-transversals of Ω. If $I \subseteq J \subseteq [k]$ and e is a J-partial transversal then the restriction of e to I is an I-partial transversal which we denote by $\mathrm{pr}_I(e)$. For a set E of J-partial transversals, we define the *projection* $\mathrm{pr}_I(E) = \{\mathrm{pr}_I(e) : e \in E\}$. Most of the time, we shall use these definitions with $J = [k]$.

Definition 3.2.1. (see Figure 3.1) A *k-partite k-hypergraph* $X = (V, E)$ is defined by a k-partite set $V = (V_1, \dots, V_k)$ of vertices and a set $E \subseteq \pi(V)$ of edges. We call the V_i the "parts" of the vertex set V. The *product-size* of X is $\Pi_0(X) = \prod_i |V_i|$.

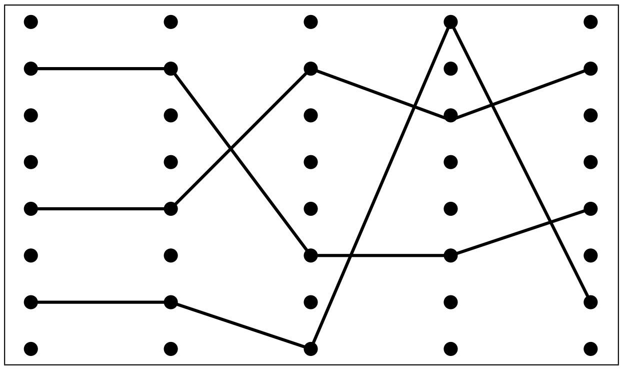
Figure 3.1: A 5-partite 5-hypergraph. The dots represent the vertices, the three transversals are three edges.

A *partial edge* is a subset of an edge. The partial edges are partial transversals; and *I-partial edge* is an I-partial transversal.

Let $X = (V; E)$ and $X' = (V'; E')$ be k-partite k-hypergraphs, where $V = (V_1, \dots, V_k)$ and $V' = (V'_1, \dots, V'_k)$. An isomorphisms of X and X' is a $V \to V'$ bijection which preserves edges, i. e., $e \in E$ if and only iff $e^\sigma \in E'$. If this is the case, we write $X' = X^\sigma$. $\mathrm{ISO}(X, X')$ denotes the set of $X \to X'$ isomorphisms; and $\mathrm{Aut}(X) = \mathrm{ISO}(X, X)$ is the automorphism group of X.

W.l.o.g. in discussing the isomorphism problem for X and X' we may restrict ourselves to the case when $V = V'$, i. e., $V_i = V'_i$ for all i.

We use the following notation: for a t-partite set $\Omega = (\Omega_1, \dots, \Omega_t)$, let $\mathrm{SymPr}(\Omega)$ denote the group $\mathrm{Sym}(\Omega_1) \times \cdots \times \mathrm{Sym}(\Omega_t)$ in its action on $\mu(\Omega)$.

The following is immediate from Theorem 3.1.2.

Theorem 3.2.2. (Coset Intersection for k-partite sets) *Given a k-partite set $\Omega = (\Omega_1, \dots, \Omega_k)$, subgroups $G, H \le \mathrm{SymPr}(\Omega)$,and elements $\sigma, \tau \in \mathrm{SymPr}(\Omega)$, one can determine $G\sigma \cap H\tau$ in time $\mathrm{poly}(k)m^{O(\sqrt{m})}$, where $m = \max_i |\Omega_i|$.*

For the rest of this section, let $X = (V, E)$ and $Y = (V, F)$ be two k-partite k-hypergraphs over the same vertex set $V = (V_1, \dots, V_k)$. With this notation, the isomorphisms of X and Y form a subcoset of $\mathrm{SymPr}(V_1, \dots, V_k)$; our goal is to find this subcoset. We need to generalize this problem to finding all isomorphisms within a subcoset.

Definition 3.2.3. Let $L \subseteq \mathrm{SymPr}(V_1, \ldots, V_k)$ be a set of permutations. The set of L-isomorphisms between X and Y is $\mathrm{ISO}(L; X, Y) = L \cap \mathrm{ISO}(X, Y)$. The set of L-automorphisms of X is $\mathrm{Aut}(L; X) = \mathrm{ISO}(L; X, X) = L \cap \mathrm{Aut}(X)$.

If L is a subcoset of $\mathrm{SymPr}(V_1, \ldots, V_k)$ then so is $\mathrm{ISO}(L; X, Y)$. This is then the form in which we prove our main result:

Problem 3.2.4 (H-isomorphism of k-partite k-hypergraphs)**.** *Let X and Y be two k-partite k-hypergraphs over the same vertex set $V = (V_1, \ldots, V_k)$. Let H be a subcoset of* $\mathrm{SymPr}(V_1, \ldots, V_k)$*. Find the subcoset* $\mathrm{ISO}(H; X, Y)$*.*

Theorem 3.2.5. *Problem 3.2.4 can be solved in* $\exp\left(\widetilde{O}(k^2\sqrt{n})\right)$ *time, where* $n = \max_{i=1}^{k} |V_i|$*.*

Note that $\mathrm{ISO}(H\sigma; X, Y) = \mathrm{ISO}(H; X, Y^{\sigma^{-1}})$, for any set $H \subseteq \mathrm{SymPr}(V_1, \ldots, V_k)$. So we may assume H is a group.

Observation 3.2.6. *For a coset H, H-isomorphism of hypergraphs of rank k with n vertices reduces to H^*-isomorphism of k-partite k-hypergraphs with parts of size at most n, and an increase of a factor of at most $k!$ in the number of edges. The k-partite k-hypergraphs and the coset H^* can be constructed in time linear in the size of the output.*

Proof. Let Y_1 and Y_2 be hypergraphs of rank k over the same vertex set W, with edge sets F_1 and F_2, resp. We construct the k-partite k-hypergraphs $X_i = (W, \ldots, W; E_i)$, $i = 1, 2$ where $E_i = \{(w_1, \ldots, w_k) : \{w_1, \ldots, w_j\} \in F_i$ for some $j \leq k$ and $w_{j+1} = w_{j+2} = \cdots = w_k \in \{w_1, \ldots, w_j\}\}$ (so $|F_i| \leq k!|E_i|$)). Let moreover $H^* = \{(\sigma, \ldots, \sigma) : \sigma \in H\}$. Then the hypergraphs Y_1 and Y_2 are H-isomorphic if and only if the k-partite k-hypergraphs X_1 and X_2 are H^*-isomorphic. □

3.2.2 Relational structures

In fact, essentially the same reduction we showed above allows us to reduce isomorphism for explicit relational structures to isomorphism of k-partite k-hypergraphs (see Section 2.3.1 for the definition of relational structure). Note that while isomorphism of explicit relational structures reduces to Graph Isomorphism in polynomial time (cf. Theorem 2.3.5), the reduction does not preserve the number of vertices (much like the reduction from Hypergraph Isomorphism to Graph Isomorphism), so the moderately exponential Graph Isomorphism test does not yield a moderately exponential isomorphism test for relational structures, but our algorithm for hypergraph isomorphism does.

Observation 3.2.7. *For a coset H, H-isomorphism of explicit relational structures of arity k defined by s relations, and over a set of size n reduces to H^* isomorphism of colored k-partite k-hypergraphs with n vertices in each part, $m = \sum_{i=1}^{s} |R_i|$ edges, and s color classes.*

Proof. Simple modification of the reduction in observation 3.2.6 □

3.2.3 Linked actions

Throughout this section, $\Omega = (\Omega_1, \ldots, \Omega_k)$ will be a k-partite set and $G \leq \mathrm{SymPr}(\Omega)$; $G_i \leq \mathrm{Sym}(\Omega_i)$ will be the action on Ω_i.

We say that the Ω_i are *linked* under the G-action (or "the domains of the G_i are linked") if $\mu(\Omega)$ admits a G-invariant partition into transversals $B_1, \ldots, B_m \subseteq \mu(\Omega)$ (for each i, j, $|B_j \cap \Omega_i| = 1$). In this case, $|\Omega_1| = \cdots = |\Omega_k|$; the transversals define bijections between the Ω_i.

Observation 3.2.8. *If Ω_i and Ω_j are linked for every pair $i < j \leq k$ then all the Ω_i are linked. Moreover, the statement $S(i, j)$ that "Ω_i and Ω_j are linked" is an equivalence relation on $[k]$.*

Note that linking of the domains is the same concept as permutational diagonals (see Section 1.2.7).

Discovering links will be one of our main structural goals; the following lemma will serve us in this.

Lemma 3.2.9. *Let $Y = (\Omega_1, \Omega_2; E)$ be a biregular bipartite graph (regular on both parts) of positive density $\leq 1/2$ (i. e., $1 \leq |E| \leq |\Omega_1||\Omega_2|/2$). Let $G \leq \mathrm{SymPr}(\Omega_1, \Omega_2)$. (a) If $2\sqrt{n} \leq |\Omega_i| \leq n$ for $i = 1, 2$, and the restriction of $\mathrm{Aut}(G; Y)$ to Ω_1 contains $\mathrm{Alt}(\Omega_1)$ then the degree of each vertex in Ω_2 is 1. (b) If, in addition, the restriction of $\mathrm{Aut}(G; Y)$ to Ω_2 contains $\mathrm{Alt}(\Omega_2)$ then Ω_1 and Ω_2 are linked.*

Proof. Let d be the degree of the vertices in Ω_2; assume $d \geq 2$. As $\mathrm{Alt}(\Omega_1)$ acts on Ω_1, every set of d vertices in Ω_1 must share a neighbor. There are $\binom{|\Omega_1|}{d} > |\Omega_2|$ such sets; therefore, by the pigeon hole principle, two of them must share a common neighbor. But such a common neighbor has degree greater than d, a contradiction, proving part (a). For part (b), we see that Y is regular of degree 1, so the edges of Y form the desired block system. $\square$

If G is isomorphic to all the G_i, i. e., the G-action is faithful on each Ω_i, then we say that G acts *diagonally* on Ω. Fact 1.2.14 states that if each G_i is nonabelian simple then G is a direct product of diagonal actions. Therefore if all the G_i are nonabelian simple then one can partition $[k]$ as $[k] = I_1 \dot\cup \ldots \dot\cup I_t$ such that $G = D_1 \times \cdots \times D_t$ where D_i acts diagonally on the $|I_i|$-partite set $(\Omega_j : j \in I_i)$ and it acts trivially on all the other Ω_j.

We shall say that the G_i corresponding to the same I_i are *linked.* We note that being linked is an equivalence relation.

Now we turn to the case of greatest importance to us: when each G_i is an alternating group. In the language of linking, we can re-write Fact 1.2.15 as follows

Fact 3.2.10. Assume $G_i = \mathrm{Alt}(\Omega_i)$, $|\Omega_i| \geq 5$ and $|\Omega_i| \neq 6$. If $G_1, \ldots, G_k$ are linked then their domains are linked.

We need one more well-known fact about the alternating groups.

Fact 3.2.11. In A_n, the subgroup of smallest index has index n if $n \geq 5$, the second smallest $\binom{n}{2}$ if $n \geq 9$.

3.2.4 Giant threshold, giant action

Our algorithm will operate with a global constant n, the initial number of vertices of the hypergraphs. Throughout the algorithm, including its recursive calls, each "part" V_i of our k-partite k-hypergraphs will satisfy $|V_i| \leq n$.

We say that a permutation group $G \leq S_d$ is a *giant* if (a) G is either S_d or A_d; (b) $\gamma < d \leq n$ where $\gamma = \max\{7, 2\sqrt{n}\}$ is our *giant threshold.*

Definition 3.2.12. The G-action on V is of **giant type** if the following hold. G acts transitively on each V_i; each V_i admits a G-invariant partition $\mathfrak{B}_i$; and the G-action on each $\mathfrak{B}_i$ (by permuting the blocks of $\mathfrak{B}_i$) is a giant.

An application of Luks's divide-and-conquer strategy [55] (cf. Section 2.5.2) will show that within our target time bound, the only case we need to worry about is when G is of giant type.

Definition 3.2.13. Assume the G-action on V is of giant type. If the action of G on $\mathfrak{B}_i$ is $\mathrm{Sym}(\mathfrak{B}_i)$ then let $G[i]$ be the subgroup of index 2 in G which acts as $\mathrm{Alt}(\mathfrak{B}_i)$ on $\mathfrak{B}_i$; otherwise set $G[i] = G$. We say that the G-action is of **strong giant type** if $(\forall i)(G[i]$ acts transitively on $V_i)$.

Note that in the alternative, some V_i is split into two orbits of equal size under $G[i]$; these orbits are blocks of imprimitivity under G.

3.3 Color Reductions, Regularization

In this section we generalize the isomorphism problem to k-partite k-hypergraphs with colored vertices and colored edges; colors are preserved by isomorphisms by definition. We show how to get rid of the colors using recursion or group intersection. Since any observed irregularity provides us with an invariant coloring, a result of this homogenization is that we only need to deal with highly regular objects.

3.3.1 Vertex-color reduction

Let $f, f' : V \to \Sigma$ be two colorings of V. Let $X = (V, E, f)$ and $Y = (V, F, f')$ be two vertex-colored k-partite k-hypergraphs. Now $\mathrm{ISO}(X, Y) = H \cap \mathrm{ISO}((V, E), (V, F))$ where $H = \mathrm{ISO}((V, f), (V, f'))$ is the subcoset of color-preserving maps from the colored set (V, f) to the colored set (V, f').

It is trivial to compute H in polynomial time. We are interested in the G-isomorphisms of X and Y. If $G \cap H = \emptyset$ then X and Y are not G-isomorphic. Otherwise let $H = K\sigma$ where $K \leq \mathrm{SymPr}(V)$; then $\mathrm{ISO}(G; X, Y^{\sigma^{-1}}) = \mathrm{ISO}(G \cap K; X, Y^{\sigma^{-1}})$. Here $Y^{\sigma^{-1}} = (V, F^{\sigma^{-1}}, f)$, so we obtained the following (with $G^* = G \cap K$).

Reduction 3.3.1. *G-isomorphism of vertex-colored k-partite k-hypergraphs X and Y reduces to G^*-isomorphism of X' and Y', color-refined versions of X and Y, resp., where $G^* \leq G$; X' and Y' are colored identically; and this coloring is G^*-invariant. The cost of the reduction is a single application of coset intersection.*

We note that this reduction in particular permits the *elimination of isolated vertices*. In this case we shall reduce G to G_i and apply Luks's divide-and-conquer.

3.3.2 Edge-color reduction

Let $f : E \to \Sigma$, $f' : F \to \Sigma$ be two edge-colorings. Isomorphisms preserve edge-color by definition; and for any $L \subseteq \mathrm{SymPr}(V_1, \dots, V_k)$, $\mathrm{ISO}(L; (X, f), (Y, f')) = L \cap \mathrm{ISO}((X, f), (Y, f'))$.

Observation 3.3.2. *Let L be a subcoset of* $\mathrm{SymPr}(V)$*. Then we can find* $\mathrm{ISO}(L; (X, f), (Y, f'))$ *by solving $|\mathrm{Im}(f)|$ instances of Problem 3.2.4 and applying coset intersection.*

Proof. Indeed, if $\mathrm{Im}(f) \neq \mathrm{Im}(f')$ then (X, f) and (Y, f') are not isomorphic. Otherwise, for each color i, create two k-partite k-hypergraphs, X_i, and Y_i, over the vertex set $V = (V_1, \dots, V_k)$, with edge sets $E_i = \{e \in E | f(e) = i\}$, and $F_i = \{e \in F | f'(e) = i\}$, resp. We have $\mathrm{ISO}(L; (X, f), (Y, f')) = \bigcap_{i \in \mathrm{Im}(f)} \mathrm{ISO}(L; X_i, Y_i)$. □

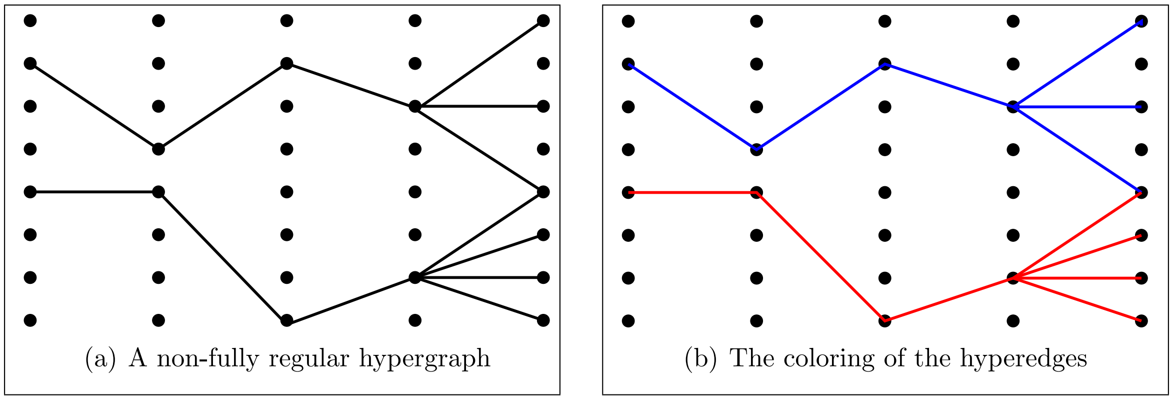
(a) A non-fully regular hypergraph
(b) The coloring of the hyperedges

Figure 3.2: (a) In this example some partial hyperedges have 3 extensions to the last part, and some have 4; so (b) we color the hyperedges differently (blue and red respectively).

Therefore if we find an invariant coloring of the edges of X and Y, we reduce Problem 3.2.4 to the color classes. The resulting hypergraphs will be increasingly "regular" ("irregularities" can be used to obtain an invariant split).

In particular we obtain the following:

Reduction 3.3.3. *The general case reduces to the case when all edges of X and Y belong to a single G-orbit (on $\pi(V)$). The reduction is additive in terms of the number of edges. The cost is $m-1$ applications of Coset Intersection, where m is the number of G-orbits into which the edges of X are divided.*

3.3.3 Regularization

We say that a k-partite k-hypergraph is **fully regular** if any two partial edges over the same set have the same number of extensions to any superset. More formally

Definition 3.3.4. The k-partite k-hypergraph $X = (V, E)$ is *fully regular* if $\forall I \subseteq J \subseteq [k], \quad \forall e_1, e_2 \in \mathrm{pr}_I(E)$

$$|\{e_3 \in \mathrm{pr}_J(E) : e_1 \subseteq e_3\}| = |\{e_3 \in \mathrm{pr}_J(E) : e_2 \subseteq e_3\}|.$$

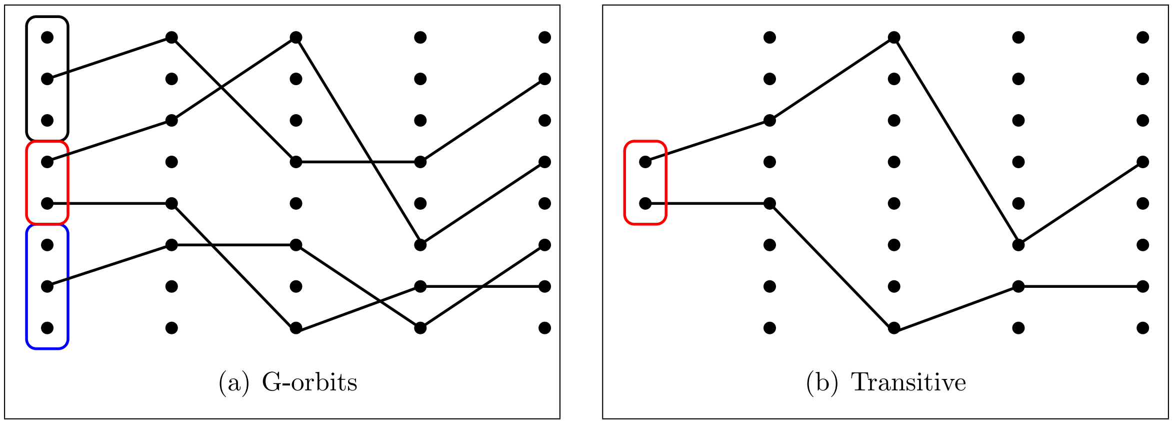

Figure 3.3: Luks's reduction allows us to look at each G-orbit in each part separately.

Reduction 3.3.5. *The general case reduces to the case when X and Y are fully regular with the same parameters. The reduction is additive in terms of the number of edges. The cost is $m-1$ applications of Coset Intersection, where m is the number of fully regular subsets into which the edges of X are divided.*

Indeed, if the hypergraph is not fully regular then then we label the I-partial edges by the number of J-partial edges containing them, and then color each edge by the label of its restriction to I (see Figure 3.2). This is a non-trivial invariant coloring, and we can apply the edge-color reduction.

