The Complexity of some Exact and Approximate Isomorphism Problems

By

Yadu Vasudev

MATH10200805001

The Institute of Mathematical Sciences, Chennai

A thesis submitted to the

Board of Studies in Mathematical Sciences

(Theoretical Computer Science)

In partial fulfillment of requirements

For the Degree of

DOCTOR OF PHILOSOPHY

of

HOMI BHABHA NATIONAL INSTITUTE

July, 2014

Homi Bhabha National Institute

Recommendations of the Viva Voce Board

As members of the Viva Voce Board, we certify that we have read the dissertation prepared by Yadu Vasudev entitled "The Complexity of some Exact and Approximate Isomorphism Problems" and recommend that it maybe accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

_____ Date:

Chair - Meena Mahajan

_____ Date:

Guide/Convener - V. Arvind

_____ Date:

Member 1 - Vikram Sharma

_____ Date:

Member 2 (External Examiner) - Piyush P. Kurur

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

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

Date:

Place: Guide

STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at Homi Bhabha National Institute (HBNI) and is deposited in the Library to be made available to borrowers under rules of the HBNI.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgement of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the Competent Authority of HBNI when in his or her judgement the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

(Yadu Vasudev)

DECLARATION

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

(Yadu Vasudev)

ACKNOWLEDGEMENTS

I would like to thank my advisor, V. Arvind, for all the support and encouragement that he has given me throughout my years in IMSc as his student. His guidance has been invaluable in this thesis seeing the light of the day. His clarity in explaining new ideas and the breadth of his knowledge are qualities that have inspired me to become a better researcher. He was always available whenever I needed any help - academic or otherwise. I am very fortunate to have spent six years working with him.

I would also like to thank Johannes Köbler for hosting me at the Humboldt University of Berlin for three wonderful summers; he made sure that I had a comfortable and productive stay. I thank Sebastian Kuhnert for all the discussions that we had during these visits. I learnt a great deal from Johannes and Sebastian during these visits and was also fortunate to collaborate with them on two papers whose work went into this thesis.

I also thank Kristoffer Hansen for hosting me at Aarhus university, Jacobo Toràn for hosting me at the University of Ulm and Eldar Fischer for hosting me at Technion.

I learnt a lot about Computer Science and research from the wonderful courses and seminars at IMSc. I thank all the faculty members at IMSc for all the and for creating a great environment for research and learning. I also thank my teachers at NIT, Calicut, for providing an excellant atmosphere for learning. They were instrumental in my deciding to take up research in Computer Science.

I owe many thanks to all the friends in IMSc for making my stay here enjoyable. They made sure that there were enough distractions. Special thanks to Karteek, Madhushree, Mubeena, Anoop, Sreejith, Baskar, Ramchandra, and Rohan for many wonderful days here.

Finally, a big thanks to Amma, Acchan, Muthassi, Valliamma, Kuttettammavan and Edathiammumma for the unconditional love and support during these six years.

Contents

Synopsis

In this thesis we study the algorithmic complexity of Graph Isomorphism and Boolean Function Isomorphism and some variants of these problems.

First, we define and study an optimization problem of the Graph Isomorphism problem that we call the Approximate Graph Isomorphism problem. Given two graphs G and H, we are interested in finding a bijection π from $V(G)$ to $V(H)$ that maximizes the number of matches (edges mapped to edges or non-edges mapped to non-edges). We give an α-approximation algorithm for this problem that runs in time $n^{O(\log n/\alpha^2)}$ for any constant α. We prove this by combining the $n^{O(\log n)}$ time additive error approximation algorithm of Arora et al. [AFK02] with a simple averaging algorithm. We also consider the corresponding minimization problem (of mismatches) and we prove that it is NP-hard to α-approximate for any constant factor α. Furthermore, we show that it is also NP-hard to approximate the maximum number of edges mapped to edges beyond a factor of 0.94. We also explore these optimization problems for bounded color class graphs which is a well studied tractable special case of Graph Isomorphism [Bab79, FHL80] and we show hardness of approximation results for these problems.

Next, we study the complexity of the Boolean Isomorphism problem which is defined as follows: Given two graphs G and H, test whether there exists a permutation $\pi : [n] \to [n]$ such that $f^\pi(x_1, \cdots, x_n) = f(x_{\pi(1)}, \cdots, x_{\pi(n)})$ and $g(x_1, \cdots, x_n)$ are equivalent. We study the computational complexity of the exact and approximate versions of the Boolean Isomorphism problem.

The objective in the Approximate Boolean Isomorphism problem is to construct a permutation $\pi : [n] \to [n]$ such that $\Pr[f^\pi(x) \neq g(x)] \leq \varepsilon$ for two given isomorphic Boolean functions f and g. We show that for Boolean functions computable by constant-depth polynomial-size circuits as well as for functions representable as linear threshold functions, there is a randomized $2^{\widetilde{O}(\sqrt{n}(\log(n/\varepsilon))^{O(1)})}$-time algorithm that computes a permutation π such that $\Pr[f^\pi(x) \neq g(x)] \leq \varepsilon$.

We then study the complexity of the Boolean Isomorphism problem based on the representation in which the functions f and g are given with the aim of bounding the complexity of the isomorphism problem by the complexity of the satisfiability problem for that representation. We show that when f and g are given as Horn-CNFs, then the Boolean Isomorphism problem is polynomial-time equivalent to the Graph Isomorphism problem. We also show that the Boolean Isomorphism problem is polynomial-time equivalent to Graph Isomorphism when f and g are given as bounded-rank decision trees. As a consequence we obtain a $2^{O((\log s)^{O(1)}\sqrt{n})}$-time algorithm for testing whether two functions given as size-s decision trees are isomorphic. Furthermore, we prove a trichotomy theorem regarding the complexity of the Boolean Isomorphism problem when the functions are given as decision lists.

List of Figures

Chapter 1

Introduction

The topic of this thesis is the computational complexity of the isomorphism problem of graphs, the isomorphism problem for Boolean functions for various representations, and some variants of these problems, especially centered on notions of approximate isomorphism.

Classifying computational problems based on the resources - time and space - required to solve them is one of the important objectives of computer science, and in particular, of complexity theory. The class P of problems that are decidable in polynomial time by deterministic Turing machines is widely accepted as the correct notion of problems that are efficiently solvable [Edm65, Cob65]. Proving that P is different from NP, the class of problems that are decidable in polynomial time by non-deterministic Turing machines is an outstanding open problem in complexity theory.

Cook [Coo71] introduced the theory of NP-completeness by showing that SAT is NP-complete. He proved that the computation of a polynomial-time non-deterministic Turing machine on an input can be *encoded* as a Boolean formula in time polynomial in the size of the description of the Turing machine such that the non-deterministic Turing machine accepts the input if and only if the Boolean formula is satisfiable. A consequence of this theorem is that if $\mathsf{SAT} \in \mathsf{P}$ then $\mathsf{NP} = \mathsf{P}$. Karp, in [Kar72], defined the notion of *polynomial-time many-one reductions* and showed that a wide variety of seemingly different problems are NP-complete.

Given an instance of $\mathcal{I}$ of a problem A, a *reduction* is an encoding $R(\mathcal{I})$ of the instance $\mathcal{I}$ to another problem $\mathcal{B}$. A polynomial-time many-one reduction is a function f computable by a polynomial time deterministic Turing machine that takes as input an instance $\mathcal{I}$ of a problem A and outputs an input instance $f(\mathcal{I})$ of B with the property that $\mathcal{I}$ is a "yes"-

instance of A if and only if $f(\mathcal{I})$ is a "yes"-instance of B. If A is reducible to B, it is denoted as $A \leq B$ since B is computationally at least as hard as A. As a consequence, if there is a polynomial time algorithm for B, then we can combine that with the reduction f to obtain a polynomial time algorithm for A.

1.1 Graph Isomorphism

The Graph Isomorphism problem is a very natural algorithmic question about graphs. Given two graphs G and H on n vertices, a graph isomorphism is a bijection $\pi : V(G) \to V(H)$ such that for each pair (u, v) of vertices in G, $(u, v) \in E(G)$ if and only if $(\pi(u), \pi(v)) \in E(H)$. In other words, if G is isomorphic to H, then G can be "redrawn" to look identical to H. If there is an isomorphism between the two graphs G and H, we will say that G and H are isomorphic and denote it by $G \cong H$. The Graph Isomorphism problem (for convenience we will sometimes write GI to denote Graph Isomorphism) is the following algorithmic question: Given two graphs G and H, on n vertices, are the two graphs isomorphic?

The problem is clearly in the complexity class NP, and whether or not Graph Isomorphism has a polynomial-time algorithm is a celebrated open question in the field of algorithmic complexity. Garey and Johnson, in their book on NP-completeness [GJ79], highlight Graph Isomorphism as one of the challenging open problems of the field along with integer factoring.

The computational complexity of Graph Isomorphism has been well-studied for over four decades. Goldreich, Micali and Wigderson [GMW87] proved that the graph non-isomorphism problem has a two-round interactive protocol. Building on that, Boppana, Hastad and Zachos [BHZ87] showed that if coNP-complete problems have constant-round interactive protocols, then the Polynomial-Time Hierarchy (PH) collapses to the second level (Σ_2^p). It follows as a consequence of these results that if GI is NP-complete then the polynomial-time hierarchy collapses to the second level (i.e. $\mathsf{PH} = \Sigma_2^p$). On the other hand, Torán [Tor04] has shown that GI is hard for the class of problems that are NC^1-reducible[1] to computing the determinant of an integer matrix. This is the best hardness result known for Graph Isomorphism. It is an open problem whether Graph Isomorphism is hard for the class P under logspace reductions.

[1] A language A is NC^1-reducible to B if for each n, there is a circuit of size $n^{O(1)}$, depth $O(\log n)$ with AND, OR gates with fan-in 1 and oracle gates for the language B with unbounded fan-in which decides if a string $x \in \{0,1\}^n$ is in A. The size of an oracle gate is the number of wires w that feed into it and contributes $\log w$ to the depth of the circuit.

On the algorithmic side, the fastest known algorithm for testing the isomorphism of two graphs, due to Luks [BL83], runs in time $2^{O(\sqrt{n \log n})}$. This algorithm combines the bounded degree graph isomorphism algorithm of Luks [Luk82] with a degree reduction trick due to Zemlyachenko [ZKT85]. These algorithms use permutation group theory and reduce the isomorphism problem to finding the automorphisms of the graph. Currently known techniques are even unable to give an algorithm with running time $2^{O(n^{1/2-\varepsilon})}$ for general graphs and this remains an open problem. For testing the isomorphism of strongly regular graphs, Spielman [Spi96] gave a deterministic $2^{O(n^{1/3})}$-time algorithm. More recently, Babai et al [BCS+13] have given a deterministic algorithm for the same problem which runs in time $2^{O(n^{1/5})}$.

It is interesting to note that Graph Isomorphism has also been studied in the area of pattern recognition as the graph matching problem; see [LR13] for a recent survey. In graph-based pattern recognition, images are represented as graphs and the problem of checking whether the images are identical essentially amounts to checking if there is a permutation of the vertices of one graph such that the most of edges in the first graph is correctly mapped to edges in the other graph and most of the non-edges in the first graph is correctly mapped to the non-edges in the second graph. Various heuristic methods have been used successfully in this area; see [GXTL10]. This motivates a theoretical study of approximate graph isomorphism in this thesis.

Approximate Graph Isomorphism

We define and study four variants of the *approximate graph isomorphism* problem in this thesis. The basic version of approximate Graph Isomorphism is the following optimization problem: we are given two graphs G and H as input, and the objective is to compute a bijection $\pi : (V) \to V(H)$ such that *most edges* of G are mapped to edges of H by π, and *most non-edges* of G are mapped to non-edges of H. Unlike Graph Isomorphism, for which techniques from permutation group theory turn out to be algorithmically useful, for approximate Graph Isomorphism such techniques do not appear meaningful because approximate isomorphisms from graph G to graph H do not have a group-theoretic structure. We study the following versions of the approximate graph isomorphism problem [AKKV12]:

1. Max-EGI: Given G_1, G_2, find a bijection $\pi : V(G_1) \to V(G_2)$ that maximizes the number of edges of G_1 that are mapped to edges of G_2. Assuming the hardness of an average-case instance of Subgraph Isomorphism, explained in detail in Chapter 2, we show that there is no polynomial-time algorithm that approximates the

optimum better than a factor of $1/2 + \varepsilon$ of the optimum for any ε. The proof, detailed in Chapter 2, is via a reduction that preserves *approximability*. More recently, O'Donnell et al [OWWZ14] have shown that there is no constant factor approximation algorithm for Max-EGI assuming the hardness of the *random 3XOR hypothesis* of Feige [Fei02] which we explain in Section 2.3 of Chapter 2.

2. Max-PGI: Given G_1, G_2, find a bijection $\pi : V(G_1) \to V(G_2)$ that maximizes the number of vertex pairs (u, v) that are mapped correctly. I.e. the number of vertex pairs (u, v) in the graph G_1 such that (u, v) is an edge in G_1 if and only if $(\pi(u), \pi(v))$ is an edge in graph G_2. For this maximization problem, we obtain an α-approximation algorithm that runs in time $n^{O(\log n/\alpha^2)}$ in [AKKV12]. We combine an additive error approximation algorithm for the more general quadratic assignment problem, due to [AFK02], with the observation that a random permutation performs well when the difference in the number of edges in the two graphs is large.

3. Min-EGI: Given G_1, G_2, find a bijection $\pi : V(G_1) \to V(G_2)$ that minimizes the number of edges in G_1 that are mapped to non-edges in G_2. This is the minimization version of Max-EGI. In [AKKV12], we observe that for this problem there is no polynomial-time constant factor approximation algorithm unless P = NP.

4. Min-PGI: Given G_1, G_2, find a bijection $\pi : V(G_1) \to V(G_2)$ that minimizes the total number of vertex pairs (u, v) in G_1 that are incorrectly mapped by π (i.e. edges to non-edges or non-edges to edges). For this problem, we show a weaker result than for Min-EGI that there is no polynomial-time approximation scheme unless P = NP. We also observe that designing a polynomial-time algorithm that approximates the optimum to any constant is at least as hard as GI.

5. In Chapter 2, we also study the four problems mentioned above for the case of vertex-colored graphs. We denote the problems as Max-EGI$_k$, Max-PGI$_k$, Min-EGI$_k$, and Min-PGI$_k$ respectively, where the k indicates that the input graphs G_1 and G_2 are vertex-colored and there are at most k vertices with a particular color. Isomorphism testing of vertex-colored graphs with bounded color-class size is a well-studied special case of graph isomorphism for which polynomial-time algorithms are known [FHL80, Bab79].

   In Chapter 2, we show that the optimization problem for graphs with bounded color-class size is harder to approximate than the general case. Specifically, we show that there is no polynomial-time 0.94- approximation algorithm for Max-EGI$_k$ and Max-PGI$_k$ unless P = NP. For Min-EGI$_k$, we show that there is no polynomial-time constant-factor approximation algorithm, for any constant unless P = NP.

Similarly for Min-PGI$_k$, we show that there is no polynomial-time constant-factor approximation algorithm for any constant assuming the Unique Games Conjecture [Kho02].

1.2 Boolean Isomorphism

Another set of problems that we consider in this thesis concerns the isomorphism problem for Boolean functions, which we will refer to in the rest of the thesis as the *Boolean Isomorphism problem*. The Boolean Isomorphism problem is the following decision problem: Given two Boolean functions $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^n \to \{0,1\}$ on n Boolean variables, test if there is a bijection $\pi : [n] \to [n]$ such that $f(x_{\pi(1)}, \ldots, x_{\pi(n)})$ and $g(x_1, \ldots, x_n)$ are *equivalent* Boolean functions.

Naturally, the computational complexity of this problem depends on the input representation. For instance, when the inputs f and g are given as CNF formulas, then the Boolean Isomorphism problem is coNP-hard since a CNF formula f is unsatisfiable if and only if f is isomorphic to the function g that is 0 on all inputs. When the Boolean functions f and g are given as formulas or circuits, notice that the isomorphism problem is in the complexity class Σ_2^p. For, we can guess the permutation π of the variables with a polynomially bounded existential quantifier and with a polynomially bounded universal quantifier we check if the two functions $f(x_{\pi(1)}, \ldots, x_{\pi(n)})$ and $g(x_1, \ldots, x_n)$ are equivalent.

Agrawal and Thierauf, in [AT96] showed that the Boolean isomorphism problem for circuits is not complete for Σ_2^p unless the polynomial hierarchy collapses to the third level. Their proof is on the lines of the Boppana-Hastad-Zachos argument [BHZ87]. They give an interactive protocol for Boolean Nonisomorphism in which the randomized polynomial-time verifier has oracle access to SAT. Furthermore, Thierauf [Thi00], has studied the Boolean isomorphism problem for various other representations like read-once branching programs, probabilistic branching programs etc.

When the two n-variate Boolean functions f and g are given as truth-tables, then the input size is already $N = 2^{n+1}$. Therefore, even the brute-force algorithm of testing if each permutation $\pi \in S_n$ is an isomorphism between the functions f and g takes only $N^{O(\log \log N)}$ time. A Boolean function $f : \{0,1\}^n \to \{0,1\}$ on n variables can be encoded as a hypergraph $\mathcal{H}(V, \mathcal{E})$ where the vertex set V is the set of the variables of the Boolean function f and a subset $E \subseteq [n]$ is a hyperedge if and only if for the characteristic vector x of the set E, $f(x) = 1$. In [Luk99], Luks has given an algorithm for testing the isomorphism of two hypergraphs on n vertices in time $2^{O(n)}$. Consequently, there is a polynomial-time

algorithm for the isomorphism problem when the functions f and g are given as truth tables.

Since Boolean Isomorphism, when the functions f and g are input in a particular representation, is computationally at least as hard as the unsatisfiability problem for that representation, obtaining a $2^{o(n)}$ time algorithm for Boolean Isomorphism even for 3-CNF formulas is difficult. At this point, we can ask the following two motivating questions:

1. Given two Boolean functions f and g, suppose we are only interested in computing an approximate isomorphism? I.e. a permutation π such that for most inputs $x \in \{0,1\}^n$ we have $f(x_{\pi(1)}, \cdots, x_{\pi(n)}) = g(x_1, \cdots, x_n)$. Specifically, is there a $2^{o(n)}$-time algorithm for computing such an approximate solution when f and g are CNF formulas?

2. Suppose f and g are input in representations for which satisfiability is in polynomial time. For instance, suppose f and g are given 2-CNFs or Horn-CNFs. Is there a $2^{o(n)}$ time algorithm for Boolean Isomorphism in these cases?

In the rest of this chapter we elaborate on these questions and describe our results in this context.

Approximate Boolean Isomorphism

As we described before, given two Boolean functions as circuits or formulas computing them, there is a $2^{O(n)}$ time algorithm known for testing the isomorphism. But even for the case when the functions are given as 3-CNFs, obtaining an algorithm with running time $2^{o(n)}$ for the isomorphism problem appears difficult. In particular, it will contradict the exponential time hypothesis[2] [IPZ01].

Moreover, a hypergraph with m hyperedges on n vertices can be encoded as a disjunction of m terms, where each term corresponds to a hyperedge and is the conjunction of the variables corresponding to the vertices in the hyperedge. Therefore, the isomorphism of monotone DNFs is also at least as hard as Hypergraph Isomorphism. Since there is no algorithm known for Hypergraph Isomorphism which has a running time better than $2^{O(n)}$, a $2^{o(n)}$ time algorithm for Boolean Isomorphism problem for the special class of monotone DNFs, seems difficult.

[2]Let s_k be the infimum over all δ such that there is a $2^{\delta n}$-time algorithm for k-SAT. The Exponential Time Hypothesis (ETH) states that for all k, $s_k > 0$. Sometimes ETH also refers to the weaker statement that there is is no $2^{o(n)}$-time algorithm for k-SAT.

This motivates the question whether it is easier to find *approximate isomorphisms*, which we will now define. Two Boolean functions f and g are said to be ε-close if

$$\Pr_{x\in\{0,1\}^n}[f(x) \neq g(x)] \leq \varepsilon,$$

for $x \in \{0,1\}^n$ picked uniformly at random. In other words, f and g differ in at most $\varepsilon 2^n$ many points in $\{0,1\}^n$. We say that Boolean functions f and g are *ε-approximately isomorphic* if there is a bijection $\pi : [n] \to [n]$ such that $f(x_{\pi(1)}, \ldots, x_{\pi(n)})$ and $g(x_1, \ldots, x_n)$ are ε-close. Since $2^{o(n)}$ time algorithms for Boolean Isomorphism, when f and g are given as circuits, appears difficult to solve, we ask the following question: Given two isomorphic Boolean functions f and g on n variables as formulas or circuits, is there an algorithm for constructing an ε-approximate isomorphism π that runs in time $2^{o(n)}$?

In Chapter 3, one of the problems that we study is approximate Boolean Isomorphism for Boolean functions f and g given by constant depth polynomial-size circuits, also known as AC^0 circuits. More precisely, given isomorphic Boolean functions f and g input by circuits of size s and depth d, we give a randomized $2^{O((\log s)^{O(d)}\sqrt{n})}$ time algorithm to construct an approximate isomorphism. Our algorithm is based on the seminal work of Linial, Mansour and Nisan [LMN93] in which they showed that Boolean functions computable by circuits of size s and depth d have their Fourier spectrum ε-concentrated on sets of poly-logarithmic size. More precisely, they proved the following statement for any function f computable by circuits of size s and depth d:

$$\sum_{|S|\leq(\log(2s/\varepsilon))^d} \hat{f}(S)^2 \leq \varepsilon,$$

where $\hat{f}(S)$ are the Fourier coefficients of f. We use this result and approximate the Boolean function by estimating the Fourier coefficients.

In fact, in Chapter 3 we prove the more general statement that if f and g are two isomorphic Boolean functions, f and g input as oracles, such that $\sum_{|S|\leq t} \hat{f}(S)^2 \leq \varepsilon$ and $\sum_{|S|\leq t} \hat{g}(S)^2 \leq \varepsilon$, then there is a randomized algorithm that runs in time $2^{O((\log n/\delta)^{O(1)}t^{O(1)}\sqrt{n})}$ which constructs an δ-approximate isomorphism, where δ depends on ε.

Exact Boolean Isomorphism

In Chapters 4 and 5, we turn back to the *exact* Boolean isomorphism problem. We study the complexity of Boolean Isomorphism for representations like Horn-CNFs, decision trees and decision lists.

Our aim is to explore representations of Boolean functions for which there are $2^{o(n)}$-time algorithms for Boolean Isomorphism. Observe that given a graph $G(V, E)$ on n vertices $v_1, \ldots, v_n$, we can construct the following 2-DNF f_G on n variables $x_1, \ldots, x_n$: $f_G = \vee_{(v_i,v_j)\in E} x_i \wedge x_j$. This 2-DNF has $|E|$ many minimal satisfying assignments and therefore two graphs G and G' are isomorphic if and only if the Boolean functions represented by the 2-DNFs, f_G and $f_{G'}$ are isomorphic. Hence, we cannot expect to obtain an algorithm for Boolean Isomorphism with running time better than $2^{O(\sqrt{n \log n})}$ as this would give a faster algorithm for testing Graph Isomorphism.

Since for any representation, Boolean Isomorphism is computationally at least as hard as the satisfiability problem for that representation, an interesting question is to compare the complexity of Boolean Isomorphism with SAT for that representation. More specifically, is Boolean Isomorphism reducible to Graph Isomorphism or Hypergraph Isomorphism given oracle access to Boolean satisfiability?

In particular, if a representation has polynomial-time satisfiability algorithm, then is Boolean Isomorphism for that representation polynomial-time reducible to Graph Isomorphism? Examples of such representations are 2-CNFs, Horn-CNFs, decision trees etc. We study the isomorphism problem for these models and also for a generalization of decision trees, called decision lists.

In Chapter 4 of the thesis, we show the following results.

- Given two Boolean functions f and g as decision trees of size s, there is a deterministic algorithm that runs in time $n^{O(\log s)}$ that constructs two hypergraphs $\mathcal{H}_f$ and $\mathcal{H}_g$ on n vertices with the property that each hyperedge is of size $O(\log s)$ such that the Boolean functions f and g are isomorphic if and only if $\mathcal{H}_f$ and $\mathcal{H}_g$ are isomorphic. Using the Hypergraph Isomorphism algorithm of Babai and Codenotti [BC08], for hypergraphs with bounded hyperedge size, this gives an algorithm for testing isomorphism of Boolean functions represented as decision trees in time $2^{O((\log s)^{O(1)}\sqrt{n})}$.

- When the two Boolean functions f and g on n variables are given as Horn-CNFs with m clauses, we give a polynomial-time many-one reduction of the Boolean Isomorphism problem to Graph Isomorphism problem. Further, we show that the Boolean Isomorphism problem when the functions are represented as Horn-CNFs is complete for GI under polynomial-time many-one reductions.

- We then study Boolean Isomorphism when the inputs are represented as decision lists. A decision list is a sequence of tuples $\langle f_1, b_1\rangle, \ldots, \langle f_m, b_m\rangle, \langle \mathbf{1}, b_{m+1}\rangle$, where f_is are Boolean functions and b_is are Boolean constants [Riv87]. The

function value computed by this decision list at an input x is the Boolean value b_i such that $f_j(x) = 0$ for all $j < i$ and $f_i(x) = 1$. Observe that any CNF $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ can be written as the decision list in the following way easily: $\langle \neg C_1, 0\rangle, \langle \neg C_2, 0\rangle, \cdots, \langle \neg C_m, 0\rangle, \langle \mathbf{1}, 0\rangle$. Since the function computed by the decision list $\langle f_1, \neg b_1\rangle, \ldots, \langle f_m, \neg b_m\rangle, \langle \mathbf{1}, \neg b_{m+1}\rangle$ is the complement of the function computed by the decision list $\langle f_1, b_1\rangle, \ldots, \langle f_m, b_m\rangle, \langle \mathbf{1}, b_{m+1}\rangle$, decision lists also encode DNFs. We show that the complexity of the isomorphism problem for decision lists depends on the representation of the f_is by proving a trichotomy theorem for the complexity of the isomorphism problem, similar to the trichotomy theorem proved for the isomorphism problem for constraint satisfaction problem by Böhler et al [BHRV04].

The crucial ingredient in all the results stated above is the notion of a *permutation preserving normal form*. A permutation preserving normal form for f is a representation N_f such that, (i) if g is a Boolean function that is equivalent to f, then N_g is identical to N_f, and (ii) if g is a Boolean function isomorphic to f, then there is a way to rename the variables in N_g such that it becomes identical to N_f. To reduce the Boolean isomorphism problem to graph or Hypergraph Isomorphism, we construct a permutation preserving normal form by using the satisfiability problem for that representation as an oracle and then encode this normal form as a graph or hypergraph. Permutation preserving normal forms were first formally used by Böhler et al [BHRV04] in order to study the isomorphism problem for constraint satisfaction problems.

In all the normal form constructions in Chapter 4, the queries to the satisfiability oracle are themselves functions in the same representation as f and on n variables. Furthermore, the graphs or hypergraphs that are constructed have $O(n)$ vertices. Therefore, if the satisfiability problem for that representation has an algorithm that runs in time $t(n)$, then this gives an algorithm for Boolean Isomorphism that runs in time $O(t(n) + 2^{O(\sqrt{n \log n})})$. Reductions of this form were studied by Impagliazzo, Paturi and Zane [IPZ01], who called these as *Sub-Exponential time Reduction Families*.

It is interesting to ask which representations of Boolean functions have this property. We explore this question in Chapter 5 and show an example where this is true: If f is a k-CNF then there is an $O(\alpha^n + 2^{O(\sqrt{n \log n})})$ time algorithm for testing isomorphism, where α^n is the running time of the satisfiability algorithm for k-CNFs. Schöning [Sch99] has proved that there is an algorithm to test the satisfiability of k-CNFs in time $(2(1-1/k))^n$. Consequently, given two k-CNFs f and g, there is a $(2(1 - 1/k))^n + 2^{O(\sqrt{n \log n})}$ time algorithm to test if f and g are isomorphic, which is many times faster than the $2^{O(n)}$-time algorithm using Luks' [Luk99] hypergraph isomorphism testing algorithm.