3.4 Group Theoretic Reductions

3.4.1 G transitive on each part

Suppose $V_i = V_i' \dot\cup V_i''$ where V_i', V_i'' are nonempty and G-invariant. We use Luks's divide-and-conquer idea to reduce this case to a case where V_i is reduced to V_i' and then a case where V_i is reduced to V_i'' (the latter to be solved within the output of the former cf. Section 2.5.2, Equation (2.2)). As a result we obtain:

Reduction 3.4.1. *The general case recursively reduces to the sub-cases when each part is a G-orbit (see Figure 3.3). The product-size $\Pi_0(X) = \prod |V_i|$ is additive under this recursion.*

We note that this reduction does not work directly for hypergraphs (even for graphs); this is the main reason why we had to switch to k-partite hypergraphs.

3.4.2 Reduction to strong giant type

Now we use the other element of Luks's divide-and-conquer strategy to reduce to giant type (cf. Section 2.5.2). Let $\mathfrak{B}_i$ be a minimal block system on V_i. If the G-action on $\mathfrak{B}_i$ is not a giant, let K_i be the kernel of the G-action on $\mathfrak{B}_i$. Then we split G into cosets of K_i and solve the isomorphism problem within each coset (cf. Equation (2.3)). This reduces the problem to $|G : K_i|$ instances of K_i-isomorphism. Since each block of $\mathfrak{B}_i$ is K_i-invariant, we then use the previous section to reduce V_i successively to each block of $\mathfrak{B}_i$ (cf. Equation (2.4)).

We use part (a) of Theorem 1.2.4 to assess the cost of this recursion. While the multiplicative size of the problem was divided by $|\mathfrak{B}_i|$, the number of instances was multiplied by $|G : K_i| < |\mathfrak{B}_i|^{\sqrt{|\mathfrak{B}_i|}} < |\mathfrak{B}_i|^{\sqrt{n}} < |\mathfrak{B}_i|^{\gamma}$ if $|\mathfrak{B}_i| \geq \gamma$, where γ is our "giant threshold" (Section 3.2.4); and $|G : K_i| \leq |\mathfrak{B}_i|! < |\mathfrak{B}_i|^{\gamma}$ if $|\mathfrak{B}_i| < \gamma$. So in all cases, $|G : K_i| < |\mathfrak{B}_i|^{\gamma}$, giving a recurrence of the type $T(N) \leq k^{\gamma} T(N/k)$ (where $k = |\mathfrak{B}_i|$), resulting in a recursion tree with fewer than N^{γ} leaves, where N is the product-size $\Pi_0(X)$. The leaves correspond to cases where G is either of giant type or trivial.

Finally if the G-action is of giant type but not of strong giant type, we find a V_i with an imprimitive action with 2 blocks (see end of Section 3.2.4) and proceed as above.

3.4.3 Link-type

Assume G is of strong giant type. Then each V_i has a unique minimal block system $\mathfrak{B}_i$ on which G acts as a giant. Consider the G-action on the k-partite set $\mathfrak{B} = (\mathfrak{B}_1, \ldots, \mathfrak{B}_k)$. We write $i \approx j$ and say that the parts V_i and V_j are **block-linked** if the G-actions on $\mathfrak{B}_i$ and $\mathfrak{B}_j$ are linked. In this case we also say that v_i and v_j are block-linked if $v_i \in V_i$ and $v_j \in V_j$ belong to linked blocks.

We write $i \sim_e j$ and say that the parts V_i and V_j are edge-linked under edge $e =$

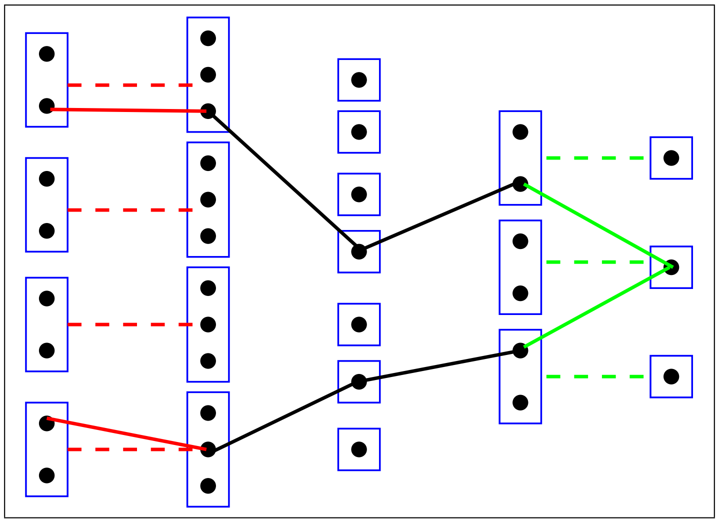

Figure 3.4: The dotted lines represent linking between the blocks. The two hyper-edges depicted have the same link-type: edge-linked for the red linking, and not for the green.

$(v_1, \ldots, v_k) \in E$ if the block containing $v_i \in V_i$ and the block containing $v_j \in V_j$ are linked. We call the equivalence relation $\sim_e$ the **link-type** of the edge e; it is a refinement of the equivalence relation $\approx$. We note that G preserves link-type; therefore, after Reduction 3.3.3, *all edges have the same link-type* and we may refer to $\sim$ without specifying e.

The link-type of I-partial edges is the restriction of $\sim$ to $I \subseteq [k]$.

Let $\mathfrak{J}$ denote the set of $\approx$-equivalence classes, and $\mathfrak{L}$ the set of $\sim$-equivalence classes (each of these partition $[k]$). We call the elements of $\mathfrak{J}$ **block-linked sections** and the elements of $\mathfrak{L}$ **edge-linked sections**.

Some further notation. For an edge-linked section L, let $M_L^1, \ldots, M_L^{k(L)}$ be a G-invariant partition of $\{\mathfrak{B}_i : i \in L\}$ into transversals. We call the M_L^j **macro-blocks** (see Figure 3.5). Note that each macro-block consists of one block from each part of an edge-linked section. Note further that all the edges in an edge-linked section will be within the macro-blocks. Let $\mathfrak{M}_L = \{M_L^s : s \in [k(L)]\}$ be the set of macro-blocks of edge-linked section L.

For any $v \in V$, we call block(v) the block containing v, and macroblock(v) the macro-block containing v.

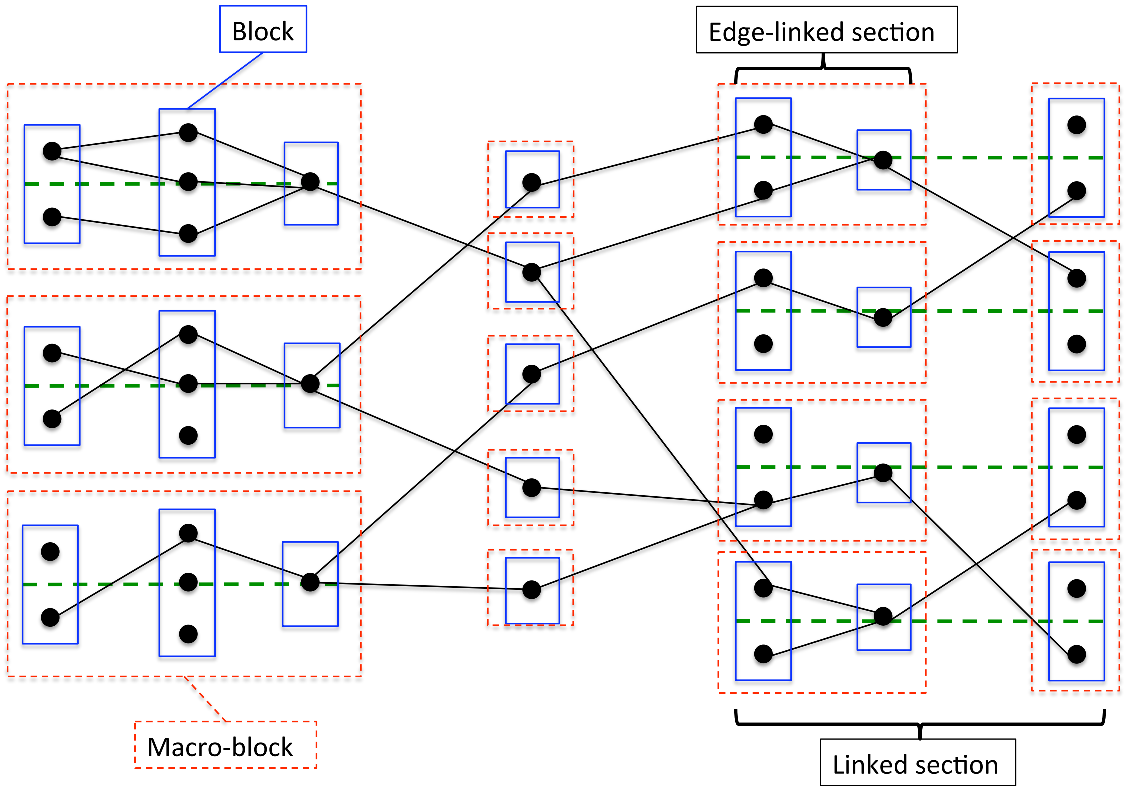

Figure 3.5: Vertices are depicted as black dots, and edges as black lines. The dotted green lines represent linking of the group action. The blue rectangles are the blocks, the dotted red rectangles are the macro-blocks.

3.4.4 The ultimate structure

Assume G is of strong giant type, all edges of $X = (V, E)$ belong to the same G-orbit, and X has no isolated vertices. We use the notation of Section 3.4.3. Let $f = (v_1, \dots, v_k) \in \pi(V)$ be a transversal ($v_i \in V_i$). We say that $f \in \hat{E}$ if (a) for each edge-linked section $L \in \mathfrak{L}$, there is an edge $e \in E$ such that $e|_L = f|_L$; and (b) if $i \not\sim j$ then v_i and v_j are not block-linked. Obviously $E \subseteq \hat{E}$.

We say that the pair (X, G) has the **ultimate structure** if the assumptions of the first sentence hold and $\hat{E} = E$.

Lemma 3.4.2. *Assume both (X, G) and (Y, G) have the ultimate structure. Then the G-isomorphisms of X and Y can be determined in time $O(n^{k\sqrt{n}})$.*

Proof. Let $\hat{G}$ be the largest supergroup of G in $\mathrm{SymPr}(V)$ which has the same blocks and the same block-link relation $\approx$. It is trivial to determine $\hat{G}$ in polynomial time. It suffices to find all $\hat{G}$-isomorphisms and then apply coset intersection with G. The $\hat{G}$-isomorphisms can be pieced together from the $\hat{G}$-isomorphisms of the restrictions of X and Y to each $\sim$ class. But each connected component of such a restriction is contained in a macro-block, so the pairwise isomorphisms of the connected components can be computed by brute force in $((\sqrt{n})!)^k$. □

3.5 Breaking Symmetry

3.5.1 The situation

Our overall strategy is to find either irregularities or structure. When we find irregularities, we have various techniques to reduce to subproblems that are more regular. In the end we will obtain the ultimate structure (cf. Section 3.4.4), which can be dealt with by Lemma 3.4.2.

The techniques we have discussed so far allow us to reach the following situation of regularity and apparent symmetry: the hypergraphs are fully regular, and we the action of the group is of strong giant type, i.e., the group has a giant action on blocks of each of the parts. If the action of the automorphism group on the blocks of each part is indeed giant, then we will observe the ultimate structure. Otherwise we must *constructively disprove* the giant action, and thus break the apparent symmetry. This is the crux of the problem, and our main technical contribution is to deal with this situation. We have more techniques that allow us to break symmetry. Individualization, with the goal of finding splits, or in order to obtain a large reduction in $|G|$ will be an important tool to break the apparent symmetry..

3.5.2 Individualization

Recall that to *individualize* a vertex $x \in V_i$ in X means to assign x a special color, say "red," in X, resulting in the structure X_x (cf. Section 2.4.2). For each $y \in V_i$, let $\sigma_y \in G$ move x to y. Now we have $\mathrm{ISO}(G, X, Y) = \bigcup_{y \in V_i} \mathrm{ISO}(G_x\, \sigma_y; X_x, Y_y)$.

This recurrence thus reduces an instance of the G-isomorphism to $\leq n$ instances of G_x-isomorphism. To individualize a set R of vertices means to individualize each $x \in R$ in succession; the cost is a factor of $\leq n^{|R|}$. By breaking the symmetry we hope to make sufficient progress to offset this factor.

3.5.3 Splits

A *split* is a G-invariant partition of the vertices. Finding a split will allow us to reduce the problem to the parts.

Let G act on V, with blocks of imprimitivity $\mathfrak{B}_i$ on V_i. Let $R \subseteq \mu(V)$, and $S \subseteq E$ or $S \subseteq V_i$ or $S \subseteq \mathfrak{B}_i$, such that S is invariant under the pointwise stabilizer $\mathrm{Aut}(G; X)_R$.

Definition 3.5.1. Let $S' \subseteq S$, $S' \neq \emptyset$, $S' \neq S$. We say that S' is a **split of S relative to R** if S' is invariant under $\mathrm{Aut}(G; X)_R$. If $R = \emptyset$, we say that this is an **absolute split**. If $|S|/\gamma \leq |S'| \leq (1 - 1/\gamma)|S|$ and $|R| \leq 2k$ then we say this is a **large split**; otherwise it is a **small split**.

Note that if we have an absolute split we can partition one of the V_i (or the $\mathfrak{B}_i$ or E), and recurse on the parts. If we find a “large” split relative to a “small” set R, we will individualize R pointwise, and recurse. Our R will always be the union of at most two partial transversals, hence the inequality $|R| \leq 2k$. When we have a “small” relative split, we need one more technique, “split amplification,” which will allows us to either get a larger split, or an absolute split.

3.5.4 Reduction of G

Most steps of the algorithm involve reducing G to some group H. The new group H is always a quotient of a subgroup of the original; in particular, $|H| \leq |G|$. If this reduction is achieved through individualization of a subset $R \subseteq \mu(V)$ then we say we have a *large relative reduction of* G if (a) $|H| < |G| \exp(-\gamma/2)$; and (b) $|R| \leq 2k$.

This reduction will be used in the context of Fact 3.2.11.

3.6 Measures of Progress

3.6.1 Events of progress

We now define a list of structures which will allow us *to make progress.* If we discover one of these structures, we take the recursive action stated in parentheses next to each structure. Henceforth, if we say “we make progress,” this phrase will always refer to one of the structures on this list.

1. $|V_i| = 1$ for some i (reduction: $k \leftarrow k-1$);
2. Aut(X)-invariant vertex-coloring (this includes the case when the action of G is intransitive on any of the V_i) (reduce to color-homogeneous parts, Section 3.3.1);
3. Aut(X)-invariant edge-coloring (reduce to color-homogeneous parts, Section 3.3.2; this includes Reduction 3.3.3 (all edges belong to a single G-orbit));

4. The G-action on a set of maximal blocks in one of the V_i is not a giant (divide-and-conquer, Section 3.4.2);
5. Absolute split or large relative split (reduce to the two parts of the split);
6. Large relative reduction of G (reduce G).

To analyze our algorithms, we consider the following operations as **elementary steps:** computing the coset of isomorphisms of a pair of graphs with $O(n)$ vertices; computing coset intersection of permutation groups acting on a k-partite domain in which each part has size $O(n)$; and operations requiring time $n^{O(k)}$. A **composite step** consists of any combination of $n^{O(k)}$ elementary steps.

The following result describes the nature of our algorithm; its proof follows in Section 3.7.

Theorem 3.6.1. *Our algorithm proceeds in a sequence of recursive phases each of which makes progress. The cost of each reduction is a single composite step.*

3.6.2 Potential – timing analysis

We introduce three potential functions on the pairs (X, G):

$$\Pi_1(X) = |E|\Pi_0(X) = |E| \cdot \prod_i |V_i|,$$

$$\Pi_2(G) = \exp \frac{4k \ln |G| \ln n}{\gamma},$$

$$\Pi(G, X) = \Pi_1(X)^{\gamma(2k+1)\ln n}\Pi_2(G).$$

The following theorem justifies the claimed running time for our algorithm.

Theorem 3.6.2. *Let $T(G, X)$ denote the number of composite steps required by the algorithm. Then*

$$T(G, X) \leq \Pi(G, X).$$

We show that this bound follows from Theorem 3.6.1. For notational convenience, set $\eta = \gamma(2k+1)\ln n$.

Proof. By induction on $\Pi(G, X)$. The base cases are $k \leq 2$ (a single application of Graph Isomorphism) and the "ultimate structure" (Section 3.4.4) which is settled by Lemma 3.4.2.

We note that neither Π_1 nor Π_2 increases during recursive calls. We distinguish cases according to the last event of progress; the numbering follows that of Section 3.6.1.

1. The reduction $k \leftarrow k-1$ decreases Π_1 by at least 1, and so takes care of the unit cost.

2-3. Invariant coloring. We reduce the problem (X, G) to instances (X_i, G_i), where $(\forall i)(\Pi_2(G_i) \leq \Pi_2(G))$, and

$$\Pi_1(X) = \sum_i \Pi_1(X_i). \tag{3.1}$$

Moreover, $T(G, X) \leq 1 + \sum_i T(G_i, X_i)$. (The one added composite step covers the cost of the reduction.) By the inductive hypothesis $T(G_i, X_i) \leq \Pi(G_i, X_i)$. So, by (3.1),

$$\frac{T(G,X)}{\Pi_2(G)} \leq 1 + \sum_i \Pi_1(X_i)^\eta < \Pi_1(X)^\eta.$$

4. Divide-and-conquer for imprimitive groups. Following the argument outlined in Section 3.4.2, let b be the number of blocks; we reduce to at most b^γ instances (X_i, G_i) where $(\forall i)(\Pi_2(G_i) \leq \Pi_2(G))$, and $(\forall i)(\Pi_1(X_i) = (1/b)\Pi_1(X))$. Again we have $T(G, X) \leq 1 + \sum_i T(G_i, X_i)$; hence

$$\frac{T(G,X)}{\Pi_2(G)} < 1 + \sum_i \Pi_1(X_i)^\eta \leq 1 + b^\gamma \left((1/b)\Pi_1(X)\right)^\eta < \Pi_1(X)^\eta.$$

5. Splits. We reduce the instance (G, X) to two instances $(G_1, X_1), (G_2, X_2)$. Again $(\forall i)(\Pi_2(G_i) \leq \Pi_2(G))$, and $\Pi_1(X) = \Pi_1(X_1) + \Pi_1(X_2)$. If the split is absolute then $T(G, X) \leq 1 + \sum_i T(G_i, X_i)$. By the inductive hypothesis, $T(G_i, X_i) \leq \Pi(G_i, X_i)$, hence $T(G, X)/\Pi_2(G) < 1 + \sum_i \Pi_i(X_i)^\eta < \Pi_1(X)^\eta$. If the split is a large relative split

then we have $\Pi_1(X_i) < (1-1/\gamma)\Pi_1(X)$ but we incur a multiplicative cost of at most n^{2k} for individualization. So $T(G,X)/\Pi_2(G) < 2n^{2k}\left((1-1/\gamma)\Pi_1(X)\right)^\eta < \Pi_1(X)^\eta$. (Note that in all other events of progress, we could have set $\eta = \gamma$; here we needed the factor $\eta/\gamma = (2k+1)\ln n$ to counter the multiplicative cost of n^2k due to individualization.)

6. Large relative reduction of G. In this case, $\Pi_2(G)$ is reduced by a factor of $\leq n^{-2k}$, offsetting the $\leq n^{2k}$ multiplicative cost of individualization. □

It follows that our algorithm has the claimed running time since initially $\Pi_1(X) \leq n^{2k}$ and

$$|G| \leq (n!)^k < n^{nk} \Rightarrow \Pi_2(G) \leq \exp\left(\frac{4k^2 n \ln^2 n}{\gamma}\right) < \exp(2k^2\sqrt{n}\ln^2 n).$$

Therefore

$$T(G,X) \leq \Pi(G,X) < \exp((4k\sqrt{n}\,(2k+1)\,\ln n) + (2k^2\sqrt{n}\ln n) = \exp(\sqrt{n}\,6k^2\,\ln^2 n).$$

3.7 The Algorithm

We focus on $X = (V,E)$; we repeat the same operations for Y, if there is a difference, we reject isomorphism. Our goal is to keep making progress (or reject isomorphism) until the ultimate structure (Section 3.4.4) is reached. As described in Section 3.5.1, through a sequence of "progress events" (Section 3.6.1) and reductions described in Sections 3.3 and 3.4, we reach a state where (1) G is of strong giant type; (2) all edges lie in the same G-orbit; (3) X is fully regular and has no isolated vertices.

One of our goals when individualizing will be to obtain a split in either some set of vertices V_i, some set of blocks $\mathfrak{B}_i$, or some set of macro-blocks $\mathfrak{M}_i$. If we can split off those vertices (or (macro)blocks) that are linked to the vertices we are individualizing, this split does not give us new information. We are therefore interested in splits of those vertices not linked to what we are individualizing. The following notation will help in the exposition of these kinds of situations.

Notation 3.7.1. *Let $t \subseteq \mu(V)$. Let Z be either $\mathfrak{B}_i$ or V_i or $\mathfrak{M}_r$. We define $Z(t)$ as the subset of Z consisting of the elements not block-linked to any vertex in t.*

3.7.1 Vertex splitting – split amplification

Let $T \subseteq \mathcal{P}(\mu(V))$ ($\mathcal{P}$ denotes the power-set). We say that an assignment $f : T \to \mathcal{P}(\mu(V))$ is *equivariant* if $f(t^\sigma) = f(t)^\sigma$ for every $t \in T$, $\sigma \in \operatorname{Aut}(G; X)$.