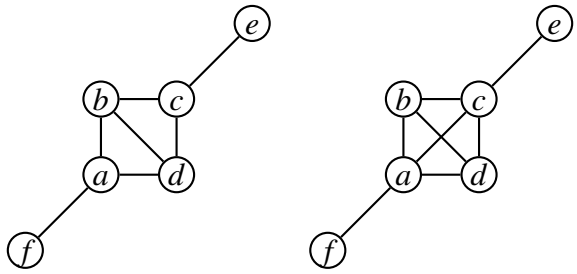

Figure 1.1: Two Hamiltonian-path equivalent graphs that are different, but have the same set of Hamiltonian paths: $f-a-b-d-c-e, f-a-d-b-c-e, e-c-d-b-a-f, e-c-b-d-a-f$

Motivated by this connection between Boolean satisfiability and isomorphism, we define and study a notion of generalized isomorphisms of graphs. We will explain generalized isomorphisms using the example of Hamiltonian paths in graphs. First, we will define a notion of Hamiltonian-path equivalence. Two graphs G and H are Hamiltonian-path equivalent if the set of Hamiltonian paths in the two graphs are identical. It is possible that two graphs are different, yet are Hamiltonian-path equivalent; Figure 1.1 shows an example.

Similarly, we call two graphs Hamiltonian-path isomorphic if there is a bijection between the vertices of the two graphs such that the set of Hamiltonian paths in one graph is mapped by the permutation to the set of Hamiltonian paths in the other graph. For any property $\mathcal{P}$, we can define and the study if there is a faster algorithm for the $\mathcal{P}$-isomorphism problem using the algorithm for testing whether a graph has the property $\mathcal{P}$. This is a generalized notion of isomorphism because if $\mathcal{P}$ is the property that the graph has an edge, then $\mathcal{P}$-isomorphism is precisely the Graph Isomorphism problem.

In Chapter 5, we define the $\mathcal{P}$-isomorphism problem, when (i) $\mathcal{P}$ is the set of graphs with Hamiltonian paths, (ii) when $\mathcal{P}$ is the set of 3-colorable graphs, and (iii) when $\mathcal{P}$ is the set of graphs homomorphic to a fixed graph H.

1.3 Organization of the thesis

We now give a brief chapter-wise outline of the thesis.

Approximate Graph Isomorphism: In Chapter 2 of the thesis we study approximate Graph Isomorphism.

Approximate Boolean Isomorphism We extend the idea of approximate isomorphism to

Boolean functions in Chapter 3. We show that if f and g are two Boolean functions such that the Fourier spectrum of the functions is ε-concentrated on sets of size $t(\varepsilon)$, then there is a randomized algorithm for the approximate Boolean Isomorphism problem which runs in time $2^{O((\log n)^{O(1)} t^{O(1)}\sqrt{n})}$.

Complexity of Boolean Isomorphism In Chapter 4, we study the complexity of exact Boolean Isomorphism. For Horn-CNFs, decision trees and decision lists, we show how the problem is reducible to Hypergraph Isomorphism by constructing permutation preserving normal forms for these representations.

In Chapter 5, we show that for k-CNFs (and more generally k-decision lists), if there is a satisfiability algorithms that runs in time α^n, then there is an algorithm for the Boolean isomorphism problem that runs in time $O(\alpha^n + 2^{O(\sqrt{n \log n})}$. In the final section of the chapter, we will also explore a notion of generalized Graph Isomorphism, motivated by the connection between satisfiability and Boolean Isomorphism.

Chapter 2

Approximate Graph Isomorphism

The Graph Isomorphism problem (GI for short) is the computational problem of deciding, given two graphs G_1 and G_2 on n vertices, if there exists a bijection $\pi\colon V(G_1) \to V(G_2)$ such that $(u, v) \in E_1$ iff $(\pi(u), \pi(v)) \in E_2$. There is no polynomial time algorithm currently known for GI and it is also known that if GI is NP-hard, then $\mathsf{PH} \subseteq \Sigma_2^p$ [GMW87, BHZ87].

Although the fastest known algorithm for the Graph Isomorphism problem has running time $2^{O(\sqrt{n \log n})}$ [BL83, ZKT85], polynomial-time algorithms are known for many interesting subclasses, e.g. bounded degree graphs [Luk82], bounded genus graphs [Mil83], and bounded eigenvalue multiplicity graphs [BGM82].

In this chapter we study a natural optimization problem corresponding to the Graph Isomorphism problem, which we will call the Approximate Graph Isomorphism problem. Intuitively, an approximate isomorphism between two graphs G and H is a permutation $\pi : V(G) \to V(H)$ such that the graphs $\pi(G)$ and H are "close" to each other. To define this problem formally, we need a suitable definition of distance between graphs. We study different versions of the approximate isomorphism problem depending on two notions of distance which we will describe formally in the next section. The main motivation for this study is to explore if approximate isomorphisms can be computed efficiently, given that the best known algorithm for computing exact isomorphisms has running time $2^{O(\sqrt{n \log n})}$. To the best of our knowledge, the only previous theoretical study of Approximate Graph Isomorphism is the work of Arora, Frieze and Kaplan [AFK02]. More recently, there has been work by O'Donnell et al [OWWZ14] who study a version of the Approximate Graph Isomorphism problem, that they call *Robust Graph Isomorphism*. Approximate Graph Isomorphism has also been studied in the setting of property testing [FM06] and the testing algorithm uses ideas similar to [AFK02].

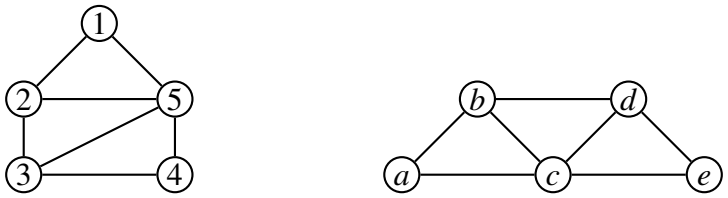

Figure 2.1: Two isomorphic graphs G and H

2.1 Preliminaries

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two input graphs on n of vertices. The *pairwise distance* between G_1 and G_2, denoted by $d_p(G_1, G_2)$, is defined as $|E(G_1) \triangle E(G_2)|$. This is set of edges that are present in either G_1 or G_2 but not in both. Similarly, the *edgewise distance* between G_1 and G_2, denoted by $d_e(G_1, G_2)$, is defined as $|E(G_1) \setminus E(G_2)|$. This quantity is the set of edges in G_1 that are not present in G_2. We study the complexity of the following optimization problems:

- Max-EGI: Given G_1, G_2, find a bijection $\pi\colon V_1 \to V_2$ that maximizes the number of matched edges, i.e., $\overline{d}_e(G_1^\pi, G_2) = |E_1| - d_e(G_1^\pi, G_2)$.
- Max-PGI: Given G_1, G_2, find a bijection $\pi\colon V_1 \to V_2$ that maximizes matched vertex pairs, i.e., $\overline{d}_p(G_1^\pi, G_2) = \binom{n}{2} - d_p(G_1^\pi, G_2)$.
- Min-EGI: Given G_1, G_2, find a bijection $\pi\colon V_1 \to V_2$ that minimizes mismatched edges, $d_e(G_1^\pi, G_2)$.
- Min-PGI: Given G_1, G_2, find a bijection $\pi\colon V_1 \to V_2$ that minimizes mismatched pairs, $d_p(G_1^\pi, G_2)$.

Example 2.1.1. *Figure 2.1 shows two graphs G and H that are isomorphic under the permutation $\pi = \left(\begin{smallmatrix}1&2&3&4&5\\a&b&d&e&c\end{smallmatrix}\right)$. For this permutation π, $d_p(G^\pi, H) = d_e(G^\pi, H) = 0$ while $\overline{d}_p(G^\pi, H) = 10$ and $\overline{d}_e(G^\pi, H) = 7$.*

The permutation $\sigma = \left(\begin{smallmatrix}1&2&3&4&5\\a&b&c&d&e\end{smallmatrix}\right)$ is not an isomorphism since $(2, 5)$ is an edge in G whereas (b, e) is not an edge in H. This is an approximate isomorphism and the distance between the graphs under this approximate isomorphism is as follows: $d_p(G^\pi, H) = 4$ and $d_e(G^\pi, H) = 2$ while $\overline{d}_p(G^\pi, H) = 6$ and $\overline{d}_e(G^\pi, H) = 5$.

We first recall the notion of an α-approximation algorithm for an optimization problem. We call an algorithm $\mathcal{A}$ for a maximization problem an α-approximation algorithm, where

$\alpha < 1$, if given an instance $\mathcal{I}$ of the problem with an optimum $\mathsf{OPT}(\mathcal{I})$, $\mathcal{A}$ outputs a solution with value $\mathcal{A}(\mathcal{I})$ such that $\mathcal{A}(\mathcal{I}) \geq \alpha\mathsf{OPT}(\mathcal{I})$. Similarly, for a minimization problem, we say $\mathcal{B}$ is a β-approximation algorithm for $\beta > 1$, if for any instance $\mathcal{I}$ of the problem with an optimum $\mathsf{OPT}(\mathcal{I})$, $\mathcal{B}$ outputs a solution with value $\mathcal{B}(\mathcal{I})$ such that $\mathcal{B}(\mathcal{I}) \leq \beta\mathsf{OPT}(\mathcal{I})$.

Arora, Frieze and Kaplan, in [AFK02], studied the maximization problem Max-PGI as an instance of the quadratic assignment problem. They formulate Max-PGI as an instance of a quadratic assignment problem. Given a pair of n-vertex isomorphic graphs their algorithm, based on randomized rounding, computes a permutation that has *additive error* εn^2 and runs in time $n^{O(\log n/\varepsilon^2)}$. The problem Max-EGI can also be viewed as an optimization variant of Subgraph Isomorphism. Very recently O'Donnell et al [OWWZ14] have studied this problem as *Robust Graph Isomorphism*.

Observe that $\overline{d}_p(G_1^\pi, G_2) + d_p(G_1, G_2) = \binom{n}{2}$ and $\overline{d}_e(G_1^\pi, G_2) + d_e(G_1^\pi, G_2) = |E_1|$. Thus solving one of the maximization problems with additive error is equivalent to solving the corresponding minimization problem with the same additive error. However, the minimization problems behave differently for multiplicative factor approximations, so we study them separately.

A natural restriction of the Graph Isomorphism problem is to vertex-colored graphs (G_1, G_2) where $V(G_1) = C_1 \cup C_2 \cup \cdots \cup C_m$ and $V(G_2) = C_1' \cup C_2' \cup \cdots \cup C_m'$, and C_i, C_i' contain the vertices of G_1 and G_2, respectively, that are colored i. The problem is to compute a color-preserving isomorphism π between G_1 and G_2, i.e., an isomorphism π such that for any vertex u, both u and $\pi(u)$ have the same color. The bounded color-class version GI_k of the Graph Isomorphism problem consists of instances such that $|C_i| = |C_i'| \leq k$ for all i. For GI_k, randomized [Bab79] and deterministic [FHL80] polynomial time algorithms are known.

It is, therefore, natural to study the optimization problems defined above in the setting of vertex-colored graphs where the objective function is optimized over all color-preserving bijections $\pi\colon V_1 \to V_2$. We denote these problems as $\mathsf{Max\text{-}PGI}_k$, $\mathsf{Max\text{-}EGI}_k$, $\mathsf{Min\text{-}PGI}_k$ and $\mathsf{Min\text{-}EGI}_k$, where k is a bound on the number of vertices having the same color.

2.1.1 Outline of the Chapter

We now give an outline of the theorems that we prove in this chapter.

Section 2.2 deals with the maximization problems Max-PGI and Max-EGI. The first

theorem that will be proved in this chapter is the following:

Theorem 2.1.2. *For any constant $\alpha < 1$, there is an α-approximation algorithm for* Max-PGI *running in time $n^{O(\log n/(1-\alpha)^4)}$.*

The proof uses a result of Arora, Frieze and Kaplan [AFK02] which gives an additive error approximation algorithms for a quadratic assignment problem based on randomized rounding. Among the various problems they study, they also observe that approximate graph isomorphisms between n vertex graphs can be computed up to *additive error* εn^2 in time $n^{O(\log n/\varepsilon^2)}$. We show that this algorithm can be modified to obtain a multiplicative error approximation scheme for the problem. We obtain the α-approximation algorithm for Max-PGI by combining the $n^{O(\log n)}$ time additive error algorithm of [AFK02] with a simple averaging algorithm.

Next we consider the Max-EGI problem. Langberg et al. [LRS06] proved that there is no polynomial-time $(1/2 + \varepsilon)$-approximation algorithm for the Maximum Graph Homomorphism problem (see Definition 2.2.5) for any constant $\varepsilon > 0$ assuming average-case hardness for a version of the Subgraph Isomorphism problem [Fei02]. We give a factor-preserving reduction from the Maximum Graph Homomorphism problem to Max-EGI thus obtaining the following result.

Theorem 2.1.3. *There is no $(\frac{1}{2} + \varepsilon)$-approximation algorithm for* Max-EGI *for any constant $\varepsilon > 0$ assuming the hardness for the refutation problem for Subgraph Isomorphism (Definition 2.2.6).*

The paper of O'Donnell et al [OWWZ14], published recently have improved this result to state that there is no constant factor approximation algorithm for Max-EGI under the random 3XOR hypothesis of Feige [Fei02].

Unlike in the case of GI_k, where polynomial time algorithms are known [Bab79, FHL80, Luk86], we show that in the optimization setting these problems are computationally harder. We prove the following theorem by giving a factor-preserving reduction from Max-2Lin-2 (e.g. see [KKMO04]) to Max-PGI$_k$ and Max-EGI$_k$. The input to the problem Max-2Lin-2 is a set of equations on two variables over $\mathbb{GF}(2)$ and the goal is to maximize the number of equations that are satisfied.

Theorem 2.1.4. *For any $k \geq 2$,* Max-PGI$_k$ *and* Max-EGI$_k$ *are* NP*-hard to approximate beyond a factor of* 0.94.

Since, assuming the Unique Games Conjecture (UGC for short) of Khot [Kho02], it is NP-hard to approximate Max-2Lin-2 beyond a factor of 0.878 [KKMO04], the same bound

holds under UGC for Max-PGI$_k$ and Max-EGI$_k$ by the same reduction. Since Max-PGI$_k$ and Max-EGI$_k$ are easily seen to be instances of generalized 2CSP, they have constant factor approximation algorithms, for a constant factor depending on k. In fact, it turns out that Max-EGI$_2$ and Max-PGI$_2$ are tightly classified by Max-2Lin-2 with almost matching upper and lower bounds (details are given in Section 2.2). However, we do not know of similar gap-preserving reductions from general unique games (with alphabet size more than 2) to Max-PGI$_k$ or Max-EGI$_k$ for any k.

In Section 2.3, we study the corresponding minimization problems for graphs and vertex-colored graphs with bounded color-class size. The main results of that section show that the complexity of Min-PGI and Min-EGI is significantly different from Max-PGI and Max-EGI.

Theorem 2.1.5. *There is no polynomial-time approximation algorithm for* Min-PGI *with any multiplicative approximation guarantee unless* GI $\in$ P.

Theorem 2.1.6. *There is no α-approximation algorithm for* Min-PGI *that runs in time polynomial in n and $1/\alpha$ unless* P = NP.

Theorem 2.1.7. *There is no polynomial-time approximation algorithm for* Min-EGI *with any multiplicative approximation guarantee unless* P = NP.

In the case of Min-PGI$_k$ and Min-EGI$_k$, we prove that Min-PGI$_k$ is as hard as the minimization version of Max-2Lin-2, known in literature as the Min-Uncut problem, and that Min-EGI$_4$ is inapproximable for any constant factor unless P = NP by reducing the Nearest Codeword Problem (NCP) to it.

Remark. We remark that there does not seem to be any coset structure to the set of approximate isomorphisms that we can exploit to obtain approximate isomorphism algorithms. For instance, it is not clear how to define a notion of an approximate automorphism. If we were to define an approximate automorphism as a permutation π of vertices such that $d_e(G^\pi, G) = k$, for some number k, then the graph G in Figure 2.1 shows a simple counter-example. Here the permutation $\pi = \left(\begin{smallmatrix}1&2&3&4&5\\1&3&2&4&5\end{smallmatrix}\right)$ has $d_e(G^\pi, G) = 1$ and $\sigma = \left(\begin{smallmatrix}1&2&3&4&5\\4&5&2&3&1\end{smallmatrix}\right)$ has $d_e(G^\sigma, G) = 1$. But $\pi \circ \sigma$ is the permutation $\left(\begin{smallmatrix}1&2&3&4&5\\4&2&5&3&1\end{smallmatrix}\right)$ which has $d_2(G^{\pi\circ\sigma}, G) = 2$. This shows that composing approximate isomorphism can increase the distance between the graphs. Hence, we do not believe that the group-theoretic techniques which were used in the algorithms for exact Graph Isomorphism are useful in the setting of approximate isomorphisms.

2.2 Maximizing the number of matches

We first observe that computing optimal solutions to Max-PGI is NP-hard via a reduction from CLIQUE.

Proposition 2.2.1. *Computing optimal solutions to* Max-PGI *instances is* NP*-hard.*

Proof. Let (G, k) be an instance of the CLIQUE problem. Define the graphs $G_1 = G$ and $G_2 = K_k \cup \overline{K}_{n-k}$, i.e., a k-clique and $n - k$ isolated vertices. Let π_{opt} be a bijection that achieves the optimum value for this Max-PGI instance. Then G has a k-clique if and only if $\overline{d}_p(G^{\pi_{opt}}, G_2) = \binom{n}{2} - |E_G| + \binom{k}{2}$. □

Next we give a general method for combining an additive error approximation algorithm for Max-PGI with a simple averaging approximation algorithm to design an α-approximation algorithm for Max-PGI for any constant $\alpha < 1$.

Lemma 2.2.2. *Suppose $\mathcal{A}$ is an algorithm such that for any $\varepsilon > 0$, given a* Max-PGI *instance in form of two n-vertex graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, computes a bijection $\pi\colon V_1 \to V_2$ such that $\overline{d}_p(G_1^{\pi}, G_2) \geq \mathsf{OPT} - \varepsilon n^2$ in time $T(n, \varepsilon)$. Then there is an algorithm that computes for each $\alpha < 1$ an α-approximate solution for any* Max-PGI *instance (G_1, G_2) in time $O(T(n, (1-\alpha)^2/9) + n^3)$.*

Proof. Without loss of generality we can assume $V_1 = V_2 = [n]$. We denote the number of edges in G_i by t_i and the number of non-edges by $\bar{t}_i$. Observe that the optimum for Max-PGI satisfies $\mathsf{OPT} \leq t_1 + \bar{t}_2$ since the best permutation is an isomorphism. Let $\pi\colon [n] \to [n]$ be a permutation chosen uniformly at random. Then, the following calculation shows that the expected number s of matched pairs is

$$s = \frac{t_1 t_2 + \bar{t}_1 \bar{t}_2}{\binom{n}{2}} = \frac{\binom{n}{2} - \bar{t}_2}{\binom{n}{2}} t_1 + \frac{\bar{t}_2}{\binom{n}{2}} \left(\binom{n}{2} - t_1 \right) = t_1 + \bar{t}_2 - \frac{2 t_1 \bar{t}_2}{\binom{n}{2}}.$$

We can deterministically compute a permutation σ such that $\overline{d}_p(G_1^{\sigma}, G_2) \geq s$ using the method of conditional probabilities; we defer this detail to the end of the proof. We now show how this can be combined with the additive error approximation algorithm $\mathcal{A}$ for Max-PGI to obtain an α-approximation algorithm for Max-PGI. This combined algorithm distinguishes two cases based on the number of edges and non-edges in G_1 and G_2, respectively.

Case 1 ($\min\{t_1, \bar{t}_2\} \leq (1-\alpha)\binom{n}{2}/2$): In this case we compute a permutation σ with $\overline{d}_p(G_1^\sigma, G_2) \geq s$. Since

$$t_1\bar{t}_2 = \max\{t_1, \bar{t}_2\}\min\{t_1, \bar{t}_2\} \leq \left(t_1 + \bar{t}_2\right)(1-\alpha)\binom{n}{2}/2,$$

it follows that

$$t_1 + \bar{t}_2 - 2t_1\bar{t}_2/\binom{n}{2} \geq \alpha\left(t_1 + \bar{t}_2\right) \geq \alpha\mathsf{OPT}.$$

Case 2 ($\min\{t_1, \bar{t}_2\} > (1-\alpha)\binom{n}{2}/2$): In this case we use algorithm $\mathcal{A}$ with $\varepsilon = (1-\alpha)^2/9$ to obtain a permutation π with $\overline{d}_p(G_1^\pi, G_2) \geq \mathsf{OPT} - \varepsilon n^2$. Since $t_1 + \bar{t}_2 + \bar{t}_1 + t_2 = 2\binom{n}{2}$, either $t_1 + \bar{t}_2 \leq \binom{n}{2}$ or $\bar{t}_1 + t_2 \leq \binom{n}{2}$. Without loss of generality assume $t_1 + \bar{t}_2 \leq \binom{n}{2}$ (otherwise we interchange G_1 and G_2), implying that either $t_1 \leq \binom{n}{2}/2$ or $\bar{t}_2 \leq \binom{n}{2}/2$. Further, since the expected value $\mathbb{E}_\sigma\left[\overline{d}_p(G_1^\sigma, G_2)\right] = t_1 + \bar{t}_2 - 2t_1\bar{t}_2/\binom{n}{2}$, it follows that for sufficiently large n,

$$\mathsf{OPT} \geq t_1 - t_1\bar{t}_2/\binom{n}{2} + \bar{t}_2 - t_1\bar{t}_2/\binom{n}{2} \geq \frac{\min\{t_1, \bar{t}_2\}}{2} > \frac{1-\alpha}{4}\binom{n}{2} \geq \frac{\varepsilon n^2}{1-\alpha}.$$

Hence, $\overline{d}_p(G_1^\pi, G_2) \geq \mathsf{OPT} - \varepsilon n^2 \geq \alpha\mathsf{OPT}$.

It remains to show how a permutation which achieves at least the expected number s of matched pairs can be computed deterministically. Suppose that $\sigma\colon [i] \to [n]$ is a *partial permutation*. Let $\pi\colon [n] \to [n]$ be a random permutation that extends σ, i.e., $\pi(j) = \sigma(j)$ for $j \in [i]$. Let $s(\sigma)$ denote the expected number of matched pairs over random permutations π that extend σ. It is easy to see that we can compute $s(\sigma)$ in polynomial time. We do this by counting the pairs in three parts: (a) pairs with both end points in $[i]$, (b) pairs with both end points in $[n] \setminus [i]$, and (c) pairs with one end point in $[i]$ and the other in $[n] \setminus [i]$. Matched pairs of type (a) depend only on σ and can be counted straightaway. The expected number of matched pairs of type (b) is computed exactly as s above (since π restricted on $[n] \setminus [i]$ is random). The expected number of matched pairs of type (c) is given by $\sum_{j\in[i]} \frac{n_j n_{\sigma(j)} + (n-i-n_j)(n-i-n_{\sigma(j)})}{n-i}$, where n_j is the number of neighbors of j in the graph G_1 contained in $[n] \setminus [i]$ and $n_{\sigma(j)}$ is the number of neighbors of $\sigma(j)$ in the graph G_2 contained in $[n] \setminus \{\sigma(l) \mid l \in [i]\}$. The entire computation of $s(\sigma)$ takes $O(n^2)$ time.

Now, for $k \in [n] \setminus \{\sigma(l) \mid l \in [i]\}$, let $\sigma_k\colon [i+1] \to [n]$ denote the extension of σ by setting $\sigma(i+1) = k$. Since a random extension π of σ can map $i+1$ uniformly to any $k \in [n] \setminus \{\sigma(l) \mid l \in [i]\}$ it follows that

$$s(\sigma) = \frac{1}{n-i}\sum_k s(\sigma_k),$$

where the summation is over all $k \in [n] \setminus \{\sigma(l) \mid l \in [i]\}$.

Furthermore, each $s(\sigma_k)$ is efficiently computable, as explained above. Reusing partial computations, we can find k such that $s(\sigma_k) \geq s(\sigma)$ in time $O(n^2)$. Continuing thus, when we fix the permutation on all of $[n]$ we obtain a σ with $\overline{d}_p(G_1^\sigma, G_2) \geq s$ in $O(n^3)$ time. □

Note that any polynomial time additive ε-error algorithm for Max-PGI, i.e., an algorithm running in time $n^{\text{poly}(1/\varepsilon)}$ with an additive error $\leq \varepsilon n^2$, gives a polynomial time α-approximation algorithm for Max-PGI running in time $n^{\text{poly}(1/(1-\alpha))}$.

To complete the proof of Theorem 2.1.2, we formulate Max-PGI as an instance of a quadratic optimization problem called the Quadratic Assignment Problem (QAP for short) as was done in [AFK02] and use an additive error approximation algorithm for the Quadratic Assignment Problem due to Arora, Frieze and Kaplan [AFK02].

Quadratic Assignment Problem. The quadratic assignment problem is an assignment problem where the objective is to maximize a quadratic polynomial under linear constraints. Given $\{c_{ijkl}\}_{1 \leq i,j,k,l \leq n}$, the maximization version of the *quadratic assignment problem* is to find an $n \times n$ permutation matrix $x = (x_{ij})$ that maximizes $val(x) = \sum_{i,j,k,l} c_{ijkl} x_{ij} x_{kl}$. The minimization version of the problem is to find an $n \times n$ permutation matrix $x = (x_{i,j})$ that minimizes $val(x) = \sum_{i,j,k,l} c_{ijkl} x_{ij} x_{kl}$.

The quadratic assignment problem generalizes many combinatorial optimization problems like maximum acyclic subgraph, betweenness etc. [AFK02]. We now give a simple proposition from [AFK02] regarding the approximability of the quadratic assignment problem.

Proposition 2.2.3. *There is no α-approximation algorithm for the minimization version of the quadratic assignment problem for any $\alpha < 1$ unless* P = NP.

Proof. We will reduce the Hamiltonian cycle problem to the minimization version of the quadratic assignment problem. Let $G(V, E)$ be an instance of the Hamiltonian cycle problem. Define the instance of quadratic assignment problem where $c_{ijkl} = 1$ if and only if $k = i + 1$ and $(j, l) \notin E$. The variable $x_{i,j}$ is an indicator variable for the fact that the i^{th} vertex in the Hamiltonian path is j. This minimum of the objective function $\sum_{i,j,k,l} c_{ijkl} x_{ij} x_{kl}$ is 0 if and only if the graph has a Hamiltonian cycle. Thus, if there is an α-approximation algorithm for the quadratic assignment problem for any $\alpha > 1$, then there is a polynomial-time algorithm for Hamiltonian cycle. □

Arora, Frieze and Kaplan in [AFK02] give a general quasipolynomial-time algorithm for QAP with an additive error. Formally, they prove the following theorem.

Theorem 2.2.4 ([AFK02]). *There is an algorithm that, given an instance of* QAP *where each of the c_{ijkl} is bounded in absolute value by a constant c and given an ε, finds an assignment to x_{ij} such that $val(x) \geq val(x^*) - \varepsilon n^2$ where x^* is the assignment which attains the optimum. The algorithm runs in time $n^{O(c^2 \log n/\varepsilon^2)}$.*

An instance of Max-PGI consisting of graphs $G_1 = ([n], E_1)$ and $G_2 = ([n], E_2)$ can be naturally expressed as a QAP instance by setting

$$c_{ijkl} = \begin{cases} 1 & \text{if } (i,k) \in E_1 \text{ and } (j,l) \in E_2 \text{ or } (i,k) \notin E_1 \text{ and } (j,l) \notin E_2 \\ 0 & \text{otherwise.} \end{cases}$$

This ensures that $val(x) = \overline{d}_p(G_1^{\pi_x}, G_2)$ for all permutation matrices x with corresponding permutation π_x; in particular, the optimum solutions of the Max-PGI and QAP instances achieve the same value.

Thus for the Max-PGI problem, using Theorem 2.2.4 we can find a permutation π such that $\overline{d}_p(G_1^{\pi}, G_2) \geq \mathsf{OPT} - \varepsilon n^2$ in time $n^{O(\log n/\varepsilon^2)}$. Combining this with Lemma 2.2.2, we get an α-approximation algorithm for Max-PGI running in time $n^{O(\log n/(1-\alpha)^4)}$ and this completes the proof of Theorem 2.1.2.

In contrast to the quasipolynomial-time approximation scheme for Max-PGI, we now show that Max-EGI is likely to be $(\frac{1}{2} + \varepsilon)$-hard to approximate. To this end, we define the Maximum Graph Homomorphism problem (MGH) first studied in [LRS06].

Definition 2.2.5 (Maximum Graph Homomorphism Problem). *Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, compute a mapping $\phi\colon V_1 \to V_2$ that maximizes the cardinality of the set $\{(u,v) \in E_1 \mid (\phi(u), \phi(v)) \in E_2\}$.*

Langberg et al. [LRS06] proved that MGH is hard to approximate beyond a factor of $1/2 + \varepsilon$ assuming the hardness of certain average case instances of Subgraph Isomorphism. We now define the average case instance of Subgraph Isomorphism that is studied in [LRS06], which they refer to as the refutation problem for Subgraph Isomorphism. We note that this assumption is a rather weak assumption as the computational hardness of this average-case instance is not known, although Subgraph Isomorphism is an NP-complete problem. The work in [LRS06] is along the lines of the work of Feige [Fei02] who used average-case hardness assumptions to prove hardness of approximating optimization problems.

Let $\triangle_n$ be the set of all triangle free graphs on n vertices. Let $\triangle_{n,p}$ be the distribution obtained by choosing a random graph $G \in \mathcal{G}_{n,p}$ and then considering edges in some order and deleting any edge that is part of a triangle.

Definition 2.2.6 (Refutation problem for Subgraph Isomorphism, [LRS06])**.** *Given a parameter $c > 0$ and a pair of graphs $G \in \triangle_{n,p}$, $H \in \triangle_{n,p'}$, where $p = c \ln n/n$ and $p' \gg p$, find an algorithm that (a) returns "yes" if H contains G as a subgraph and (b) return "no" on most instances, i.e. with probability at least $1/2$, the algorithm return "no".*

To prove Theorem 2.1.3, we give a factor-preserving reduction from MGH to Max-EGI.

Lemma 2.2.7. *There is a polynomial-time algorithm that for a given* MGH *instance $\mathcal{I}$, constructs a* Max-EGI *instance $\mathcal{I}'$ with* $\mathsf{OPT}(\mathcal{I}) = \mathsf{OPT}(\mathcal{I}')$.

Proof. Given an MGH instance $\mathcal{I} = (G_1, G_2)$, we construct the Max-EGI instance $\mathcal{I}' = (G_1', G_2')$ as follows. The graphs G_1' and G_2' both have vertex set $V_1 \times V_2$. For each edge (u_1, v_1) in the graph G_1, we put a single edge between the vertices (u_1, w_2) and (v_1, w_2) in E_1', where w_2 is an arbitrary but fixed vertex in V_2, and for each edge (u_2, v_2) in the graph G_2, we put all $|V_1|^2$ edges between $V_1 \times \{u_2\}$ and $V_1 \times \{v_2\}$ in E_2'. It suffices to prove the following claim.

Claim. There is a mapping $\phi\colon V_1 \to V_2$ such that $\left|\{(u,v) \in E_1 \mid (\phi(u), \phi(v)) \in E_2\}\right| = k$ if and only if there is a permutation $\pi\colon V_1 \times V_2 \to V_1 \times V_2$ such that $\left|\left\{(u,v) \in E_1' \mid (\pi(u), \pi(v)) \in E_2'\right\}\right| = k$.

Given the mapping ϕ, we construct the permutation π as follows: For each $u_1 \in V_1$, π maps the vertex (u_1, w_2) of G_1' to the vertex $(u_1, \phi(u_1))$ in G_2'. The remaining $|V_1| \cdot |V_2| - |V_1|$ vertices of G_1' are mapped arbitrarily.

Then each edge $(u_1, v_1) \in E_1$ is satisfied by ϕ if and only if the corresponding edge between (u_1, w_2) and (v_1, w_2) in E_1' is satisfied by π. This follows from the fact that $(\phi(u_1), \phi(v_1)) \in E_2$ if and only there is an edge between $(u_1, \phi(u_1))$ and $(v_1, \phi(v_1))$ in E_2'.

Similarly, given a permutation π between G_1' and G_2', we can obtain a mapping $\phi\colon V_1 \to V_2$ achieving the same number of matched edges by letting $\phi(u_1) = v_2$, where v_2 is the second component of the vertex $\pi(u_1, w_2)$. □

Unlike in the case of Max-PGI, we observe that there cannot be constant factor approximation algorithms for Max-PGI$_k$ for all constants. This is in interesting contrast to the fact that GI for graphs with bounded color-class size is in P. We now prove the hardness of approximating Max-PGI$_k$ and Max-EGI$_k$ for any $k \geq 2$.

We prove the hardness by exhibiting a factor-preserving reduction from Max-2Lin-2, which is hard to approximate above a guarantee of 0.94 unless P = NP [Hås01]. The problem of Max-2Lin-2 can be stated thus

Problem 2.2.8. *Given a set $E \subseteq \{x_i + x_j = b \mid i, j \in [n], b \in \{0,1\}\}$ of m equations over $\mathbb{F}_2$, find an assignment to the variables $x_1, \ldots, x_n$ that maximizes the number of equations satisfied.*

The following lemma proves the factor-preserving reduction from Max-2Lin-2 to Max-PGI_k. The proof for Max-EGI_k is similar.

Lemma 2.2.9. *For any $k \geq 2$, there is a polynomial-time algorithm that for a given* Max-2Lin-2 *instance $\mathcal{I}$ constructs a* Max-PGI_{2k} *instance $\mathcal{I}'$ such that $\mathsf{OPT}(\mathcal{I}') = (2k)^2\mathsf{OPT}(\mathcal{I})$.*

Proof. Let $E \subseteq \{x_i + x_j = b \mid i, j \in [n], b \in \{0,1\}\}$ be the equations of $\mathcal{I}$. As a first step, if there is a pair of equations $x_i + x_j = 1$ and $x_i + x_j = 0$ in E, remove both these equations and add a new equation $y_i + y_j = 1$ on two new variables y_i and y_j. Let E' be the new set of equations obtained. Notice that $\mathsf{OPT}(E) = \mathsf{OPT}(E')$. We now describe the construction of the instance $\mathcal{I}'$ of Max-PGI_{2k}. For each variable x_i, put two sets of vertices V_i^0 and V_i^1 with k vertices each of color i. Let $x_l + x_m = b$ be an equation in E'. In the graph G_1, add a complete bipartite graph between V_l^0 and V_m^0 and another complete bipartite graph between V_l^1 and V_m^1. Similarly, add the complete bipartite graph between V_l^0 and V_m^b and between V_l^1 and $V_m^{1\oplus b}$ in G_2. If there is no equation in E' connecting the variables x_l and x_m, add a complete bipartite graph between the color classes l and m in G_1 and the empty graph between l and m in G_2. Similarly, make all color classes cliques in G_1 and independent sets in G_2. The idea is that assigning $x_i \mapsto 0$ corresponds to mapping V_i^0 and V_i^1 to themselves, respectively, while assigning $x_i \mapsto 1$ corresponds to mapping V_i^0 to V_i^1 and vice versa.

Given an assignment $\sigma\colon [n] \to \{0,1\}$ that satisfies t of the equations in E, let π_σ be the permutation that maps the j^{th} vertex in V_i^b to the j^{th} vertex in $V_i^{b\oplus\sigma(i)}$. For each satisfied equation $x_i + x_j = b$, this guarantees that all $(2k)^2$ pairs in $(V_i^0 \cup V_i^1) \times (V_j^0 \cup V_j^1)$ are matched. Thus $\mathsf{OPT}(\mathcal{I}') \geq \overline{d}_p(G_1^{\pi_\sigma}, G_2) = (2k)^2 t$.

To prove the converse, let $\pi\colon [n] \to [n]$ be a permutation with $\overline{d}_p(G_1^\pi, G_2) = t$. Define f_i as the number of vertices in V_i^0 that are mapped to V_i^1 by π (it is also the number of vertices in V_i^1 mapped to V_i^0). If $f_i \in \{0, k\}$ for all i, it is straightforward to reverse the above construction, obtaining an assignment that satisfies $\overline{d}_p(G_1^\pi, G_2)/(2k)^2$ equations.

If there is an i with $f_i \notin \{0, k\}$, then the number of matched pairs between color classes i and j is given by the sum $4\left[(k - f_j)f_i + (k - f_i)f_j\right]$. Define $\overline{d}_p^{\,i}(G_1^\pi, G_2)$ as

$$\overline{d}_p^{\,i}(G_1^\pi, G_2) = \sum_j 4\left[(k - f_j)f_i + (k - f_i)f_j\right].$$

This can be re-written as

$$\overline{d}_p^i(G_1^\pi, G_2) = k^2 \sum_j 4\left[\left(1 - \frac{f_j}{k}\right)\frac{f_i}{k} + \left(1 - \frac{f_i}{k}\right)\frac{f_j}{k}\right].$$

Let $mp_i'(\pi) = (1/k^2)\overline{d}_p^i(G_1^\pi, G_2) = \sum_j 4\left[(1-f_j')f_i' + (1-f_i')f_j'\right]$ where $f_i' = f_i/k$ and $f_j' = f_j/k$.

Define $\pi_{i,b}$ (for $b \in \{0,1\}$) as the permutation that maps the j^{th} vertex of $V_i^{b'}$ to the j^{th} vertex of $V_i^{b\oplus b'}$, and that acts like π on all other color classes. Thus, $\overline{d}_p^i(G_1^{\pi_{i,0}}, G_2) = 4\sum_j f_j'$ and $\overline{d}_p^i(G_1^{\pi_{i,1}}, G_2) = 4\sum_j(1-f_j')$.

Since $mp_i'(\pi)$ is a convex combination of $\overline{d}_p^i(G_1^{\pi_{i,0}}, G_2)$ and $\overline{d}_p^i(G_1^{\pi_{i,1}}, G_2)$, one of the two must be at least as large as $mp_i'(\pi)$. Replace π by that permutation, and repeat this process until $f_i \in \{0, k\}$ for all i. □

This construction still works if we replace $\overline{d}_p(G_1^\pi, G_2)$ with $\overline{d}_e(G_1^\pi, G_2)$, as for all equations $x_i + x_j = b$ in E, exactly half of the possible edges between color classes i and j are present. It follows that there is a factor-preserving reduction from Max-2Lin-2 to Max-EGI$_{2k}$.

Lemma 2.2.10. *For any $k \geq 2$, there is a polynomial-time algorithm that for a given* Max-2Lin-2 *instance $\mathcal{I}$ constructs a* Max-EGI$_{2k}$ *instance $\mathcal{I}'$ such that* $\mathsf{OPT}(\mathcal{I}') = 2k^2\mathsf{OPT}(\mathcal{I})$.

Since there is no α-approximation algorithm for Max-2Lin-2 for $\alpha > 0.94$ unless P = NP [Hås01], Lemmas 2.2.9 and 2.2.10 complete the proof of Theorem 2.1.4 that there is no α-approximation algorithm for Max-PGI$_k$ and Max-EGI$_k$ for $\alpha > 0.94$ unless P = NP.

We now exhibit a simple reduction from the Max-PGI$_2$ problem to Max-2CSP(2) problem. The same method can be used to reduce the Max-PGI$_k$ problem to the more general Max-2CSP(q), where $q = k!$. The Max-2CSP(q) is a constraint satisfaction problem where the input is a set of constraints of arity two and the domain is a set of values from $\{1, \cdots, q\}$. Guruswami and Raghavendra [GR08] proved that the there is a constant factor approximation algorithm for the Max-2CSP(q) problem where the approximation guarantee depends on q.

Lemma 2.2.11. *There is a polynomial-time algorithm that for two given vertex-colored graphs G_1 and G_2 where each color class has size at most 2, outputs a* Max-2CSP(2) *instance $\mathcal{F} = \{f_1, \ldots, f_m\}$ where $m = |E(G_1)|$ and $f_i \colon \{0,1\}^2 \to \{0,1\}$ such that there is a color-preserving bijection $\pi \colon V(G_1) \to V(G_2)$ with $\overline{d}_e(G_1^\pi, G_2) = k$, if and only if there is an assignment which satisfies k constraints in $\mathcal{F}$.*

Proof. For each color class C_i, we assign a variable x_i. For an edge e from C_i to C_j in G_1, construct the function $f_e\colon \{0,1\}^2 \to \{0,1\}$ over the variables x_i and x_j as follows. Any Boolean assignment to the variables can be looked upon as a permutation: If $x_i \mapsto 0$, then we have the identity permutation on C_i, otherwise the permutation swaps the vertices of C_i. The value f_e on that particular assignment is 1 if the permutation that it corresponds to sends the edge e to an edge in G_2. Hence there is an assignment that satisfies k constraints if and only if there is a permutation π with $\overline{d}_e(G_1^\pi, G_2) = k$. □

Thus, both Max-PGI$_k$ and Max-EGI$_k$ have constant factor approximation algorithms by virtue of the semidefinite programming based approximation algorithm for Max-2CSP(q) due to [GR08].

As the problem of Max-2CSP(2) has an approximation algorithm with a guarantee of 0.874, due to [LLZ02], this implies an approximation algorithm for Max-EGI$_2$ with the same guarantee and since Max-2Lin-2 is hard to approximate beyond 0.878 under UGC [KKMO04], we have almost matching upper and lower bounds for Max-EGI$_2$ under UGC.

Remark. We remark that the Unique Games Conjecture(UGC), introduced by Khot [Kho02], is a weaker hardness assumption than NP-hardness. The UGC conjectures that it is NP-hard to decide the promise problem *label cover with unique constraints*, which we explain now. For a directed graph $G(V,E)$, an alphabet Σ, and a set of permutations $\{\pi_e : \Sigma \to \Sigma \mid e \in E\}$, the value of a map $\phi : V \to \Sigma$, is defined as $val_\phi(G) = |\{e = (u,v) \in E \mid \pi_e(\phi(u)) = \phi(v)\}|$. The value of the graph, $val(G)$ is the maximum of this quantity over all maps $\phi : V \to \Sigma$. Given parameters $\varepsilon, \delta > 0$, the objective of label cover with unique constraints is to test if $val(G) \leq \varepsilon$ or $val(G) \geq 1 - \delta$. The UGC states for every ε, δ, there is some Σ such that this decision problem is NP-hard. The relevance of UGC comes from the fact that under this assumption, we can prove many optimal lower bounds for optimization problems [Kho02].

2.3 Minimizing the number of mismatches

In this section we consider the minimization problems Min-PGI and Min-EGI, where the objective is to minimize the number of mismatched pairs and edges, respectively.

Theorem 2.1.5. *There is no polynomial-time approximation algorithm for* Min-PGI *with any multiplicative approximation guarantee unless* GI ∈ P.

Proof. Assume that there is a polynomial time α-approximation algorithm $\mathcal{A}$ for Min-PGI. We will give a polynomial time algorithm for graphs isomorphism using the approximation algorithm $\mathcal{A}$. Let G_1 and G_2 be the inputs to the Graph Isomorphism problem. If the two input graphs G_1 and G_2 are isomorphic, then there is a bijection $\pi\colon V_1 \to V_2$ such that $d_p(G_1^\pi, G_2) = 0$, and if G_1 and G_2 are not isomorphic, then $d_p(G_1^\pi, G_2) > 0$ for all π. Thus, it immediately follows that G_1 and G_2 are isomorphic, if and only if $\mathcal{A}$ outputs a bijection $\sigma\colon V(G_1) \to V(G_2)$ with $d_p(G_1^\sigma, G_2) = 0$ (i.e., an isomorphism). □

The hardness assumption in the theorem above is weak since it is generally believed that Graph Isomorphism has a polynomial-time algorithm. Next we show that Min-PGI is unlikely to have a polynomial-time approximation scheme unless P = NP. In order to show that it is unlikely that Min-PGI has a polynomial-time approximation scheme, we give a gap-preserving reduction from the Vertex-disjoint Triangle Packing problem (VTP) defined as follows: Given a graph G find the maximum number of vertex-disjoint triangles that can be packed into G. We look at the corresponding gap version of the VTP problem:

Problem 2.3.1 (Gap-VTP$_{\alpha,\beta}$)**.** *Given a graph G and $\alpha > \beta$,*

1. *Answer YES, if at least $\alpha n/3$ triangles can be packed into G.*
2. *Answer NO, if at most $\beta n/3$ triangles can be packed into G.*

It is known that the VTP problem does not have an algorithm which when given a graph and parameter α as input, computes a vertex-disjoint triangle packing of size at least αOPT in time $O(n^{\text{poly}(1/(1-\alpha))})$ unless P = NP [CR02]. It is also known that for a fixed value of $\beta < 1$, Gap-VTP$_{1,\beta}$ is NP-hard on graphs of bounded degree [GI03, Pet94]. Indeed, Petrank [Pet94] gives a gap-preserving reduction from 3Sat to 3DimensionalMatching. It is not hard to see that replacing the hyperedges in the generated instances with triangles results in a gap-preserving reduction to VTP, as all triangles in the resulting graph correspond to a hyperedge. All vertices in the generated graph G have degree 4 or 6. Thus there is a β such that Gap-VTP$_{1,\beta}$ is NP-hard on such graphs. By attaching the gadget depicted in Fig 2.2 to each vertex of degree 4 in G, we obtain a 6-regular graph G', which we again consider as VTP instance.

Let n and n' denote the number of vertices in G and G', respectively. If G can be packed with $n/3$ vertex-disjoint triangles, then G' can also be packed fully by vertex-disjoint triangles. If $\text{OPT}(G) \leq \beta n/3$, then $\text{OPT}(G') \leq (1 - \frac{1-\beta}{13})n'/3$. Thus there is a β' such that Gap-VTP$_{1,\beta'}$ is NP-hard on 6-uniform graphs.

Lemma 2.3.2. *Given a* Gap-VTP$_{\alpha,\beta}$ *instance $\mathcal{I}$ (a 6-uniform graph on n vertices), in*

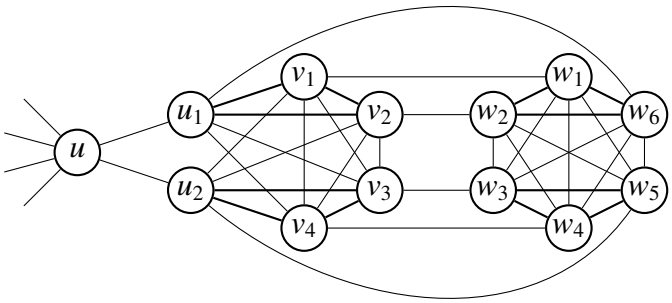

Figure 2.2: Converting a VTP instance G of degrees 4 and 6 to a 6-uniform VTP instance G'

polynomial time we can find an Min-PGI *instance* $\mathcal{I}'$ *such that*

$$\mathsf{OPT}(\mathcal{I}) \geq \frac{\alpha n}{3} \Rightarrow \mathsf{OPT}(\mathcal{I}') \leq 2n(2-\alpha)$$
$$\mathsf{OPT}(\mathcal{I}) \leq \frac{\beta n}{3} \Rightarrow \mathsf{OPT}(\mathcal{I}') \geq \frac{2n}{3}(4-\beta)$$

This reduction together with the hardness of VTP proves Theorem 2.1.6.

Proof. Let the instance $\mathcal{I}$ of VTP be a 6-regular graph G on n vertices. We construct a Min-PGI instance $\mathcal{I}' = (G_1, G_2)$ as follows: $G_1 := G$ and G_2 is a collection of $n/3$ vertex-disjoint triangles on the same vertex set as G_1, without any further edges. Suppose $\mathsf{OPT}(\mathcal{I}) \geq \alpha n/3$, then there is a permutation π that maps at least $\alpha n/3$ triangles to vertex-disjoint triangles of G_1. Hence the number of edges of G_1 that are mapped to non-edges of G_2 is at most $3n-\alpha n$. Similarly, the number of edges of G_2 that are images of non-edges of G_1 is at most $(1-\alpha)n$. Therefore, $\mathsf{OPT}(\mathcal{I}') \leq d_p(G_1^\pi, G_2) \leq 2n(2-\alpha)$.

Now suppose $\mathsf{OPT}(\mathcal{I}) \leq \beta n/3$. Since G_1 has at most $\beta n/3$ disjoint triangles, any permutation π maps at least $(1-\beta)n/3$ non-edges of G_1 to edges of G_2. Further, since G_1 has at least $2n$ edges more than G_2 and since already at least $(1-\beta)n/3$ of the edges of G_2 are images of non-edges of G_1, π maps at least $2n + (1-\beta)n/3$ edges of G_1 to non-edges of G_2. Thus we have

$$d_p(G_1^\pi, G_2) \geq \frac{n}{3}(1-\beta) + 2n + \frac{n}{3}(1-\beta) = \frac{2n}{3}(4-\beta).$$

□

Next we prove Theorem 2.1.7.

Theorem 2.1.7. *There is no polynomial time approximation algorithm for* Min-EGI *with any multiplicative approximation guarantee unless* P = NP.

Proof. The theorem follows from the following reduction from the CLIQUE problem. Given an instance (G, k) of CLIQUE, we construct the instance of Min-EGI as follows. G_1 consists of a k-clique and $n - k$ independent vertices, and $G_2 := G$. $(G, k) \in$ CLIQUE if and only if there exists a π such that in the Min-EGI problem $d_e(G_1^\pi, G_2) = 0$. Hence any polynomial-time approximation algorithm with a multiplicative guarantee for Min-EGI gives a polynomial time algorithm for CLIQUE. □

Now we will prove two results that show that approximating Min-PGI$_k$ and Min-EGI$_k$ to any constant factor is hard. First, we will prove the inapproximability of the Min-PGI$_k$ problem. For this, we will use the Min-Uncut problem. The input for the Min-Uncut problem is a set $E \subseteq \{x_i + x_j = 1 \mid i, j \in [n]\}$ of m equations. The objective is to minimize the number of equations that must be removed from the set E so that there is an assignment to the variables that satisfy all the equations. If we construct a graph on n vertices, with an edge between i and j if the equation $x_i + x_j = 1$ is present in the set of equations, then the objective of the Min-Uncut problem is to find a cut that partitions the vertex set into two parts such that the number of edges within each part is minimized. This problem is known to be MaxSNP-hard [KSTW00], and assuming the Unique Games Conjecture, hard to approximate within any constant factor [Kho02]. The following lemma shows that Min-PGI$_k$ is as hard as the Min-Uncut problem.

Lemma 2.3.3. *Let $\mathcal{I}$ be an instance of* Min-Uncut *and let k be a positive integer. There is a polynomial-time algorithm that constructs an instance $\mathcal{I}'$ of* Min-PGI$_{2k}$ *such that* $\mathsf{OPT}(\mathcal{I}') = (2k)^2\mathsf{OPT}(\mathcal{I})$.

The proof of this lemma is similar to the proof of Lemma 2.2.9. Given a set $E \subseteq \{x_i + x_j = 1 \mid i, j \in [n]\}$ of equations over $\mathbb{F}_2$, we construct an instance $\mathcal{I}'$ of Min-PGI$_{2k}$ exactly as described in the proof of Lemma 2.2.9. If the minimum number of equations that have to be deleted from E to make the rest satisfiable is at most t, then there is an assignment such that at most t equations in E are not satisfied. This implies that there is a permutation π such that the only edges that are mapped to non-edges and vice-versa are from at most t pairs of color classes. The same argument as in the proof of Lemma 2.2.9 shows that for any permutation π there is a permutation σ such that $d_p(G_1^\sigma, G_2) \leq d_p(G_1^\pi, G_2)$ and σ has the following property: For any color class j, σ maps all the vertices in V_j^0 to V_j^1 and vice-versa or is the identity mapping on that color class.

Finally we show that Min-EGI$_4$ is hard to approximate.

Theorem 2.3.4. *For any constant $\alpha > 1$, there is no α-approximation algorithm for* Min-EGI$_4$ *unless* P = NP.

An instance of NCP consists of a subspace $\mathcal{S}$ of $\mathbb{F}_2^n$ given as a set of basis vectors $\mathcal{B} = \{s_1, \ldots, s_k\}$ and a vector $v \in \mathbb{F}_2^n$. The objective is to find a vector $u \in \mathcal{S}$ which minimizes the hamming weight $\mathrm{wt}(u + v)$, i.e., the number of bits where u and v differ. It is NP-hard to approximate NCP within any constant factor [ABSS97]. The following lemma gives a reduction that transfers this hardness to Min-EGI$_4$.

Lemma 2.3.5. *There is a polynomial-time algorithm that for a given* NCP *instance $\mathcal{I}$, constructs a* Min-EGI$_4$ *instance $\mathcal{I}'$ with* $\mathsf{OPT}(\mathcal{I}') = \mathsf{OPT}(\mathcal{I})$.

The idea of the proof is to construct two graphs G_1 and G_2 such that any vector from the given subspace $\mathcal{S}$ that is equal to v in all but k positions, can be converted into a color-preserving bijection from $V(G_1)$ to $V(G_2)$ that maps all but k edges to edges, and vice versa.

Let the instance $\mathcal{I}$ be given by the vector $v \in \mathbb{F}_2^n$ and the basis $\mathcal{B} = \{s_1, \ldots, s_m\}$ of the subspace $\mathcal{S}$, i.e., $\mathcal{S} = \{\sum_{i=1}^m \alpha_i s_i \mid \alpha_i \in \{0, 1\}\}$. The computation of a vector $u \in \mathcal{S}$ can be thought of as n circuits $C_1, \ldots, C_n$. Thus C_i computes the i^{th} bit of u, i.e., $C_i(\alpha) = \bigoplus_{j \in [m], s_{j,i}=1} \alpha_j$, where $\alpha = \alpha_1 \cdots \alpha_m$ is the input and $s_{j,i}$ is the i^{th} bit of s_j. We assume that these circuits contain only parity gates with fanin 2.

We now proceed to construct a graph G from these circuits such that there is a one-one correspondence between all assignments of values to α and all automorphisms of G. For each input bit α_j, add two vertices $\alpha_{j,0}, \alpha_{j,1}$ of the same color. Assigning $\alpha_j = 0$ corresponds to the identity permutation on this color class, assigning $\alpha_j = 1$ corresponds to exchanging these vertices. We also add two vertices of the same color for the output of each parity gate. To get the desired correspondence between assignments and automorphisms, we use the graph gadget of Torán [Tor04]: For a parity gate with inputs x and y which computes $z = x \oplus y$, the gadget $G_\oplus$ connects the vertices x_0, x_1 corresponding to x, y_0, y_1 corresponding to y, and z_0, z_1 corresponding to z using four additional intermediate vertices $w_{0,0}, w_{0,1}, w_{1,0}, w_{1,1}$ that receive the same (new) color. For $b \in \{0, 1\}$, the vertex x_b is connected to $w_{b,0}$ and $w_{b,1}$, while y_b is connected to $w_{0,b}$ and $w_{1,b}$. The vertex w_{b_1,b_2} is connected to $z_{b_1 \oplus b_2}$ for $b_1, b_2 \in \{0, 1\}$. The construction is depicted in Figure 2.3.

The gadget is useful due to the following lemma.