The following theorem is a key tool that will allow us to make progress any time we find a split. For $\mathfrak{R} \subseteq \mathfrak{L}$ we use $\operatorname{pr}_{\mathfrak{R}}(E)$ to denote the projection of E to $\bigcup_{I \in \mathfrak{R}} I$.

Theorem 3.7.2 (split amplification)**.** *Let $\mathfrak{R} \subseteq \mathfrak{L}$, $i \in [k]$, and let $\{S_t \subseteq V_i | t \in \operatorname{pr}_{\mathfrak{R}}(E)\}$ be an equivariant assignment of subsets of V_i to elements of $\operatorname{pr}_{\mathfrak{R}}(E)$. If $\exists t^* \in \operatorname{pr}_{\mathfrak{R}}(E)$ such that S_{t^*} is a split of $V_i(t^*)$ then in a single composite step we can make progress.*

Proof. We will prove this by reverse induction on $|\mathfrak{R}|$. If $|\mathfrak{R}| = 0$, then we have an absolute split, hence progress.

The strategy is to find a large relative split or an absolute split. If the initial splits were large, then we can individualize one partial transversal t, progress. Otherwise the goal is to obtain splits relative to smaller partial edges (see Figure 3.6). Since all of the splits are small, we can take the union over a block (which is also small). Continuing in this way, we obtain either a large split, or splits relative to macro-blocks. Then we make use of Lemma 3.2.9, which allows us to either make progress or "un-individualize" further (i.e., make our splits relative to smaller hyperedges). If we never make progress along the way, we will eventually obtain an absolute split.

The algorithm proceeds in steps; for each step either we make progress, or we have some additional structure. We will break down the proof into claims, each of which will have the form "either we make progress, or we can impose some further condition on X." After each claim we will assume X satisfies all the conditions of the previous claims.

Claim 3.7.3. *Either we make progress or* $(\forall t, t' \in \operatorname{pr}_{\mathfrak{R}}(E))(|S_t| \neq |S_{t'}|,$ *and* $|S_t| < |V_i|/\gamma)$.

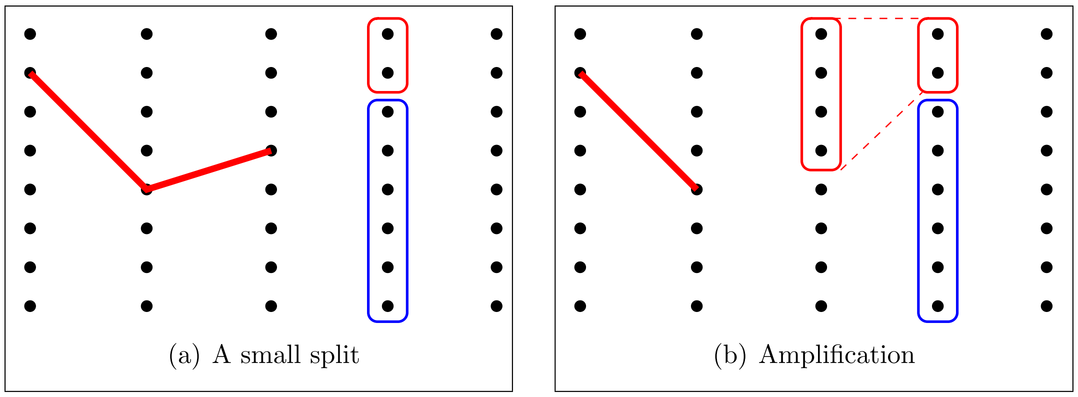

Figure 3.6: (a) We are given a small split (red vs. blue vertices) relative to the red partial edge. (b) Our strategy is to obtain a large split or a split relative to a smaller partial edge.

Proof. If $|S_t| \neq |S_{t'}|$ for some $t, t' \in \mathrm{pr}_{\mathfrak{R}}(E)$, then we color $\mathrm{pr}_{\mathfrak{R}}(E)$ by $|S_t|$, which is progress. W.l.o.g. $|S_t| \leq |V_i|/2$. If S_t is a large split of V_i relative to t, progress. $\square$

Since S_t is a small split of V_i, not all blocks $\mathfrak{B}_i(t)$ contain elements of S_t. Call the blocks in $\mathfrak{B}_i(t)$ that contain any elements of S_t **special**, and let the set of macro blocks containing a special block be $\mathrm{sp}(t)$. Let $L \in \mathfrak{L}$ such that $i \in L$.

Claim 3.7.4 (Extending the splits to splits of the macro-blocks)**.** *Either we make progress, or* $(\forall t \in \mathrm{pr}_{\mathfrak{R}}(E))(|\mathrm{sp}(t)| < |\mathfrak{M}_L(t)|/\gamma)$.

Proof. If for any t, $\mathrm{sp}(t)$ is a large split of $\mathfrak{M}_L(t)$, then we have a large split of V_i, progress. $\square$

Pick any $R = \{r_1, \ldots, r_\alpha\} \in \mathfrak{R}$. Let $\mathrm{sp}(\alpha+1; t) = \mathrm{sp}(t)$. For every $1 < \beta \leq \alpha$, and every $u \in \mathrm{pr}_{\mathfrak{R}\setminus\{r_\beta,\ldots,r_\alpha\}}(E)$, let $V(u,\beta) = \{v \in V_{r_\beta} : (u,v) \in \mathrm{pr}_{\mathfrak{R}\setminus\{r_{\beta+1},\ldots,r_\alpha\}}(E)\}$, and let $\mathrm{sp}(\beta; u) = \bigcup_{v \in V(u,\beta)} \mathrm{sp}(\beta+1; (u,v))$. Here (u,v) denotes the partial edge that is u augmented by v in the appropriate coordinate. For each $M \in \mathfrak{M}_R$ and $u \in \mathrm{pr}_{\mathfrak{R}\setminus\{R\}}(E)$, let $\mathrm{sp}(u, M) = \bigcup_{v \in M \cap V_{r_1}} \mathrm{sp}(1; (u,v))$.

Claim 3.7.5. *Either we make progress, or* $(\forall M \in \mathfrak{M}_R)(\forall u \in \mathrm{pr}_{\mathfrak{R}\setminus\{R\}}(E))(|\mathrm{sp}(u,M)| < |\mathfrak{M}_L|/\gamma)$.

Proof. If any of the sets $\mathrm{sp}(\beta; u)$ constructed along the way is a large split of $\mathfrak{M}_L$, progress. Else, we claim that $|\mathrm{sp}(\beta; u)| \leq |\mathfrak{M}_L|/\gamma = |\mathfrak{B}_i|/\gamma$, where $\gamma = 2\sqrt{n}$ is our "giant threshold."

This follows by reverse induction on β. Indeed, by induction, $\operatorname{sp}(\beta; u)$ is the union of $\leq n/\gamma$ sets, each of size $\leq |\mathfrak{M}_L|/\gamma$, so $|\operatorname{sp}(\beta; u)| \leq |\mathfrak{M}_L|/2$ and therefore $|\operatorname{sp}(\beta; u)| \leq |\mathfrak{M}_L|/\gamma$ (otherwise it would be a large split). □

Claim 3.7.6. *Either we make progress or we obtain an equivariant assignment* $\{S_u \mid u \in \operatorname{pr}_{\mathfrak{R}\setminus\{R\}}(E)\}$ *such that one* S_{u^*} *is a split.*

Proof. We create graphs $Y(u)$ for each $u \in \operatorname{pr}_{\mathfrak{R}\setminus\{R\}}(E)$. If $R \neq L$ then $Y(u)$ will be bipartite with vertex set $(\mathfrak{M}_R(u), \mathfrak{M}_L(u))$; if $R = L$ then $Y(u)$ is a graph with vertex set $\mathfrak{M}_L(u)$. In both cases, the edge set is $\{(M \in \mathfrak{M}_R(u), N \in \mathfrak{M}_L(u)) : \; N \in \operatorname{sp}(u, M)\}$. We will refer to $\mathfrak{M}_R(u)$ and $\mathfrak{M}_L(u)$ as the parts of $Y(u)$.

Now for each $u \in \operatorname{pr}_{\mathfrak{R}\setminus\{R\}}$, we use the Graph Isomorphism algorithm to compute $\operatorname{Aut}(Y(u))$. Let S_u be a largest orbit of $\operatorname{Aut}(Y(u))$ on V_j. If any of the S_u is a large split, progress. Else, if any S_u is a split, we have found the desired equivariant assignment. Else, $\operatorname{Aut}(Y(u))$ is transitive on each part of $Y(u)$.

If the action of $\operatorname{Aut}(Y(u))$ is transitive on each part of $Y(u)$ but the not giant on either part, then we obtain a large reduction of G relative to u (Fact 3.2.11), progress. Else, we claim that $R \neq L$ and $Y(u)$ is a matching. This follows by Lemma 3.2.9 from the fact that the density of $Y(u)$ is positive and $\leq 1/2$. This means we found a linking between previously unlinked blocks, yielding a large relative reduction in G, progress. □

If we did not make progress so far, then we found an equivariant assignment $\{S_u\}$ of subsets of V_j to elements of $\operatorname{pr}_{\mathfrak{R}\setminus\{R\}(E)}$, and one u^* such that S_{u^*} is a split, and hence we are done by induction on $|\mathfrak{R}|$.

To analyze the cost note that for each partial edge, we do polynomial work and one Graph Isomorphism test. □

Theorem 3.7.7 (double split amplification). *Let* $\mathfrak{R}_j \subseteq \mathfrak{L}$ *for* $j = 1, 2$; *let* $i \in [k]$; *and let* $\{S_T | T \in \operatorname{pr}_{\mathfrak{R}_1}(E) \times \operatorname{pr}_{\mathfrak{R}_2}(E)\}$ *be an equivariant assignment of subsets of* $V_i(T)$ *to elements of* $\operatorname{pr}_{\mathfrak{R}_1}(E) \times \operatorname{pr}_{\mathfrak{R}_2}(E)$. *If* $\exists T \in \operatorname{pr}_{\mathfrak{R}_1}(E) \times \operatorname{pr}_{\mathfrak{R}_2}(E)$ *such that* S_T *is a split of* $V_i(T)$ *then in* *a single composite step we can make progress.*

Proof. Fix $t_1 \in \mathrm{pr}_{\mathfrak{R}_1}(E)$ and perform the split amplification with respect to $\mathrm{pr}_{\mathfrak{R}_1}(E)$. If the split amplification makes progress that involves individualization, then we will individualize all of t_1 and the partial transversal that was individualized by split amplification, and make the same progress. Otherwise split amplification makes progress by finding a small absolute split; but this is a split relative to t_1. We repeat this process for all $t_1 \in \mathrm{pr}_{\mathfrak{R}_1}(E)$. If for all of them we do not make progress, then we have an equivariant assignment of splits to $\mathrm{pr}_{\mathfrak{R}_1}(E)$, and hence we can make progress by invoking split amplification once more. The cost is $\mathrm{poly}(n)$ $+O(n^k)$ calls to split amplification. $\square$

3.7.2 Structure

Theorem 3.7.8. *If we do not observe the ultimate structure then we make progress.*

Proof. For $L \in \mathfrak{L}$, let $\overline{L} = [k]\backslash L$. Fix any $L \in \mathfrak{L}$. For each $t \in \mathrm{pr}_{\overline{L}}(E)$ and $M \in \mathfrak{M}_L$ we define a hypergraph $X(t, M) = (M, E(t, M))$ where $E(t, M) = \{u \in \mathrm{pr}_L(E) \cup \pi(M) \,:\, (t, u) \in E\}$, where (t, u) stands for the concatenation of t and u.

As in the proof of Theorem 3.7.2, we present a series of claims of the form "either we make progress, or we can impose some further condition on X". We will use these claims to finally prove that if we do not observe the ultimate structure, then we make progress. After each claim, we will assume X satisfies the conditions from all the previous claims.

Claim 3.7.9. *Either we make progress, or* $(E(t, M) = \emptyset \iff M \notin \mathfrak{M}_L(t))$.

Proof. If $M \notin \mathfrak{M}_L(t)$ then $E(t, M) = \emptyset$ since all edges in an edge-linked-section are within macro-blocks (see Section 3.4.3).

If $E(t, M) = \emptyset$ for some $M \in \mathfrak{M}_L(t)$, then for every $s \in \mathrm{pr}_{\overline{L}}(E)$, the set of macro-blocks with empty hypergraphs is an invariant partition of $\mathfrak{M}_L(s)$. Moreover, for t the set of empty macro blocks is a split of $\mathfrak{M}_L(t)$, since we are assuming $\exists M$ such that $E(t, M) = \emptyset$, but

$\exists M'$ such that $E(t, M') \neq \emptyset$ since t is a partial edge. Thus we make progress via the split amplification (Theorem 3.7.2).

$\square$

Claim 3.7.10. *Either we make progress, or all the $X(t, M)$ are isomorphic. More precisely,* $(\forall\, t, t' \in \mathrm{pr}_{\overline{L}}(E))(\forall M \in \mathfrak{M}_L(t))(\forall M' \in \mathfrak{M}_L(t'))(X(t, M) \cong X(t', M'))$

Proof. Let $t, t' \in \mathrm{pr}_{\overline{L}}(E)$, $M \in \mathfrak{M}_L(t)$, $M' \in \mathfrak{M}_L(t')$, such that $X(t, M) \ncong X(t', M')$. Note by the previous claim that $E(t, M), E(t', M') \neq \emptyset$, but then the following is an invariant coloring of E. For any $e \in E$, let $t(e) = \mathrm{pr}_{\overline{L}}(e)$, and let $M(e) = \mathrm{macroblock}(\mathrm{pr}_L(e))$ be the macro-block containing $\mathrm{pr}_L(e)$. Then color e by the isomorphism type of $X(t(e), M(e))$.

$\square$

If X does not have the structure, we can find some L^*, $t \in \mathrm{pr}_{\overline{L^*}}(E)$, and $u \in \mathrm{pr}_{L^*}(E)$ such that $(t, u) \notin E$. But since $u \in \mathrm{pr}_{L^*}(E)$, $\exists s \in \mathrm{pr}_{\overline{L^*}}(E)$ such that $(s, u) \in E$. Let $R = L^*$. Note that in particular, we will have $X(s, M) \neq X(t, M)$ for some $M \in \mathfrak{M}_R$.

For $s, t \in \mathrm{pr}_{\overline{R}}(E)$, and $M \in \mathfrak{M}_R$, let $X(s, t; M)$ be the superposition of the hypergraphs $X(s, M)$ and $X(t, M)$, that is, the colored hypergraph with the edges of $X(s, M)$ of one color and the edges of $X(t, M)$ of another color. We denote the edges of $X(s, t; M)$ by $E(s, t; M)$. Note that if X has the structure, then $X(s, t; M)$ will be the superposition of two identical hypergraphs. We can compute isomorphisms of the $X(s, t; M)$ by exhaustive search in time $\exp(O(k\sqrt{n}))$.

Claim 3.7.11. *Either we make progress, or all the $X(s, t; M)$ are isomorphic.*

Proof. Let $\mathcal{H}$ be the set of isomorphism types of the $|R|$-partite $|R|$-hypergraphs $X(s, t, M)$ ($s, t \in \mathrm{pr}_{\overline{R}}(E)$, $M \in \mathfrak{M}_R$). Fix $H \in \mathcal{H}$. Let $S(s, t) = \{M \in \mathfrak{M}_R : X(s, t; M) \cong H\}$. This is an equivariant map on the pairs (s, t). If $S(s, t)$ splits $\mathfrak{M}_R$ for some s, t, H then we have a split of V_j ($j \in R$) and we make progress by the double split amplification (Theorem 3.7.7).

Else we conclude that the isomorphism type of $X(s, t, M)$ does not depend on M, since the same argument applies to all the isomorphism classes. Let us again fix $H \in \mathcal{H}$. Let

$S(t) = \{s \in \mathrm{pr}_{\overline{R}}(E) : X(s,t;M) \cong H\}$. This again is an equivariant map. Again, if for some t, H this is a split, we make progress by the split amplification (Theorem 3.7.2). Else we conclude that the isomorphism type of $X(s,t;M)$ does not depend on s; by symmetry, it also does not depend on t.

□

Claim 3.7.12. *Either we make progress, or the automorphism groups* $A(s,t;M) := \mathrm{Aut}(X(s,t;M))$ *act transitively on the partial edges in* M *(i. e., on* $E(s,t;M)$*).*

Proof. If $A(s,t;M)$ is not vertex transitive on the vertices of one of the parts of M then we have a split into orbits relative to s, t, and M. It is the same size split in all the macro blocks in $\mathfrak{M}_R$, so it is a large split since there are $\geq \gamma$ macro blocks. Progress.

Our next goal is to achieve that $A(s,t;M)$ is transitive on $E(s,t;M)$.

We iterate through R as follows. Let $R = \{i_1, \ldots, i_{|R|}\}$ and let $R_\ell = \{i_1, \ldots, i_\ell\}$. We prove by induction on ℓ that either $A(t,s;M)$ acts transitively on $\mathrm{pr}_{R_\ell}(X(t,s;M))$ or we make progress. We have shown this for $\ell = 1$ (vertex-transitivity). Let $\ell \geq 2$ be the smallest value such that $A(t,s;M)$ acts intransitively on $\mathrm{pr}_{R_\ell}(X(t,s;M))$. Then for any partial edge $e \in \mathrm{pr}_{R_{\ell-1}}(X(t,s;M))$, we get a split of $\mathrm{pr}_{R_\ell}(X(t,s;M))$ into orbits, and hence a split of V_{i_ℓ} relative to t, s and e. The split is large, since it is a split in every block of V_{i_ℓ}, progress.

□

Claim 3.7.13. *Either we make progress, or* $(\forall s,t \in \mathrm{pr}_{\overline{R}}(E))(\forall M \in \mathfrak{M}_R = \mathfrak{M}_R)(X(t,M) = X(s,M))$ *and therefore we have the ultimate structure.*

Let $M \in \mathfrak{M}_R$ be a macro-block, and $s, t \in \mathrm{pr}_{\overline{R}}(E)$ be a pair of partial edges, such that $X(s,M) \neq X(t,M)$. Look at the smallest ℓ (over all choices of $s, t \in \mathrm{pr}_{\overline{R}}(E)$, and $M \in \mathfrak{M}_R$) such that $X(s,M)$ and $X(t,M)$ are not identical when restricted to the first ℓ parts of M.

Note that ℓ cannot be 1: $A(s,t;M)$ acts transitively on the vertices in each part of M.

Note further that because $A(s,t;M)$ acts transitively both on $E(t,M)$ and on $E(s,M)$, these sets are either identical or disjoint, and so are any of their projections.

Define equivalence classes on $\mathrm{pr}_{\overline{R}}(E)$, where two partial edges s and t are equivalent if and only if $X(s,M)$ and $X(t,M)$ are identical up to level ℓ. We individualize one of the equivalence classes; this splits E. We claim this is a large (relative) split.

Indeed, fix a partial edge e of length $\ell-1$ in $X(s,M)$. (Thus is also a partial edge of $X(t,M)$ for all t.) Let B be the block of M in V_{i_ℓ}. Let $B(t) = \{v \in B \text{ s.t. } (t,e,v) \text{ is a partial edge}\}$. The $B(t)$ partition B. On the other hand, the relation $v \in B(t)$ defines a biregular bipartite graph on $(\mathrm{pr}_{\overline{R}}(E), B)$ (because of full regularity). It follows that all equivalence classes are the same size. Their number is at most $|B| \le n/\gamma$. $\square$

This concludes the proof of Theorem 3.7.8. $\square$

3.8 Open Problems

The big open problem continues to be to reduce the $\widetilde{O}(\sqrt{n})$ term in the exponent of the complexity of Graph Isomorphism to $n^{1/2-\epsilon}$. Our result for 4-hypergraphs may open the path to a reduction of the exponent to $\widetilde{O}\left(n^{1/4}\right)$.

Another problem is to extend our moderately exponential isomorphism test to hypergraphs of rank $n^{1-\epsilon}$.

Finally, our work brings new life to an old dilemma: *isomorphism testing vs. canonical forms* (cf. [23, 38], Section 2.2.2). Because of the use of Coset Intersection, our algorithm does not yield canonical forms. As Gene Luks points out, his simply exponential (C^n) isomorphism test for hypergraphs (of arbitrary rank) [57] does not yield canonical forms either. So the problem is, find canonical forms (a) for hypergraphs in simply exponential time and (b) for hypergraphs of bounded rank in moderately exponential time.