Lemma 2.3.6 ([Tor04])**.** *There is a unique automorphism ϕ for $G_\oplus$ which maps x_i to $x_{a \oplus i}$ and y_i to $y_{b \oplus i}$ for $a, b, i \in \{0, 1\}$. This automorphism ϕ maps z_i to $z_{a \oplus b \oplus i}$.*

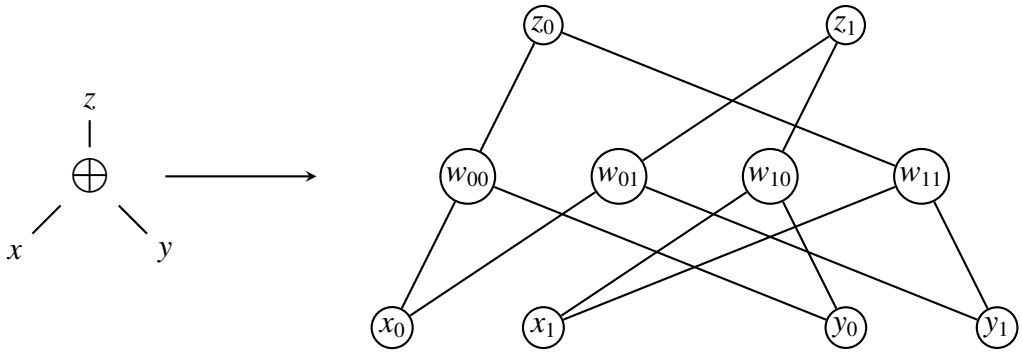

Figure 2.3: Gadget $G_\oplus$ corresponding to a parity gate $z = x \oplus y$ [Tor04]

Lemma 2.3.6 implies that the automorphisms of G exactly correspond to the valid computations of the circuits $C_1, \ldots, C_n$ on all possible 2^m assignments. We obtain the two graphs G_1 and G_2 for the Min-EGI$_4$ instance $\mathcal{I}'$ from the graph G by adding marker gadgets to the vertices corresponding to the output bits. Let $u_{i,0}$ and $u_{i,1}$ be the vertices corresponding to the output bit of C_i. For each circuit, we add a new vertex u'_i (with a new color) in G_1 as well as in G_2. In G_1, we connect u'_i to $u_{i,0}$ if $v_i = 0$, and to $u_{i,1}$ otherwise, whereas in G_2, we connect u'_i to $u_{i,0}$ unconditionally. Now we are ready to prove Lemma 2.3.5.

Proof of Lemma 2.3.5. Given an instance of NCP specified by a subspace $\mathcal{S}$ generated by the basis vectors $\mathcal{B} = \{s_1, \ldots, s_m\}$ and a vector $v \in \mathbb{F}_2^n$, we construct graphs G_1 and G_2 as described above.

Suppose there exists a vector $u = \sum_{i=1}^{m} \alpha_i s_i$ such that $\mathrm{wt}(u + v) \leq t$. Given this α, we construct an automorphism π_α of G as follows: For each input node of C_i, apply the automorphism on the vertices corresponding to the value of α_i to it. For each parity gate, Lemma 2.3.6 specifies how to extend an automorphism to the output vertices of the gadget, given a permutation of the input vertices. Continuing this process for the whole graph we get an automorphism of G that maps the vertex $u_{i,0}$ to u_{i,u_i}. We extend this automorphism to a mapping from G_1 to G_2, fixing the output marker vertices u'_i. The only unmatched edges are those incident to the vertices u'_i with $u_i \neq v_i$, so all but at most t edges of G_1 are mapped to edges of G_2.

Now suppose that there is a permutation π such that $d_e(G_1^\pi, G_2) \leq t$. By construction, each parity gate is used for only one output bit, so at most t output bits are affected by the mismatched edges. Thus we can convert this permutation π to a new permutation σ such that $d_e(G_1^\sigma, G_2) \leq d_e(G_1^\pi, G_2)$ where the only edge that is mapped to a non-edge is $(u_{i,b}, u'_i)$. This is because for each circuit C_j, starting from a permutation of its inputs, we can consistently extend the permutation till the output gate of C_j. Thus depending on

whether the input vertices were flipped by the permutation or not, we can assign a value to each α_j and hence get a vector $u \in \mathcal{S}$ such that $\mathrm{wt}(u + v) \leq t$. This completes the proof of the lemma and finishes the proof of Theorem 2.3.4. □

2.4 Summary and Open Problems

Although GI expressed as an optimization problem was mentioned in [AFK02], as far as we know this is the first time that the complexity of the other three variants of this optimization problem has been studied. Considering the upper and lower complexity bounds that we have proved in this paper, the following questions seem particularly interesting.

1. In Theorem 2.1.2 we describe an α-approximation algorithm for Max-PGI that runs in quasi-polynomial time. Does Max-PGI also have a polynomial-time approximation scheme? It is also interesting that the property tester for graph isomorphism [FM06], also has a quasi-polynomial time bound on the running time. That paper also uses ideas that are similar to [AFK02].

2. Recently, it has been shown in [OWWZ14] that Max-EGI has no constant factor approximation algorithm under the random 3XOR hypothesis of Feige [Fei02]. The reduction uses the idea of encoding an XOR gate with a graph gadget similar to [Tor04, CFI92]. It remains open if the same hardness result can be proved under stronger assumptions.

3. In the case of vertex-colored graphs, even though we can rule out the existence of a PTAS for Max-PGI$_k$ and Max-EGI$_k$ for $k > 2$, it remains open whether these problems have efficient approximation algorithms providing a good constant factor approximation guarantee.

Chapter 3

Approximate Boolean Isomorphism

Boolean Isomorphism is the decision problem of checking if the input Boolean functions $f, g : \{0,1\}^n \to \{0,1\}$ are isomorphic: is there a permutation $\pi : [n] \to [n]$ of the variables $\{x_1, x_2, \ldots, x_n\}$ such that the Boolean functions $f(x_1, x_2, \ldots, x_n)$ and $g(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$ are equivalent. As explained in Chapter 1, the complexity of Boolean Isomorphism depends on the representation of the inputs.

Consider any representation of the Boolean functions f and g such that the representations can be evaluated on any given input in $\{0,1\}^n$ in time polynomial in n and the size of the representation. We have the following simple algorithm for Boolean Isomorphism that runs in time $O^*(2^{O(n)})$[1]: First, compute the truth-tables of the functions f and g in time $O^*(2^n)$ by evaluating the representations on all 2^n inputs. Now, the truth tables for f and g can be encoded as hypergraphs G_f and G_g with vertex set $[n]$ in a straightforward manner. A subset $S \subseteq [n]$ is an edge in G_f if and only if $f(x_S) = 1$, where $x_S \in \{0,1\}^n$ is the characteristic vector corresponding to S. I.e. $\chi_S(i) = 1$ if and only if $i \in S$. Hypergraph Isomorphism for n-vertex and m-edge hypergraphs has a $2^{O(n)}m^{O(1)}$ algorithm due to Luks [Luk99] which yields the claimed $O^*(2^{O(n)})$ time algorithm for testing if f and g are isomorphic.

As observed in Chapter 1, even for f and g represented as 3-CNF formulas, solving Boolean Isomorphism in $2^{o(n)}$ time appears very difficult since it is at least as hard as obtaining such algorithms for 3-CNF satisfiability. On the other hand, there are representations for which the satisfiability problem can be solved in polynomial time, but a $2^{o(n)}$ time algorithm for Boolean Isomorphism appears difficult due to a different bottleneck. For instance, consider monotone DNF formulas for which satisfiability is a trivial problem. We give a simple reduction showing that Hypergraph Isomorphism is polynomial-

[1]Here $O^*()$ suppresses factors that are polynomial in the input size.

time reducible to Boolean Isomorphism for monotone DNFs.[2]

Proposition. Hypergraph Isomorphism is polynomial-time reducible to Boolean Isomorphism for monotone DNF formulas.

Proof. Suppose $\mathcal{H}_1$ and $\mathcal{H}_2$ are hypergraphs on vertex set $[n]$. Let $\mathcal{E}_1$ and $\mathcal{E}_2$ be their respective hyperedge sets. The easy case for the reduction is when the hyperedges in $\mathcal{E}_1$ are not subsets of each other (which must also hold in $\mathcal{E}_2$, for otherwise the hypergraphs are not isomorphic).

In that case, for each hyperedge $E \in \mathcal{E}_1$, such that $E = \{i_1, \ldots, i_r\}$, we include a monotone term $T_E = \wedge_{j=1}^{r} x_{i_j}$ in the DNF formula $\mathcal{F}_1$ so that $\mathcal{F}_1 = \vee_{E \in \mathcal{E}_1} T_E$. The monotone DNF formula $\mathcal{F}_2 = \vee_{E \in \mathcal{E}_2} T_E$ is similarly defined.

Now, we claim that the hypergraphs $\mathcal{H}_1$ and $\mathcal{H}_2$ are isomorphic if and only if $\mathcal{F}_1$ and $\mathcal{F}_2$ are isomorphic as Boolean functions. The forward direction is obvious. For the reverse direction suppose $\pi : [n] \to [n]$ is an isomorphism between $\mathcal{F}_1$ and $\mathcal{F}_2$. Since $\mathcal{F}_1^{\pi}$ and $\mathcal{F}_2$ are equivalent Boolean functions, it follows every term T_2 in $\mathcal{F}_2$ must contain some term T_1^{π} in $\mathcal{F}_1^{\pi}$. In turn, T_1^{π} must contain some term T_2' of $\mathcal{F}_2$. However, by the assumed subset free property of the hyperedges in the two hypergraphs it follows that $T_1^{\pi} = T_2$. Thus, π must map hyperedges of $\mathcal{H}_1$ to hyperedges of $\mathcal{H}_2$ making it an isomorphism.

We now consider the general case when the hypergraphs $\mathcal{H}_1$ and $\mathcal{H}_2$ need not have the subset-free property. In this case we first transform them to new hypergraphs $\mathcal{H}_1'$ and $\mathcal{H}_2'$ on a vertex set of size $O(n)$ and whose hyperedges are subset-free. Letting $[n]$ denote the original vertex set, we introduce new vertices $\{w_0, w_1, \ldots, w_n\}$. For each hyperedge E of $\mathcal{H}_1$ include the hyperedge $E \cup \{w_{|E|}\}$ in $\mathcal{H}_1'$. Additionally, include hyperedges $\{w_0, w_i\}$ $1 \leq i \leq n$ in $\mathcal{H}_1'$ as well as hyperedges $\{i, w_i\} 1 \leq i \leq n$. By construction the hypergraph $\mathcal{H}_1'$ is subset-free. Furthermore, we claim that $\mathcal{H}_1$ and $\mathcal{H}_2$ are isomorphic if and only if $\mathcal{H}_1'$ and $\mathcal{H}_2'$ are isomorphic. The construction enforces that any isomorphism from $\mathcal{H}_1'$ to $\mathcal{H}_2'$ must fix the vertex w_0 and consequently each w_i is also fixed.

Combined with the reduction for the subset-free case above completes the proof. To wit, the two hypergraphs $\mathcal{H}_1$ and $\mathcal{H}_2$ are isomorphic if and only if the monotone DNF formulas $F_{\mathcal{H}_1}$ and $F_{\mathcal{H}_2}$ are isomorphic (as Boolean functions). □

Thus, a $2^{o(n)}$ time algorithm for Boolean Isomorphism when f and g are monotone DNF formulas would yield a $2^{o(n)}$ time algorithm for Hypergraph Isomorphism, improving upon Luks's $2^{O(n)}$ time algorithm for Hypergraph Isomorphism, which is an open problem for

[2]There is also a simple polynomial-time reduction in the other direction, from Boolean Isomorphism for monotone DNFs to Hypergraph Isomorphism.

over a decade now. Since Boolean Isomorphism is as hard as Hypergraph Isomorphism, even for monotone DNFs, we investigate *approximate* Boolean Isomorphism. Our approach to the approximation problem is via Fourier analytic techniques for Boolean functions. It turns out that this works for certain restricted circuit representations for the input functions f and g.

Remark. We note here that approximate function isomorphism has been studied in the framework of property testing. We recall that in property testing the objective is to test whether two given Boolean functions are close to being isomorphic or far apart. The main goal is to design a property tester with low *query* complexity and time complexity is of secondary importance. Nearly matching upper and lower bounds are known ([AB10, BO10, CGSM11]) for the Boolean isomorphism problem in the setting of property testing. In contrast, the results in this chapter are algorithmic and the goal is to efficiently compute a good approximate isomorphism.

3.1 Preliminaries

For the rest of this chapter, it is convenient to consider Boolean functions with domain $\{\pm 1\}^n$ and range $\{\pm 1\}$. As we will explain shortly, this notation is useful for representing Boolean functions as polynomials in the Fourier basis representation, which helps in the analysis of the algorithms in this chapter. We will denote the set of all n-ary Boolean functions $f : \{\pm 1\}^n \to \{\pm 1\}$ by $\mathcal{B}_n$. Let $g : \{\pm 1\}^n \to \{\pm 1\}$ be a Boolean function and let $\pi : [n] \to [n]$ be any permutation. The Boolean function $g^\pi : \{\pm 1\}^n \to \{\pm 1\}$ obtained by applying the permutation π to the function g is defined as follows: $g^\pi(x_1, x_2, \ldots, x_n) = g(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$.

This defines a (faithful) group action of the permutation group S_n on the set $\mathcal{B}_n$. I.e. $g^{(\pi\psi)} = (g^\pi)^\psi$ for all $g \in \mathcal{B}_n$ and $\pi, \psi \in S_n$, and $g^\pi = g^\psi$ for all $g \in \mathcal{B}_n$ if and only if $\pi = \psi$.

Definition 3.1.1. *Two Boolean functions $f, g \in \mathcal{B}_n$ are said to be isomorphic (denoted by $f \cong g$) if there exists a permutation $\pi : [n] \to [n]$ such that $\forall x \in \{\pm 1\}^n, f(x) = g^\pi(x)$.*

Our notion of approximate isomorphism of Boolean functions is based on the notion of *closeness* of Boolean functions which we now recall.

Definition 3.1.2. *Two Boolean functions f, g are $\frac{1}{2^\ell}$-close if* $\Pr_{x \in \{0,1\}^n}[f(x) \neq g(x)] \leq \frac{1}{2^\ell}$.

Definition 3.1.3. *Two Boolean functions f, g are* $\frac{1}{2^\ell}$-approximate isomorphic *if there is a permutation $\pi : [n] \to [n]$ such that the functions f and g^π are $\frac{1}{2^\ell}$-close.*

We now give a brief overview of Fourier analysis of Boolean functions that we will use in this chapter.

3.1.1 Fourier Analysis Cheat Sheet

The study of Boolean functions by using Fourier analysis of the abelian group $\mathbb{Z}_2^n$ was initiated in the seminal work of Kahn, Kalai and Linial [KKL88]. Linial, Mansour and Nisan [LMN93] built on this to study the properties of the *Fourier spectrum* of *bounded depth* Boolean circuits.

The fundamental idea is to study the linear space of real functions on $\mathbb{Z}_2^n$. The set $\mathcal{F} = \{f : \{-1,1\}^n \to \mathbb{R}\}$ of real-valued functions forms a 2^n-dimensional vector space over $\mathbb{R}$, where vector addition is defined as $(f+g)(x) = f(x) + g(x)$. The vector space $\mathcal{F}$ forms an inner product space with inner product defined as follows:

$$\langle f, g \rangle = \mathbb{E}_{x \in \{-1,1\}^n}[f(x)g(x)] = \frac{1}{2^n} \sum_{x \in \{-1,1\}^n} f(x)g(x).$$

The ℓ_2-norm of a function $f \in \mathcal{F}$ is $\|f\|_2 = \sqrt{\langle f, f \rangle}$. Clearly any Boolean function $f : \{-1,1\}^n \to \{-1,1\}$ has unit norm under this inner product since

$$\langle f, f \rangle = \frac{1}{2^n} \sum_{x \in \{\pm 1\}^n} f(x)^2.$$

The standard basis for the vector space consists of the functions $\{f_a \mid a \in \{\pm 1\}^n\}$ where

$$f_a(x) = \begin{cases} -1 & \text{if } x = a, \\ 1 & \text{otherwise.} \end{cases}$$

In order to analyze the properties of Boolean functions, it is useful to look at the *Fourier basis* of $\mathcal{F}$, which is the set $\{\chi_S \big| S \subseteq [n]\}$ defined as $\chi_S(x) = \prod_{i \in S} x_i$. These are precisely the 2^n parity functions in the $\{0,1\}^n$ domain. It is easy to observe the following proposition.

Proposition 3.1.4. *For any $S \subseteq [n]$, $\mathbb{E}_x[\chi_S(x)] = 0$ for $S \neq \emptyset$ and $\mathbb{E}_x[\chi_\emptyset(x)] = 1$.*

As a consequence, we have the following proposition:

Proposition 3.1.5. *For any $S, T \subseteq [n]$,*

$$\langle \chi_S, \chi_T \rangle = \begin{cases} 1 & \text{if } S = T, \\ 0 & \text{otherwise.} \end{cases}$$

Furthermore, $\langle \chi_S, \chi_T \rangle = \mathbb{E}_x[\chi_{S \triangle T}(x)]$. It follows that the Fourier basis is an orthonormal basis with respect to the inner product. Thus, any $f \in \mathcal{F}$ can be written as $f = \sum \hat{f}_S \chi_S$. This is the *Fourier representation* of f, and the numbers $\hat{f}_S = \langle f, \chi_S \rangle$ are the *Fourier coefficients* of f. The *Fourier spectrum* of the Boolean functions f is the set of its Fourier coefficients. The orthonormality of the Fourier basis yields the following useful observations.

Theorem 3.1.6 (Plancheral's identity)**.** *Let f a real valued function on $\{\pm 1\}^n$. Then $\langle f, f \rangle = \sum_{S \subseteq [n]} \hat{f}(S)^2$.*

Proof. Since $\langle f, f \rangle = \langle \sum_{S \subseteq [n]} \hat{f}(S)\chi_S, \sum_{S \subseteq [n]} \hat{f}(S)\chi_S \rangle$, by the linearity of the inner product $\langle f, f \rangle = \sum_{S,T \subseteq [n]} \hat{f}(S)\hat{f}(T)\langle \chi_S, \chi_T \rangle$. By Proposition 3.1.5, we have $\langle f, f \rangle = \sum_{S \subseteq [n]} \hat{f}(S)^2$. □

The following important corollary is a direct consequence of the fact that Boolean functions have unit norm under the inner product that we defined.

Corollary 3.1.7 (Parseval's identity)**.** *Let $f : \{\pm 1\}^n \to \{\pm 1\}$ be any Boolean function. Then $\sum_{S \subseteq [n]} \hat{f}(S)^2 = 1$.*

3.1.2 Outline of the chapter

We will now define the notion of $(t(\varepsilon), \varepsilon)$-concentration of a Boolean function and state the main result of this chapter.

Definition 3.1.8. *A Boolean function $f : \{\pm 1\}^n \to \{\pm 1\}$ is said to be $(t(\varepsilon), \varepsilon)$-concentrated, if $\sum_{|S| > t(\varepsilon)} \hat{f}(S)^2 \leq \varepsilon$.*

From now on we will write $t(\varepsilon)$ as t while keeping in mind that the number t is a function of the concentration ε. The main theorem in this chapter can be stated as follows.

Theorem 3.1.9. *Let f and g be isomorphic (t, ε)-concentrated Boolean functions. Then, there is a randomized $2^{\widetilde{O}(t'(\delta)\sqrt{n})}$-time algorithm that makes oracle queries to the Boolean functions f and g and constructs a permutation π such that f and g^π and δ-close.*

The rest of the chapter is organized as follows. In Section 3.2 we state Theorem 3.1.9 formally and prove it. As corollaries to this theorem, we will see what this says about the approximate isomorphism problem for AC^0 functions and linear threshold functions. Finally, in Section 3.3 we examine a general problem: given two n-variable Boolean functions f and g as Boolean formulas, consider the optimization problem where the objective is to find a permutation π that maximizes $|\{x \in \{0,1\}^n \mid f(x) = g^\pi(x)\}|$. We observe that this problem is coNP-hard under polynomial-time Turing reductions.

A natural question to ask next is whether there is a polynomial-time approximation algorithm for this problem. The trivial algorithm to compute the permutation π that maximizes $|\{x \in \{0,1\}^n \mid f(x) = g^\pi(x)\}|$ is the brute-force algorithm of computing the truth-table of the functions and then picking the π that maximizes $|\{x \in \{0,1\}^n \mid f(x) = g^\pi(x)\}|$ by checking this for each permutation. This algorithm takes takes $2^{O(n \log n)}$ time.

The interesting question to ask is if we can do better than the brute-force search over all permutations. In the final section of the chapter, we give a simple $2^{O(n)}$ time deterministic approximation algorithm for this problem which takes as input Boolean functions f and g, represented as Boolean circuits, such that f and g^π agree on a constant fraction of the inputs for some permutation π, outputs a permutation σ such that f and g^σ agree on an $O(\frac{1}{\sqrt{n}})$ fraction of the inputs. We do not know if there is a faster algorithm or a $2^{O(n)}$ time algorithm that achieves a better approximation ratio.

3.2 Testing isomorphism using Fourier coefficients

We now consider the problem of computing an approximate isomorphism for two Boolean functions f, g that are (t, ε)-concentrated. We start with two propositions that relate the isomorphism of Boolean functions f and g to their Fourier coefficients.

Proposition 3.2.1. *Let $\pi : [n] \to [n]$ be any permutation, g any Boolean function and $S \subseteq [n]$, then $\hat{g^\pi}(S) = \hat{g}(S^\pi)$ where $S^\pi = \{\, i \mid \pi(i) \in S \,\}$.*

Proof. From the definition, we have

$$\begin{aligned}\hat{g^\pi}(S) &= \frac{1}{2^n} \sum_{x \in \{-1,1\}^n} g^\pi(x) \chi_S(x) \\ &= \frac{1}{2^n} \sum_{x_1,\ldots,x_n \in \{-1,1\}} g(x_{\pi(1),\ldots,\pi(n)}) \chi_S(x_1, \ldots, x_n).\end{aligned}$$

The permutation π defines a bijection from $\{-1,1\}^n$ to $\{-1,1\}^n$ where $\pi(b_1, \ldots, b_n) =$

$(b_{\pi(1)}, \ldots, b_{\pi(n)})$. Hence,

$$\sum_{x_1,\ldots,x_n\in\{-1,1\}} g(x_{\pi(1),\ldots,\pi(n)})\chi_S(x_1,\ldots,x_n) = \sum_{x_1,\ldots,x_n\in\{0,1\}} g(x_1,\ldots,x_n)\prod_{i\in S} x_{\pi^{-1}(i)}.$$

This completes the proof since χ_{S^π} is exactly $\prod_{i\in S} x_{\pi^{-1}(i)}$. □

Proposition 3.2.2. *Two Boolean functions $f, g : \{-1.1\}^n \to \{-1, 1\}$ are isomorphic via permutation π if and only if $\hat{f}(S) = \hat{g}(S^\pi)$ for each subset S.*

Proof. Suppose $\pi : [n] \to [n]$ is an isomorphism. I.e. $f(x) = g^\pi(x)$ for all $x \in \{-1, 1\}^n$. Consider any subset $S \subseteq [n]$

$$\hat{f}(S) = \frac{1}{2^n}\sum_{x\in\{-1,1\}^n} f(x)\chi_S(x) = \frac{1}{2^n}\sum_{x\in\{-1,1\}^n} g^\pi(x)\chi_S(x) = \hat{g^\pi}(S)$$

Since $\hat{g^\pi}(S) = \hat{g}(S^\pi)$ from the previous proposition, this completes the proof of the forward direction of the proposition.

Conversely, if $\hat{f}(S) = \hat{g}(S^\pi)$ for each subset S, again by the previous proposition we have $\hat{f}(S) = \hat{g^\pi}(S)$ which implies that $f = g^\pi$. □

3.2.1 Approximate Isomorphism for (t, ε)-concentrated functions

In this subsection we prove that the problem of checking if two (t, ε)-concentrated Boolean functions f, g are $1/2^\ell$-approximately isomorphic is efficiently reducible, via a randomized reduction with oracle access to f and g, to checking if two "low-degree" polynomials (whose degrees depend on ℓ) are isomorphic. We first outline this randomized reduction using the Fourier spectrum of the Boolean functions f and g.

Since f and g are (t, ε)-concentrated, we know that

$$\sum_{S\subseteq[n],|S|>t} \hat{f}(S)^2 \le \varepsilon \quad \text{and} \quad \sum_{S\subseteq[n],|S|>t} \hat{f}(S)^2 \le \varepsilon.$$

We will consider the function $\widetilde{f} = \sum_{|S|\le t} \hat{f}(S)\chi_S$. Notice that each $\chi_S = \prod_{i\in S} x_i$ is a monomial, and hence $\widetilde{f}$ is a degree-t polynomial that approximates f. Given (t, ε)-concentrated Boolean functions f, g as an instance of Boolean Isomorphism, our aim is to replace them

with the polynomials $\widetilde{f}$ and $\widetilde{g}$ in the isomorphism problem:

$$\widetilde{f} = \sum_{S\subseteq[n],|S|\leq t} \hat{f}(S)\chi_S \quad \text{and} \quad \widetilde{g} = \sum_{S\subseteq[n],|S|\leq t} \hat{g}(S)\chi_S. \tag{3.1}$$

This is because $\widetilde{f}$ and $\widetilde{g}$ are of degree t and have only n^t terms. The idea is that checking if there is an isomorphism between the functions $\widetilde{f}$ and $\widetilde{g}$ is likely to be easier. Since the polynomials $\widetilde{f}, \widetilde{g} : \{-1,1\}^n \to \mathbb{R}$ are no longer Boolean-valued functions, we need to formalize an appropriate notion of isomorphism here.

Definition 3.2.3. *Let $f', g' : \{-1,1\}^n \to \mathbb{R}$ be two functions from $\mathcal{F}$. We say that f' and g' are $\frac{1}{2^\ell}$-approximate isomorphic if there exists a permutation $\pi : [n] \to [n]$ such that $\|f' - g'^\pi\|_2^2 \leq \frac{1}{2^\ell}$.*

Notice that this definition of $\frac{1}{2^\ell}$-approximate isomorphism for real-valued functions differs from the notion of $\frac{1}{2^\ell}$-approximate isomorphism for Boolean functions in Definition 3.1.3. However, the next proposition shows that if Boolean functions f and g are $\frac{1}{2^\ell}$-close then $\|f - g\|$ is small.

Proposition 3.2.4. *If $f, g : \{-1,1\}^n \to \{-1,1\}$ are $\frac{1}{2^\ell}$-close, then $\|f - g\|_2^2 \leq 4\frac{1}{2^\ell}$*

Proof. From the definition, we have,

$$\|f - g\|_2^2 = \frac{1}{2^n} \sum_{x\in\{-1,1\}^n} (f(x) - g(x))^2.$$

Observing that $(f(x) - g(x))^2$ is 4 if $f(x) \neq g(x)$ and zero otherwise, we have

$$\|f - g\|_2^2 = \frac{4|\{x \mid f(x) \neq g(x)\}|}{2^n}$$

This completes the proof. □

We now explain the connection between $\frac{1}{2^\ell}$-approximate isomorphism of two functions and their Fourier coefficients.

Lemma 3.2.5. *Let f and g be two Boolean functions that are $\frac{1}{2^\ell}$-approximate isomorphic via permutation $\pi : [n] \to [n]$. Then $\forall S \subseteq [n] : \left|\hat{f}(S) - \hat{g^\pi}(S)\right| \leq \frac{2}{2^{\ell/2}}$.*

Proof. Notice that $\sum_{S\subseteq[n]}(\hat{f}(S) - \hat{g^\pi}(S))^2 = \|f - g^\pi\|_2^2$. Suppose f, g are $\frac{1}{2^\ell}$-approximate isomorphic via permutation π. By Proposition 3.2.4 we know that $\sum_{S\subseteq[n]}(\hat{f}(S) - \hat{g^\pi}(S))^2 = \|f - g^\pi\|_2^2 \leq \frac{4}{2^\ell}$. Hence for each subset $S \subseteq [n]$ we have $\left(\hat{f}(S) - \hat{g^\pi}(S)\right)^2 \leq \frac{4}{2^\ell}$. □

Let f and g be two Boolean functions that are $\frac{1}{2^\ell}$-approximate isomorphic via permutation $\pi : [n] \to [n]$. By the above proposition $|\hat{f}(S) - \hat{g^\pi}(S)|$ is bounded by $\frac{2}{2^{\ell/2}}$. Furthermore, since both $\hat{f}(S)$ and $\hat{g^\pi}(S)$ are Fourier coefficients of Boolean functions f and g^π, we have $0 \leq |\hat{f}(S)| \leq 1$ and $0 \leq |\hat{g^\pi}(S)| \leq 1$. Hence, the bound implies that the $\lfloor \ell/2 \rfloor - 1$ most significant positions in the binary representation of $\hat{f}(S)$ and $\hat{g^\pi}(S)$ are identical.