3.9 Appendix: Pseudocode

In this appendix we present the pseudocode for the hypergraph isomorphism algorithm. Table 3.1 describes the subroutines for individualization, Table 3.2 the color reduction, Ta-

Table 3.1: Our main procedure, HYPISO, will use two subroutines for individualization, both of which take as input a partial edge $t \in pr_\ell(E(X))$. The macro **individualize** individualizes t and iterates over all choices; the procedure REC_INDIV, recursively calls HYPISO after individualization (cf. Section 3.5.2).

```
individualize(t)
00    $G \leftarrow G_t$
01    for $w \in t^G$
02        $Y \leftarrow Y^{\sigma_w}$
(: All the commands that follow this macro will be understood to be inside the for loop :)
```

```
Procedure REC_INDIV(t)
00    $G \leftarrow G_t$
01    for $w \in t^G$
02        $Y \leftarrow Y^{\sigma_w}$
03        HYPISO$(G; X, Y)$
```

ble 3.3 the split amplification, and Table 3.4 the double split amplification. We give the pseudocode for the preliminary reductions of the main algorithm in Table 3.5 and the crux of the main algorithm in Table 3.6.

Table 3.2: If there is more than one color class, the following procedure refines the coloring g on the partial edges to a coloring on the edges, and calls HYPISO for each color class (cf. Section 3.3.2).

Procedure COLOR_REFINE(G; X, Y; g; L)

Input: the group G, and hypergraphs X and Y
with a coloring g of partial transversals on the parts in $L \subseteq [k]$.

Output: calls HYPISO on each of the color classes.

```
00    for $t \in \mathrm{pr}_L(E(X))$ for $e = (t, u) \in E(X)$
01        $f(e) \leftarrow g(t)$
02    for $Z = X, Y$ for $c$ a color class
03        $E(Z)^c \leftarrow \{e \in E(X) | f(e) = c\}$
04    return $\cap_c$ HYPISO$(G; (V, E(X)^c), (V, E(Y)^c))$
```

Table 3.3: This procedure allows us to make progress whenever we find a split (cf. Theorem 3.7.2).

```
Procedure SPLIT_AMPLIFICATION($\mathfrak{R}$, $\{S_t\}$, $u$)

Input:
$\mathfrak{R} \subseteq \mathfrak{L}$ is a set of edge-linked sections,
$\{S_t \subseteq V_i | t \in \mathrm{pr}_{\mathfrak{R}}(E)\}$ is an equivariant assignment of partial edges to subsets of $V_i$,
$u \in \mathrm{pr}_{\mathfrak{R}}(E)$ is a partial edge such that $S_u$ is a split of $V_i$.

Output:
progress (i.e., recursive calls to HYPISO).

1. (The $S_t$ are not all of the same size)
00    Color the partial edges $t$ by $|S_t|$
01    COLOR_REFINE
(: Henceforth all the $S_t$ are of the same size :)

2. (One of the $S_t$ is a large split)
00    REC_INDIV $t$
(: Henceforth all $|S_t|$ are "small" :)

3. (One of the special sets is a large split)
00    $\mathrm{sp}(t) \leftarrow \{\mathrm{macroblock}(v) | v \in S_t\}$
01    if $\mathrm{sp}(t)$ a large split for any $t$, REC_INDIV($t$)
02    construct $\mathrm{sp}(u; M)$
03    if a large split is found along the way, REC_INDIV
(: Henceforth all $|^{(}u; M)|$ are "small" :)

4. (Auxiliary graphs $Y(u)$)
00    construct the graphs $Y(u)$
01    use GRAPH ISOMORPHISM to compute $\mathrm{Aut}(Y(u))$
02    if $\mathrm{Aut}(Y(u))$ is not transitive then
03        if any orbit is a large split REC_INDIV($t$)
04        else construct equivariant assignment and recurse
05    else if $\mathrm{Aut}(Y(u))$ not giant
06        REC_INDIV($u$)
07    else $Y(u)$ is a matching $\Rightarrow$ new linking
```

Table 3.4: The following procedure allows us to make progress whenever we find a split relative to two partial edges (cf. Theorem 3.7.7).

Procedure DOUBLE_SPLIT_AMPLIFICATION($\mathfrak{R}_1, \mathfrak{R}_2$, i, $\{S_T\}$, U)

Input:
$\mathfrak{R}_1, \mathfrak{R}_2 \subseteq \mathfrak{L}$ are sets of edge-linked sections,
$\{S_T \subseteq V_i \mid T \in \mathrm{pr}_{\mathfrak{R}_1}(E) \times \mathrm{pr}_{\mathfrak{R}_2}(E)\}$ is an equivariant assignment of partial edges to subsets of V_i,
$U \in \mathrm{pr}_{\mathfrak{R}_1}(E) \times \mathrm{pr}_{\mathfrak{R}_2}(E)$ is a pair of partial edges such that S_U is a split of $V_i(T)$.

Output:
Recursive calls to HYPISO.

```
00  forall $t_1 \in \mathrm{pr}_{\mathfrak{R}_1}(E)$
01      SPLIT_AMPLIFICATION($\mathfrak{R}_2$, $\{S_{(t_1,t)}\}$, $pr_{\mathfrak{R}_2}(U)$)
02      if SPLIT_AMPLIFICATION finds a small absolute split then
03          this is a split $S_t$ relative to $t_1$
04      else
05          made progress
06  SPLIT_AMPLIFICATION($\mathfrak{R}_1$, $\{S_t\}$, $t_1$)
```

Table 3.5: The preliminary steps of the main algorithm (cf. Sections 3.3 and 3.4).

```
Procedure HYPISO($G$; $X$, $Y$)

Input:
$X$ and $Y$ are $k$-partite $k$-hypergraphs over the same vertex set $V$,
$G \leq \mathrm{Sym}(G)$ is a group

Output:
$\mathrm{ISO}(G; X, Y)$.

0. ($|V_i| = 1$ for some $i$)
(: Henceforth for each $i \in [k]$, $|V_i| > 1$ :)

1. (Intransitive G-action)
00    Luks divide and conquer on orbits
      (cf. Section 3.4.1)
(: Henceforth for each $i \in [k]$, $G$ acts transitively on $V_i$ :)

2. (Non-strong giant G-action)
00    Luks divide and conquer on non-giant primitive actions
      (cf. Section 3.4.2)
(: Henceforth for each $i \in [k]$, $G$ acts as a giant on $\mathfrak{B}_i$ :)

3. (Non-strong giant G-action)
00    Find a $V_i$ with imprimitive action on 2 blocks.
      (cf. Section 3.4.2)
(: Henceforth $G$ is of strong giant type :)

4. (Not fully regular)
00    Color the edges by link-type
01    COLOR_REFINE
(: Henceforth $X$ and $Y$ are fully regular :)
```

Table 3.6: The crux of the main algorithm (cf. Theorem 3.7.8).

```
5. (A residual hypergraph $X(t, M)$ with $t$ non-linked to $M$ is empty)
00  split relative to this partial edge.
01  SPLIT_AMPLIFICATION
(: Henceforth residual hypergraphs $X(t, M)$ are empty if and only if $t$ and $M$ are linked :)

6. (Two non-empty residual hypergraphs not isomorphic)
00  invariant coloring of $E$
01  COLOR_REFINE
(: Henceforth all nonempty residual hypergraphs are isomorphic :)

7. (Superposition residual hypergraphs not all isomorphic)
00  if iso type of $X(s, t; M)$ depends on $M$ then
01      DOUBLE_SPLIT_AMPLIFICATION
02  else
03      SPLIT_AMPLIFICATION
(: Henceforth all the superposition residual hypergraphs are isomorphic :)

Fix $R \in \mathfrak{L}$, as in discussion after Claim 3.7.10.

8. (Aut(X(s, t; M)) not transitive on E(s, t; M))
00  if not vertex transitive then
01      large relative split $\rightarrow$ individualize
02  else
03      for $\ell = 2$ to $|R|$
04          if Aut intransitive on first $\ell$ parts then
05              large split of $V_i$ relative to $s, t, e, R$
06              for any $e \in \text{pr}_{R_{\ell-1}}(X(t, s; M))$
07              individualize $s, t, e$, and $R$.
(: Henceforth $\text{Aut}(X(s, t; M))$ acts transitively on $E(s, t; M)$ :)

9. (Two residual hypergraphs on the same macro-block are not identical)
00  equivalence classes on $\text{pr}_{\overline{R}}(E)$
01  individualize one equivalence class
(: Henceforth all residual hypergraphs on any macro-block are identical
   Hence we have the ultimate structure :)

10. (Ultimate structure- Section 3.4.4)
00  Compute $ISO(\hat{G}; X, Y)$ by brute force on edge linked hypergraphs
01  return $G \cap ISO(\hat{G}; X, Y)$
```

CHAPTER 4

CODES

4.1 Definitions

Definition 4.1.1 (code)**.** A **code** of length n over a finite alphabet Γ is a subset of Γ^A for some set A with $|A| = n$.

An equivalence of the codes $\mathcal{A} \subseteq \Gamma^A$ and $\mathcal{B} \subseteq \Gamma^B$ is a bijection $\pi : A \to B$ that takes $\mathcal{A}$ to $\mathcal{B}$. More formally, for $x \in \Gamma^A$, define $x^\pi \in \Gamma^B$ by $x^\pi(b) = x(b^{\pi^{-1}})$. For a subset $S \subseteq \mathcal{A}$, let $S^\pi = \{x^\pi \mid x \in S\}$.

Definition 4.1.2 (Equivalence)**.** A bijection π is an **equivalence** of $\mathcal{A}$ and $\mathcal{B}$ if $\mathcal{A}^\pi = \mathcal{B}$.

If $|\Gamma| = 2$ then the code is a boolean function (i.e. a hypergraph), so the problem of deciding equivalence of codes generalizes the hypergraph isomorphism problem. Inspired by Luks's dynamic programming algorithm for hypergraph isomorphism [57], we showed that the code equivalence problem can be solved in time $(c|\Gamma|)^{2n}$, where c is an absolute constant, and n is the length of the codes ($|A| = |B| = n$) [17].

In [18] we introduced a generalization of the code equivalence problem, necessary for our algorithm to test isomorphism of semisimple groups. In addition to permuting the coordinates, we let a permutation group $G \leq \mathrm{Sym}(\Gamma)$ act on the symbols, independently on each coordinate. We call this more general problem **twisted code equivalence**. We were able to solve twisted code equivalence by generalizing the algorithm for code equivalence and improving its data management. The improvement in data management allows us an improved analysis of the algorithm, taking the size of the code into account. For the special case of code equivalence we obtain a running time of $O(c^n \cdot |\Gamma||\mathcal{A}|^{2.01})$, where c is an absolute constant. Note that the new running time has a polynomial dependence on $|\Gamma|$, to be contrasted with the exponential dependence in the previous algorithm.

Formally, given a bijection $\pi : A \to B$ and a function $g : B \to G$, we associate with $\psi = (\pi, g)$ a map $\psi : \Gamma^A \to \Gamma^B$, $\psi : x \mapsto x^\psi$, defined by $x^\psi(b) = (x(b^{\pi^{-1}}))^{g(b^{\pi^{-1}})}$. Twisted code equivalence is defined as follows.

Definition 4.1.3 (Twisted Equivalence)**.** Let $\mathcal{A} \subseteq \Gamma^A$, $\mathcal{B} \subseteq \Gamma^B$ be codes of length n over Γ. Let G act on Γ. Given a bijection $\pi : A \to B$ and a function $g : B \to G$, we say $\psi = (\pi, g)$ is a G**-twisted equivalence** of the codes $\mathcal{A}$ and $\mathcal{B}$ if $\mathcal{A}^\psi = \mathcal{B}$. We denote the set of all twisted equivalences of $\mathcal{A}$ and $\mathcal{B}$ by $\mathrm{EQ}_G(\mathcal{A}, \mathcal{B})$.

Note that if $A = B$ (which can be assumed w.l.o.g.), then $\mathrm{EQ}_G(\mathcal{A}, \mathcal{B}) \subseteq G \wr \mathrm{Sym}(A)$ is either empty or a coset of $\mathrm{EQ}_G(\mathcal{A}, \mathcal{A})$.

We shall need a generalization of the above to codes over multiple alphabets: let $\Gamma_1, \ldots, \Gamma_r$ be disjoint finite alphabets. A string of **length** $(k_1, \ldots, k_r)$ over $(\Gamma_1, \ldots, \Gamma_r)$ is a set of maps $x_i \colon A_i \to \Gamma_i$, denoted collectively as x, where $|A_i| = k_i$. A code of length $(k_1, \ldots, k_r)$ has **total length** $n = \sum_{i=1}^r k_i$. The set of all strings of length $(k_1, \ldots, k_r)$ over the alphabets Γ_i is $\prod_{i=1}^r \Gamma_i^{A_i}$. A **code** of length $(k_1, \ldots, k_r)$ with domain $(A_1, \ldots, A_r)$ is a subset $\mathcal{A} \subseteq \prod_{i=1}^r \Gamma_i^{A_i}$.

To define twisted code equivalence over multiple alphabets, for every $i = 1, \ldots, r$, let $G_i \leq \mathrm{Sym}(\Gamma_i)$ be a group acting on the alphabet Γ_i, and let $\mathcal{G} = (G_i)_{i=1}^r$. Given bijections $\pi_i : A_i \to B_i$ and functions $g_i : B_i \to G_i$, for $i = 1, \ldots, r$, let $\psi_i := (\pi_i, g_i)$, and $\psi = (\psi_i)_{i=1}^r$. For $a = (a_1, \ldots, a_r) \in \prod_{i=1}^r \Gamma_i^{A_i}$, define $a^\psi := (a_i^{\psi_i})_{i=1}^r$. As before a $\mathcal{G}$**-twisted equivalence** of two codes $\mathcal{A} \subseteq \prod \Gamma_i^{A_i}$, $\mathcal{B} \subseteq \prod \Gamma_i^{B_i}$ is a ψ such that $\mathcal{A}^\psi = \mathcal{B}$. The set of all $\mathcal{A} \to \mathcal{B}$ $\mathcal{G}$-twisted code equivalences will be denoted by $\mathrm{EQ}_{\mathcal{G}}(\mathcal{A}, \mathcal{B})$.

Each $\psi \in \mathrm{EQ}_{\mathcal{G}}(\mathcal{A}, \mathcal{B})$ naturally induces a bijection between $\mathcal{A}$ and $\mathcal{B}$, by sending $a \in \mathcal{A}$ to $a^\psi \in \mathcal{B}$. We denote by $\widehat{\psi}$ the induced map on strings, and let $\widehat{\mathrm{EQ}}_{\mathcal{G}}(\mathcal{A}, \mathcal{B}) = \{\widehat{\psi} : \psi \in \mathrm{EQ}_{\mathcal{G}}(\mathcal{A}, \mathcal{B})\}$.

Again note that if $(\forall i)(A_i = B_i)$, $\mathrm{EQ}_{\mathcal{G}}(\mathcal{A}, \mathcal{B})$ is either empty or a coset of $\mathrm{EQ}_{\mathcal{G}}(\mathcal{A}, \mathcal{A})$ in $\prod_{i=1}^r G_i \wr \mathrm{Sym}(A_i)$.

4.2 The Algorithm for Twisted Code Equivalence

Because we will use the algorithm for Coset Intersection (Theorem 2.3.10) as a subroutine, the degree of the permutation representation of $\mathrm{EQ}_{\mathcal{G}}(\mathcal{A}, \mathcal{A})$ will come into play. In particular, we will get good bounds on the running time if we have low-degree faithful permutation representations of the G_i.

Theorem 4.2.1. *Let $\mathcal{A} \subseteq \prod_{i=1}^{r} \Gamma_i^{A_i}$ and $\mathcal{B} \subseteq \prod_{i=1}^{r} \Gamma_i^{B_i}$ be codes of total length n. For $i = 1, \ldots, r$, let G_i be a permutation group acting on Γ_i, and assume we are given a faithful permutation representation of G_i of degree d_i. Let $G_{\max}$ and $\Gamma_{\max}$ such that $|G_{\max}| = \max_{i\in[r]} |G_i|$ and $|\Gamma_{\max}| = \max_{i\in[r]} |\Gamma_i|$. Then $\mathrm{EQ}_{(G_1,\ldots G_r)}(\mathcal{A}, \mathcal{B})$ can be found in time $O(n^2 \cdot 2^{n+d} \cdot |G_{\max}||\Gamma_{\max}| \cdot |\mathcal{A}|^2 \log |\mathcal{A}|)$, where $d = \sum_{i=1}^{r} |A_i| d_i$ is the degree of a faithful permutation representation of $\prod_{i=1}^{r} G_i \wr \mathrm{Sym}(A_i)$.*

Proof. For subsets $U_i \subseteq A_i$ we call a function $y\colon \bigcup_{i=1}^{r} U_i \to \bigcup_{i=1}^{r} \Gamma_i$ a "partial string over $A = (A_1, \ldots, A_r)$." We call the tuple $(|U_1|, \ldots, |U_r|)$ the *length* of y. For every partial string y over A, let $\mathcal{A}_y$ be the set of strings in $\mathcal{A}$ that are extensions of y. We make analogous definitions for $\mathcal{B}$. Let $\mathcal{G} = (G_1, \ldots, G_r)$.

We construct a dynamic programming table with an entry for each pair (y, z) of partial strings, y over A and z over B, of equal length such that $\mathcal{A}_y \neq \emptyset$, and $\mathcal{B}_z \neq \emptyset$. For each such pair (y, z), we store the set $I(y, z)$ of $\mathcal{G}$-twisted equivalences of the restriction of $\mathcal{A}_y$ to $A \setminus \mathrm{dom}(y)$ with the restriction of $\mathcal{B}_z$ to $B \setminus \mathrm{dom}(z)$. Note that these sets are either empty or cosets, so we store them by a set of generators and a coset representative. Also note the condition $\mathcal{A}_y \neq \emptyset$ and $\mathcal{B}_z \neq \emptyset$ implies that only partial strings that actually appear in $\mathcal{A}$ and $\mathcal{B}$ will be recorded. This will help us bound the size of the dynamic programming table.[1]

We start with full strings $y \in \mathcal{A}$, $z \in \mathcal{B}$ and work our way down to $\mathrm{dom}(y) = \mathrm{dom}(z) = \emptyset$, at which point we shall have constructed all $\mathcal{A} \to \mathcal{B}$ twisted $\mathcal{G}$-equivalences.

1. This is the "data management improvement" over the algorithm in [17] in the case of code equivalence, critical for our analysis.

When y, z are full strings, we have $|\mathcal{A}_y| = 1$, $|\mathcal{B}_z| = 1$, and $I(y, z)$ is trivial to compute.

Now let y, z be proper partial strings of total length ℓ, and assume we have constructed $I(y', z')$ for all y', z' of total length greater than ℓ. To construct $I(y, z)$ we augment the domain of y by one index $r \in A$, and the domain of z by one index, $s \in B$. We fix r, say $r \in A_i$, and make all possible choices of $s \in B_i$. For each $s \in B_i$ and $g \in G_i$, we will separately find the set of all elements of $I(y, z)$ that move s to r, and act on the symbol in that position by g. More formally, for $\gamma \in \Gamma_i$, let $y(r, \gamma)$ be the partial string extending y by γ at position r. Then

$$I(y, z) = \bigcup_{s \in B_i} \bigcup_{g \in G_i} \bigcap_{\gamma \in \Gamma_i : \mathcal{A}_{y(r,\gamma)} \neq \emptyset} I(y(r, \gamma), z(s, \gamma^{g^{-1}})).$$

If $z(s, \gamma^{g^{-1}}) \in \mathcal{B}$, then we can look up the value of $I(y', z')$ in the table. If for some g and γ, $z(s, \gamma^{g^{-1}}) \notin \mathcal{B}$, then $I(y(r, \gamma), z(s, \gamma^{g^{-1}}))$ is empty.

Analysis. Recall that $n = \sum_i |A_i|$. The number of partial strings of $\mathcal{A}$ is $2^n |\mathcal{A}|$, since given a subset of A there are at most $|\mathcal{A}|$ partial strings appearing in $\mathcal{A}$. So the number of pairs of partial strings to store is less than $4^n \cdot |\mathcal{A}|^2$. (We can assume $|\mathcal{A}| = |\mathcal{B}|$, otherwise we reject equivalence.) In fact, we can first fix one subset of size ℓ in A for every $\ell \in [n]$, and then enumerate all subsets of B. In this way we only need to consider $n \cdot |\mathcal{A}|$ partial strings for $\mathcal{A}$, and $2^n |\mathcal{B}|$ for $\mathcal{B}$. With this modification we only need to store $n \cdot 2^n \cdot |\mathcal{A}|^2$ table entries.

The cost of computing each dynamic programming entry when extending at an index from A_i is $n|\Gamma_i||G_i|$ coset intersection operations, and $n|\Gamma_i||G_i|$ times the cost of checking whether $\mathcal{A}_{y'}$ is empty for a partial string y' (and the same for $\mathcal{B}$). By Theorem 2.3.10 the cost of coset intersection is at most $2^{\sqrt{d} \log^2 d} = O(2^d)$ (w.l.o.g. $A = B$, and the cosets we are considering are in $\prod G_i \wr \mathrm{Sym}(A_i)$). The cost of checking if $\mathcal{A}_{y'}$ is empty is $\log |\mathcal{A}|$ if we add a preprocessing step to sort $\mathcal{A}$. So our overall running time is

$$O(n \cdot 2^n \cdot |\mathcal{A}|^2 \cdot (n|\Gamma_{\max}||G_{\max}|) \cdot (2^d + \log|\mathcal{A}|)) = O(n^2 \cdot 2^{n+d} \cdot |\Gamma_{\max}||G_{\max}| \cdot |\mathcal{A}|^2 \log|\mathcal{A}|).$$

$\square$

The following corollary gives the improved result for code equivalence.