For each subset S, let $\hat{f_\ell}(S)$ denote the truncation of $\hat{f}(S)$ to the first $\lfloor \ell/2 \rfloor - 1$ bits. Thus, $|\hat{f_\ell}(S) - \hat{f}(S)| \leq \frac{1}{2^{\lfloor \ell/2 \rfloor - 1}}$ for each S. Similarly, $\hat{g}_\ell(S)$ denotes the truncation of $\hat{g}(S)$ to the first $\lfloor \ell/2 \rfloor - 1$ bits. We define the following two functions f_ℓ and g_ℓ from $\{-1, 1\}^n \to \mathbb{R}$:

$$f_\ell = \sum_{S \subseteq [n]} \hat{f_\ell}(S)\chi_S \quad \text{and} \quad g_\ell = \sum_{S \subseteq [n]} \hat{g}_\ell(S)\chi_S. \tag{3.2}$$

The following lemma formally summarizes the above discussion and gives us a way to reduce isomorphism of Boolean functions to exact isomorphism of low-degree polynomials.

Lemma 3.2.6. *Let f and g be two Boolean functions that are $\frac{1}{2^\ell}$-approximate isomorphic via permutation $\pi : [n] \to [n]$. Then $f_\ell = g_\ell^\pi$, i.e. the functions f_ℓ and g_ℓ are (exactly) isomorphic via the permutation π.*

Lemma 3.2.5 and Proposition 3.2.4 yield the following observation.

Lemma 3.2.7. *Suppose f, g are two Boolean functions that are $\frac{1}{2^\ell}$-approximate isomorphic via permutation π. Then $\|\widetilde{f} - \widetilde{g^\pi}\|_2^2 \leq \frac{4}{2^\ell}$. I.e. $\widetilde{f}$ and $\widetilde{g}$ are $\frac{4}{2^\ell}$-approximate isomorphic via the same permutation π. Furthermore, $|\hat{f}(S) - \hat{g^\pi}(S)| \leq \frac{2}{2^{\ell/2}}$ for all $S : |S| \leq t$.*

Proof. By Lemma 3.2.5 and Proposition 3.2.4 we have $\sum_{S \subseteq [n]} (\hat{f}(S) - \hat{g^\pi}(S))^2 \leq \frac{4}{2^\ell}$, which implies $\|\widetilde{f} - \widetilde{g^\pi}\|_2^2 = \sum_{|S| \leq t} (\hat{f}(S) - \hat{g^\pi}(S))^2 \leq \frac{4}{2^\ell}$. It follows that $|\hat{f}(S) - \hat{g^\pi}(S)| \leq \frac{2}{2^{\ell/2}}$ for all $S : |S| \leq t$. □

Now, if $|\hat{f}(S) - \hat{g^\pi}(S)| \leq \frac{2}{2^{\ell/2}}$ for all $S : |S| \leq t$, it implies that $\hat{f_\ell}(S) = \hat{g_\ell^\pi}(S)$ for all $S : |S| \leq t$, where $\hat{f_\ell}(S)$ and $\hat{g}_\ell(S)$ are defined in Equation 3.2. Indeed, if we truncate the coefficients of the polynomials $\widetilde{f}$ and $\widetilde{g}$ also to the first $\lfloor \ell/2 \rfloor - 1$ bits we obtain the polynomials:

$$\widetilde{f_\ell}(x_1, \ldots, x_n) = \sum_{S:|S|\le t} \hat{f}_\ell(S)\chi_S \tag{3.3}$$

$$\widetilde{g_\ell}(x_1, \ldots, x_n) = \sum_{S:|S|\le t} \hat{g}_\ell(S)\chi_S. \tag{3.4}$$

It clearly follows that π is an exact isomorphism between $\widetilde{f_\ell}$ and $\widetilde{g_\ell}$. We summarize the above discussion in the following lemma which is crucial for our algorithm.

Lemma 3.2.8. *Suppose f, g are two Boolean functions that are $\frac{1}{2^\ell}$-approximate isomorphic via permutation π. Then:*

1. *$\|\widetilde{f} - \widetilde{g^\pi}\|_2^2 \le \frac{4}{2^\ell}$. I.e. $\widetilde{f}$ and $\widetilde{g}$ are $\frac{4}{2^\ell}$-approximate isomorphic via the same permutation π, and hence $|\hat{f}(S) - \hat{g^\pi}(S)| \le \frac{2}{2^{\ell/2}}$ for all $S : |S| \le t$.*
2. *Consequently, π is an exact isomorphism between the polynomials $\widetilde{f_\ell}$ and $\widetilde{g}_\ell$.*

Now, if π is an exact isomorphism between $\widetilde{f_\ell}$ and $\widetilde{g}_\ell$ what can we infer about π as an approximate isomorphism between f and g? The following lemma quantifies it.

Lemma 3.2.9. *Suppose f and g are (t, δ)-concentrated Boolean functions such that π is an exact isomorphism between $\widetilde{f_\ell}$ and $\widetilde{g}_\ell$ (where $\widetilde{f}$ and $\widetilde{g}$ are given by Equation 3.1). Then π is an $(\delta + \varepsilon)^2$-approximate isomorphism between f and g, where $\varepsilon = \frac{2n^{t/2}}{2^{(\ell-1)/2}}$.*

Proof. Since π is an exact isomorphism between $\widetilde{f_\ell}$ and $\widetilde{g}_\ell$ we have $\widetilde{f_\ell} = \widetilde{g^\pi_\ell}$. Now consider $\|f - g^\pi\|_2^2$. By triangle inequality

$$\begin{aligned} \|f - g^\pi\|_2 &\le \|f - \widetilde{f}\| + \|\widetilde{f} - \widetilde{f_\ell}\| + \|\widetilde{f_\ell} - \widetilde{g^\pi}_\ell\| + \|\widetilde{g^\pi} - \widetilde{g^\pi}_\ell\| + \|g^\pi - \widetilde{g^\pi}\| \\ &= \|f - \widetilde{f}\| + \|\widetilde{f} - \widetilde{f_\ell}\| + \|\widetilde{g^\pi} - \widetilde{g^\pi}_\ell\| + \|g^\pi - \widetilde{g^\pi}\|. \end{aligned}$$

Since f and g are (t, δ)-concentrated, both $\|f - \widetilde{f}\|$ and $\|g^\pi - \widetilde{g^\pi}\|$ are bounded by δ. Furthermore,

$$\|\widetilde{f} - \widetilde{f_\ell}\|_2^2 = \sum_{S:|S|\le t} (\hat{f}(S) - \hat{f}_\ell(S))^2 \le \sum_{S:|S|\le t} \frac{4}{2^{\ell-1}} \le \frac{4n^t}{2^{\ell-1}}.$$

Hence, $\|\widetilde{f} - \widetilde{f_\ell}\| \le \frac{2n^{t/2}}{2^{(\ell-1)/2}}$ and, likewise, $\|\widetilde{g^\pi} - \widetilde{g^\pi}_\ell\| \le \frac{2n^{t/2}}{2^{(\ell-1)/2}}$. Putting it together with Proposition 3.2.4 we get

$$4\Pr[f \ne g^\pi] = \|f - g^\pi\|_2^2 \le \left(2\delta + \frac{4n^{t/2}}{2^{(\ell-1)/2}}\right)^2.$$

It follows that f and g are $(\delta+\varepsilon)^2$-approximate isomorphic via the permutation π. □

Our goal now is to design an efficient algorithm that will compute the polynomials $\widetilde{f_\ell}$ and $\widetilde{g_\ell}$, where ℓ will be appropriately chosen in the analysis. In order to compute $\widetilde{f_\ell}$ and $\widetilde{g_\ell}$ we need to estimate to $\lfloor \ell/2 \rfloor - 1$ bits of precision, the Fourier coefficients $\hat{f}(S)$ and $\hat{g}(S)$ for each subset $S : |S| \leq t$. Now, by definition, $\hat{f}(S)$ is the average of $f(x)\chi_S(x)$ where x is uniformly distributed in $\{-1,1\}^n$. Hence, following a standard Monte-Carlo sampling procedure, we can estimate $\hat{f}(S)$ quite accurately from a random sample of inputs from $\{-1,1\}^n$ and with high probability we can exactly compute $\hat{f}_\ell(S)$ for all $S : |S| \leq t$. We formally explain this in the next lemma.

Lemma 3.2.10. *Given $f : \{-1,1\}^n \to \{-1,1\}$ as a black-box, there is a randomized algorithm C with running time $\mathrm{poly}(n^t, 2^\ell)$ that outputs the set $\{\hat{f}_\ell(S) \mid |S| \leq t\}$ with probability at least $1 - \frac{1}{2^{\Omega(n)}}$.*

Proof. We use the same technique as [LMN93] to estimate the required Fourier coefficients.

1. For each subset $S \subset [n]$ such that $|S| \leq t$ do the following two steps:
2. Pick $x_i \in_r \{-1,1\}^n$ and compute the value $f(x_i)\chi_S(x_i)$ for $i \in [m]$.
3. Estimate the Fourier coefficient as $\alpha_f(S) = \frac{1}{m}\sum_{i=1}^{m} f(x_i)\chi_S(x_i)$.

Applying Chernoff bounds, for each subset S we have $\Pr[\left|\hat{f}(S) - \alpha_f(S)\right| \geq \lambda] \leq 2e^{-\lambda^2 m/2}$. In our case we set $\lambda = \frac{1}{2^{\lfloor \ell/2 \rfloor - 1}}$. In order to estimate $\hat{f}(S)$ for each $S : |S| \leq t$ within the prescribed accuracy and with small error probability, we set $m = tn \log n 2^\ell$. The entire procedure runs in $\mathrm{poly}(n^t, 2^\ell)$ time. Furthermore, by a simple union bound it follows that with probability at least $1 - 2^{-\Omega(n)}$ we have $\alpha_f(S) = \hat{f}_\ell(S)$ for each $S : |S| \leq t$ with high probability. Thus, the randomized algorithm computes the polynomial $\widetilde{f_\ell}$ with high probability. □

3.2.2 Exact isomorphism test for low degree polynomials

We now deal with the problem of checking if the polynomials $\widetilde{f_\ell} = \sum_{S:|S|\leq t} \hat{f}_\ell(S)\chi_S$ and $\widetilde{g_\ell} = \sum_{S:|S|\leq t} \hat{g}_\ell(S)\chi_S$ are isomorphic, and if so to compute an exact isomorphism π. To this end, we will encode f_ℓ and g_ℓ as *weighted* hypergraphs G_f and G_g respectively.

The vertex sets for both graphs is $[n]$. Let E denote the set of all subsets $S \subset [n]$ of size at most t. The weight functions for the edges are w_f and w_g for G_f and G_g defined as follows:

$$w_f(S) = \begin{cases} \hat{f}_\ell(S) & \forall S \subseteq [n], |S| \leq t \\ 0 & \textit{otherwise}, \end{cases} \quad \text{and} \quad w_g(S) = \begin{cases} \hat{g}_\ell(S) & \forall S \subseteq [n], |S| \leq t \\ 0 & \textit{otherwise}. \end{cases}$$

The isomorphism problem for the polynomials f_ℓ and g_ℓ is now the edge-weighted hypergraph isomorphism problem, where G_f and G_g are the two edge-weighted graphs, and the problem is to compute a permutation on $[n]$ that maps edges to edges (preserving edge weights) and non-edges to non-edges. Our aim is to apply the Babai-Codenotti isomorphism algorithm for hypergraphs with hyperedge size bounded by k [BC08]. Their algorithm has running time $2^{\widetilde{O}(k^2\sqrt{n})}$. We adapt their algorithm to work for hypergraphs with edge weights as follows: Since the edge weights for the graphs G_f and G_g can be encoded by $\lfloor \ell/2 \rfloor - 1$ length bit strings, we encode the weights into the hyperedges by introducing new vertices.

More precisely, we create new graphs G'_f and G'_g corresponding to f and g, where the number of vertices is now $n + O(\ell)$. Let the set of new vertices be $\{v_1, \ldots, v_r\}$, where $r = O(\ell)$. Let $S \subset [n]$ be a hyperedge in the original graph G_f. A subset $T \subset \{v_1, \ldots, v_r\}$ encodes an r-bit string via a natural bijection (the j^{th} bit is 1 if and only if $v_j \in T$). Let $T(S) \subset \{v_1, \ldots, v_r\}$ denote the encoding of the number $\hat{f}_\ell(S)$ for each hyperedge $S \in E$. Similarly, let $T'(S) \subset \{v_1, \ldots, v_r\}$ denote the encoding of the number $\hat{g}_\ell(S)$. The hyperedge $S \cup T(S)$ encodes S along with its weight $\hat{f}_\ell(S)$ for each S in G'_f. Similarly, $S \cup T'(S)$ encodes S along with its weight $\hat{g}_\ell(S)$ for each S in G'_g. The following lemma is an easy consequence of our construction of the hypergraphs.

Lemma 3.2.11. *There exists a permutation $\pi : [n] \to [n]$ such that $\widetilde{f_\ell} = \widetilde{g_\ell^\pi}$ if and only if $\mu : [n + r] \to [n + r]$ defined by*

$$\mu(i) = \begin{cases} \pi(i) & \textit{if } i \leq n \\ i & \textit{if } i > n \end{cases}$$

is a hypergraph isomorphism between G'_f and G'_g.

Since, as candidate isomorphisms between G'_f and G'_g we wish to consider only permutations on $[n] \cup \{v_1, \ldots, v_r\}$ that fix each v_i, $1 \leq i \leq r$, we perform the following operation on G'_f and G'_g so that any isomorphism between G'_f and G'_g is an identity on the set $\{v_1, \ldots, v_r\}$. For each vertex V_i, we add a $n + i$ length path starting at v_i. Since the hypergraphs G'_f

and G'_g did not have paths of length more than n originally, any isomorphism between the hypergraphs cannot map v_i to a vertex other than v_i.

With this construction, it is sufficient to construct an isomorphism between the hypergraphs G'_f and G'_g to compute an isomorphism between $\widetilde{f_\ell}$ and $\widetilde{g}_\ell$. Now we invoke the algorithm of [BC08] on G'_f and G'_g which will yield an isomorphism ψ between $\widetilde{f_\ell}$ and $\widetilde{g}_\ell$. In summary, the algorithm for isomorphism testing f_ℓ and g_ℓ carries out the following steps.

Isomorphism Test for Polynomials

1. Construct the hypergraphs G'_f and G'_g as defined above.
2. Run the algorithm of Babai and Codenotti [BC08] on the hypergraphs G'_f and G'_g and output isomorphism ψ or report they are non-isomorphic.

Lemma 3.2.12. *The isomorphism of polynomials $\widetilde{f_\ell}$ and $\widetilde{g}_\ell$ (defined by Equation 3.2) can be tested in time $2^{O(\sqrt{n})(\ell+t)^2 \log^{O(1)} n}$. If the polynomials are isomorphic, then an exact isomorphism can be computed in the same running time bound.*

3.2.3 Approximate Isomorphism Algorithm

We now give an outline of the entire algorithm.

> **Input**: Two (t, δ)-concentrated Boolean functions f, g along with parameters t and ℓ.
>
> **Step 1.** Compute the polynomials $\widetilde{f_\ell}$ and $\widetilde{g}_\ell$ using the randomized algorithm of Lemma 3.2.10.
>
> **Step 2.** Check if $\widetilde{f_\ell}$ and $\widetilde{g}_\ell$ are isomorphic using the polynomial isomorphism algorithm described above. If they are not isomorphic *reject* else **output** the computed exact isomorphism π.

Suppose π is an exact isomorphism between $\widetilde{f_\ell}$ and $\widetilde{g}_\ell$ computed by the above algorithm. By Lemma 3.2.9 π is a $(\delta + \varepsilon)^2$-approximate isomorphism between f and g, where $\varepsilon = \frac{2n^{t/2}}{2^{(\ell-1)/2}}$. From Lemmas 3.2.10 and 3.2.12 it follows that the overall running time of the algorithm is $\text{poly}(n^t, 2^\ell) + 2^{O(\sqrt{n})(\ell+t)^2 \log^{O(1)} n}$ and the error probability, as argued in Lemma 3.2.10, is at most $2^{-\Omega(n)}$.

This gives us the following theorem about approximate isomorphism of (t, ε)-concentrated Boolean functions.

Theorem 3.2.13. *Given two (t, δ)-concentrated Boolean functions f, g which are $\frac{1}{2^\ell}$-isomorphic, there is a randomized algorithm running in time $2^{O((\ell+t)^2 \log^{O(1)} n \sqrt{n})}$ that computes a $(\delta + \varepsilon)$-approximate isomorphism between f and g where $\varepsilon = 2n^{t/2}/2^{(\ell-1)/2}$ and the error probability of the algorithm is bounded by $2^{-\Omega(n)}$.*

If the two Boolean functions f and g are isomorphic, then we get the following theorem about approximate isomorphism for f and g.

Theorem 3.2.14. *Given two isomorphic Boolean functions f and g that are (t, δ)-concentrated and a parameter $\theta > 0$, there is a randomized $2^{t(\sqrt{\theta}/2) \log(n/\theta) \sqrt{n}}$-time algorithm that computes an θ-approximate isomorphism between f to g, and the error probability of the algorithm is bounded by $2^{-\Omega(n)}$.*

Proof. Notice first that if f and g are isomorphic, then for any $\ell > 0$, f and g are $\frac{1}{2^\ell}$-approximately isomorphic. Also, by Lemma 3.2.9, we know that if π is an isomorphism between between $\widetilde{f_\ell}$ and $\widetilde{g_\ell}$, then π is a $(\delta + \varepsilon)^2$-approximate isomorphism between f and g. For $\delta + \varepsilon \leq \sqrt{\theta}$, it is enough to set δ and ε such that $\delta \leq \sqrt{\theta}/2$ and $\varepsilon \leq \sqrt{\theta}/2$. For this, fix the value of $t = t(\delta) = t(\sqrt{\theta}/2)$ and $\ell = O(t \log(n/\theta))$ in Lemma 3.2.9. Using Lemma 3.2.10, we can compute $\widetilde{f_\ell}$ and $\widetilde{g_\ell}$ in time $\text{poly}(2^{t(\log(n/\theta)})$ and construct an isomorphism between $\widetilde{f_\ell}$ and $\widetilde{g_\ell}$ using Lemma 3.2.12 in randomized time $2^{t^2 \log(n/\theta) \sqrt{n}}$. □

We will now see two applications of Theorem 3.2.14.

3.2.3.1 AC^0 Functions

The first application is the isomorphism problem for the class of Boolean functions that are computable by Boolean circuits of depth d and size s, where the gates allowed are unbounded fan-in AND and OR gates, and negation gates. The class of constant depth unbounded fan-in circuits is well-studied in complexity theory. The class AC^0 consists of languages $L \subseteq \{0,1\}^*$ for which there is a nonuniform family of circuits $\{C_n\}_{n>0}$ such that: (i) For each n the circuit C_n takes n input bits and accepts precisely the length n strings in L, and (ii) There are a constant d and a polynomial $p(n)$ such that C_n is an unbounded fan-in circuit of depth bounded by d and size bounded by $p(n)$ for each n. Furst, Saxe and Sipser [FSS81] proved that the language of all binary strings of odd parity (i.e. the range of the parity function) is not in AC^0. A far reaching improvement of this result was due to

Håstad [Hås86] who obtained essentially optimal lower bounds for computing the parity of n variables.

Let $\mathcal{AC}_{s,d,n}$ denote the class of n-input Boolean functions computable by circuits of size at most s and depth at most d. Linial, Mansour and Nisan proved the following theorem about the Fourier coefficients of such Boolean functions in their seminal work [LMN93].

Theorem 3.2.15 ([LMN93]). *Let $f : \{\pm 1\}^n \to \{\pm 1\}$ be computable by a Boolean circuit of size s and depth d. Then, for any integer $t > 0$*

$$\sum_{S \subseteq [n], |S| > t} \hat{f}(S)^2 \leq 2s2^{-t^{1/d}}$$

In other words, any Boolean function f computable by a circuit of size s and depth d is $(\log(\varepsilon/2s)^{-d}, \varepsilon)$-concentrated.

Now, suppose $f, g \in \mathcal{AC}_{s,d,n}$ are isomorphic Boolean functions. As a consequence of Theorem 3.2.14 there is a randomized algorithm that computes an ε-approximate isomorphism between f and g in time $2^{\log(n/\varepsilon)^{O(d)}}\sqrt{n}$ for any positive ε. This is substantially faster than the $2^{O(n)}$ time algorithm for computing an exact isomorphism.

Remark. Given a Boolean function $f(x_1, \cdots, x_n)$, we used a real polynomial $\widetilde{f}(x_1, \cdots, x_n)$ that approximates f to reduce the isomorphism problem of Boolean functions to the isomorphism problem for hypergraphs. This is possible because for any permutation π of the Boolean functions, the approximating polynomial for the function $f(x_{\pi(1)}, \cdots, x_{\pi(n)})$ is obtained by permuting the monomials in $\widetilde{f}$ according to π. One could think of the polynomial as a sort of an approximate *permutation preserving normal form* (as described in Chapter 1) for the function.

Razborov [Raz87] and Smolensky [Smo87] in their seminal work on constant-depth circuits with *Mod* gates, also have shown that functions computed by AC^0 circuits are well approximated by polynomials of degree $(\log n)^{O(d)}$ over, say, rationals. However, these approximating polynomials do not seem to have a crucial property that we used for computing an approximate isomorphism between f and g. What we require is that f and g are approximately isomorphic via π if and only if the corresponding polynomials that approximate f and g are exactly isomorphic via π. In fact the approximating polynomials construction in [Raz87, Smo87] crucially use the circuit structure for the given functions f and g. Hence, if we construct approximating polylog-degree polynomials p_1 and p_2 starting from two different $\mathcal{AC}_{s,n,d}$ circuits for the same function f, it appears unlikely that p_1 and p_2 will be isomorphic. Similarly, it is not clear if the approximation of constant-

depth circuits by probabilistic polynomials [Tar93] can be used in designing isomorphism testing algorithms.

3.2.3.2 Linear Threshold Functions

A second class of functions where this result is applicable is the set of linear threshold functions.

Definition 3.2.16. *A function* $f : \{\pm 1\}^n \to \{\pm 1\}$ *is a linear threshold function if* $f(x_1, \cdots, x_n) = \mathrm{sgn}(\sum_{i\in[n]} w_i x_i + \theta)$ *where* $w_1, \cdots, w_n, \theta \in \mathbb{R}$.

The class of linear threshold functions is one of most well-studied classes of Boolean functions with applications in a wide variety of areas. The following theorem was proved by Klivans, O'Donnell and Servedio about the concentration of the Fourier coefficients of linear threshold functions.

Theorem 3.2.17 ([KOS04])**.** *Let* $f : \{\pm 1\}^n \to \{\pm 1\}$ *be any linear threshold function and let* $\varepsilon < 1/2$*. Then, the function* f *is* $((21/\varepsilon)^2, \varepsilon)$*-concentrated.*

As a corollary, using Theorem 3.2.14 we can see that there is a randomized algorithm that computes an ε-approximate isomorphism between linear threshold functions f and g in time $2^{(\log n/\varepsilon)\sqrt{n}}$.

3.3 A general approximate isomorphism algorithm

In this section we study the approximate isomorphism problem for general Boolean functions. Given two n-variable Boolean functions f and g as Boolean circuits computing them, consider the optimization problem of finding a permutation π that minimizes $|\{x \in \{\pm 1\}^n \mid f(x) \neq g^\pi(x)\}|$, which we will call MinBooleanIso. A brute-force search that runs in $n!$ time by cycling through all permutations yields a trivial algorithm for this optimization problem.

We show that MinBooleanIso is coNP-hard under Turing reductions. We give a polynomial-time Turing reduction from the coNP-complete problem TAUTOLOGY (checking if a propositional formula is a tautology) to MinBooleanIso.

Lemma 3.3.1. TAUTOLOGY *is polynomial-time Turing reducible to* MinBooleanIso.

Proof. Given $f : \{\pm 1\}^n \to \{\pm 1\}$ as an n-variable propositional formula, we define functions $g_i : \{\pm 1\}^n \to \{\pm 1\}$ for $i \in [n]$ such that

$$g_i(x) = \begin{cases} 1 & \text{if } x = -1^i 1^{n-i} \\ -1 & \text{otherwise} \end{cases}$$

The Boolean function f is a tautology if $f(x) = -1$ for all $x \in \{\pm 1\}^n$. Observe that if f is a tautology then for each i, $|\{x \in \{0,1\}^n \mid f(x) \neq g_i^\pi(x)\}| = 1$ for all permutations π.

We now describe a polynomial-time algorithm for TAUTOLOGY with MinBooleanIso as oracle. For each g_i, compute (with a query to the function oracle MinBooleanIso) a permutation π_i that minimizes $|\{x \in \{0,1\}^n \mid f(x) \neq g_i^{\pi_i}(x)\}|$. If $f(\pi_i^{-1}(-1^i 1^{n-i})) = -1$ for each i, the algorithm describing the Turing reduction "accepts" f as a tautology and otherwise it "rejects" f.

We now show the correctness of this reduction. If f is a tautology, then clearly for each π_i we have $f(\pi_i^{-1}(-1^i 1^{n-i})) = -1$.

Conversely, suppose f is not a tautology. We want to show that for some i, the oracle query to MinBooleanIso with f and g_i as inputs returns a permutation π_i such that $f(\pi_i^{-1}(-1^i 1^{n-i})) = 1$. Since f is not a tautology, the set $f^{-1}(1) = \{x \in \{\pm 1\}^n \mid f(x) = 1\}$ is nonempty. Let $|f^{-1}(1)| = N$. For any permutation π, either (i) it maps all the elements in $f^{-1}(1)$ incorrectly, in which case $|\{x \in \{\pm 1\}^n \mid f(x) \neq g_i^\pi(x)\}| = N + 1$, or (ii) it maps all except one element from $f^{-1}(1)$ incorrectly, in which case $|\{x \in \{\pm 1\}^n \mid f(x) \neq g_i^\pi(x)\}| = N - 1$. Suppose $x \in f^{-1}(1)$ is a string with i number of -1s. Then, there is a permutation π such that $|\{x \in \{\pm 1\}^n \mid f(x) \neq g_i^\pi(x)\}| = N - 1$ since any permutation that maps x to $-1^i 1^{n-1}$ has this property. Therefore, when the algorithm queries the MinBooleanIso oracle with f and g_i, it will return a π with this property. In this case, the pre-image of $-1^i 1^{n-1}$ will be in $f^{-1}(1)$. In other words, $f(\pi^{-1}(-1^i 1^{n-i})) = 1$ and the algorithm will reject. □

Corresponding to the minimization problem, we look at the *maximization problem*: Find π that maximizes $|\{x \in \{\pm 1\}^n \mid f(x) = g^\pi(x)\}|$. Of course computing an optimal solution to this problem is polynomial-time equivalent to MinBooleanIso. In the remainder of this section we design a simple approximate isomorphism algorithm for the *maximization* problem. Our simple algorithm is based on the method of conditional probabilities. We first examine how good a random permutation is as an approximate isomorphism. Then we describe a deterministic algorithm for computing a permutation with the same solution quality.

3.3.1 Estimating the guarantee of a random permutation

For Boolean functions f and g, we now estimate the random variable $|\{x \mid f(x) = g^{\pi}(x)\}|$ when the permutation π is picked uniformly at random from the symmetric group S_n.