Corollary 4.2.2. *Given codes $\mathcal{A} \subseteq \prod_{i=1}^{r} \Gamma_i^{A_i}$ and $\mathcal{B} \subseteq \prod_{i=1}^{r} \Gamma_i^{B_i}$ (as explicit lists of strings), $\mathrm{EQ}(\mathcal{A}, \mathcal{B})$ can be found in time $c^n \operatorname{poly}(|\mathcal{A}|, |\Gamma_{\max}|)$ where $n = \sum_{i=1}^{r} |A_i|$ is the total length of the codes.*

4.3 Related Problems

In our applications to Permutational Isomorphism and Semisimple Group Isomorphism, our codes have the additional property that the alphabet Γ is a group, and the codes are subgroups $\mathcal{A} \leq \Gamma^n$. (For simplicity we state these problems for a single alphabet.) We call such codes group codes. More precisely

Definition 4.3.1. A **group code** of length n over a group Γ is a subgroup $\mathcal{A} \leq \Gamma^n$.

We now ask the question of determining equivalence (Definition 4.1.2) of group codes.

Problem 4.3.2. Group Code Equivalence
INPUT: *A group Γ; and two group codes $\mathcal{A}, \mathcal{B} \leq \Gamma^n$.*
OUTPUT: *Are $\mathcal{A}$ and $\mathcal{B}$ equivalent?*

For general codes, we cannot hope to improve the bound in Corollary 4.2.2 below $|\mathcal{A}|$, since that is the size of the input. However, for group codes, the input could be succinctly represented by generators, and we can ask for a simply-exponential algorithm.

Open Problem 4.3.3. Solve Group Code Equivalence in time $c^n \operatorname{poly}\log(|\Gamma|)$.

When $\Gamma = \mathbb{Z}_p$, then Γ is a field, and the code is a subspace. In this special case the code is called a **linear code**. The open problem above has been solved for linear codes ([10], cf. [17]).

Theorem 4.3.4 (Babai [10]). *Equivalence of linear codes of dimension d and length n can be reduced to $\binom{n}{d}$ instances of* Graph Isomorphism *on n vertices, at a cost of* $\operatorname{poly}(n)$ *field operations per instance.*

Combined with the moderately exponential Graph Isomorphism test, this yields the desired bound.

Theorem 4.3.5 (Babai [10]). *Equivalence of linear codes of length n over a field Γ can be decided in time $(2.01)^n \log^2 |\Gamma|$.*

We note that while the problem has been solved for $\Gamma = \mathbb{Z}_p$, it remains open even for elementary abelian groups ($\mathbb{Z}_p \times \cdots \times \mathbb{Z}_p$) and cyclic groups ($\mathbb{Z}_m$).

CHAPTER 5

PERMUTATIONAL ISOMORPHISM

5.1 Introduction

We prove the following theorem for permutational isomorphism of permutation groups.

Theorem 5.1.1. *Given two permutation groups G and H of degree d, we can find* $\mathrm{PISO}(G, H)$ *in time* $\mathrm{poly}(|G|) \cdot c^d$*, for an absolute constant c.*

To prove this result, we first prove that for *transitive* permutation groups we can in fact *list* all isomorphisms in the desired time. This is not true in the general case (there can be too many isomorphisms). We shall reduce the general case to twisted code equivalence.

5.1.1 Giant primitive groups

Let $G \leq S_n$. We call G a *giant*[1] if $n \geq 7$ and $G = S_n$ or A_n. These two groups are far larger than any other primitive group, as shown in Theorem 1.2.4 (cf. Cameron [33]). We can now re-state Lemma 1.2.6 in terms of giants, and relax the bound to be simply-exponential.

Lemma 5.1.2. *There exists a constant c_2 such that if $G \leq S_n$ is a non-giant primitive group, then: (a) $|\mathrm{PAut}(G)| \leq c_2^n$; and (b) for every $H \leq S_n$, we can list* $\mathrm{PISO}(G, H)$ *in time c_2^n.*

5.2 Permutational Isomorphism of Transitive Permutation Groups

Recall our result for transitive permutation groups.

1. We remark that S_n and A_n are primitive for $n \geq 3$. We look at $n \geq 5$ and $n \neq 6$ since A_n is simple when $n \geq 5$; and $\mathrm{Aut}(A_n) \cong S_n$ when $n \geq 4$, $n \neq 6$.

Theorem 5.2.1. *There exists an absolute constant c such that (a) if G is a transitive permutation group of degree n, then $|\mathrm{PAut}(G)| \leq |G| \cdot c^n$; (b) given two transitive permutation groups G and H of degree n, all their permutational isomorphisms can be listed in time $O(|G| \cdot c^n)$.*

The bound in Theorem 5.2.1(a) is nearly tight: Kovács and Newman [52] (cf. [65]) showed examples where the number of permutational automorphisms is $|G|2^{k/\sqrt{\log k}}$. In an unpublished work, Guralnick and Pyber proved a matching upper bound for the number of permutational automorphisms of transitive groups [41]. While our bound is slightly worse, it comes with a simple algorithm to list all permutational isomorphisms, which is necessary for our applications.

The general strategy for our algorithm is the following. We fix a structure tree of G and work by induction on its depth. (See Section 1.2.6 for the basics on structure trees.) The base case is when G is primitive; this case is easily settled based on Lemma 5.1.2. (In fact, permutational isomorphism of primitive groups can be tested –but not listed– in time, quasipolynomial in the degree.)

We call a structure tree T of G and a structure tree U of H **compatible** if their depth is the same and for every ℓ, the primitive groups arising on level ℓ in G and H (as actions of the stabilizers of a node on level ℓ on the children of that node) are permutationally isomorphic.

At each step in the induction we will have to deal with primitive groups. For non-giant primitive groups, we will use Lemma 5.1.2 to bound the number of permutational isomorphisms and list them. For giant primitive groups, we have to delve into the structure in greater depth.

The following situation, corresponding to one layer of the structure tree, will be the central inductive step in our algorithm. Let $G \leq \mathrm{Sym}(\Omega)$, $H \leq \mathrm{Sym}(\Delta)$ with $|\Omega| = |\Delta| = n$. Let $\{\Omega_1, \ldots, \Omega_m\}$ be a system of imprimitivity for G. Let K be the kernel of the action of G on the set of blocks (the stabilizer of the blocks setwise), and let G^* be the action on the set of blocks. Similarly, let $\{\Delta_1, \ldots, \Delta_m\}$ be a system of imprimitivity for H, let L be the

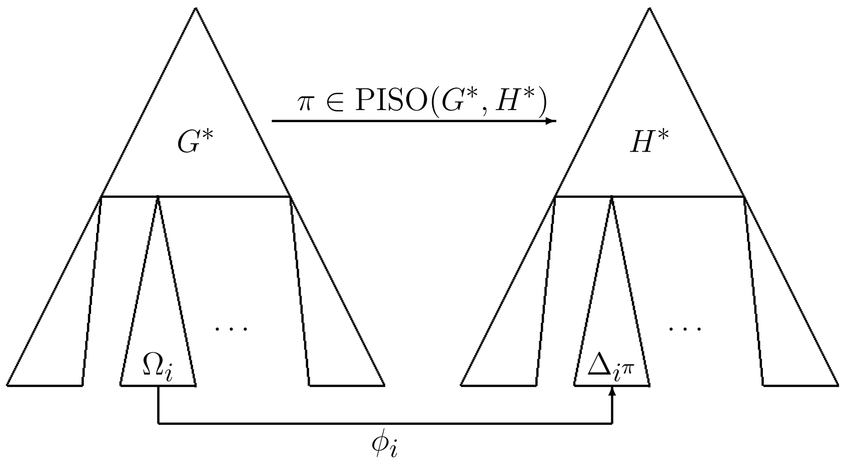

Figure 5.1: A permutational isomorphism ϕ that extends $\pi \in \mathrm{PISO}(G^*, H^*)$ is uniquely defined by π and the projections $\phi_i = \phi|_{\Omega_i}$.

kernel of the H-action on the set of blocks, and let H^* be the H-action on the set of blocks. Note that K is the kernel of the restriction homomorphism $G \to G^*$.

We say that $\phi \in \mathrm{PISO}(G, H)$ **extends** $\pi \in \mathrm{PISO}(G^*, H^*)$ if ϕ acts as π on the set of blocks, in other words, if $(\forall i)(\phi(\Omega_i) = \Delta_{i^\pi})$. Let $\mathrm{PISO}(G, H; \pi) = \{\phi \in \mathrm{PISO}(G, H) \mid \phi \text{ extends } \pi\}$. Let $G(i) = G_{\{\Omega_i\}}^{\Omega_i}$ and $K(i) = K^{\Omega_i}$ be the restrictions of the groups G and K respectively to the blocks Ω_i. Similarly define $H(i)$ and $L(i)$ (see figure 5.2).

When $K(i)$ is a giant, we will use the following lemma to bound the number of extensions.

Lemma 5.2.2. *For $i = 1, \dots, m$, let Ω_i and Δ_i be sets of size k, for $k \geq 5, k \neq 6$. Let $K \leq \mathrm{Alt}(\Omega_1) \times \cdots \times \mathrm{Alt}(\Omega_m)$, $L \leq \mathrm{Alt}(\Delta_1) \times \cdots \times \mathrm{Alt}(\Delta_m)$ be subdirect products. Given $\pi \in S_m$, (a) there are at most $2^m|K|$ permutational isomorphisms $\phi \in \mathrm{PISO}(K, L; \pi)$; moreover, (b) we can list all such permutational isomorphisms in time $O(2^m|K|km)$.*

Proof. For $\phi \in \mathrm{PISO}(K, L)$, let $\phi_i : \Omega_i \to \Delta_{i^\pi}$ be the restriction of ϕ to the ith part, $\phi_i := \phi|_{\Omega_i}$. By Fact 1.2.15, K and L are the direct product of permutational diagonals. Let $\sim_K$ be the equivalence relation on $[m]$ corresponding to the partition into diagonal sections of K, and $\sim_L$ the corresponding relation under L. If π does not transform $\sim_K$ into $\sim_L$ then no extension of π exists. Suppose now it does. For $i \sim_K j$, let $f_{ij} : \Omega_i \to \Omega_j$ denote the bijection defining the permutational diagonal; and for $i \sim_L j$, let $g_{ij} : \Delta_i \to \Delta_j$ be defined analogously.

Assume $\phi \in \mathrm{PISO}(K, L; \pi)$. Then for any $i, j \in [m]$, we must have $\phi_j = g_{i^\pi, j^\pi}\, \phi_i\, f_{i,j}^{-1}$. Let $R \subseteq [m]$ be a set of representatives of the diagonal sections. Any list of bijections $(\phi_i : \Omega_i \to \Delta_{i^\pi})_{i \in R}$ uniquely determines the remaining ϕ_j, and therefore all of ϕ. The number of such lists is $(k!)^{|R|} \leq 2^m |K|$, since $|K| = (k!/2)^{|R|}$. This proves (a).

To list all the ϕ that extend π, we consider all possible lists $(\phi_i)_{i \in R}$, and for each construct the unique bijection $\phi : \Omega \to \Delta$ that the list induces. We can find the diagonal sections of K and L, pick the set R, and compute the bijections induced by the diagonal action in time $O(|K|)$. Finally, given the $(\phi_i)_{i \in R}$, we can construct ϕ in time $O(mk)$ by using the formula $\phi_j = g_{i^\pi, j^\pi}\, \phi_i\, f_{i,j}^{-1}$. Therefore the total running time is bounded by $O(|K| 2^m km)$ □

We use the following results when $G(i)$ is a giant, but $K(i)$ is trivial.

Observation 5.2.3. *If G acts transitively on the blocks and $(\exists i)(K(i) = \mathrm{id})$, then $K = \mathrm{id}$.*

For completeness, we include the easy proof.

Proof. Recall that $K(i) = K^{\Omega_i} = K/K_{\Omega_i}$, where K_{Ω_i} is the pointwise stabilizer of Ω_i in K. Since G acts transitively on the blocks,

$$(\forall i)(\exists \psi_i \in G)(G_{\Omega_i}^{\psi_i} = G_{\Omega_1}).$$

Note that $K_{\Omega_i} = K \cap G_{\Omega_i}$, and $(\forall \psi \in G)(K^\psi = K)$ (since K is normal), so we have:

$$K_{\Omega_i}^{\psi_i} = K^{\psi_i} \cap G_{\Omega_i}^{\psi_i} = K \cap G_{\Omega_1} = K_{\Omega_1}.$$

Therefore $(\forall i)(K(i) \cong K(1))$, and so if $(\exists i)(K(i) = \mathrm{id})$, then $(\forall i)(K(i) = \mathrm{id}) \Rightarrow K = \mathrm{id}$.

□

Lemma 5.2.4. *If the system of imprimitivity* $\{\Omega_i\}_{i\in[m]}$*is maximal and two distinct points* $x \neq y$ *belong to the same block* Ω_i*, and* $G(i) \neq \mathrm{id}$*, then* $G_x \neq G_y$*.*

Proof. The equivalence relation $x \sim y \Leftrightarrow G_x = G_y$ forms an invariant partition:

$$(\forall \pi \in G)(x \sim y \iff G_x = G_y \iff G_x^\pi = G_y^\pi \iff G_{x^\pi} = G_{y^\pi} \iff x^\pi \sim y^\pi).$$

Since Ω_i was assumed to be a minimal block, the partition can only be the trivial partition into one set.

□

Lemma 5.2.5. *If* $(\exists i)(G(i) \neq \mathrm{id},\ \text{and}\ K(i) = \mathrm{id})$ *then there exists at most one extension for a given* $\pi \in \mathrm{PISO}(G^*, H^*)$*, and if such an extension exits, we can find it in time, polynomial in* n*.*

Proof. By Observation 5.2.3, $K = \mathrm{id}$, hence $G \cong G^*$. If $L \neq \mathrm{id}$, no extension of π exists, so w.l.o.g. $H \cong H^*$. Then

$$(\forall x \in \Omega_i)(\forall \phi \in \mathrm{PISO}(G, H; \pi))(\exists y \in \Delta_{i^\pi})((G_x)^\phi = H_y).$$

In fact by Lemma 5.2.4, there is a unique such y. Therefore ϕ is also unique given π. To find ϕ, let $\sigma : G \to H$ be the isomorphism induced by π, check if $\exists y \in \Delta_{i^\pi}$ such that $G_x^\sigma = H_y$; if so let $x^\phi = y$, otherwise there is no extension of π. □

The inductive step in the proof of Theorem 5.2.1 is based on the following main technical lemma.

Lemma 5.2.6. *Let* $G \leq \mathrm{Sym}(\Omega)$*, and* $H \leq \mathrm{Sym}(\Delta)$ *be transitive permutation groups of degree* n*, and let* $\Omega_1, \ldots, \Omega_m$ *and* $\Delta_1, \ldots, \Delta_m$ *be maximal systems of imprimitivity for* G *and* H*, respectively. Let* G^**,* H^* *be the actions of* G *and* H*, respectively, on the blocks.*

Given any $\pi \in \mathrm{PISO}(G^*, H^*)$, (a) $|\mathrm{PISO}(G, H; \pi)| \leq |K|\, c_2^n$; moreover (b) we can list $\mathrm{PISO}(G, H; \pi)$ in time $|K|\, c_2^n \operatorname{poly}(n)$.

Proof. W.l.o.g. $(\forall i, j)$ $G(i)$ and $H(j)$ are permutationally isomorphic to some primitive $T \leq S_k$ (otherwise reject isomorphism extending π). We consider the following two cases: (1) if T is not a giant, (2) if T is a giant.

(1) If T is not a giant, we try extending by all possible lists of permutational isomorphisms $(\sigma_i \in \mathrm{PISO}(G(i), H(i^\pi)))_{i=1}^m$. Given such a list, and π, there is a unique bijection $\phi : \Omega \to \Delta$, given by $\phi(\Omega_i) = \Delta_{i^\pi}$, $\phi|_{\Omega_i} = \sigma_i$. We can check whether $\phi \in \mathrm{PISO}(G, H)$ in polynomial time in n (by Proposition 1.2.2). By part (a) of Lemma 1.2.6, $|\mathrm{PISO}(G(i), H(j))| \leq c_2^k$, hence the number of possible extensions is bounded by $(c_2^k)^m = c_2^n$. Moreover, by part (b) of Lemma 1.2.6, we can list all of them in time $O(\operatorname{poly}(n) c_2^n)$.

(2) If T is a giant, we need to look at the structure of the group in more detail. Since $K \lhd G$, it is also the case that $K(i) := K^{\Omega_i} \lhd G^{\Omega_i}_{\{\Omega_i\}} = G(i)$ (we make analogous observations for H). But since $G(i)$ is permutationally isomorphic to A_k or S_k, for $k \geq 5$, $K(i)$ is either S_k, or A_k, or id. If $(\exists i)(K(i) = \mathrm{id})$, then by Lemma 5.2.5, there is at most one extension, and we can find it in time polynomial in n. Otherwise, note that if $\phi \in \mathrm{PISO}(G, H; \pi)$, then $\phi \in \mathrm{PISO}(K', L'; \pi)$, and K' and L' are subdirect products of A_k^m. (Recall that $R' = [R, R]$ denotes the commutator subgroup.) Now for $k \geq 5$, $k \neq 6$, by part (a) of Lemma 5.2.2, there are at most $2^n |K'| \leq 2^n |K|$ extensions. Moreover, by part (b) of Lemma 5.2.2, we can list all $\phi \in \mathrm{PISO}(K', L'; \pi)$ in time $O(2^n |K'|) \leq O(2^n |K|)$. And by Proposition 1.2.2 for each $\phi \in \mathrm{PISO}(K', L'; \pi)$ we can check whether $\phi \in \mathrm{PISO}(G, H; \pi)$ in time $\operatorname{poly}(n)$. $\square$

For the purposes of induction, we need a slightly stronger version of Theorem 5.2.1.

Theorem 5.2.7. *Let G, H be transitive permutation groups of degree n, and fix structure trees for G and H. Then (a) the number of permutational isomorphisms respecting the structure trees is at most $|G| \cdot c_2^{2n}$, where c_2 is as in Lemma 5.1.2. (b) All the permutational isomorphisms of G and H that respect the structure trees can be listed in time $O(|G| \cdot c_0^n)$,* *for* $c_0 = c_2^2 + \epsilon$, *for any* $\epsilon > 0$.

Theorem 5.2.1 follows by fixing a structure tree for G and trying all possible structure trees for H. By Lemma 1.2.11, there are at most $n^{2\log n}$ such choices. So setting $c = c_0 + \epsilon$, for any $\epsilon > 0$, we obtain the desired result.

Proof. We proceed by induction on the depth of the structure trees. Let $c_0 = c_2^2 + \epsilon$, for any $\epsilon > 0$. We will prove that then number of permutational isomorphisms that respect this choice of structure trees is at most $c_2^{2n}|G|$, and we can list all such permutational isomrophisms in time $c_0^n|G|$. If the depth is 1, the the groups are primitive. If they are both non-giant, then the theorem follows by Lemma 5.1.2. If both groups are A_n or S_n, then $\text{PISO}(G, H) = S_n$; and $n! \leq 2|G|$ so we can list all $n!$ permutations in the desired time bound. Otherwise $\text{PISO}(G, H) = \emptyset$.

Now consider structure trees of depth d, and assume by inductive hypothesis that the theorem holds for depth $d-1$. The last layer of the structure trees is a maximal system of imprimitivity. Using the notation from Lemma 5.2.6, by inductive hypothesis the number of permutational isomorphisms $\pi \in \text{PISO}(G^*, H^*)$ is at most $|G^*|c_2^{2m}$ and we can list them in time $O(|G^*|c_0^m)$. By Lemma 5.2.6, for each such π, the number of extensions is at most $c_2^n|K|$, and we can find all of them in time $O(c_2^n|K|\text{poly}(n))$. Since $m \leq n/2$, and $|G| = |G^*||K|$, the total number of permutational isomorphisms for this structure tree is at most $c_2^{2m}\,|G^*| \cdot c_2^n\,|K| = c_2^{2n}\,|G|$. And so we can list all those $\text{PISO}(G, H)$ respecting this structure tree in time $c_2^{2m}\,|G^*| \cdot c_2^n\,|K| \cdot \text{poly}(n) = c_0^n|G|$. $\square$

5.3 Intransitive Permutation Groups – Reduction to Twisted Code Equivalence

We reduce permutational isomorphism of intransitive groups to twisted code equivalence.