Lemma 3.3.2. *Let $f : \{-1,1\}^n \to \{-1,1\}$ and $g : \{-1,1\}^n \to \{-1,1\}$ be δ-close Boolean functions for some $\delta > 0$. Then,*

$$\mathbb{E}_{\pi}\left[|\{x \mid f(x) = g^{\pi}(x)\}|\right] \geq \frac{\delta^2 2^n}{64\sqrt{n}}.$$

Proof. Let $s_i(f)$ denote the cardinality $|\{x \in \{0,1\}^n| \text{ wt}(x) = i, f(x) = 1\}|$ where $\text{wt}(x)$ is the hamming weight of the Boolean string x. Clearly, $s_i(f) \leq \binom{n}{i}$. For each $u \in \{0,1\}^n$ define the 0-1 random variable X_u which takes value 1 if and only if $f(u) = g^{\pi}(u)$ for $\pi \in S_n$ picked uniformly at random. If $\text{wt}(u) = i$, then

$$\Pr_{\pi}[X_u = 1] = \frac{s_i(g)}{\binom{n}{i}} f(u) + \frac{\binom{n}{i} - s_i(g)}{\binom{n}{i}} (1 - f(u)).$$

The sum $X = \sum_{u \in \{0,1\}^n} X_u$ is the random variable $|\{x | f(x) = g^{\pi}(x)\}|$ for a random permutation $\pi \in S_n$. We have

$$\begin{aligned}
\mathbb{E}_{\pi}[X] &= \sum_{i=0}^{n} \sum_{u:\text{wt}(u)=i} \frac{s_i(g)}{\binom{n}{i}} f(u) + \sum_{i=0}^{n} \sum_{u:\text{wt}(u)=i} \frac{\binom{n}{i} - s_i(g)}{\binom{n}{i}} (1 - f(u)) && (3.5)\\
&= \sum_{i=0}^{n} \frac{s_i(g)s_i(f)}{\binom{n}{i}} + \sum_{i=0}^{n} \frac{(\binom{n}{i} - s_i(g))(\binom{n}{i} - s_i(f))}{\binom{n}{i}} && (3.6)\\
&\geq \max_i \left(\frac{s_i(f)s_i(g)}{\binom{n}{i}}, \frac{(\binom{n}{i} - s_i(g))(\binom{n}{i} - s_i(f))}{\binom{n}{i}} \right). && (3.7)
\end{aligned}$$

Since f and g are δ-close for some constant $\delta > 0$, the fraction $\delta = \max_{\sigma \in S_n} |\{x | f(x) = g^{\sigma}(x)\}|/2^n$ is a constant (independent of n). For $0 \leq i \leq n$, define δ_i as the following:

$$\delta_i = |\{x \mid f(x) = g^{\tau}(x), \text{wt}(x) = i\}| / \binom{n}{i}.$$

Thus $\sum_{i=0}^{n} \delta_i \binom{n}{i} = \delta 2^n$ which we can write as

$$\sum_{i=0}^{\sqrt{n}} (\delta_i + \delta_{n-\sqrt{n}+i}) \binom{n}{i} + \sum_{i=n/2-\sqrt{n}}^{n/2+\sqrt{n}} \delta_i \binom{n}{i} = \delta 2^n.$$

Since each $\delta_i \leq 1$, $\sum_{i=0}^{\sqrt{n}} (\delta_i + \delta_{n-\sqrt{n}+i}) \binom{n}{i} \leq 2n^{\sqrt{n}+1} \leq 2^{2\sqrt{n}\log n}$ for sufficiently large n.

Let A denote the sum $\sum_{i=n/2-\sqrt{n}}^{n/2+\sqrt{n}} \delta_i \binom{n}{i}$. Then,

$$A \geq \delta 2^n \left(1 - \frac{2^{2\sqrt{n}\log n}}{\delta 2^n}\right) \geq \frac{\delta}{2} 2^n.$$

By averaging, there is some hamming weight i in the range $n/2 - \sqrt{n} \leq i \leq n/2 + \sqrt{n}$, such that

$$\delta_i \binom{n}{i} = |\{u \mid \mathrm{wt}(u) = i \text{ and } f(u) = g^\pi(u)\}| \geq \frac{\delta 2^n}{4\sqrt{n}}.$$

We fix this value of i and let S denote the set $\{u \mid \mathrm{wt}(u) = i \text{ and } f(u) = g^\pi(u)\}$. Assume without loss of generality that $|f^{-1}(1) \cap S| > \frac{\delta 2^n}{8\sqrt{n}}$ (Otherwise we consider $f^{-1}(0) \cap S$). Thus, we have $s_i(f) \geq |f^{-1}(1) \cap S| = |(g^\pi)^{-1}(1) \cap S| \geq \frac{\delta 2^n}{8\sqrt{n}}$. Similarly, $s_i(g) = \{u \mid \mathrm{wt}(u) = i \text{ and } g^\pi(u) = 1\} \supseteq (g^\pi)^{-1}(1) \cap S$. Hence $|(g^\pi)^{-1}(1) \cap S| \leq |\{u \mid \mathrm{wt}(u) = i \text{ and } g^\pi(u) = 1\}| = |\{u \mid \mathrm{wt}(u) = i \text{ and } g(u) = 1\}| = s_i(g)$. Therefore, both $s_i(f)$ and $s_i(g)$ are at least $\delta 2^n / 8\sqrt{n}$.

Combined with Equation 3.5 and using the inequality $\binom{n}{i} \leq \frac{2^n}{\sqrt{n}}$ for large enough n, we get the desired lower bound on $\mathbb{E}[X]$:

$$\mathbb{E}[X] \geq \frac{s_i(f) s_i(g)}{\binom{n}{i}} \geq \frac{\delta^2 2^{2n}}{64n\binom{n}{i}} \geq \frac{\delta^2 2^n}{64\sqrt{n}}.$$

□

Using the estimate for the expected value of the size of the set $\{x \mid f(x) = g^\pi(x)\}$, where π is picked at random, we get an approximate isomorphism algorithm which is given in the next theorem.

Theorem 3.3.3. *There is a deterministic $2^{O(n)}$ time algorithm that takes two Boolean functions $f, g : \{0,1\}^n \to \{0,1\}$ as input (give either by Boolean circuits or by black-box access) and outputs a permutation σ with the following property: If f and g^π are δ-close for some permutation π and constant δ, then*

$$|\{x \mid f(x) = g^\sigma(x)\}| \geq \Omega\left(\frac{2^n}{\sqrt{n}}\right).$$

Proof. Let X denote the random variable $|\{x \mid f(x) = g^{\pi}(x)\}$ where π is picked uniformly at random from S_n. From the previous lemma we know that the $\mathbb{E}_{\pi}[X] \geq \Omega(2^n/\sqrt{n})$. Now we show how to compute a permutation σ such that $|\{x|f(x) = g^{\sigma}(x)\}| \geq \mathbb{E}_{\pi}[X]$,

Let σ_i be an injective map $\sigma_i : \{1, 2, \ldots, i\} \to [n]$ and let $S_n^{\sigma_i}$ denote the set of permutations $\{\pi \in S_n \mid \pi(l) = \sigma(j) \text{ for } 1 \leq l \leq i\}$. Given the partial permutation σ_i defined on $\{1, 2, \ldots, i\}$, define random variables $X_{\sigma_i,u}$ for each $u \in \{0,1\}^n$ where $X_{\sigma_i,u} = 1$ if $f(u) = g^{\pi}(u)$ for π picked uniformly at random from $S_n^{\sigma_i}$. Let $X_{\sigma_i} = \sum_u X_{\sigma_i,u}$. Similar to Equation 3.5, we can write an expression for $\mathbb{E}[X_{\sigma_i}]$ and compute it exactly in time $2^{O(n)}$ for a given σ_i. For $j \in [n] \setminus \{\sigma_i(1), \sigma_i(2), \ldots, \sigma_i(i)\}$ let $\sigma_{i,j}$ denote the extension of σ_i that maps $i+1$ to j. In time $2^{O(n)}$ we can compute $\mathbb{E}[X_{\sigma_{i,j}}]$ for every j, and choose the permutation σ_{i+1} as that $\sigma_{i,j}$ which maximizes $\mathbb{E}[X_{\sigma_{i,j}}]$. In particular, this will satisfy $\mathbb{E}[X_{\sigma_{i+1}}] \geq \mathbb{E}[X_{\sigma_i}]$. Continuing this process until $i = n$ yields $\sigma_n = \sigma$ such that $|\{x \mid f(x) = g^{\sigma}(x)\}| \geq \mathbb{E}_{\pi}[X]$, where π is randomly picked from S_n. □

3.4 Summary and Open Problems

Motivated by the question whether Boolean Isomorphism has algorithms faster than Luks's $2^{O(n)}$ time algorithm [Luk99], we initiate the study of approximate Boolean Isomorphism. The main result of this chapter is an approximate isomorphism testing algorithm for Boolean functions whose Fourier spectrum is concentrated in a small set. The obvious question here, which we have been unable to answer until now, is if there are other properties that can be utilized to obtain approximation testing algorithms for other classes of Boolean functions.

Another interesting direction to pursue is whether there are good property testing algorithms for this problem. Tight bounds are known for property testing Boolean function isomorphism [AB10], though they make no assumption about the structural properties of the functions. If query complexity is measured, then the algorithm given in this chapter can be used to obtain a good tester. Unfortunately, the running time of the algorithm is $n!$ which makes it uninteresting. A concrete problem to look at this stage is whether the ideas in this chapter can be used to obtain a better tester for Boolean isomorphism.

An interesting question that remains at this point is whether there is a polynomial-time approximation scheme for approximate Boolean Isomorphism when the functions are given as Boolean formulas or circuits. It is open whether there is a constant factor approximation algorithm for approximate Boolean Isomorphism even when the functions are represented as 3-CNFs. More generally, it would be interesting to investigate how the

representation of the Boolean functions influences the computational complexity of computing approximate isomorphisms. We do now see how to approach this question.

In the next chapter, we will study how the representation of the Boolean functions that are given as input influences the complexity of exact Boolean Isomorphism.

Chapter 4

Boolean Isomorphism for some Representations

In this chapter and the next, we will study the Boolean Isomorphism problem (not the approximate but the exact version) for certain representation classes. The results in these chapters are motivated by the question: how closely is the complexity of Boolean Isomorphism for a particular representation class related to the satisfiability problem? More precisely, we seek answers to the following questions:

- Is it possible to efficiently transform Boolean Isomorphism for a particular representation to Graph Isomorphism (or bounded rank Hypergraph Isomorphism) if there is an efficient Satisfiability test for that representation? This seems the most natural approach to obtain an algorithm that runs faster than $O^*(2^{O(n)})$ for Boolean Isomorphism.
- Since bounded rank Hypergraph Isomorphism has a $2^{o(n)}$ time algorithm, a related question would be if Boolean Isomorphism for a particular representation can be solved as fast as Satisfiability or Equivalence for that representation.

As we described in the previous chapter, Hypergraph Isomorphism is polynomial-time reducible to Boolean Isomorphism for monotone DNFs. Therefore, even for classes of Boolean functions that have polynomial-time satisfiability algorithms, it is difficult to design algorithms for Boolean Isomorphism that run faster than the hypergraph isomorphism testing algorithm of Luks [Luk99].

An interesting question that we can ask here, is for which representations of Boolean functions that have efficient satisfiability algorithms, can we give solve Boolean Isomorphism

as fast as Graph Isomorphism? As we explained in Chapter 1, Boolean Isomorphism for 2-CNFs is already as hard as Graph Isomorphism.

With this question in mind, we study Boolean Isomorphism for two representations of Boolean functions that have efficient satisfiability testing algorithms: Horn-CNFs and decision trees. We also study Boolean Isomorphism for decision lists which is a more general class of representations (4.1.3).

Two well-studied representations of Boolean functions are 2-CNFs and Horn-CNFs. Recall that Horn-CNF formulas are conjunctive normal form formulas with the restriction that each clause contains at most one positive literal. It turns out that for both representations Boolean Isomorphism is polynomial-time equivalent to GI. The result for 2-CNF representation is already shown in [BHRV04]. We prove a somewhat stronger result for decision lists.

We also study Boolean Isomorphism for Boolean decision trees. Decision trees are a natural representation for Boolean functions and are fundamental to Boolean function complexity due to their conceptual simplicity. See, for example, the survey by Buhrman and de Wolf [BdW02] on complexity measures for Boolean functions and the central role of decision tree complexity in the field. Decision lists were introduced as a flexible representation for Boolean functions by Rivest [Riv87] in the context of machine learning. In the field of algorithmic learning theory, decision trees too have played a significant role in learnability of Boolean functions. The polynomial time PAC learning algorithm of decision trees given by Kushilevitz and Mansour [KM93] is an example.

4.1 Preliminaries

We start with some basic definitions that will be useful in this chapter.

Decision Trees

Definition 4.1.1. *A decision tree T_f on variables $X = \{x_1, \ldots, x_n\}$ is an ordered binary tree in which each leaf is labeled with a Boolean value and each inner node is labeled with a variable in X and has exactly two children. Any assignment $b_1, \ldots, b_n$ defines a path from the root of T_f to a leaf: At an inner node labeled with x_i, proceed to the left child if $b_i = 0$ and to the right child otherwise. The function value $T_f(b_1, \ldots, b_n)$ is the label of the leaf node reached along this path.*

Decision trees are a natural representation for Boolean functions and are fundamental to Boolean function complexity. The *size* $|T|$ of a decision tree T is the number of its leaves. We may assume that on the path from the root to any leaf, each variable occurs at most once as label of an inner node. Indeed, querying the same variable a second time will always yield the same result as before, so the second occurrence can be removed together with the subtree rooted at its non-reachable child without changing the represented function.

The satisfiability and equivalence problems for decision trees have simple polynomial-time algorithms. Given a decision tree T, the Boolean function represented by T is satisfiable if and only if one of the leaves of T is labeled with the constant 1. For checking the equivalence of Boolean functions f and g given as decision trees T_f and T_g, we can construct a decision tree T for the function $f \oplus g$. Two Boolean functions f and g are equivalent if and only if $f \oplus g$ is not satisfiable. To construct the decision tree T for $f \oplus g$, we attach the decision tree T_g to the leaves of T_f labeled with 0 and we attach the decision tree $T_{\overline{g}}$ (obtained by complementing the leaves of the decision tree T_g) to the leaves of T_f labeled with 1. We can then prune this decision tree to remove nodes with the same label in a path to obtain the decision tree T. To check equivalence it is sufficient to check if all the leaves of the decision tree T are labeled with the constant 0.

Thus, Boolean Isomorphism for decision trees, denoted DT-Iso, is in NP.

We give a polynomial time reduction from Boolean Isomorphism for *bounded rank decision trees* to Hypergraph Isomorphism for *bounded rank hypergraph*. One corollary of this is a $2^{\sqrt{s}(\log s)^{O(1)}}$ time algorithm for Boolean Isomorphism of size-s decision trees.

We first define the notion of rank for decision trees as given by Ehrenfeucht and Haussler in [EH89]. Let T be a decision tree and v be a node in T.

The *rank* of the node v, $\mathrm{rk}(v)$, is defined as follows,

$$\mathrm{rk}(v) = \begin{cases} 0 & \text{if } v \text{ is a leaf,} \\ \max\{\mathrm{rk}(v_1), \mathrm{rk}(v_2)\} & v_1 \text{ and } v_2 \text{ are children of } v \text{ with } \mathrm{rk}(v_1) \neq \mathrm{rk}(v_2), \\ \mathrm{rk}(v_1) + 1 & \text{otherwise.} \end{cases}$$

The rank of the decision tree T is rank of its root node. For a Boolean function f, the rank of f is defined as follows:

$$\mathrm{rk}(f) = \min\{\mathrm{rk}(T) \mid T \text{ is a decision tree that represents } f\}.$$

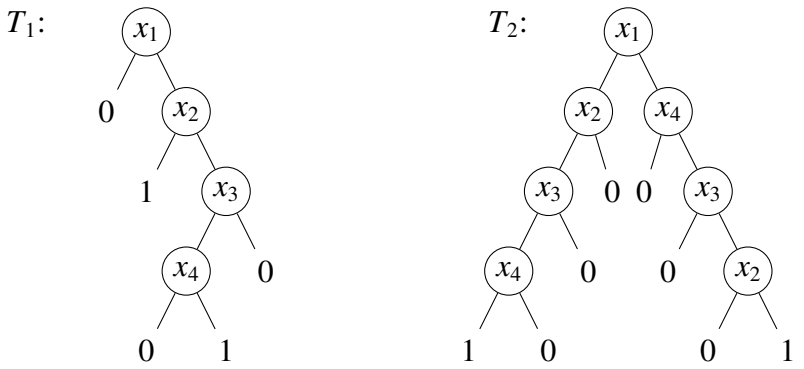

Figure 4.1: The decision tree T_1 computes the function $x_1 \wedge (\overline{x_2} \vee (\overline{x_3} \wedge x_4))$ and has rank 1. The decision tree T_2 computes the function $(x_1 \wedge x_2 \wedge x_3 \wedge x_4) \vee (\overline{x_1} \wedge \overline{x_2} \wedge \overline{x_3} \wedge \overline{x_4})$ and has rank 2.

In other words, the rank of a decision tree T is the depth of the largest full binary tree that can be embedded into T. The rank of a hypergraph is the maximum hyperedge size in the hypergraph. In Section 4.2, we prove the following theorem:

Theorem 4.1.2. *Let f and g be two Boolean functions of rank r given as decision trees T_1 and T_2. There is a deterministic algorithm running in time $n^{O(r)}$ that constructs two hypergraphs $\mathcal{H}_1$ and $\mathcal{H}_2$, each of which have hyperedges of size $O(r)$ such that $f \cong g$ if and only if $\mathcal{H}_1 \cong \mathcal{H}_2$.*

Along the way we prove that the isomorphism of rank-1 decision trees is complete for deterministic logspace. In Section 4.2, we prove a converse to Theorem 4.1.2, that Graph Isomorphism is polynomial-time reducible to Boolean Isomorphism for decision trees of rank 2.

Decision Lists

The next important class of functions that we study in this chapter are the class of decision lists, which were introduced by Rivest [Riv87] in learning theory.

Definition 4.1.3 ([Riv87])**.** *A C-decision list (C-DL) L, where C is a finite class of Boolean functions, is a sequence of pairs $\langle f_i, b_i \rangle_{i \leq m}$ where $b_i \in \{0,1\}$, $f_m = 1$, and for $i = 1, \ldots, m-1$, $f_i(x_1, \ldots, x_n) = g_i(x_{i_1}, \ldots, x_{i_k})$ for some $g_i \in C$, where $1 \leq i_1, \ldots, i_k \leq n$. For a Boolean assignment x, the decision list L has the value $L(x) = b_i$, where $i = \min\{j \geq 1 \mid f_j(x) = 1\}$.*

If $C = \{x_1, \overline{x_1}\}$, then C-DL coincides with rank-1 decision trees (defined in the previous chapter). Similarly, if C consists of conjunctions of r literals then every r-CNF or r-DNF

formula has a C-decision list. We call such decision lists r-decision lists (or r-DLs, in short). For $r \geq 3$, the satisfiability problem for r-DLs is clearly NP-complete, and the equivalence problem is coNP-complete. Furthermore, every rank-r decision tree of size s has an r-decision list of length $O(s)$ [Blu92]. Our results on the complexity of Boolean Isomorphism for C-DLs, denoted C-DL-Iso, are summarized below.

Example 4.1.4. *The function* $\mathsf{AND}(x_1, x_2, x_3, x_4) = x_1 \wedge x_2 \wedge x_3 \wedge x_4$ *can be represented as the* 1*-decision list* $\langle \neg x_1, 0\rangle, \langle \neg x_2, 0\rangle, \langle \neg x_3, 0\rangle, \langle \neg x_4, 0\rangle, \langle 1, 1\rangle$. *Any* k*-CNF* $\mathsf{F}(x_1, \ldots, x_n) = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ *can be represented as the* k*-decision lists* $\langle \neg C_1, 0\rangle, \langle \neg C_2, 0\rangle, \cdots, \langle \neg C_m, 0\rangle, \langle 1, 0\rangle$.

Horn-CNFs

Definition 4.1.5 (Horn-CNF)**.** *A CNF formula* $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ *is a Horn-CNF if each clause* C_i *is of the form* $x_{i_1} \wedge x_{i_2} \wedge \cdots \wedge x_{i_k} \Rightarrow x_{i'}$*, where* x_{i_j}*s are variables and* $x_{i'}$ *is either a variable or a Boolean constant. In other words, each clause of the Horn-CNF has at most one positive literal. We will call the variables* x_{i_j}*s as the* antecedent *and the literal* $x_{i'}$ *as the* consequent*.*

The satisfiability problem for Horn-CNFs is P-complete and hence the equivalence problem is solvable in polynomial time. We prove the following theorem.

Theorem 4.1.6. *Let* f *and* g *be two Boolean functions on* n *variables given as Horn-CNFs. There is an algorithm running in time* $n^{O(1)}$ *that constructs two hypergraphs* $\mathcal{H}_1$ *and* $\mathcal{H}_2$ *such that* $f \cong g$ *if and only if* $\mathcal{H}_1 \cong \mathcal{H}_2$*.*

Since hypergraph isomorphism is reducible to graph isomorphism, this also shows that testing isomorphism of Horn-CNFs is polynomial-time many-one reducible to graph isomorphism.

Further in the same section, we note a simple observation (already shown in [BHRV04]) that Boolean Isomorphism for 2-CNFs is reducible to Graph Isomorphism. We also exhibit a polynomial time reduction from Graph Isomorphism to Boolean Isomorphism for monotone 2-CNF.

By a *representation* of a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ we mean a finite description R for f, such that for any input $x \in \{0,1\}^n$ we can evaluate $R(x) = f(x)$ in time polynomial in the size of R. Examples of representations include circuits, branching programs, formulas, decision trees etc. Two representations R and R' are *equivalent* (denoted $R \equiv R'$) if they describe the same Boolean function.

Let π be a permutation of the input variables $x_1, x_2, \ldots, x_n$. Then f^π denotes the Boolean function $f(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$. Similarly, we assume that we can transform any representation R of the function f into a representation R^π for f^π by replacing each input variable x_i in R by $x_{\pi(i)}$.

Let $\mathcal{R}$ and $\mathcal{R}'$ be sets of representations of Boolean functions. A *permutation preserving normal form representation* (in short, *normal form*) for $\mathcal{R}$ is a mapping $N\colon \mathcal{R} \to \mathcal{R}'$ such that (i) $N_R \equiv R$ for any $R \in \mathcal{R}$, (ii) $R_1 \equiv R_2$ implies $N_{R_1} = N_{R_2}$, and (iii) for each permutation π we have $N_{R^\pi} = (N_R)^\pi$.

A *canonical form representation* for $\mathcal{R}$ is a mapping $C\colon \mathcal{R} \to \mathcal{R}'$ such that (i) for any $R \in \mathcal{R}$, the function represented by C_R is isomorphic to the one described by R, and (ii) for any two representations R_1 and R_2, the functions described by R_1 and by R_2 are isomorphic if and only if $C_{R_1} = C_{R_2}$.

Given a Boolean function f defined on the variables X, a variable $x \in X$ and a bit $b \in \{0, 1\}$, the restricted function $f|_{x \leftarrow b}$ is the Boolean function defined on the variables $X \setminus \{x\}$ that is obtained from f by fixing the value of x to b. For $X' \subset X$ we also use the notation $f|_{X' \leftarrow b}$ for the restricted function where all variables in X' are fixed to b.

4.2 Boolean Isomorphism for Decision Trees

This section consists of two parts. In Section 4.2.1, we show that Boolean Isomorphism for rank-1 Boolean functions is in polynomial time. In fact, we will give a polynomial-time algorithm for computing a canonical form representation for rank-1 Boolean functions. If the rank-1 function is given as a rank-1 decision tree, we show that Boolean Isomorphism is complete for deterministic logspace. In Section 4.2.2, we build on the rank-1 case to give a polynomial-time reduction from Boolean Isomorphism for bounded rank Boolean functions to Hypergraph Isomorphism for bounded rank hypergraphs. This yields a moderately exponential-time algorithm for Boolean Isomorphism for bounded rank decision trees.

We begin with a few simple observations.

Proposition 4.2.1. *Let f be a rank-r Boolean function on variables $x_1, \ldots, x_n$. Then there exists some variable x_i such that $f|_{x_i \leftarrow 0}$ or $f|_{x_i \leftarrow 1}$ has rank at most $r - 1$. Additionally, for all variables x_i the restricted functions $f|_{x_i \leftarrow 0}$ and $f|_{x_i \leftarrow 1}$ have rank at most r.*

Proof. Since f is a rank-r Boolean function, there is some decision tree T_f computing f

which is of rank r. Let x_i be the variable that the root of T_f is labelled with. Since the decision tree T_f is of rank r, one of the children of the root must have at most rank $r-1$. If it is the left child, then $f|_{x_i \leftarrow 0}$ has rank at most $r-1$; otherwise $f|_{x_i \leftarrow 1}$ has rank at most $r-1$.

The second part holds because fixing any variable to a constant does not increase the rank of T_f. □

Proposition 4.2.2. *Let T be a decision tree that represents some Boolean function f on the variables X, and let $x \in X$ and $b \in \{0,1\}$. Then a decision tree $T|_{x \leftarrow b}$ for the restricted function $f|_{x \leftarrow b}$ can be computed in time $O(|T|)$.*

Proof. To obtain $T|_{x \leftarrow b}$ from T, remove each inner node labelled with x along with the subtree rooted at its child selected by b. □

Proposition 4.2.3. *Given a decision tree T that represents some Boolean function f, it can be checked whether $f = 0$ or $f = 1$ in time $O(|T|)$.*

Proof. It suffices to check whether all leaves of T are labelled with the desired constant. □

Using these observations, we can minimize the rank of decision trees.

Theorem 4.2.4. *Given as input a decision tree T, we can check if the represented Boolean function f has rank r and, if so, construct a rank-r decision tree for it in time* $\mathrm{poly}(n^r \cdot |T|)$.

Proof. The algorithm is recursive. As base case, suppose $r = 1$ and let f be the function represented by the given decision tree T. Find a variable x and a bit b such that $f|_{x \leftarrow b}$ is constant; this can be checked using Propositions 4.2.2 and 4.2.3. If no such x and b exist, reject T as it has rank more than 1. Proceeding recursively with the function $f|_{x \leftarrow \bar{b}}$ (which has only $n-1$ variables), we can check if f is of rank 1 and also compute a rank-1 decision tree for it.

For checking if f has rank r, we sketch a simple recursive procedure: Find a variable x and a bit $b \in \{0,1\}$ such that $f|_{x \leftarrow b}$ is of rank at most $r-1$ (checked recursively). If $f|_{x \leftarrow \bar{b}}$ has rank at most r (checked recursively) then f is of rank at most r, else f has rank more than r. If no such variable exists then f has rank more than r.

By Proposition 4.2.1, the desired variables can always be found if f has rank r. The running time follows since if $t(n, r, s)$ is the time to check if an n-variate function represented by a decision tree T of size s is of rank r, then by the above algorithm we have the

following recurrence:

$$t(n, r, s) \leq 2n \cdot t(n-1, r-1, s-1) + t(n-1, r, s-1) + |T|.$$

It is easy to verify by induction that $t(n, r, s) = O((n^r s)^3)$. □

4.2.1 Boolean Isomorphism for Rank-1 Boolean Functions

In this section, we describe a normal form for rank-1 Boolean functions, whose structure allows to decide Boolean Isomorphism efficiently. We then show that when the input is a rank-1 decision tree, Boolean Isomorphism (and even computing canonical forms) can be done in logspace. To show that this is optimal, we give a reduction from a known logspace-complete problem.

4.2.1.1 Normal Form for Rank-1 Boolean Functions

Let f be a rank-1 Boolean function given by some decision tree T_f which is not necessarily rank 1. For Boolean constants $c, b \in \{0, 1\}$, let $X_{c,b}(f) = \{x \mid f|_{x \leftarrow b} = c\}$. By Propositions 4.2.2 and 4.2.3, these sets can be computed efficiently.

Next, we define $f_0 = f$ and, for $j \geq 1$, $f_j = f_{j-1}|_{V_{j,0}(f) \leftarrow 1, V_{j,1}(f) \leftarrow 0}$, where $V_{j,b}(f) = X_{j \bmod 2, b}(f_{j-1})$ for $b \in \{0, 1\}$. The variable set $\{x_1, x_2, \ldots, x_n\}$ is thus partitioned into $k \leq n$ subsets $V_{1,0}(f) \cup V_{1,1}(f), \ldots, V_{k,0}(f) \cup V_{k,1}(f)$. The *level* of a variable x is the index j such that $x \in V_{j,0}(f) \cup V_{j,1}(f)$.