Let $G \leq \text{Sym}(\Omega)$ and $H \leq \text{Sym}(\Delta)$ be (possibly intransitive) permutation groups. Let G have orbits of r isomorphism types, and k_i orbits of the ith isomorphism type. For

$i = 1, \ldots, r$, and $j = 1, \ldots, k_i$, let Ω^i_j be the jth orbit of isomorphism type i. Similarly define Δ^i_j for H. For every $i = 1, \ldots, r$, $j = 1, \ldots, k_i$, let $G^i_j = G^{\Omega^i_j}$ be the restriction of G to the orbit Ω^i_j. Similarly, let $H^i_j = H^{\Delta^i_j}$. We use the algorithm for transitive permutation group isomorphism (Theorem 5.2.1) to find such a labeling of the orbits. Note that $(\forall i \leq r)(\forall j \leq k_i)(G^i_j$ and H^i_j are permutationally isomorphic to $G^i_1)$.

Now $\forall i \in [r], j \in [k_i]$, let us fix arbitrary permutational isomorphisms $\phi^i_j \in \mathrm{PISO}(G^i_j, G^i_1)$, and $\psi^i_j \in \mathrm{PISO}(H^i_j, G^i_1)$. Let $\Phi : \Omega \to \bigcup_{i=1}^r (k_i \cdot \Omega^i_1)$ and $\Psi : \Delta \to \bigcup_{i=1}^r (k_i \cdot \Omega^i_1)$ be the compositions of the ϕ^i_j and ψ^i_j respectively (where $k \cdot S$ means the disjoint union of k copies of S). These define embeddings $\widehat{\Phi} : G \hookrightarrow \prod_{i=1}^r (G^i_1)^{k_i}$, and $\widehat{\Psi} : H \hookrightarrow \prod_{i=1}^r (G^i_1)^{k_i}$. The following clearly follows from the definitions:

$$\mathrm{PISO}(G, H) = \Phi \mathrm{PISO}(\widehat{\Phi}(G), \widehat{\Psi}(H)) \Psi^{-1}. \tag{5.1}$$

Therefore it is enough to compute $\mathrm{PISO}(\widehat{\Phi}(G), \widehat{\Psi}(H))$.

Lemma 5.3.1. *Let $\Gamma_i = G^i_1$, $W_i = \widehat{\mathrm{PAut}}(G^i_1)$, and $\mathcal{W} = (W_i)_{i=1}^r$. Then*

$$\mathrm{PISO}(\widehat{\Phi}(G), \widehat{\Psi}(H)) = \mathrm{EQ}_{\mathcal{W}}(\widehat{\Phi}(G), \widehat{\Psi}(H))$$

Proof. Follows from the definitions. A permutational isomorphism of $\widehat{\Phi}(G)$ and $\widehat{\Psi}(H)$ can be decomposed into its action on the orbits, and a set of permutational automorphisms within the orbits. The action on the orbits is the permutation of the coordinates of the code, and the permutational automorphisms induce the automorphisms applied by the "twisting" groups.

□

The following concludes the proof of Theorem 5.1.1.

Corollary 5.3.2. *We can find the coset of permutational isomorphisms of two permutation groups G and H of degree d in time $c^d \operatorname{poly}(|G|)$, for an absolute constant c.*

Proof. By equation 5.1 and Lemma 5.3.1, $\mathrm{PISO}(G, H) = \Phi\, \mathrm{EQ}_{\mathcal{W}}(\widehat{\Phi}(G), \widehat{\Psi}(H))\Psi^{-1}$, where the alphabets are $\Gamma_i = G_1^i$, and the groups acting on the alphabets are $W_i = \mathrm{PAut}(G_1^i)$. The total length of the strings is $n = \sum_{i=1}^{r} k_i \leq d$. By part (a) of Theorem 5.2.1,

$$|W_i| = |\mathrm{PAut}(G_1^i)| \leq |G_1^i|\, c^{|\Omega_1^i|} \leq |G|\, c^d.$$

The groups W_i have permutation representations of degree $|\Omega_1^i|$, and $|A_i| = |B_i| = k_i$, so the degree of the permutation representation of $\prod W_i \wr S_{k_i}$ is at most $\sum_{i=1}^{r} |A_i||\Omega_1^i| = d$. By Theorem 4.2.1, we can compute $\mathrm{EQ}_{\mathcal{W}}(\widehat{\Phi}(G), \widehat{\Psi}(H))$ in time

$$O(n^2\, 2^{n+d} |W_{\max}||\Gamma_{\max}||G|^2 \log |G|) = O(c^d \cdot |G|^{4.01}).$$

To this we have to add the cost of computing the W_i, which is $O(|G|\, c^d)$ (by part (b) of Theorem 5.2.1), and so the asymptotic running time is the same as the running time of the twisted code equivalence routine.

□

CHAPTER 6
SEMISIMPLE GROUP ISOMORPHISM

6.1 Introduction

6.1.1 Main result

Definition 6.1.1. A group is *semisimple* if it has no abelian normal subgroups.

Theorem 6.1.2 (Main result)**.** *Isomorphism of semisimple groups given by their Cayley tables can be decided in polynomial time.*

We prove this result in Sections 6.4 and 6.5. The key ingredients are: (a) a study of the group theoretic structure of semisimple groups, introduced in [17]; (b) the algorithm to decide permutational isomorphism of permutation groups of degree k and order n in time linear in n and simply exponential in k from Chapter 5; (c) the algorithm to test twisted code equivalence from Chapter 4.

6.1.2 The structure of semisimple groups – link to permutation groups

Our approach is motivated by the BB-filtration of groups [12] (cf. Section 6.2.2). Specifically, the **socle** of a group is the product of its minimal normal subgroups. The socle of a semisimple group G is the direct product of nonabelian simple groups, and G acts on the set of simple factors of the socle by conjugation, producing a permutation group of degree at most $\log_{60} |G|$. It follows, as pointed out in [17], that any polynomial-time isomorphism test for semisimple groups yields an algorithm for permutational isomorphism of permutation groups in time polynomial in the order of the group and simply exponential in the degree. Conversely, we demonstrate in this chapter that a solution to the transitive permutation group isomorphism problem within such time bounds can be used to solve the isomorphism problem for semisimple groups in polynomial time, thus we use Theorem 1.1.7 to prove our main result.

6.1.3 Strategy for the main result

First we observe that isomorphism of groups that are direct products of simple groups can be tested in polynomial time (Proposition 6.2.6). So we can assume that our semisimple groups G and H have isomorphic socles. In fact we can assume further that the socles decompose "isomorphically" into the direct product of their minimal normal subgroups. We look at the conjugation action of the groups G and H on the simple factors of their socles. The orbits of this action are the simple factors of each minimal normal subgroup. Using the algorithm for transitive permutational isomorphism, we compute the isomorphism types of these actions. We then reduce to the problem to twisted code equivalence where the alphabets are the automorphism groups of the minimal normal subgroups; the twisting groups consist of certain automorphisms of these automorphism groups.

6.2 Group Theoretic Preliminaries

6.2.1 Characteristically simple groups; the socle

Definition 6.2.1. A subgroup $H \leq G$ is a **characteristic subgroup** if H is invariant under all automorphisms of G. A group is **characteristically simple** if it has no nontrivial characteristic subgroups.

Fact 6.2.2. Every characteristically simple group is the direct product of isomorphic (abelian or non-abelian) simple groups.

Fact 6.2.3. If A is a characteristic subgroup of B, and $B \triangleleft C$, then $A \triangleleft C$.

Definition 6.2.4. $N \triangleleft G$ is a *minimal normal subgroup* if $|N| > 1$ and N does not contain any nonidentity normal subgroup of G other than itself.

Fact 6.2.5. Every minimal normal subgroup is characteristically simple.

If $G = \prod T_i$, where the T_i are non-abelian simple, then this product decomposition is unique (not just up to isomorphism). Note how this is not true for $\mathbb{Z}_p \times \cdots \times \mathbb{Z}_p$.

Recall that the socle of a group G, denoted by $\operatorname{Soc}(G)$, is the product of the minimal normal subgroups of G. If G is semisimple, $\operatorname{Soc}(G)$ is the direct product of all minimal normal subgroups. We group this direct product by isomorphism types of the minimal normal subgroups as

$$\operatorname{Soc}(G) = \prod_{i=1}^{d} \prod_{j=1}^{z_i} N_{i,j} \cong \prod_{i=1}^{d} K_i^{z_i}, \tag{6.1}$$

where the $N_{i,j}$ are the minimal normal subgroups and $(\forall i,j)(N_{i,j} \cong K_i)$. The K_i are pairwise non-isomorphic characteristically simple groups.

We refine decomposition (6.1) to simple factors, and lump the isomorphic simple factors together to obtain the following decomposition:

$$\operatorname{Soc}(G) = \prod_{i=1}^{r} \prod_{j=1}^{k_i} V_{i,j} \cong \prod_{i=1}^{r} T_i^{k_i}, \tag{6.2}$$

where $(\forall i,j)(V_{i,j} \cong T_i)$, and the T_i are pairwise non-isomorphic non-abelian simple groups.

The following is an easy consequence of the fact that (non-abelian) simple groups have 2 generators (cf. [17, Proposition 2.1]). We include a proof for completeness.

Proposition 6.2.6. *Let G be a direct product of non-abelian simple groups given by its Cayley table. Then: (a) the (unique) direct product decomposition of G into its simple factors can be found in polynomial time; (b) isomorphism of G and any other group H can be decided, and the set of isomorphisms found, in polynomial time.*

Proof. (a) Take the normal closure of each element; take the minimal ones among the subgroups obtained. These are the direct factors. (b) Do the same to H and verify that a direct decomposition was found; if not, reject isomorphism. Otherwise decide which pairs among the simple factors of G and H are isomorphic and find all their isomorphisms. If the multiplicities of isomorphism types don't match, reject isomorphism. Otherwise, find an isomorphism along matched factors of G and H; combine this with the automorphism group of G. The automorphism group of $G = \prod T_i^{k_i}$ is $\prod \operatorname{Aut}(T_i) \wr S_{k_i}$.

We used the fact that all isomorphisms of two simple groups (and therefore all automorphisms of a simple group) can be listed in polynomial time. The reason is that simple groups can be generated by 2 elements.

□

While the proof above is straightforward, we mention that the same result holds, non-trivially, in the context of permutation groups given by generators; in fact, if a permutation group is a product of simple groups, it can be split into its simple factors in NC [24]. We mention that a direct decomposition of a permutation group is also known to be computable in polynomial time even if the direct factors are not simple; this was done in [50] for groups given by Cayley tables and by Wilson [83] for permutation groups.

By Proposition 6.2.6, we can decide isomorphism of $\mathrm{Soc}(G)$ and $\mathrm{Soc}(H)$ in polynomial time. Indeed, we can find the two product decompositions of the socles described above in polynomial time, and decide isomorphism of the factors.

6.2.2 Relationship to the Babai-Beals filtration

Our approach is motivated by the following chain of characteristic subgroups, introduced by Babai and Beals [12] and since used extensively in the algorithmic theory of matrix groups and black-box groups (see [13]):

$$1 \leq \mathrm{Rad}(G) \leq \mathrm{Soc}^*(G) \leq \mathrm{Pker}(G) \leq G.$$

We now explain the terms of this chain. $\mathrm{Rad}(G)$, the **solvable radical,**, is the unique maximal solvable normal subgroup of G. $\mathrm{Soc}^*(G)$ is the preimage of the socle $\mathrm{Soc}(G/\mathrm{Rad}(G))$ under the natural projection $G \to G/\mathrm{Rad}(G)$. (Recall that the socle is the product of the minimal normal subgroups.)

Note that the group $\mathrm{Soc}^*(G)/\mathrm{Rad}(G) = \mathrm{Soc}(G/\mathrm{Rad}(G))$ is the direct product of non-abelian simple groups $T_1, \ldots T_k$. The group G acts by conjugation on $\mathrm{Soc}(G/\mathrm{Rad}(G))$; this action permutes the k simple groups involved, so we obtain a homomorphism $G \to S_k$. We denote by $\mathrm{Pker}(G)$ the kernel of this homomorphism (permutation representation).

In a sense, this normal structure provides a layering to the group isomorphism problem; the layers are

1st layer $G/\mathrm{Pker}(G)$: a permutation group of logarithmic degree;

2nd layer $\mathrm{Pker}(G)/\mathrm{Soc}^*(G)$, a solvable group satisfying strong structural constraints;

3rd layer $\mathrm{Soc}^*(G)/\mathrm{Rad}(G) = \mathrm{Soc}(G/\mathrm{Rad}(G))$, a direct product of non-abelian simple groups;

4th layer $\mathrm{Rad}(G)$, a solvable group,

While it is by no means the case that solving the isomorphism problem for the layers would automatically solve it for the entire group, solving it for the layers is definitely a prerequisite. (This statement was formalized for the top layer in Proposition 7.1 in [17].) Then the task remains to control the "glue" that holds these layers together.

The bottom layer is a solvable group and testing isomorphism in polynomial time for solvable groups remains elusive (they include the notorious class-2 nilpotent groups). We consider semisimple groups, i. e., we assume $\mathrm{Rad}(G)$ is trivial and therefore $\mathrm{Soc}^*(G) = \mathrm{Soc}(G)$ is a direct product of non-abelian simple groups.

Recall that the isomorphism problem for direct products of non-abelian simple groups (third layer) is easily solved in polynomial time (Proposition 6.2.6).

The second layer ("outer automorphism layer") is solvable but is no cause for panic; we glue it right to the second layer (Corollary 6.3.3 gives a key link). The top layer is a permutation group of logarithmic degree. This is where the permutational isomorphism of permutation groups problem comes into the picture. The orbits of the top layer correspond to the simple factors of the minimal normal subgroups. Therefore, analogously to what we did for permutation groups, we first examine the "transitive" case, that is, groups with a unique minimal normal subgroup. In this case we will be able to *list* all isomorphisms in time polynomial in the order of the group. We will then use the algorithm for the unique minimal normal subgroup case to reduce the general case to twisted code equivalence.

6.3 The Framework

In this section we give some group theoretic reductions that we introduced in [17]. In the next section we use this framework and the permutational isomorphism test for transitive permutation groups (Theorem 5.2.1) to obtain an isomorphism test of semisimple groups with a unique minimal normal subgroup [18]. This result is an ingredient in our isomorphism test for all semisimple groups, analogously to the way in which the transitive case was used for the intransitive case of permutational isomorphism.

6.3.1 Restriction of isomorphisms to the socle

If $N \trianglelefteq G$ is a normal subgroup of G, then G acts on N by conjugation. This action defines a homomorphism $\gamma = \gamma_{G,N} \colon G \to \mathrm{Aut}(N)$. For $g \in G$ and $n \in N$, we write $n^g = n^{\gamma(g)} = g^{-1}ng$. G is said to **act faithfully** on N if γ is injective. Note that $\ker(\gamma) = C_G(N)$, the **centralizer** of N in G. If $C_G(N) = 1$, then γ is an embedding. If we take $N = G$, the automorphism of the form $\gamma(g)$ are the **inner automorphisms**. Recall from Section 1.2 that the group of inner automorphisms is denoted $\mathrm{Inn}(G)$ ($\mathrm{Inn}(G) := G^\gamma$); it is a normal subgroup of $\mathrm{Aut}(G)$, and note that $\mathrm{Inn}(G) \cong G/Z(G)$ where $Z(G)$ denotes the center of G. If $Z(G) = 1$ then $\gamma_G = \gamma_{G,G}$ is a canonical isomorphism $G \cong \mathrm{Inn}(G)$.

Fact 6.3.1. Let G be a group, and $N \trianglelefteq G$ a normal subgroup with trivial centralizer. Then every isomorphism $\varphi \colon N \to M$ extends uniquely to an embedding $\Phi \colon G \hookrightarrow \mathrm{Aut}(M)$ with $\Phi|_N = \varphi\gamma_M$. In particular, there is a bijection between $\mathrm{ISO}(N, M)$ and the set of embeddings $\Phi \colon G \hookrightarrow \mathrm{Aut}(M)$ such that $\Phi(N) = \mathrm{Inn}(M)$.

$$\begin{tikzcd}
G^* \arrow[r, "f^*"] & H^* \\
G \arrow[u, "\alpha"] \arrow[r, "\overline{f}"] & H \arrow[u, "\beta"'] \\
R \arrow[u, "\trianglelefteq"] \arrow[r, "f"] & S \arrow[u, "\trianglelefteq"']
\end{tikzcd}$$

Figure 6.1: The commutative diagram for the groups in Observation 6.3.2.

Observation 6.3.2 (cf. [69, Claim 3.3.19, page 90], and [17, Lemma 3.1])**.** *Let G and H be groups and $R \lhd G$ and $S \lhd H$ normal subgroups with trivial centralizers. Let $\alpha : G \to \mathrm{Aut}(R)$ and $\beta : H \to \mathrm{Aut}(S)$ be the permutation representations of G and H via conjugation action on R and S, respectively. Let $G^* = \mathrm{Im}(\alpha)$, and $H^* = \mathrm{Im}(\beta)$. Let $f : R \to S$ be an isomorphism. Then f extends to an isomorphism $\overline{f} : G \to H$ if and only if f is a permutational isomorphism between G^* and H^*; and if so, $\overline{f} = \alpha\widehat{f}\beta^{-1}$ where $\widehat{f}$ is the isomorphism $G^* \to H^*$ induced by f (see Figure 6.1).*

For completeness, we include a proof.

Proof. By applying the inverse of f, we may assume $R = S$, and f is the identity. We claim that if $\widehat{f}$ exists, it must be the identity. Suppose $\widehat{f}$ exists. Let $\overline{f}$ denote the corresponding $G \to H$ isomorphism. So $\overline{f}|_R = \mathrm{id}$. Let $g \in G$ and let $g^* = G^*$ be the corresponding automorphism of R. We need to show that $\widehat{f}(g^*) = g^*$, that is, for all $r \in R$, $r^g = r^{\overline{f}(g)}$, i. e.,

$$g^{-1}rg = \overline{f}(g)^{-1}r\overline{f}(g). \tag{6.3}$$

But $\overline{f}(g)^{-1}r\overline{f}(g) = \overline{f}(g)^{-1}\overline{f}(r)\overline{f}(g) = \overline{f}(g^{-1}rg) = g^{-1}rg$ because $g^{-1}rg \in R$, proving (6.3). □

Corollary 6.3.3. *Let G and H be two groups given by Cayley tables. Let $R \lhd G$ and $S \lhd H$ be normal subgroups with trivial centralizers. Assume $f : R \to S$ is an isomorphism. Then* *(a) f extends in at most one way to an isomorphism $\overline{f} : G \to H$; and (b) given f we can decide if $\overline{f}$ exists, and find it if it does, in polynomial time.*

Proof. Part(a) follows from Observation 6.3.2. Part (b) follows from (a) and Proposition 1.2.2.

□

6.3.2 Diagonal-respecting isomorphisms

We will be interested in isomorphisms that respect diagonals (cf. Section 1.2.7 for the definition of diagonal). In order to define this concept, we need to talk about isomorphisms that respect the decomposition of the groups into direct products.

Definition 6.3.4 (ISOp)**.** Given two groups X, Y, along with product decompositions $X = \prod_{i=1}^{r}\prod_{j=1}^{k_i} V_{i,j}$, $Y = \prod_{i=1}^{r}\prod_{j=1}^{k_i} U_{i,j}$, where $(\forall i,j)(U_{i,j} \cong V_{i,j} \cong T_i)$, we say that an isomorphism $\chi : X \to Y$ *respects* the decompositions $\mathcal{V} = (V_{i,j})$ and $\mathcal{U} = (U_{i,j})$ if $(\forall i,j)(\exists j')(\chi(V_{i,j}) = U_{i,j'})$. We denote the set of isomorphisms that respect decompositions $\mathcal{V}$ and $\mathcal{U}$ by $\mathrm{ISOp}((X,\mathcal{V}),(Y,\mathcal{U}))$, where the 'p' stands for product decomposition. If the decompositions are understood from context, we will write $\mathrm{ISOp}(X,Y)$.

Definition 6.3.5 (ISOd)**.** Given two groups X, Y, along with product decompositions $X = \prod_{i=1}^{r}\prod_{j=1}^{k_i} V_{i,j}$, and $Y = \prod_{i=1}^{r}\prod_{j=1}^{k_i} U_{i,j}$, where $(\forall i,j)(U_{i,j} \cong V_{i,j} \cong T_i)$, and let φ, ψ be diagonal products of $\mathcal{V} = (V_{i,j})$ and $\mathcal{U} = (U_{i,j})$ respectively. We say that an isomorphism $\chi \in \mathrm{ISOp}((X,\mathcal{V}),(Y,\mathcal{U}))$ *respects* the diagonal products φ and ψ if $\varphi\chi = \psi$. We denote the set of diagonal product respecting isomorphisms by $\mathrm{ISOd}((X,\mathcal{V}),(Y,\mathcal{U});\varphi,\psi)$. Again, if $\mathcal{V}$ and $\mathcal{U}$ are understood from context, we will omit them.

The following fact is clear from the definitions, and will be used to reduce to testing diagonal-respecting isomorphisms.