The normal form of T_f is defined as $N_f = S_{1,0}(f), S_{1,1}(f), \ldots, S_{k-1,0}(f), S_{k-1,1}(f), \langle 1, k \bmod 2 \rangle$, where $S_{j,b}(f) = \{\langle x_i^b, j \bmod 2 \rangle \mid x_i \in V_{j,b}(f)\}$; we here use the notation $x_i^1 = x_i$ and $x_i^0 = \overline{x_i}$. See Figure 4.2 for an example.

This sequence N_f represents the same function as the decision list obtained from it by replacing each $S_{j,b}(f)$ with the list of pairs contained in it. Clearly, N_f is equivalent to T_f and only depends on f (i.e., not on the structure of T_f). The next two lemmas imply that N_f is a permutation preserving normal form representation for T_f.

Lemma 4.2.5. *Suppose f and g are two isomorphic rank-1 Boolean functions and π is an isomorphism from f to g. For any input variable x, if x is in level j for f then variable $\pi(x)$ is in level j for g.*

Proof. For all bits $c, b \in \{0, 1\}$, we have $f|_{x \leftarrow b} = c$ if and only if $g|_{\pi(x) \leftarrow b} = c$ as π is an

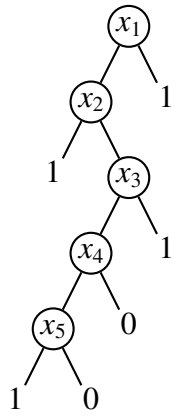

Figure 4.2: A rank-1 function f where $V_{1,1}(f) = \{x_1, x_3\}$, $V_{1,0}(f) = \{x_2\}$, $V_{2,1}(f) = \{x_4, x_5\}$

isomorphism; so π maps $X_{c,b}(f)$ to $X_{c,b}(g)$. This implies that π is an isomorphism from f_j to g_j for all levels j.

Now let x be a variable in level j for f, i.e., there is a bit $b \in \{0, 1\}$ such that $x \in X_{j \bmod 2,b}(f_{j-1})$. By the above observation, we have $\pi(x) \in X_{j \bmod 2,b}(g_{j-1})$, so $\pi(x)$ is in level j for g. □

This implies that the number of variables in each level coincides for two isomorphic Boolean functions of rank 1. The next lemma is in the converse direction.

Lemma 4.2.6. *Let T_f and T_g be decision trees for two Boolean functions f and g of rank* 1*, defined on the n variables $x_1, \ldots, x_n$. Let N_f and N_g be the corresponding normal form sequences obtained as in the discussion above. Suppose for each level j, $|V_{j,0}(f)| = |V_{j,0}(g)|$ and $|V_{j,1}(f)| = |V_{j,1}(g)|$, then f and g are isomorphic.*

Proof. For each level j, we have $|V_{j,0}(f)| = |V_{j,0}(g)|$ and $|V_{j,1}(f)| = |V_{j,1}(g)|$, so there are bijections between the sets $V_{j,0}(f)$ and $V_{j,0}(g)$ and between $V_{j,1}(f)$ and $V_{j,1}(g)$; we may assume that each of these bijections preserves the relative order of the variables w.r.t. the indices. As these variable sets are disjoint, these bijections can be combined into a single bijection π. Rename the variables in N_f according to π; then N_f^π and N_g are identical because of the way they are sorted. Since the normal form for the function f^π is obtained from the normal form of f by renaming the variables according to π, this implies $f^\pi = g$ and thus $f \cong g$. □

Hence, the defined sequences are normal forms for rank-1 functions and in time polynomial in the size of the input decision tree we can compute the normal form. Given

T_f and T_g, Lemma 4.2.6 shows that by comparing the sizes of the sets $V_{j,0}$ and $V_{j,1}$ for the two functions we can check if the Boolean functions are isomorphic or not. This gives us the following theorem.

Theorem 4.2.7. *Given Boolean functions of rank* 1 *by decision trees, there is a polynomial time algorithm that checks if the functions are isomorphic.*

4.2.1.2 Isomorphism of Rank-1 Decision Trees in Logspace

We now show that if the rank-1 function f is given as a decision tree T_f which is of rank 1, then the canonization problem is in logspace. We first show how the sets $V_{j,b}(f)$ can be computed in logspace.

For each internal node of T_f, at least one of its children is a leaf (labeled by a constant). We can partition the internal nodes in the tree into subsets $L_1, \ldots, L_m$, where L_1 has consecutive nodes starting from the root that have a leaf child labelled with 1, L_2 is the next set of consecutive nodes with a leaf child labelled with 0, and so on. We further classify the variables in each L_j into the subset $L_{j,0}$ of nodes in L_j whose left child is a leaf and subset $L_{j,1}$ of nodes whose right child is a leaf. These sets can be computed in logspace by inspection of the input decision tree.

Lemma 4.2.8. $L_{j,b} = V_{j,b}(f)$ *for all levels* j *and bits* $b \in \{0, 1\}$.

Proof. Since for each $x \in L_{1,b}$, the restriction $f|_{x \leftarrow b}$ is 1, we have $L_{1,b} \subseteq V_{1,b}(f)$. No variable outside the set $L_{1,b}$ has this property, so $L_{1,b} = V_{1,b}(f)$. A simple inductive argument generalizes this to the higher levels. □

Together with Lemma 4.2.6 this gives a logspace isomorphism test for rank-1 decision trees. With the following theorem, we observe that this can be strengthened to canonization.

Theorem 4.2.9. *Given a rank-1 decision tree* T*, a canonical form* C_T *of* T *can be computed in logspace, i.e.,* $C_T = C_{T'}$ *whenever* T *and* T' *describe isomorphic functions.*

Proof. To obtain C_T from T, we first order the variables of L_j such that all the nodes whose left child is a constant come first followed by the nodes whose right child is a constant. We do this for all the sets $L_1, \ldots, L_m$. Now starting from the root, rename the variables with the root node getting the variable x_1 followed by x_2 and so on. The resulting decision tree C_T can clearly be computed in logspace. As C_T only depends on the sizes of

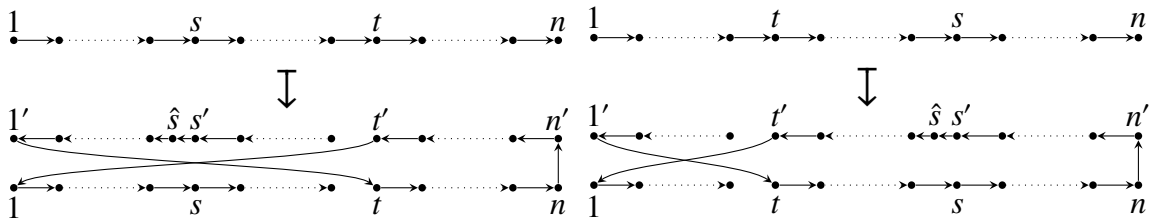

Figure 4.3: The reduction from Ord to DiOddPathCenter

the sets $L_{j,b}$, Lemmas 4.2.6 and 4.2.8 imply that decision trees that represent isomorphic functions receive the same canonical form. □

4.2.1.3 Hardness for Logspace

We now show the logspace completeness. We will give a reduction from the problem Ord, which is known to be complete for L [Ete97]. The input to Ord is a directed path P given as a set of edges and two vertices s and t. The problem is test if s occurs before t is the path P.

We will first reduce Ord to the DiOddPathCenter problem which is defined as follows: Given a directed path P with an odd number of vertices and a vertex $u \in P$, test if u is the center of the path P.

Lemma 4.2.10. DiOddPathCenter *is* L*-complete.*

Proof. The problem can easily be solved in logspace. To prove the hardness, we reduce from Ord using $(P, s, t) \to (P', n)$ as reduction, where n is the vertex having no successor and 1 is the vertex having no predecessor in P, and where P' is defined by

$$\begin{aligned} V(P') &= V(P) \cup \{i' \mid i \in V(P)\} \cup \{\hat{s}\} \\ E(P') &= \{(i,j) \mid (i,j) \in E(P) \wedge j \neq t\} \cup \{(j',i') \mid (i,j) \in E(P) \wedge j \notin \{s,t\}\} \\ &\quad \cup \{(\hat{s},i') \mid (i,s) \in E(P)\} \cup \{(s',\hat{s}), (t',1), (1',t), (n,n')\}\,. \end{aligned}$$

The path P' consist of a forward and a reversed copy of P that are joined together, where the part before the first copy of t is swapped with the part after the second copy of t, and where the second copy of s is duplicated; see Fig. 4.3 for an illustration. If s precedes t in P (left side), then n is the center of P', but if t precedes s then n' is the center of P' (right side). □

We now describe how to construct two decision trees T_1 and T_2 from an instance P of DiOddPathCenter: For each $v \in V$ there is a variable x_v, and both T_1 and T_2 contain one internal node for each x_v. If v is the successor of v' in P, x_v becomes the right child of $x_{v'}$ in T_1, and $x_{v'}$ becomes the right child of x_v in T_2. The right child of x_v, where v is the vertex without successor (or without predecessor, respectively) is a leaf labeled with 1. In both trees, the left child of x_u is a leaf labeled with 1. The left children of all other variables are leaves labeled with 0.

Lemma 4.2.11. *Let T_1 and T_2 be the decision trees constructed from an instance (P, u) of* DiOddPathCenter. *Let f_1 and f_2 be the functions represented by the decision trees, respectively. Then, $f_1 \cong f_2$ if and only if $(P, u) \in$* DiOddPathCenter.

Proof. For the purposes of this proof, identify the vertices of P with the integers $1, \ldots, n$ such that the vertex i is the successor of $i - 1$. To prove the lemma, it is sufficient to show that f_1 is isomorphic to f_2 if and only if $u = (n + 1)/2$. Since T_1 and T_2 are rank 1 decision trees, Lemmas 4.2.5 and 4.2.6 imply that the functions represented by them are isomorphic if and only if $|V_{i,b}(f_1)| = |V_{i,b}(f_2)|$ for all i and b. In the case of T_1, $V_{1,0}(f_1) = \{1, \ldots, u-1\}, V_{1,1}(f_1) = \emptyset, V_{2,0}(f_1) = \{u\}, V_{2,1}(f_1) = \emptyset, V_{3,0}(f_1) = \{u+1, \ldots, n\}, V_{3,1}(f_1) = \emptyset$ and $V_{i,b}(f_1) = \emptyset$ for any larger i. Similarly for T_2, $V_{1,0}(f_2) = \{n, \ldots, u + 1\}, V_{1,1}(f_2) = \emptyset, V_{2,0}(f_2) = \{u\}, V_{2,1}(f_2) = \emptyset, V_{3,0}(f_2) = \{u - 1, \ldots, 1\}, V_{3,1}(f_2) = \emptyset$ and $V_{i,b}(f_2) = \emptyset$ for any larger i. Thus $f_1 \cong f_2$ if and only if $u = (n + 1)/2$. □

4.2.2 Isomorphism of Rank-r Boolean Functions

Like for rank-1 decision trees, we first give a normal form representation for rank-r decision trees. Next, we exploit the structure of this normal form to give a reduction from the Boolean Isomorphism problem for bounded rank decision trees to Hypergraph Isomorphism for bounded rank hypergraphs. Concluding this section, we show that Graph Isomorphism is reducible to Boolean Isomorphism for rank-r decision trees, for $r \geq 2$.

4.2.2.1 Normal Form for Bounded Rank Boolean Functions

Let T_f be a decision tree for a function f of rank r. For a bit $b \in \{0, 1\}$, let $X_b(f)$ be the subset of variables x_i such that $f|_{x_i \leftarrow b}$ has rank at most $r - 1$. Let $f_0 = f$, and for $j \geq 1$ let $f_j = f_{j-1}|_{X_0(f_{j-1}) \leftarrow 1, X_1(f_{j-1}) \leftarrow 0}$, and also define the variable sets $V_{j,b}(f) = X_b(f_{j-1})$, for $b \in \{0, 1\}$. Again, the variables of f are partitioned into $V_{1,0}(f) \cup V_{1,1}(f), \ldots, V_{k,0}(f) \cup V_{k,1}(f)$.

Generalizing the normal form for rank-1 decision trees, which is a list of pairs that comprise of one literal and one constant each and that are grouped into sets, the normal form for f is a list N_f of pairs that comprise of one literal and one normal form of a Boolean function of rank $r-1$ that are grouped into sets. We define the normal form N_f as the sequence of sets $S_{1,0}(f), S_{1,1}(f), \ldots, S_{k,0}(f), S_{k,1}(f)$, where each set $S_{j,b}(f)$ is constructed from $V_{j,b}(f)$ as follows. For each variable $x_i \in V_{j,b}(f)$, include $\langle x_i^b, N_{f|_{x_i \leftarrow b}} \rangle$ in $S_{j,b}(f)$, where $x_i^1 = x_i$ and $x_i^0 = \overline{x_i}$, and where $N_{f|_{x_i \leftarrow b}}$ is defined recursively based on the decreasing rank. To evaluate N_f on a given assignment a, find the smallest j such that $S_{j,b}(f)$ contains a pair whose first component is a literal satisfied by a, and recursively evaluate the normal form in the second component of this pair.

Since for each x_i, checking if $f|_{x_i \leftarrow b}$ has rank at most $r-1$ and to compute a decision tree for it takes $\text{poly}(n^{r-1}|T|)$ time by Theorem 4.2.4, and since this process has to be repeated for at most n steps, the normal form can be constructed in time $\text{poly}(n^r|T|)$.

Lemma 4.2.12. *Given a decision tree T_f that represents a Boolean function f of rank r, a normal form representation N_f for f can be computed in time* $\text{poly}(n^r|T|)$.

Proof. Let N_f be the normal form described above. Clearly, N_f represents the function f. Also, N_f only depends on the function f (and not the structure of T_f), so equivalent decision trees T and T' receive the same normal form $N_T = N_{T'}$. If π is an isomorphism from f to g, then a similar argument as in Lemma 4.2.5 shows that $x \in V_{j,b}(f)$ implies $\pi(x) \in V_{j,b}(g)$. Additionally, $(f_j)^\pi = g_j$ and $(f_{j-1}|_{x \leftarrow b})^\pi = g_{j-1}|_{\pi(x) \leftarrow b}$, so $(N_{f_{j-1}|_{x \leftarrow b}})^\pi = N_{g_{j-1}|_{\pi(x) \leftarrow b}}$. This implies $(S_{j,b}(f))^\pi = S_{j,b}(g)$. Thus N_f is indeed a permutation preserving normal form.

To analyze the running time, let $t(n, r, |T|)$ denote the time for constructing the normal form of an n-variate Boolean function of rank r given by a decision tree T. For each variable x_i, for each $\ell \in \{1, \ldots, r-1\}$, we have to spend at most $t(n-1, r-1, |T|-1)$ time to construct the normal form for $f|_{x_i \leftarrow b}$ if $f|_{x_i \leftarrow b}$ is of rank ℓ and at most another $t(n-1, r, |T|-1)$ time to construct the normal form for $f|_{x_i \leftarrow \overline{b}}$. To obtain the decision tree $T|_{x_i \leftarrow b}$, we have to spend $|T|$ time. This gives the following recurrence for $t(n, r, |T|)$:

$$t(n, r, |T|) \leq 2nr \cdot t(n-1, r-1, |T|-1) + t(n-1, r, |T|-1) + |T|.$$

It can be verified that $t(n, r) = n^{cr}|T|^c$ satisfies this recurrence for a constant $c \geq 3$. □

4.2.2.2 Reduction to Hypergraph Isomorphism

We now describe our reduction of rank-r decision tree isomorphism to bounded rank hypergraph isomorphism, where the rank of a hypergraph is the maximum size of any hyperedge in it.

Given a rank-r Boolean function as a decision tree, we first construct the normal form for f, N_f in time $n^{O(r)}$ as described earlier. The next step is to construct a vertex-colored hypergraph corresponding to the normal form. We will encode all the information in the normal form using hyperedges. The construction is inductive.

*Rank-*1 *functions:* The case of rank-1 functions is easy, since the normal form for a rank-1 Boolean function consists of a decision tree where for each node, one of its children is a constant. In the hypergraph corresponding to the rank-1 function f, for each variable x_i that appears in the normal form we add a vertex v_i. Add the vertices (i, b) where $1 \leq i \leq n$ and $b \in \{0, 1\}$. We also add two vertices $\mathbf{0}$ and $\mathbf{1}$ corresponding to the constants. Now for each variable x_i, if $x_i \in V_{j,b}(f)$ and one of its children is labeled with the constant c, add the hyperedge $\{v_i, (j, b), c\}$ in the hypergraph. We color all the vertices corresponding to the variables with one color and each (j, b) with a separate color. The vertices $\mathbf{0}$ and $\mathbf{1}$ are colored with different colors as well. Call the resulting rank-3 hypergraph $\mathcal{H}_f$. We have the following lemma.

Lemma 4.2.13. *Let f and g be Boolean functions of rank 1 given by decision trees, and let $\mathcal{H}_f$ and $\mathcal{H}_g$ be the hypergraphs constructed as above. Then, f and g are isomorphic as functions if and only if the hypergraphs $\mathcal{H}_f$ and $\mathcal{H}_g$ are isomorphic.*

Proof. If f and g are rank 1 functions that are isomorphic, then we have seen that $|V_{j,b}(f)| = |V_{j,b}(g)|$ for all j, b. In the construction of the hypergraph, for each variable x_i, there are fixed j, b, c such that $\{v_i, (j, b), c\}$ is a hyperedge. Also, the color of the vertex (j, b) is different for different values of j, b. Thus the hypergraphs are isomorphic if and only if the sets $V_{j,b}(f)$ and $V_{j,b}(g)$ are of the same cardinality, which proves the lemma. □

Rank-r functions: Let f be a rank-r function, and let $N_f = \langle l_i, N_{x_i} \rangle_{i \leq k}$, where $k \leq n$ and $l_i \in \{x_i, \overline{x_i}\}$, be the normal form for f. The vertex set for the hypergraph $\mathcal{H}_f$ is $\{u_1, \ldots, u_n\} \cup \{v_1^d, \ldots, v_n^d \mid 1 \leq d \leq r\} \cup \{(l, i, b, j) \mid 1 \leq l \leq r,\ 1 \leq i \leq n,\ 1 \leq j \leq r,\ b \in \{0, 1\}\} \cup \{\mathbf{0}, \mathbf{1}\}$. Intuitively, the vertices $u_1, \ldots, u_n$ will encode the variables $x_1, \ldots, x_n$ and $v_1^1, \ldots, v_n^1$ will encode the variables $x_1, \ldots, x_n$ at the outermost level in $\langle l_1, N_{x_i} \rangle$ pairs. Let $\mathcal{H}_i$ denote the hypergraph encoding N_{x_i}, constructed inductively. The vertex set of $\mathcal{H}_i$ will be $\{v_1^d, \ldots, v_n^d \mid 2 \leq d \leq r\} \cup \{(l, i, b, j) \mid 1 \leq l \leq r - 1,\ 1 \leq i \leq n,\ b \in \{0, 1\},$

$2 \leq j \leq r\} \cup \{\mathbf{0}, \mathbf{1}\}$. We define the edge set for $\mathcal{H}_f$ as follows: For every $\langle l_i, N_{x_i} \rangle$ in the normal form and every edge $e \in \mathcal{H}_i$, we include $e \cup \{v_i^1\} \cup \{(l, j, b, 1)\}$ in the edge set if $x_i \in V_{j,b}^l(f)$ (where b encodes whether l_i is x_i or $\overline{x_i}$) and the edges $\{u_i, v_i^1\}$ for all i. Assume, inductively, that $\mathcal{H}_i$ is of rank at most $2(r-1)+1$. Then clearly H_f is of rank at most $2r+1$. If $f \cong g$ via $\pi \in S_n$, then since N_f and N_g are their normal form representations $(N_f)^\pi = N_g$, where $(N_f)^\pi$ is obtained by replacing x_i by $x_{\pi(i)}$ for all i in N_f. By induction on the rank r, we can easily argue that there is a $\pi \in S_n$ such that $(N_f)^\pi = N_g$ if and only if the hypergraphs H_f and H_g are isomorphic.

Lemma 4.2.14. *Let f and g be Boolean functions of rank r given by decision trees. Let $\mathcal{H}_f$ and $\mathcal{H}_g$ be the hypergraphs constructed as above. Then, f and g are isomorphic as functions if and only if the hypergraphs $\mathcal{H}_f$ and $\mathcal{H}_g$ are isomorphic.*

Proof. Suppose f and g are isomorphic functions. From the construction of the normal form, we know that any isomorphism maps the vertices in $V_{j,b}^l(f)$ to the vertices in the set $V_{j,b}^l(g)$. Let π be such an isomorphism. Therefore we know that for $x_i \in V_{j,b}^l(f)$, $\pi(x_i) \in V_{j,b}^l(g)$. We will show that π induces an isomorphism between the hypergraphs. By induction hypothesis, the Boolean function corresponding to N_{x_i} is isomorphic to $N_{\pi(x_i)}$. Thus, extending π with the identity on all non-variable vertices results in an isomorphism between the hypergraphs $\mathcal{H}_{x_i}$ and $\mathcal{H}_{\pi(x_i)}$. Since this is true for all x_is, π induces an isomorphism between the hypergraphs $\mathcal{H}_f$ and $\mathcal{H}_g$.

To prove the other direction, let π be an isomorphism between the hypergraphs $\mathcal{H}_f$ and $\mathcal{H}_g$. Then, for any vertex v_i corresponding to a variable x_i at depth 1, the hypergraphs $\mathcal{H}_{x_i}$ and $\mathcal{H}_{\pi(x_i)}$ are isomorphic. By induction hypothesis, this implies that the functions represented by the normal forms N_{x_i} and $N_{\pi(x_i)}$ are isomorphic. Since this is true for all the vertices v_i corresponding to depth 1 variables x_i, π (restricted to the variables of f) is an isomorphism for the Boolean functions f and g. The base case of the induction is when the decision trees are of rank 1 and this is taken care of in Lemma 4.2.13. □

According to the construction, the hypergraph $\mathcal{H}_f$ corresponding to the rank-r function f has $2nr$ vertices and rank $2r+1$. The size of the hypergraph is at most $n^{O(r)}$ since any rank-r Boolean function has a rank-r decision tree of size $n^{O(r)}$. In particular, the normal form that we construct is of size at most $n^{O(r)}$. We formulate these observations in the following theorem.

Theorem 4.2.15. *Let f and g be Boolean functions of rank r given by decision trees T_f and T_g. There is an algorithm running in time $n^{O(r)}$ that outputs two hypergraphs $\mathcal{H}_f$ and $\mathcal{H}_g$ of rank $2r+1$ and size $n^{O(r)}$ such that f and g are isomorphic if and only if the hypergraphs $\mathcal{H}_f$ and $\mathcal{H}_g$ are isomorphic.*

Since any decision tree of size s has rank at most $O(\log s)$, it has a normal form representation of size $n^{O(\log s)}$ which can be computed in $n^{O(\log s)} \cdot s^{O(1)}$. Hence we have the following corollary.

Corollary 4.2.16. *Let f and g be two decision trees of size s. There is an $s^{O(\log s)}$ time algorithm which computes hypergraphs $\mathcal{H}_f$ and $\mathcal{H}_g$ of logarithmic rank and size $s^{O(\log s)}$ such that f and g are isomorphic if and only if $\mathcal{H}_f \cong \mathcal{H}_g$.*

Combining this with the isomorphism algorithm for hypergraphs of bounded rank due to Babai and Codenotti [BC08], we observe the following:

Corollary 4.2.17. *Given two Boolean functions f and g as decision trees of size s, there is a $2^{\sqrt{s}(\log s)^{O(1)}}$ time algorithm to check if $f \cong g$.*

4.2.2.3 GI-Hardness of DT-Iso

We will show that isomorphism testing even for rank-2 decision trees is GI-hard.

Let $G = (V, E)$ be a graph with $V = \{v_1, v_2 \ldots, v_n\}$ and $E = \{e_1, e_2, \ldots, e_m\}$. We encode G as a Boolean function f_G on $n + m$ Boolean variables $v_1, \ldots, v_n$ and $e_1, \ldots, e_m$ as follows: $f_G(e_1, \ldots, e_m, v_1, \ldots, v_n) = 1$ if and only if exactly three variables e_i, v_j, v_k are 1, all remaining variables are 0, and $e_i = \{v_j, v_k\} \in E$. Here the Boolean variables v_i and e_j correspond, by abuse of notation, to elements of $V \cup E$. We can write f_G as $f_G = \bigvee_{e=\{u,v\}}(e \wedge (\bigwedge_{e' \neq e} \overline{e'}) \wedge u \wedge v \wedge (\bigwedge_{w \neq u,v} \overline{w}))$.

Lemma 4.2.18. *For any graph $G = (V, E)$, the function f_G is of rank 2 and can be represented by a rank-2 decision tree of size $O(|E|^2|V|)$.*

Proof. Note that if any edge variable e is set to 1 where $e = \{u, v\}$, all the terms in f_G disappear, except the one where the variable appears un-negated. Thus, $f_G|_{e \leftarrow 1} = \bigwedge_{e' \neq e} \overline{e'} \wedge u \wedge v \wedge \bigwedge_{w \neq u,v} \overline{w}$. Since $f_G|_{e \leftarrow 1}$ is a conjunction of literals, it is a rank 1 function. Since f_G is zero if all the edge variables are set to 0, this proves that f_G is a rank-2 function. □

Theorem 4.2.19. *Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be two graphs in which all vertices have at least two neighbors, and let f_G and f_H be the functions as defined above. Then, $G \cong H$ if and only if $f_G \cong f_H$.*

Proof. The function f_G encodes the graph G in the sense that for an assignment a to the variables, $f_G(a) = 1$ exactly if a encodes an edge $e = \{u, v\} \in E_G$, i.e., $a_e = a_u = a_v = 1$ and $a_x = 0$ for all $x \in (V_G \cup E_G) \setminus \{e, u, v\}$.

Any isomorphism π from G to H can be extended to map each edge $e = \{u, v\} \in E_G$ to $\pi(e) = \{\pi(u), \pi(v)\}$. Then π sends the satisfying assignments of f_G to the satisfying assignments of f_H, implying that $f_G \cong f_H$.

Conversely, if π is an isomorphism from f_G to f_H, it induces a bijection between the satisfying assignments of the two functions. As a variable is an edge variable if and only if it occurs in only one satisfying assignment, π maps edge variables to edge variables and vertex variables to vertex variables. It follows that π restricted to V_G is an isomorphism from G to H. □

As any pair of graphs can be modified to meet the degree requirement of Theorem 4.2.19 by adding two universal vertices to each graph, we have the following corollary.

Corollary 4.2.20. GI $\leq_p^m$ DT-Iso.

4.3 Boolean Isomorphism for Decision Lists

In this section, we consider C-DL isomorphism. We first observe that satisfiability of C-DLs is related to the Constraint Satisfaction Problem (CSP) where the constraints come from the class C.

Definition 4.3.1. *A* constraint *of arity k is a Boolean function $C\colon \{0,1\}^k \to \{0,1\}$. For a constraint C of arity k and variables $x_{i_1}, \ldots, x_{i_k}$ (not necessarily different), the corresponding* constraint application *is the Boolean function $C(x_{i_1}, \ldots, x_{i_k})$. For a finite class C of constraints, a* C-CSP instance *I is a set of applications of constraints in C and represents the conjunction of these constraint applications. A constraint C is called*

- 0-valid*, if $C(\mathbf{0}) = 1$,*
- 1-valid*, if $C(\mathbf{1}) = 1$,*
- Horn*, if it is a Horn-CNF, i.e., each clause has at most one positive literal,*
- anti-Horn*, if it is an anti-Horn-CNF, i.e., each clause has at most one negative literal,*
- bijunctive*, if it is a 2-CNF, and*
- affine*, if it is a conjunction of parities.*

A class C of constraints is called 0-valid, 1-valid, Horn, anti-Horn, bijunctive or affine, if every constraint in it has the respective property; and it is called Schaefer *if it is Horn, anti-Horn, bijunctive or affine.*

Schaefer proved the following dichotomy theorem regarding the satisfiability of CSP instances.

Theorem 4.3.2 ([Sch78])**.** *Let C be a class of constraints. The satisfiability problem for C-CSP instances is in* P *if C is 0-valid, 1-valid or Schaefer, and* NP*-complete otherwise.*

Böhler et al. considered Boolean Isomorphism for CSP instances.