Lemma 6.3.6. *Let X, Y, be two groups with product decompositions $X = \prod_{i=1}^{r}\prod_{j=1}^{k_i} V_{i,j}$, $Y = \prod_{i=1}^{r}\prod_{j=1}^{k_i} U_{i,j}$, where $(\forall i,j)(U_{i,j} \cong V_{i,j} \cong T_i)$. Let $\mathcal{V} = (V_{i,j})$, and $\mathcal{U} = (U_{i,j})$ be the* *product decompositions. Fix a diagonal product $\varphi = (\varphi_1, \ldots, \varphi_r)$ of $\mathcal{V}$, and let $\mathcal{D}$ be the set of all diagonal products of $\mathcal{U}$. Then*

$$\mathrm{ISOp}((X, \mathcal{V}), (Y, \mathcal{U})) = \bigcup_{\psi \in \mathcal{D}} \mathrm{ISOd}((X, \mathcal{V}), (Y, \mathcal{U}); \varphi, \psi).$$

Proof. Given $\chi \in \mathrm{ISOp}((X, \mathcal{V}), (Y, \mathcal{U}))$, let $\psi = \varphi\chi$. Then ψ is a diagonal product of Y, since χ respects the product decomposition, and χ respects φ, ψ by definition. □

The following fact, which follows from the definitions, will help us bound the cost of using the previous lemma.

Lemma 6.3.7. *The number of diagonal products of a system of groups $((V_{i,j})_{j=1}^{k_i})_{i=1}^{r}$, where $(\forall i, j)(V_{i,j} \cong T_i)$ is $\prod_{i=1}^{r} |\mathrm{Aut}(T_i)|^{k_i}$.*

Proof. For each i, let d_i be the number of diagonals $\varphi_i : T_i \to \prod_{j=1}^{k_i} V_{i,j}$. The number of diagonal products will be $\prod_{i=1}^{r} d_i$. Now fix some i. The set of diagonals $\varphi_i : T_i \to \prod_{j=1}^{k_i} V_{i,j}$ is in bijective correspondence to the set $\prod_{j=1}^{k_i} \mathrm{ISO}(T_i, V_{i,j})$, since $\varphi_i(T_i)$ is a subdirect product. But $T_i \cong V_{i,j}$ by assumption, and hence $(\forall j)(|\mathrm{ISO}(T_i, V_{i,j})| = |\mathrm{Aut}(T_i)|)$. Therefore $d_i = |\mathrm{Aut}(T_i)|^{k_i}$.

□

In our algorithm we will encounter the case where the factors of the decomposition are non-abelian simple. If X and Y are groups that are direct products of non-abelian simple groups, when we write $\mathrm{ISOp}(X, Y)$, and $\mathrm{ISOd}(X, Y; \varphi, \psi)$ omitting the decomposition, we mean the unique decomposition given by Proposition 6.2.6.

6.3.3 Reduction to fixed diagonal products of the socles

Let G and H be two semisimple groups, with $\mathrm{Soc}(G) \cong \mathrm{Soc}(H) \cong \prod_{i=1}^{r} T_i^{k_i}$, where the T_i are pairwise non-isomorphic non-abelian simple groups. Corollary 6.3.3 applied to G and H, with $R = \mathrm{Soc}(G)$, $S = \mathrm{Soc}(H)$ implies that the isomorphisms between G and H are determined by the isomorphisms of their socles. For φ, ψ diagonal products of $\mathrm{Soc}(G)$ and $\mathrm{Soc}(H)$ respectively, let the set of **ISO**morphisms that respect **d**iagonal products of the **s**ocle be:

$$\mathrm{ISOds}(G, H; \varphi, \psi) = \{\chi \in \mathrm{ISO}(G, H) : \chi|_{\mathrm{Soc}(G)} \in \mathrm{ISOd}(\mathrm{Soc}(G), \mathrm{Soc}(H); \varphi, \psi)\}.$$

The following observation is an immediate consequence of Corollary 6.3.3 and Lemma 6.3.6.

Corollary 6.3.8. *Let G, H be semisimple, φ a diagonal product of* $\mathrm{Soc}(G)$*, and $\mathcal{D}$ the set of diagonal products of* $\mathrm{Soc}(H)$*. Then*

$$\mathrm{ISO}(G, H) = \bigcup_{\psi \in \mathcal{D}} \mathrm{ISOds}(G, H; \varphi, \psi).$$

The next fact, an easy consequence of Lemma 6.3.7, shows that this reduces ISO to polynomially many instances of ISOds.

Observation 6.3.9. *Let $\mathcal{D}$ be defined as in the previous corollary. Then $|\mathcal{D}| \leq |H|^2$.*

Proof. Each T_i is non-abelian simple, hence it is generated by 2 elements. Therefore $|\mathrm{Aut}(T_i)| = |T_i|^2$, since an automorphism is determined by the images of the generators. So, by Lemma 6.3.7, the number of diagonal products is bounded by

$$\prod_{i=1}^{r} \prod_{j=1}^{k_i} |\mathrm{Aut}(T_i)| = \prod_{i=1}^{r} \prod_{j=1}^{k_i} |T_i|^2 \leq |\mathrm{Soc}(H)|^2 \leq |H|^2.$$

□

NOTATION. If a group G is a direct product $G = \prod_{i=1}^{k} G_i$, we will denote the set of factors of G by $\mathrm{Fac}(G) = \{G_1, \ldots, G_k\}$.

Recall that $P := G/\mathrm{Pker}(G)$ is a permutation group of degree at most $\log_{60} |G|$. The orbits of P correspond to minimal normal subgroups: using the notation from decomposition (6.2), $V_{i,j}$ and $V_{i,j'}$ are in the same orbit of P if and only if they are both factors of the same minimal normal subgroup. In particular, P is transitive if and only if G has a unique minimal normal subgroup.

Lemma 6.3.10. *Let G and H be semisimple groups given by Cayley tables, with* $\mathrm{Soc}(G) \cong \mathrm{Soc}(H) \cong \prod_{i=1}^{r} T_i^{k_i}$*, and let* $P := G/\mathrm{Pker}(G)$*,* $Q := H/\mathrm{Pker}(H)$*. Then (a) every isomorphism* $\chi \in \mathrm{ISOds}(G, H; \varphi, \psi)$ *is determined by the permutational isomorphism it induces between P and Q; (b) given a permutational isomorphism* $f \in \mathrm{PISO}(P, Q)$*, we can check whether it arises as the action of some* $\chi \in \mathrm{ISOds}(G, H; \varphi, \psi)$*, and if so find that unique χ, in polynomial time.*

Proof. By Corollary 6.3.3, it suffices to prove the statement for every isomorphism χ of $\mathrm{Soc}(G)$ and $\mathrm{Soc}(H)$ that respects the diagonals φ and ψ. Applying decomposition (6.2), let us write

$$\mathrm{Soc}(G) = \prod_{i=1}^{r}\prod_{j=1}^{k_i} V_{i,j}, \qquad \text{and} \qquad \mathrm{Soc}(H) = \prod_{i=1}^{r}\prod_{j=1}^{k_i} U_{i,j},$$

where $(\forall i, j)(V_{i,j} \cong U_{i,j} \cong T_i)$. Let $\varphi = \varphi_1 \times \cdots \times \varphi_r$, where $\varphi_i : T_i \hookrightarrow \prod_{j=1}^{k_i} V_{i,j}$ is a diagonal. Similarly define $(\psi_i)_{i=1}^{r}$ for H. For all i, j, let $\varphi_{ij} : T_i \to V_{i,j}$ be the projection of φ_i to the jth coordinate of $\prod_{j=1}^{k_i} V_{i,j}$. Similarly let $\psi_{ij} : T_i \to U_{i,j}$ be the projection of ψ_i to the jth coordinate of $\prod_{j=1}^{k_i} U_{i,j}$. Notice that φ_{ij} is an isomorphism between T_i and $V_{i,j}$, and ψ_{ij} is an isomorphism between T_i and $U_{i,j}$. Now let $\chi \in \mathrm{ISOds}(G, H; \varphi, \psi)$, and let ℓ be such that $\chi(V_{i,j}) = U_{i,\ell}$; we claim that ℓ determines $\chi|_{V_{i,j}}$. Indeed in order for χ to respect the diagonal products, we must have $\chi|_{V_{i,j}} = \varphi_{ij}^{-1}\,\psi_{i\ell}$. Therefore χ is uniquely determined by its action on $\mathrm{Fac}(\mathrm{Soc}(G))$, which must be a permutational isomorphism of P and Q, since χ is an isomorphism.

To prove (b), construct $\chi_f : \mathrm{Soc}(G) \to \mathrm{Soc}(H)$ as follows. For every $i \le r, j \le k_i$, let $f(V_{i,j}) = U_{i,\ell}$. Then $\chi_f|_{V_{i,j}} = \varphi_{ij}^{-1}\,\psi_{i\ell}$. Now $\chi_f \in \mathrm{ISOd}(\mathrm{Soc}(G), \mathrm{Soc}(H); \varphi, \psi)$, and by Corollary 6.3.3, we can check in polynomial time if it extends to an isomorphism of G and H. □

6.4 Testing Isomorphism of Semisimple Groups with a Unique Minimal Normal Subgroup

As a first application of the group theoretic framework from the previous section and the permutational isomorphism test for transitive permutation groups (Theorem 5.2.1), we give an isomorphism test of semisimple groups with a unique minimal normal subgroup. This result is an ingredient in our isomorphism test for all semisimple groups, analogously to the way in which the transitive case was used for the intransitive case of permutational isomorphism.

Theorem 6.4.1. *Let G and H be semisimple groups with a unique minimal normal subgroup. Then (a) $|\mathrm{Aut}(G)| \leq |G|^{O(1)}$, and (b) we can list $\mathrm{ISO}(G, H)$ in time $|G|^{O(1)}$.*

Proof. Fix a diagonal ϕ for $\mathrm{Soc}(G)$ and try every possible diagonal ψ for $\mathrm{Soc}(H)$. By Corollary 6.3.8, we can focus on finding $\mathrm{ISOds}(G, H; \varphi, \psi)$. By Observation 6.3.9 this will only add a factor of $|\mathrm{Soc}(G)|^2 \leq |G|^2$ to our count and complexity.

Let $P = G/\mathrm{Pker}(G)$ and $Q = H/\mathrm{Pker}(H)$. Recall that P and Q are transitive, since G and H have a unique minimal normal subgroup. Part (a) then follows from the bound of the number of permutational isomorphisms of a transitive permutation group (part (a) of Theorem 5.2.1) and part (a) of Lemma 6.3.10.

To list all the isomorphisms, by part (b) of Theorem 5.2.1, we can list all $f \in \mathrm{PISO}(P, Q)$ in time $O(|P|c^{\log_{60}|G|}) \leq O(|G|^2)$, since the number of simple factors of G (and hence the degree of P) is at most $\log_{60}|G|$. By part (b) of Lemma 6.3.10, for each such f, we can check if extends to some $\chi \in \mathrm{ISOds}(G, H; \varphi, \psi)$ in polynomial time in $|G|$. $\square$

6.5 All Semisimple Groups – Reduction to Twisted Code Equivalence

In this section we give a reduction of isomorphism testing of semisimple groups to twisted code equivalence, proving Theorem 6.1.2.

6.5.1 *Embedding into the automorphism group of the socle*

Let G and H be semisimple groups, and consider decomposition (6.1) of $\mathrm{Soc}(G)$ and $\mathrm{Soc}(H)$ as the product of the minimal normal subgroups:

$$\mathrm{Soc}(G) = \prod_{i=1}^{d}\prod_{j=1}^{z_i} N_{i,j}, \qquad \mathrm{Soc}(H) = \prod_{i=1}^{d}\prod_{j=1}^{z_i} M_{i,j}.$$

Recall that $(\forall i,j)(N_{i,j} \cong M_{i,j} \cong K_i)$, and all the K_i are characteristically simple ($K_i = T_i^{t_i}$, for T_i non-abelian simple). For notational convenience, let $\mathcal{K} = \prod_{i=1}^{d} K_i^{z_i}$. Let γ and ξ be diagonal products of the systems $(N_{i,j})$ and $(M_{i,j})$ respectively. Arbitrarily pick an $\alpha_\gamma \in \mathrm{ISOd}(\mathrm{Soc}(G), \mathcal{K}; \gamma, \Delta)$, and a $\beta_\xi \in \mathrm{ISOd}(\mathrm{Soc}(H), \mathcal{K}; \xi, \Delta)$, where Δ is the standard diagonal product of $\prod_{i=1}^{d} K_i^{z_i}$. By Fact 6.3.1, the conjugation action of G and H on their socles gives us corresponding embeddings $\alpha_\gamma^* : G \hookrightarrow \mathrm{Aut}(\mathcal{K})$, and $\beta_\xi^* : H \hookrightarrow \mathrm{Aut}(\mathcal{K})$ (see Figure 6.2). Let $G^* = \alpha_\gamma^*(G)$, and $H^* = \beta_\xi^*(H)$. We have that $\mathrm{Soc}(G^*) = \mathrm{Soc}(H^*) = \mathrm{Inn}(\mathcal{K})$. Notice that in fact $G^*, H^* \leq \prod_{i=1}^{d} \mathrm{Aut}(K_i)^{z_i}$, since the conjugation action of G and H on their minimal normal subgroup fixes them (they are normal). Let

$$\mathrm{ISO}^*(G^*, H^*) = \{\widehat{f} \mid f \in \mathrm{ISO}(\mathrm{Soc}(G), \mathrm{Soc}(H))\}.$$

By Observation 6.3.2,

$$\mathrm{ISO}(G,H) = \alpha_\gamma^*\, \mathrm{ISO}^*(G^*, H^*)\, (\beta_\xi^*)^{-1}.$$

We will need to consider diagonals of the decomposition of the socle into simple groups.

$$\begin{tikzcd}
 & \mathrm{Aut}(\prod K_i^{z_i}) & \\
G \arrow[ur, "\alpha^*_\gamma"] & \mathrm{Inn}(\prod K_i^{z_i}) \arrow[u, "\trianglelefteq"] & H \arrow[ul, "\beta^*_\xi"'] \\
\mathrm{Soc}(G) \arrow[u, "\trianglelefteq"] \arrow[r, "\alpha_\gamma"'] \arrow[ur, dashed] & \prod K_i^{z_i} \arrow[u, "\cong"] & \mathrm{Soc}(H) \arrow[l, "\beta_\xi"] \arrow[ul, dashed] \arrow[u, "\trianglelefteq"']
\end{tikzcd}$$

Figure 6.2: Embedding G and H into $\mathrm{Aut}(\prod K_i^{z_i})$.

For diagonals φ of $\mathrm{Soc}(H)$ and ψ of $\mathrm{Soc}(H)$, let

$$\mathrm{ISOds}^*(G^*, H^*; \varphi, \psi) = \{\widehat{f} \mid f \in \mathrm{ISOd}(\mathrm{Soc}(G), \mathrm{Soc}(H); \varphi, \psi)\}.$$

Let φ and ψ be the diagonal products of the socles (in decomposition (6.2) as the product of the simple factors) induced by γ and ξ respectively. Then we have

$$\mathrm{ISOds}(G, H; \varphi, \psi) = \alpha^*_\gamma \, \mathrm{ISOds}^*(G^*, H^*; \Delta, \Delta) \, (\beta^*_\xi)^{-1}. \tag{6.4}$$

Here Δ is the standard diagonal product of the decomposition of the socles into simple factors.

6.5.2 Reduction to code equivalence

By Equation (6.4), Corollary 6.3.8 and Observation 6.3.9, finding $\mathrm{ISO}(G, H)$ reduces to polynomially many instances of the case of two groups $G^*, H^* \le \prod_{i=1}^d \mathrm{Aut}(K_i)^{z_i}$, with $\mathrm{Soc}(G^*) = \mathrm{Soc}(H^*) = \mathrm{Inn}(\prod_{i=1}^d \mathrm{Aut}(K_i)^{z_i})$, and we need to compute $\mathrm{ISOds}^*(G^*, H^*; \Delta, \Delta)$.

Let G^*_{ij} be the restriction of G^* to the jth copy of K_i, and similarly, H^*_{ij} the restriction of H^* to the jth copy of K_i. These are groups with a unique minimal normal subgroup, so by Theorem 6.4.1, we can compute their isomorphism types under permutational isomorphisms that preserve the standard diagonals. In other words, for every i, j, i', j', we compute $\mathrm{ISOds}^*(G^*_{ij}, H^*_{i'j'}; \Delta, \Delta)$. Let $\Gamma_1, \ldots, \Gamma_r$ be representatives of these isomorphism types. These will be our alphabets. Choose arbitrary isomorphisms χ_{ij} that preserve the standard diagonals between the G^*_{ij} and the corresponding representative, and analogously choose arbitrary δ_{ij} for the H^*_{ij}.

We create codes $\overline{G}$ and $\overline{H}$ over the alphabets Γ_i as follows. Let $\sigma \in G^*$, and let $\sigma_{ij} \in G^*_{ij}$ denote the restriction of σ to N_{ij}. The string associated with σ is $\overline{\sigma} = (\sigma_{ij}^{\chi_{ij}})_{ij}$. Then $\overline{G} = \{\overline{\sigma} \mid \sigma \in G^*\}$. Define $\overline{H}$ analogously, by setting $\overline{\pi} = (\pi_{ij}^{\delta_{ij}})_{ij}$, for $\pi \in H$.

For $\ell \in [r]$, let $W_\ell = \mathrm{Autds}^*(\Gamma_\ell; \Delta, \Delta) := \mathrm{ISOds}^*(\Gamma_\ell, \Gamma_\ell; \Delta, \Delta)$. This is the group of automorphisms of Γ_ℓ induced by permutational automorphisms that preserve the standard diagonals. By Lemma 6.3.10, these automorphisms are determined by the permutation they induce on the set of simple factors. Thus if Γ_ℓ is acting on a copy of K_i then W_ℓ has a faithful permutation representation of degree t_i (the number of simple factors of K_i).

Let m_ℓ be the number of the G^*_{ij} of isomorphism type Γ_ℓ. W.l.o.g. the number of H^*_{ij} of isomorphism type Γ_ℓ is also m_ℓ (otherwise there are no isomorphisms of G^* and H^* respecting the standard diagonals). Then $\overline{G}, \overline{H} \leq \prod_{\ell=1}^{r} \Gamma_\ell^{m_\ell}$ are subgroups; hence we view $\overline{G}$ and $\overline{H}$ as groups as well as codes.

Lemma 6.5.1. *Using the definitions from above,*

$$\mathrm{ISOds}^*(\overline{G}, \overline{H}; \Delta, \Delta) = \widehat{\mathrm{EQ}}_{(W_1, \ldots, W_r)}(\overline{G}, \overline{H})$$

Proof. Let $\chi \in \mathrm{ISOds}^*(\overline{G}, \overline{H}; \Delta, \Delta)$. By Lemma 6.3.10, χ is determined by the permutation it induces on the simple factors. Let $\pi(\chi)$ be the permutation of the minimal normal subgroups induced by χ. $\pi = \pi(\chi)$ determines χ up to elements of the W_ℓ applied to each letter of the codes $\overline{G}$ and $\overline{H}$. In other words, the set of $\mathrm{ISOds}^*(\overline{G}, \overline{H}; \Delta, \Delta)$ is exactly the set of (W_ℓ)-twisted equivalences of the codes $\overline{G}$ and $\overline{H}$. □

6.5.3 The algorithm

Theorem 6.5.2. *Given G and H semisimple, we can find $\mathrm{ISO}(G, H)$ in time $\mathrm{poly}(|G|)$.*

Proof. 1. Fix a diagonal product φ of $\mathrm{Soc}(G)$ and try all possible diagonal products ψ of $\mathrm{Soc}(H)$. By Corollary 6.3.8 we can now focus on finding $\mathrm{ISOds}(G, H; \varphi, \psi)$.
2. Use the algorithm from part (a) of Proposition 6.2.6 to compute decomposition (6.1) of the socles as the product of minimal normal subgroups, grouped by isomorphism type. If a difference is noted between G and H, reject isomorphism. Otherwise pick representatives K_i of the isomorphism types of the minimal normal subgroups, and construct G^* and H^* as in Section 6.5.1.
3. By Equation (6.4), the problem reduces to finding $\mathrm{ISOds}^*(G^*, H^*; \Delta, \Delta)$. Use part (b) of Theorem 6.4.1 to compute the Γ_ℓ and W_ℓ; construct the codes $\overline{G}$ and $\overline{H}$. The problem is now reduced to finding $\mathrm{ISOds}^*(\overline{G}, \overline{H}; \Delta, \Delta)$ (cf. Section 6.5.2).
4. Use Theorem 4.2.1 to compute $\widehat{\mathrm{EQ}}_{(W_1,\dots,W_\ell)}(\overline{G}, \overline{H})$, which is $\mathrm{ISOds}^*(\overline{G}, \overline{H}; \Delta, \Delta)$ by Lemma 6.5.1.

Analysis. Step 1 takes polynomial time by Observation 6.3.9. Step 2 takes polynomial time by Proposition 6.2.6 and step 3 by part (b) of Theorem 6.4.1. The degree of the permutation groups involved is bounded by the number of simple factors of the socles, which is at most $\log_{60}(|G|)$, and the order of the permutation groups is no larger than the order of the original groups. Therefore the algorithm for permutational isomorphism of permutation groups runs in polynomial time in $|G|$.

The running time of step 4 is determined by the twisted code equivalence algorithm. By Theorem 6.4.1, the order of each W_ℓ is $|G^*_{ij}|^{O(1)} \leq |G|^{O(1)}$, and we can list all elements of W_ℓ in polynomial time. By Lemma 6.3.10, for each W_ℓ that acts on a copy of K_i, we can compute a faithful permutation representation in S_{t_i}, where t_i is the number of simple factors of K_i. The size of the codes is $|G|$. The degree of the permutation groups involved is $\sum_{i=1}^{d} t_i z_i$ = number of simple factors of $\mathrm{Soc}(G)| \leq \log_{60}(|G|)$. Therefore by Theorem 4.2.1, the running time is $2^{2\log_{60}|G|}|G|^{O(1)} = |G|^{O(1)}$.

$\square$

This concludes the proof of Theorem 6.1.2.

CHAPTER 7
CONCLUSIONS

Our results use critically both combinatorial and group theoretic approaches to solve isomorphism problems. For isomorphism of combinatorial structures, the combination of group theoretic and combinatorial methods was used already in 1979, in the paper that introduced the group theoretic methods [2]. However, this combination had not been exploited since then. While the algorithm for isomorphism of general graph isomorphism does use both the combinatorial techniques and the group theoretic techniques, it does so separately: first it applies the combinatorial trick due to Zemlyachenko [84], and then Luks's group theoretic techniques [55]. In our application to hypergraph isomorphism we use the two techniques intertwined.

Our isomorphism test for semisimple groups is based on new connections between problems that are group theoretic and combinatorial in nature. Indeed we reduce isomorphism of semisimple groups to the transitive case of permutational isomorphism, and instances of twisted code equivalence.

7.1 Open Problems

In this section we collect the open problems mentioned in the previous chapters.

The biggest open problem in the area is to obtain a faster isomorphism test for graphs.

Open Problem 7.1.1. Solve Graph Isomorphism in time $\exp(n^{1/2-\epsilon})$.

An important question is about finding canonical forms for hypergraphs.

Open Problem 7.1.2 (Cf. Section 3.8)**.** Canonical forms for hypergraphs in moderately exponential (or even simply exponential) time.

Another question is to obtain a moderately exponential isomorphism test for hypergraphs with a running time that has a better dependence on k.

Open Problem 7.1.3 (Cf. Section 3.8)**.** Test Isomorphism of k-hypergraphs in time $\exp(\sqrt{nk})$.

The big open problem in the area of group isomorphism is to find a polynomial time isomorphism test.

Open Problem 7.1.4 (Cf. Section 1.1.2)**.** Solve Group Isomorphism in polynomial time.

We showed that abelian normal subgroups are the only obstacle to solving this problem (cf. Chapter 6). A better understanding of nilpotent groups (even nilpotent groups of class 2) is necessary. Indeed it is believed that nilpotent groups of class 2 are the hardest case. However there is no reduction to the nilpotent case, and a result in this direction would be interesting.

Open Problem 7.1.5 (Cf. Chapter 5)**.** Solve Permutational Isomorphism in time simply exponential in the degree. An interesting first step would be to answer this question for transitive permutation groups.

Open Problem 7.1.6 (Cf. Open Problem 4.3.3)**.** Solve Group Code Equivalence in time $c^n \operatorname{poly}\log(|\Gamma|)$.

While the problem is solved for $\Gamma = \mathbb{Z}_p$ (Theorem 4.3.4 cf. [10]), it remains open even for elementary abelian groups ($\mathbb{Z}_p \times \cdots \times \mathbb{Z}_p$) and cyclic groups ($\mathbb{Z}_m$).

REFERENCES

[1] László Babai. On the isomorphism problem. In *In Proc. Fundamentals of Computation Theory, Poznan-Kornik*, page 10, 1977. Article was appended to this publication.

[2] László Babai. Monte carlo algorithms in graph isomorphism testing. In *Université de Montréal Technical Report*, volume 70 of *DMS*, page 42, 1979. `http://people.cs.uchicago.edu/∼laci/lasvegas79.pdf`.

[3] László Babai. On the complexity of canonical labelling of strongly regular graphs. *SIAM Journal on Computing*, 9:212–216, 1980.

[4] László Babai. Moderately exponential bound for graph isomorphism. In *Fundamentals of Computation Theory*, pages 34–50, 1981.

[5] László Babai. On the order of uniprimitive permutation groups. *The Annals of Mathematics*, 113:553–568, 1981.

[6] László Babai. On the order of doubly transitive permutation groups. *Inventiones Mathematicae*, 65:473–484, 1982.

[7] László Babai. Permutation groups, coherent configurations, and graph isomorphism, April 1983. D.Sc. Thesis, Hungarian Academy of Sci. (Hungarian).

[8] Lászlo Babai. On the length of subgroup chains in the symmetric group. *Communications in Algebra*, 14:1729–1736, 1986.

[9] László Babai. Automorphism groups, isomorphism, reconstruction. In Ronald L. Graham, Martin Grötschel, and László Lovász, editors, *Handbook of Combinatorics*, volume II, pages 1447–1540. Elsevier Science, 1995.

[10] László Babai. Equivalence of linear codes, 2010. Unpublished manuscript.

[11] László Babai. Coset intersection in moderately exponential time. *Chicago Journal of Theoretical Computer Science*, to appear.

[12] László Babai and Robert Beals. A polynomial-time theory of black-box groups I. In C. M. Campbell, E. F. Robertson, N. Ruskuc, and G. C. Smith, editors, *Groups St Andrews 1997 in Bath, I*, volume 260 of *London Mathematical Society Lecture Notes*, pages 30–64. Cambr. U. Press, 1999.

[13] László Babai, Robert Beals, and Ákos Seress. Polynomial-time theory of matrix groups. In *41st STOC*, pages 55–64. ACM Press, 2009.

[14] László Babai, Robert Beals, and P. Takácsi-Nagy. Symmetry and complexity. In *Proc. 24th STOC*, pages 438–449. ACM Press, 1992.

[15] Lászlo Babai, Peter J. Cameron, and Peter P. Pálfy. On the orders of primitive groups with restricted nonabelian composition factors. *Journal of Algebra*, 79:161–168, 1982.

[16] László Babai and Paolo Codenotti. Isomorphism of hypergraphs of low rank in moderately exponential time. In *FOCS*, pages 667–676, 2008.

[17] László Babai, Paolo Codenotti, Joshua A. Grochow, and Youming Qiao. Code equivalence and group isomorphism. In *SODA*, pages 1395–1408, 2011.

[18] László Babai, Paolo Codenotti, and Youming Qiao. Testing isomorphism of groups with no abelian normal subgroups. in preparation, 2011.

[19] László Babai, Paul Erdős, and Stanley M. Selkow. Random graph isomorphism. *Siam Journal on Computing*, 9:628–635, 1980.

[20] László Babai, D. Yu. Grigoryev, and David M. Mount. Isomorphism of graphs with bounded eigenvalue multiplicity. In *STOC*, pages 310–324. ACM Press, 1982.

[21] László Babai, William M. Kantor, and Eugene M. Luks. Computational complexity and the classification of finite simple groups. In *Proc. 24th IEEE FOCS*, pages 162–171. IEEE Comp. Soc., 1983.

[22] László Babai and Ludek Kučera. Canonical labelling of graphs in linear average time. In *FOCS*, pages 39–46, 1979.

[23] László Babai and Eugene M. Luks. Canonical labeling of graphs. In *Proc. 15th STOC*, pages 171–183. ACM Press, 1983.

[24] László Babai, Eugene M. Luks, and Ákos Seress. Permutation groups in NC. In *Proc. 19th STOC*, pages 409–420. ACM Press, 1987.

[25] László Babai and Rudolf Mathon. Talk at the south-east conference on combinatorics and graph theory, 1980.

[26] László Babai and Shlomo Moran. Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity classes. *J. Computer and Sys. Sci.*, 36:254–276, 1988.

[27] László Babai and Youming Qiao. Efficient isomorphism test for groups with abelian sylow tower. Manuscript, 2011.

[28] K. S. Booth and C. J. Colbourn. Problems polynomially equivalent to graph isomorphism. Technical report, Computer Science Department, University of Waterloo, Waterloo, Ont, 1979.

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

[30] Armand Borel. *Linear Algebraic Groups*, volume 126 of *Graduate texts in mathematics*. Springer-Verlag, 2nd edition, 1991.

[31] Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12:389–410, 1992.

[32] Peter J. Cameron. Strongly regular graphs. In *Selected Topics in Graph Theory*, pages 337– 380. Academic Press, London, 1978.

[33] Peter J. Cameron. Finite permutation groups and finite simple groups. *The Bulletin of the London Mathematical Society*, 13(1):1–22, 1981.

[34] John Horton Conway, Robert Turner Curtis, Simon Phillips Norton, Richard A. Parker, and Robert Arnott Wilson. *Atlas of Finite Groups: Maximal Subgroups and Ordinary Characters for Simple Groups.* Oxford University Press, 1985.

[35] John D. Dixon and Brian Mortimer. *Permutation Groups*, volume 163 of *Graduate Texts in Mathematics*. Springer-Verlag, 1991.

[36] I. S. Filotti and Jack N. Mayer. A polynomial-time algorithm for determining the isomorphism of graphs of fixed genus. In *Proc. 12th STOC*, pages 236–243. ACM Press, 1980.

[37] Fedor V. Fomin, Kazuo Iwama, and Dieter Kratsch. Abstracts collection. In *Moderately Exponential Time Algorithms*, number 08431 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2008. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

[38] M. Fürer, W. Schnyder, and E. Specker. Normal forms for trivalent graphs and graphs of bounded valence. In *Proc. 15th STOC*, pages 161–170. ACM Press, 1983.

[39] Merrick L. Furst, John Hopcroft, and Eugene M. Luks. Polynomial-time algorithms for permutation groups. In *Proc. 21st FOCS*, pages 36–41. IEEE Comp. Soc., 1980.

[40] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity, or all languages in np have zero-knowledge proof systems. *Journal ACM*, 38:690–728, 1991.

[41] Robert Guralnick and László Pyber, 2008. Personal communication.

[42] J. E. Hopcroft and R. E. Tarjan. Isomorphism of planar graphs. In *Complexity of Computer Computations*, page 131152. Plenum, 1972.

[43] J. E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *Proceedings of the sixth annual ACM symposium on Theory of computing*, STOC, pages 172–184. ACM Press, 1974.

[44] James E. Humphreys. *Linear Algebraic Groups*, volume 21 of *Graduate texts in mathematics.* Springer-Verlag, 1975.

[45] Russell Impagliazzo and Ramamohan Paturi. The complexity of k-sat. *Annual IEEE Conference on Computational Complexity*, 0:237, 1999.

[46] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512 – 530, 2001.

[47] William M. Kantor. Polynomial-time algorithms for finding elements of prime order and sylow subgroups. *Journal of Algorithms*, 6(4):478 – 514, 1985.

[48] William M. Kantor. Sylow's theorem in polynomial time. *Journal of Computer and System Sciences*, 30(3):359 – 394, 1985.

[49] Telikepalli Kavitha. Linear time algorithms for abelian group isomorphism and related problems. *Journal of Computer and System Sciences*, 73(6):986–996, 2007.

[50] Neeraj Kayal and Timur Nezhmetdinov. Factoring groups efficiently. In *ICALP: Proceedings of the 36th International Colloquium on Automata, Languages and Programming*, pages 585–596. Springer-Verlag, 2009. Also availabe as ECCC Tech Report TR08-074.

[51] Donald E. Knuth. Efficient representation of perm groups. *Combinatorica*, 11:57–68, 1991.

[52] L. G. Kovács and M. F. Newman. Generating transitive permutation groups. *Quartely Journal of Mathematics*, 39:283–292, 1988.

[53] François Le Gall. Efficient isomorphism testing for a class of group extensions. In *STACS*, pages 625–636, 2009.

[54] David Lichtenstein. Isomorphism for graphs embeddable on the projective plane. In *Symposium on Theory of Computing (STOC)*, pages 218–224. ACM Press, 1980.

[55] Eugene M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comp. Sys. Sci.*, 25:42–65, 1982.

[56] Eugene M. Luks. Computing the composition factors of a permutation group in polynomial time. *Combinatorica*, 7:87–99, 1987.

[57] Eugene M. Luks. Hypergraph isomorphism and structural equivalence of boolean functions. In *Proc. 31st STOC*, pages 652–658. ACM Press, 1999.

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

[59] Brendan D. McKay. Practical graph isomorphism. In *Congressus Numerantium, 30*, pages 45–87, 1981.

[60] Gary Miller. Isomorphism testing for graphs of bounded genus. In *Proceedings of the twelfth annual ACM symposium on Theory of computing*, STOC, pages 225–235. ACM, 1980.

[61] Gary L. Miller. Graph isomorphism, general remarks. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, STOC, pages 143–150. ACM Press, 1977.

[62] Gary L. Miller. On the $n \log n$ isomorphism technique (a preliminary report). In *Proc. 10th STOC*, pages 51–58, New York, NY, USA, 1978. ACM Press.

[63] Matthias Mnich. Algorithms in moderately exponential time, 2010. Ph.D. Thesis, Eindhoven University of Technology.

[64] A. Neumaier. Stronly regular graphs with smallest eigenvalue m. *Arch. Math.*, 33:392–400, 1979.

[65] László Pyber. Asymptotic results for permutation groups. In Larry Finkelstein and William M. Kantor, editors, *Groups and Computation*, pages 197–219. DIMACS, 1991.

[66] László Pyber. The orders of doubly transitive groups, elementary estimates. *Journal Combinatorial Theory*, 62:361–366, 1993.

[67] Youming Qiao, Jayalal M. N. Sarma, and Bangsheng Tang. On isomorphism testing of groups with normal hall subgroups. In *STACS*, pages 567–578, 2011.

[68] R. C. Read and D. G. Corneil. The graph isomorphism disease. *Journal of Graph Theory*, page 339363, 1977.

[69] Derek J.S. Robinson. *A Course in the Theory of Groups.* Springer-Verlag, 2nd edition, 1996.

[70] Carla Savage. An $O(n^2)$ algorithm for abelian group isomorphism. Technical report, North Carolina State University, 1980.

[71] Ákos Seress. *Permutation Group Algorithms.* Cambridge University Press, 2003.

[72] Charles C. Sims. Computation with permutation groups. In S. R. Petrick, editor, *Proceedings Second Symposium on Symbolic and Algebraic Manipulation*, pages 23–28. ACM Press, 1971.

[73] Charles C. Sims. Some group theoretic algorithms. *Lecture Notes in Mathematics*, 697:108–124, 1978.

[74] Daniel A. Spielman. Faster isomorphism testing of strongly regular graphs. In *28th STOC*, pages 576–584. ACM Press, 1996.

[75] Tony A. Springer. *Linear Algebraic Groups*, volume 9 of *Progress in mathematics.* Birkhäuser, 1981.

[76] Michio Suzuki. *Group Theory II*, volume 248 of *A series of comprehensive studies in mathematics*. Springer-Verlag, 1986.

[77] Narayan Vikas. An $O(n)$ algorithm for abelian p-group isomorphism and an $O(n \log n)$ algorithm for abelian group isomorphism. *J. Comput. Syst. Sci.*, 53(1):1–9, 1996.

[78] B. Weisfeiler and A. A. Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction, (in russian). *Nauchno-Technicheskaya Informatsia*, Seriya 2, 9:12–16, 1968.

[79] B. Weisfeiler Ed. *On Construction and Identification of Graphs*, volume 558 of *Lecture Notes in Mathematics*. Springer, 1976.

[80] H. Wielandt. *Finite Permutation Groups*. Acad. Press, New York, 1964.

[81] James B. Wilson. Decomposing p-groups via Jordan algebras. *J. Algebra*, 322:2642–2679, 2009.

[82] James B. Wilson. Finding central decompositions of p-groups. *J. Group Theory*, 12:813–830, 2009.

[83] James B. Wilson. Finding direct product decompositions in polynomial time. Submitted for publication. Available as arXiv e-print 1005.0548., 2010.

[84] V. Zemlyachenko, N. Kornienko, and R. Tyshkevich. Graph isomorphism problem. *The Theory of Computation I*, 118, 1982.

APPENDIX A
NOTES ON TERMINOLOGY

A.1 On uses of the term 'moderately exponential'

The term 'moderately exponential' is used sometimes with different meanings. Often the definition is not precise, and just refers to algorithms that, while worse than polynomial, are not simply exponential. Many authors avoid the term altogether. In the relatively new field of exact algorithms for NP-hard problems, the term is often used with a specific meaning, different from ours (cf. e.g. [37, 63]). We chose our use to be consistent with the isomorphism literature(see e.g. [4, 57].

In this note I will talk about the connection between the different notions of moderately exponential, and related complexity classes. To avoid confusion, I will refer to our use of the term moderately exponential as ME and their use as ME^*. Recall that a problem with witness space $W(x)$ on input x is in ME if there is an algorithm with running time $\exp((\log|W(x)|)^{1-c})\operatorname{poly}(|x|)$, for some constant $c > 0$. A problem with witness space $W(x)$ on input x is in ME^* if there is an algorithm with running time $\exp((1-c)\ln|W(x)|)\operatorname{poly}(|x|)$, for some constant $c > 0$. We have that $\mathrm{ME} \subseteq \mathrm{ME}^*$, but not vice-versa. In this literature they also use 'sub-exponential time' (SE), a complexity class that is closer to our moderately exponential time. A problem with witness space $W(x)$ on input x, is in SE if there is an algorithm solving it in time $\exp(o(\log|W(x)|))\operatorname{poly}(|x|)$. We have the inclusions

$$\mathrm{ME} \subseteq \mathrm{SE} \subseteq \mathrm{ME}^* .$$

For example, for the satisfiability problem SAT on n variables, a running time of $2^n \operatorname{poly}(n)$ is exponential (that is the size of the search space), $(2-c)^n$ for some $c > 0$ is ME^*, $2^{n/\log n}$ is SE, and $2^{n^{1-c}}$ is ME. Some motivation for the use of ME^* and SE is that the Exponential Time Hypothesis states that there is no SE algorithm for 3-SAT, and the

Strong Exponential Time Hypothesis states that there is no ME* algorithm for SAT (cf. [45, 46]).

A.2 On uses of the term 'semisimple group'

The term 'semisimple group' is used with different meanings by different authors. Our definition is that a group is **semisimple** if it has no abelian normal subgroups (c.f. Definition 6.1.1, Section 6.1.1). In terms of the BB-filtration of groups (c.f. Section 6.2.2), an equivalent definition is that the solvable radical is trivial. Recall that the solvable radical of a group is the (unique) maximal solvable normal subgroup.

Below is a summary of definitions from various authors. Robinson [69] uses the same definition we use. Three books on linear algebraic groups give equivalent definitions that are different from ours, but have the same flavor: a linear algebraic group is semisimple if the radical is trivial. The difference lies in the definition of radical, which they additionally require to be connected. Suzuki [76] uses the term in yet another way, which is not of the same flavor as the others.

For arbitrary groups.

- D. J.S. Robinson [69] (page 89) has the same definition as ours: "A finite group is *semisimple* if it has no abelian normal subgroups."

- M. Suzuki [76] (page 446): "A group S is said to be *semisimple* if $S = S'$ and if $S/Z(S)$ is the direct product of nonabelian simple groups." Here S' is the commutator subgroup. In terms of the BB-filtration of groups ([12], c.f. Section 6.2.2), this means that S is perfect, $\mathrm{Rad}(S) = Z(S)$, and $S = \mathrm{Soc}^*(S)$.

For linear algebraic groups. The following three are equivalent.

- J. E. Humphreys [44] (19.5, page 125): "If $R(G)$ is trivial and $G \neq e$ is connected, we call G *semisimple*." Here $R(G)$ is the radical of G, defined as: "... an arbitrary algebraic group G possesses a unique largest normal solvable subgroup, which is automatically closed. Its identity component is then the largest connected normal solvable subgroup of G; we call it the *radical* of G, denoted $R(G)$."

A different but equivalent definition is given in the same book (13.5, page 89): "A connected algebraic group of positive dimension is *semisimple* if it has no closed connected commutative normal subgroups except $\{e\}$." This is motivated by the definition: "A Lie algebra of positive dimension is *semisimple* if it has no nonzero commutative ideal" (same page). This definition is similar in flavor to our first definition (Definition 6.1.1).

- T. A. Springer [75] (6.14, page 150): "A linear algebraic group is *semisimple* if the radical of G is the identity." Here the *radical* of a linear algebraic group is the maximal closed, connected, solvable normal subgroup. Note that the 'closed' condition is superfluous, as pointed out in the Humphreys quote above.

- A. Borel [30] (11.21, pp. 157-158): "A group is *semisimple* if $\mathcal{R}G = \{e\}$." Here $\mathcal{R}G$ is the radical of G, defined as

$$\mathcal{R}G = \left(\bigcap_{B \in \mathcal{B}} B \right)^{\circ},$$

Where G° is the connected component of e in G, and $\mathcal{B}$ is the set of all Borel subgroups. "A *Borel* subgroup is one which is maximal among the connected solvable subgroups" (11.1, page 147).