Theorem 4.3.3 ([BHRV04])**.** *Let C be a class of constraints. Boolean Isomorphism for C-CSP instances is polynomial-time reducible to* GI *if C is Schaefer, and* coNP*-hard otherwise.*

Recall that in a C-DL $L = \langle f_1, b_1\rangle, \ldots, \langle f_m, b_m\rangle$ each f_i is required to be isomorphic to some function $C \in C$ (cf. Definition 4.1.3), and thus can be viewed as application of the constraint C.

For a class of constraints C, we define its complement as $\overline{C} = \{\neg C \mid C \in C\}$, and observe the following.

Lemma 4.3.4. *For any C-CSP instance I, there is an equivalent $\overline{C}$-DL L_I.*

Proof. Given a C-CSP instance $I = \{C_i(x_{i,1}, \ldots, x_{i,k_i}) \mid i \in \{1, \ldots, m\}\}$, we define

$$L_I = \langle \neg C_1(x_{1,1}, \ldots, x_{1,k_1}), 0\rangle, \ldots, \langle \neg C_m(x_{m,1}, \ldots, x_{m,k_m}), 0\rangle, \langle 1, 1\rangle\,.$$

The decision list L_I represents the function $\bigwedge_{i=1}^{m} C_i(x_{i,1}, \ldots, x_{i,k_i})$ and thus is equivalent to I. □

Combining this lemma with Theorem 4.3.3, we observe that C-DL satisfiability is coNP-hard if $\overline{C}$ is not Schaefer. The interesting cases of C-DL Boolean Isomorphism are thus those where $\overline{C}$ is Schaefer, i.e., where C consists of either (i) disjunctions of conjunctions of literals of which at most one is negative, (ii) disjunctions of conjunctions of literals of which at most one is positive, (iii) disjunctions of parities of literals or (iv) disjunctions of conjunctions of two literals.

In Section 4.3.2, we show that in all these cases, C-DL Boolean Isomorphism is reducible to GI. Moreover, when C consists of parities of two literals, Boolean Isomorphism is in

polynomial time. We refer to this last case as 2⊕-DL and consider it in Section 4.3.1. In Section 4.3.3 we show that Graph Isomorphism is reducible to C-DL-Iso when C is any of the above four classes (with $k \geq 3$ in the case of parities). This shows the Schaefer-type dichotomy for Boolean Isomorphism of decision lists.

4.3.1 Parities of size 2

Let L be a 2⊕-DL, i.e., L is given by a sequence of pairs $\langle p_i, b_i \rangle$ where $b_i \in \{0, 1\}$ and each p_i is a parity of two literals. We say that a pair $\langle p_i, b_i \rangle$ *fires* on an assignment x, if i is the least index such that $p_i(x) = 1$. Let f_L denote the function represented by L.

We first construct a normal form representation for 2⊕-DLs to obtain an equivalent decision list where the tuples are partitioned into the sets $B_1, \ldots, B_m$, where the second component of each pair in B_k is 1 if k is odd, and is 0 if k is even (in this normal form the set B_1 could possibly be empty). We then exploit the structure of the normal form to give a polynomial time algorithm for the 2⊕-DL Boolean Isomorphism.

We now explain the normal form N_L (which is also a 2⊕-DL) for a given 2⊕-DL L. The idea is that each B_k includes as many tuples as possible, even if some of them are redundant, as this choice does not depend on the order of the variables. For all pairs of literals l_i and l_j, such that L evaluates to 1 under all assignments with $l_i \oplus l_j = 1$, we add $\langle l_i \oplus l_j, 1 \rangle$ to the set B_1. We can find such pairs by replacing all occurrences of l_j by $\overline{l_i}$ in the decision list and checking if it represents the constant function 1. To compute B_k for $k \geq 2$, find literals l_i and l_j (whose parity was not included in some B_r with $r < k$) such that L evaluates to $k \bmod 2$ under all assignments that satisfy $(l_i \oplus l_j) \wedge \bigwedge_{\langle p, r \bmod 2 \rangle \in B_r, r<k} \neg p$. For each such pair we include $\langle l_i \oplus l_j, k \bmod 2 \rangle$ in B_k. We define the normal form N_L for L by $N_L = B_1, B_2, \ldots, B_{m-1}, \langle 1, m \bmod 2 \rangle$; it represents the same function as the 2⊕-DL that results from replacing each set B_k with the sequence of the pairs that are contained in it. The following lemma summarizes this normal form construction.

Lemma 4.3.5. *If L is a 2⊕-DL then N_L is a permutation preserving normal form representation for L. Moreover, N_L is computable in time polynomial in $|L|$.*

Proof. The sets $B_1, B_2, \ldots, B_m$ can be computed in time polynomial in $|L|$ since there are at most $\binom{2n}{2}$ pairs of literals and for each pair (l_i, l_j), checking if it belongs to the set B_k reduces to checking the satisfiability of a conjunction of parities of size 2.

The construction clearly is a normal form representation since the sets B_k depend only on the function, and equivalent functions will give same sets. To show that this normal

form representation is isomorphism preserving, suppose f and g are isomorphic Boolean functions with 2⊕-DLs L_f and L_g respectively where π is the isomorphism. Then $\langle l_i \oplus l_j, k \bmod 2\rangle$ is in the set B_k of the function f if and only if $\langle l_{\pi(i)} \oplus l_{\pi(j)}, k \bmod 2\rangle$ is in the set B_k corresponding to g since π is an isomorphism between f and g. Thus $N_f^\pi = N_{f^\pi} = N_g$. □

Example 4.3.6. *The normal form of the 2⊕-DL* $L = \langle x_1 \oplus \overline{x_2}, 0\rangle\langle x_2 \oplus x_3, 1\rangle\langle 1, 0\rangle$ *is* $N_L = B_1, B_2, B_3, \langle 1, 0\rangle$ *with* $B_1 = \emptyset$, $B_2 = \{\langle x_1 \oplus \overline{x_2}, 0\rangle, \langle \overline{x_1} \oplus x_2, 0\rangle\}$ *and* $B_3 = \{\langle x_1 \oplus \overline{x_3}, 1\rangle, \langle \overline{x_1} \oplus x_3, 1\rangle, \langle x_2 \oplus x_3, 1\rangle, \langle \overline{x_2} \oplus \overline{x_3}, 1\rangle\}$. *The set* B_4*, which is not used for the normal form, has one element for each remaining parity of two literals.*

Let L be a 2⊕-DL and $N_L = B_1, B_2, \ldots, B_{m-1}, \langle 1, m \bmod 2\rangle$ be its normal form. We will efficiently encode N_L as a sequence S_L so that the following holds: For two 2⊕-DLs L_1 and L_2, the functions f_{L_1} and f_{L_2} represented by them are isomorphic if and only if the sequences S_{L_1} and S_{L_2} are equal. We first state a couple of easy observations.

Lemma 4.3.7. *If* $\langle l_i \oplus l_j, b\rangle$ *is in some set* B_k *of the normal form* N_L *of* L*, then* $\langle \overline{l_i} \oplus \overline{l_j}, b\rangle$ *is also in* B_k*.*

Proof. Suppose $\langle l_i \oplus l_j, b\rangle$ is in B_k for some $k \leq m$, i.e., L evaluates to $b = k \bmod 2$ under all assignments that satisfy $(l_i \oplus l_j) \wedge \bigwedge_{p \in B_r, r<k} \neg p$. As $l_i \oplus l_j$ and $\overline{l_i} \oplus \overline{l_j}$ are equivalent, this implies that L evaluates to b under all assignments that satisfy $(\overline{l_i} \oplus \overline{l_j}) \wedge \bigwedge_{p \in B_r, r<k} \neg p$. Hence $\langle \overline{l_i} \oplus \overline{l_j}, b\rangle$ is in B_k. □

Lemma 4.3.8. *If both* $\langle l_{i_1} \oplus l_{i_2}, b\rangle$ *and* $\langle l_{i_2} \oplus l_{i_3}, b\rangle$ *are in some set* B_k *of the normal form* N_L *of* L*, then* $\langle l_{i_1} \oplus l_{i_3}, b\rangle$ *is also in* B_k*.*

Proof. This follows from the fact that if $l_{i_1} \oplus l_{i_3} = 1$, then either $l_{i_1} \oplus l_{i_2} = 1$ or $l_{i_2} \oplus l_{i_3} = 1$. □

Lemma 4.3.9. *If* $\langle l_{i_1} \oplus l_{i_2}, b\rangle$ *is in some set* B_r *and* $\langle l_{i_1} \oplus l_j, b\rangle$ *is in some set* B_k *with* $r < k$*, then* $\langle l_{i_2} \oplus l_j, b\rangle$ *is also in* B_k*.*

Proof. In all assignments that do not let $\langle l_{i_1} \oplus l_{i_2}, b\rangle$ fire, l_{i_1} and l_{i_2} take the same value. Thus these two literals can be used interchangeably in B_k. □

We call two literals l_i and l_j *equivalent after* B_k (denoted $l_i \equiv_k l_j$) if $l_i = l_j$ or if $\langle l_i \oplus l_j, b\rangle \in B_r$ for some $b \in \{0, 1\}$ and $r \in \{1, \ldots, k\}$. This relation is transitive by Lemmas 4.3.8 and 4.3.9. The relation $\equiv_k$ partitions the set of variables X into the equivalence classes $[x_i]_k = \{x_j \in X \mid x_i \equiv_k x_j\}$, $x_i \in X$. The *complementary* equivalence class of $[x_i]_k$ is $[\overline{x_i}]_k = \{x_j \in X \mid \overline{x_i} \equiv_k x_j\}$; note that it is possible that $[\overline{x_i}]_k = \emptyset$. By definition, the equivalence classes of $\equiv_k$ are a refinement of the equivalence classes of $\equiv_{k+1}$.

Example 4.3.10. *Consider the* 2⊕*-DL* $L = \langle x_1 \oplus \overline{x_2}, 1\rangle\langle x_2 \oplus x_3, 1\rangle\langle x_5 \oplus \overline{x_6}, 1\rangle\langle x_4 \oplus x_5, 0\rangle\langle x_1 \oplus \overline{x_6}, 1\rangle\langle 1, 0\rangle$. *The equivalence classes of* $\equiv_1$ *are* $[x_1]_1 = \{x_1\}$, $[x_2]_1 = [x_3]_1 = \{x_2, x_3\}$, $[x_4]_1 = \{x_4\}$, $[x_5]_1 = \{x_5\}$, *and* $[x_6]_1 = \{x_6\}$. *The classes* $[x_1]_1$ *and* $[x_2]_1$ *are complementary to each other, as are* $[x_5]_1$ *and* $[x_6]_1$. *The complementary class of* $[x_4]_1$ *is* $\emptyset$.

The equivalence classes of $\equiv_2$ *are similar to those of* $\equiv_1$, *the only difference is that* $[x_4]_1$ *and* $[x_5]_1$ *are joined to* $[x_4]_2 = [x_5]_2 = \{x_4, x_5\}$, *which is complementary to* $[x_6]_2 = \{x_6\}$.

The equivalence classes of $\equiv_3$ *are* $[x_1]_3 = [x_4]_3 = [x_5]_3 = \{x_1, x_4, x_5\}$ *and* $[x_2]_3 = [x_3]_3 = [x_6]_3 = \{x_2, x_3, x_6\}$*; they are complementary to each other.*

For $\equiv_4$ *there is only one equivalence class, i.e., it contains all variables. It is complementary to itself.*

Given a normal form $N_L = B_1, \ldots, B_{m-1}, \langle 1, m \bmod 2\rangle$, we encode the sizes and inclusion structure of these equivalence classes as $S_L = C_1, \ldots, C_{m-1}$, where the C_k are lists that are constructed as follows. To construct C_1, consider each pair $\{[x_i]_1, [\overline{x_i}]_1\}$ of complementary equivalence classes of $\equiv_1$. We may assume $|[x_i]_1| \geq |[\overline{x_i}]_1|$ and add the pair of sizes $(|[x_i]_1|, |[\overline{x_i}]_1|)$ to C_1. The list C_1 is sorted lexicographically. For an equivalence class $[x_i]_1$ with $|[x_i]_1| \geq |[\overline{x_i}]_1|$, we use $\mathrm{pos}_1([x_i]_1, [\overline{x_i}]_1)$ to denote the position of the first occurrence of $(|[x_i]_1|, |[\overline{x_i}]_1|)$ in C_1, and for an equivalence class $[x_{i'}]_1$ with $|[x_{i'}]_1| < |[\overline{x_{i'}}]_1|$, we let $\mathrm{pos}_1([x_{i'}]_1, [\overline{x_{i'}}]_1) = -\mathrm{pos}_1([\overline{x_{i'}}]_1, [x_{i'}]_1)$.

We now describe the construction of C_k for $k \geq 2$. For each pair $\{[x_i]_k, [\overline{x_i}]_k\}$ of complementary equivalence classes of $\equiv_k$, we again assume $|[x_i]_k| \geq |[\overline{x_i}]_k|$ and include the pair of sizes $(|[x_i]_1|, |[\overline{x_i}]_1|)$ in C_k. This pair is annotated with the positions of the equivalence classes of $\equiv_{k-1}$ in C_{k-1} that are contained in $[x_i]_k$ and $[\overline{x_i}]_k$: Let $\{S_1, T_1\}, \ldots, \{S_\ell, T_\ell\}$ be the pairs of equivalence classes of $\equiv_{k-1}$ such that $\bigcup_{j=1}^{\ell} S_j = [x_i]_k$ and $\bigcup_{j=1}^{\ell} T_j = [\overline{x_i}]_k$. Define $A_{[x_i]_k}$ as the sorted list that contains $\mathrm{pos}_{k-1}(S_j, T_j)$ for each $j \in \{1, \ldots, \ell\}$, and define $A'_{[x_i]_k}$ as the sorted list that contains $\mathrm{pos}_{k-1}(T_j, S_j)$ for each $j \in \{1, \ldots, \ell\}$. If $|[x_i]_k| = |[\overline{x_i}]_k|$, we assume w.l.o.g. that $A_{[x_i]_k} \geq A'_{[x_i]_k}$. We now annotate the entry $(|[x_i]_1|, |[\overline{x_i}]_1|)$ of C_k with $A_{[x_i]_k}$, and sort C_k w.r.t. the annotations. We define $\mathrm{pos}_k([x_i]_k, [\overline{x_i}]_k)$ as the position of the first pair $(|[x_i]_1|, |[\overline{x_i}]_1|)$ in C_k that is annotated with $A_{[x_i]_k}$. In the asymmetric cases where $|[x_i]_k| > |[\overline{x_i}]_k|$ or $A_{[x_i]_k} > A'_{[x_i]_k}$, we define $\mathrm{pos}_k([\overline{x_i}]_k, [x_i]_k) = -\mathrm{pos}_k([x_i]_k, [\overline{x_i}]_k)$. In the symmetric cases where $|[x_i]_k| = |[\overline{x_i}]_k|$ and $A_{[x_i]_k} = A'_{[x_i]_k}$, we define $\mathrm{pos}_k([\overline{x_i}]_k, [x_i]_k) = \mathrm{pos}_k([x_i]_k, [\overline{x_i}]_k)$.

Example 4.3.11. *Consider again the* 2⊕*-DL* L *from Example 4.3.10. This results in the sequence* $S_L = C_1, C_2, C_3$. *For* $k = 1$, *we have* $C_1 = (1, 0), (1, 1), (2, 1)$ *and the posi-*

tions $\mathrm{pos}_1([x_4]_1, \emptyset) = 1$, $\mathrm{pos}_1(\emptyset, [x_4]_1) = -1$, $\mathrm{pos}_1([x_5]_1, [x_4]_1) = \mathrm{pos}_1([x_4]_1, [x_5]_1) = 2$, $\mathrm{pos}_1([x_2]_1, [x_1]_1) = 3$, *and* $\mathrm{pos}_1([x_1]_1, [x_2]_1) = -3$.

For $k = 2$, *we have* $C_2 = (2,1)_3, (2,1)_{1,2}$, *where the subscripts denote the annotations. The positions are* $\mathrm{pos}_2([x_2]_2, [x_1]_2) = 1$, $\mathrm{pos}_2([x_1]_2, [x_2]_2) = -1$, $\mathrm{pos}_2([x_4]_2, [x_6]_2) = 2$, *and* $\mathrm{pos}_2([x_6]_2, [x_4]_2) = -2$.

For $k = 3$, *we have* $C_3 = (3,3)_{-2,1}$ *and* $\mathrm{pos}_3([x_2]_3, [x_1]_3) = 1$ *and* $\mathrm{pos}_3([x_1]_3, [x_2]_3) = -1$.

Lemma 4.3.12. *The functions represented by two* $2\oplus$*-DLs* L *and* L' *are isomorphic if and only if the structural representations* S_L *and* $S_{L'}$ *of their normal forms* $N_L = B_1, \ldots, B_{m-1}, \langle 1, m \bmod 2\rangle$ *and* $N_{L'} = B'_1, \ldots, B'_{m-1}, \langle 1, m \bmod 2\rangle$ *are equal.*

Proof. In this proof, we denote equivalence after B_k with $\equiv_k$ and equivalence after B'_k with $\equiv'_k$. As the normal form is isomorphism preserving, any isomorphism π from L to L' maps N_L to $N_{L'}$. This implies that $x_i \equiv_k x_j$ if and only if $\pi(x_i) \equiv'_k \pi(x_j)$. In particular, the isomorphism π preserves sizes of and inclusions between the equivalence classes. It follows that $S_L = S_{L'}$.

For the reverse direction, we show that the normal form N_L can be reconstructed from S_L up to renaming of variables. This proves that non-isomorphic functions cannot receive the same structural representations. Given $S_L = C_1, \ldots, C_{m-1}$, we reconstruct $N'_L = B'_1, \ldots, B'_{m-1}, \langle 1, m \bmod 2\rangle$ as follows. For $C_1 = (u_{1,1}, v_{1,1}), \ldots, (u_{1,s}, v_{1,s})$, define disjoint sets of variables $U_{1,j}$ and $V_{1,j}$ with $|U_{1,j}| = u_{1,j}$ and $|V_{1,j}| = v_{1,j}$ for $j \in \{1, \ldots, s\}$, insert into B_1 all pairs $\langle x_i \oplus x_{i'}, 1\rangle$ and all pairs $\langle \overline{x_i} \oplus \overline{x_{i'}}, 1\rangle$ with $(x_i, x_{i'}) \in \bigcup_j U^2_{1,j} \cup V^2_{1,j}$, and all pairs $\langle x_i \oplus \overline{x_{i'}}, 1\rangle$ and all pairs $\langle \overline{x_i} \oplus x_{i'}, 1\rangle$ with $(x_i, x_{i'}) \in \bigcup_j U_{1,j} \times V_{1,j}$. For $k \geq 2$, consider an entry $(u_{k,j}, v_{k,j})$ in C_k with annotation $p_{k,j,1}, \ldots, p_{k,j,\ell}$. If some position occurs multiple times (i.e., if C_{k-1} contains multiple pairs with the same cardinality), arbitrarily disambiguate the position values so that each position occurs exactly once in all annotations in C_k. Define $U_{k,j} = \bigcup_{i:p_{k,j,i}>0} U_{k-1,p_{k,j,i}} \cup \bigcup_{i:p_{k,j,i}<0} V_{k-1,-p_{k,j,i}}$ and $V_{k,j} = \bigcup_{i:p_{k,j,i}>0} V_{k-1,p_{k,j,i}} \cup \bigcup_{i:p_{k,j,i}<0} U_{k-1,-p_{k,j,i}}$. Insert into B_k all pairs $\langle x_i \oplus x_{i'}, k \bmod 2\rangle$ and all pairs $\langle \overline{x_i} \oplus \overline{x_{i'}}, k \bmod 2\rangle$ with $(x_i, x_{i'}) \in \bigcup_j U^2_{k,j} \cup V^2_{k,j}$, and all pairs $\langle x_i \oplus \overline{x_{i'}}, k \bmod 2\rangle$ and all pairs $\langle \overline{x_i} \oplus x_{i'}, k \bmod 2\rangle$ with $(x_i, x_{i'}) \in \bigcup_j U_{k,j} \times V_{k,j}$, unless the respective parity of two literals is already included in some B_r with $r < k$.

To show that there is an isomorphism π that maps $N_L = B_1, \ldots, B_{m-1}, \langle 1, m \bmod 2\rangle$ to N'_L, we prove by induction on k that for each assignment of the pairs of complementary equivalence classes to entries in C_1 of matching cardinality, and for each choice of orientations of symmetric pairs of complementary equivalence classes, there is a π that maps $B_1, \ldots, B_k$ to $B'_1, \ldots, B'_k$ and that follows the given assignment and orientations. In the base case

$k = 1$, we take any bijection π that sends each pair of complementary equivalence classes $\{[x_i]_1, [\overline{x_i}]_1\}$, when it is assigned to the pair (u_j, v_j) to the pair of sets $\{U_{1,j}, V_{1,j}\}$. Moreover, the assignment within each such pair with $|[x_i]_1| = |[\overline{x_i}]_1|$ can be chosen according to the desired orientation. For $k \geq 2$, the prescribed assignment and orientation of the pairs of complementary equivalence classes of $\equiv_k$ to the entries in C_k induce via the annotations an assignment and orientation of the pairs of complementary equivalence classes of $\equiv_{k-1}$ to the entries in C_{k-1}. The inductive hypothesis ensures that there is a π that observes the latter and maps $B_1, \ldots, B_{k-1}$ to $B'_1, \ldots, B'_{k-1}$. This π also observes the former and maps B_k to B'_k. □

This immediately implies the following.

Theorem 4.3.13. *Boolean Isomorphism for $2\oplus$-DL is in polynomial time.*

4.3.2 Reduction to Graph Isomorphism

In this section, we show that C-DL Boolean Isomorphism is reducible to Graph Isomorphism if $\overline{C}$ is Schaefer. We give a reduction from C-DL-Iso to the label-respecting isomorphism problem of labeled trees, which is equivalent to Graph Isomorphism [RZ00]. In this problem, we are given two rooted trees and additionally each vertex has a label. We ask if there is an isomorphism between the trees which is label-respecting, i.e., two vertices in the first tree have the same label if and only if their images in the second tree have the same label. An equivalent generalized version is finding isomorphism of colored labeled trees, in which each vertex also has a color and we ask for a color-preserving, label-respecting isomorphism.

Theorem 4.3.14. *Let C be a class of functions, each depending on at most r variables, such that $\overline{C}$ is Schaefer. Then the C-DL Boolean Isomorphism is polynomial-time reducible to Graph Isomorphism.*

Proof. Given a C-DL L on variables $\{x_1, \ldots, x_n\}$, we compute a normal form in polynomial time for the associated Boolean function f_L. We may assume that in each pair $\langle f_i, b_i \rangle$ in L, f_i is a conjunction (or parity) of literals, i.e., the outer disjunctions are resolved by splitting into several pairs.

We first observe that each f_i is identified by the set of literals that occur in its representation. For such a set C of literals, we denote the identified function with f_C. For example, when $\overline{C}$ is Horn, the set $C = \{x_1, x_2, \ldots, x_{\ell-1}, \overline{x_\ell}\}$ identifies the function $f_C = x_1 \wedge x_2 \wedge \cdots \wedge x_{\ell-1} \wedge \overline{x_\ell}$ in C.

We find all sets C of at most r literals, such that all assignments that satisfy f_C force f_L to the same value, and include them in T_1. In general, T_i consists of all sets C of at most r literals, such that all assignments that satisfy $f_C \wedge \bigwedge_{C' \in T_j, j<i} \neg f_{C'}$ force f_L to the same value. Checking if some set C fulfills this condition reduces to satisfiability of $\overline{C}$-CSP instances, and thus can be implemented efficiently by Theorem 4.3.2.

The resulting sequence $T_1, \ldots, T_m$ partitions the class of all sets of at most r literals (as r is constant, there are only polynomially many). Similarly to the case of $2\oplus$-DLs, the sequence $N_L = T_1, \ldots, T_{m-1}, \langle 1, m \bmod 2 \rangle$ is a normal form; it represents the same function as the decision list obtained from N_L by replacing each T_k with the pairs $\langle f_C, k \bmod 2 \rangle$, $C \in T_k$.

We will now encode N_L as a labeled tree T_L (in the sense of [RZ00]). It turns out that two C-DLs L_1 and L_2 represent isomorphic functions if and only if there is a label-respecting tree isomorphism from T_{L_1} to T_{L_2}. We outline the encoding algorithm which takes N_L as input and computes a labeled tree T_L: Let $T_1, T_2, \ldots, T_{m-1}$ be the r-tuple sets defining N_L. We create a root node with $m-1$ children corresponding to $T_1, T_2, \ldots, T_{m-1}$, where the node for T_i is colored i. In the subtree rooted at the node corresponding to T_i we create a child c for each set $C \in T_i$. The node c will have $|C|$ children which are leaves labeled by the corresponding variable name (in $x_1, x_2, \ldots, x_n$) and colored p or n depending on whether that literal occurring in C is positive or negative. This completes the construction of the labeled tree T_L.

It is easy to verify that if the Boolean functions represented by L_1 and L_2 are isomorphic via a permutation π then, in fact, π acting on the leaf labels of T_{L_1} induces an isomorphism from T_{L_1} to T_{L_2}. Conversely, if there is a label-respecting isomorphism ψ from T_{L_1} to T_{L_2}, then ψ induces a permutation π on the leaf labels of T_{L_1}, which turns out to be an isomorphism from f_{L_1} to f_{L_2}. □

4.3.3 GI-Hardness of C-DL-Iso

Böhler et al. [BHRV04] showed that if C contains $\vee_2$ (the disjunction of two variables) or if C contains $\oplus_3$ (the parity of three variables), then GI is polynomial-time reducible to isomorphism of C-CSP instances. As $\vee_2$ is both bijunctive and anti-Horn, this implies that isomorphism of C-CSP is GI-hard if C is bijunctive, anti-Horn or affine with arity at least 3. Lemma 4.3.4 transfers the GI-hardness to isomorphism of C-DLs, where $\overline{C}$ is bijunctive, anti-Horn or affine with arity at least 3.

To extend the GI-hardness to C-DLs where $\overline{C}$ is Horn, we give a simple reduction from

graph isomorphism to 2-DNF isomorphism, where all literals are positive. This is sufficient as for each 2-DNF $f = \bigvee_{j=1}^{m} x_{i_j} \wedge x_{i'_j}$, the decision list $L_f = \langle x_{i_1} \wedge x_{i'_1}, 1\rangle, \ldots, \langle x_{i_m} \wedge x_{i'_m}, 1\rangle, \langle 1, 0\rangle$ is equivalent to f, and the complement of $x \wedge y$ is Horn.

Given a graph $G = (V, E)$, define the function $f_G = \bigvee_{e=\{u,v\}\in E} u \wedge v$ over the variable set V.

Lemma 4.3.15. *Let G, H be two graphs. Then $G \cong H$ if and only if $f_G \cong f_H$.*

Proof. It is clear that if π is an isomorphism between the graphs, then by renaming the variables in f_G according to π, we get f_H. To prove the converse, we note that the minimum weight satisfying assignments of f_G are exactly those which have two ones in positions that correspond to an edge in G. Since any isomorphism between f_G and f_H must send minimum weight satisfying assignments of f_G to those of f_H, each isomorphism between the Boolean functions f_G and f_H corresponds to an isomorphism between the graphs G and H. □

The following theorem summarizes the results of this section.

Theorem 4.3.16. GI $\leq_p^m$ *C-DL-Iso, if $\overline{C}$ is Schaefer and, if it is affine, includes a parity with arity at least* 3.

4.4 Boolean Isomorphism for Horn-CNFs

In this section we study Boolean Isomorphism for Horn-CNFs. Given two Horn-CNFs, the goal is to test if the functions computed by them are isomorphic. Similar to the previous section, the idea of the algorithm is to construct an isomorphism preserving normal form from the given Horn-CNF which is then encoded as a hypergraph. The normal form construction will run in time polynomial in the number of variables and the size of the input Horn-CNFs. It will involve queries to an oracle to the satisfiability of Horn-CNFs. It is well-known that the satisfiability of Horn-CNFs is P-complete and this will complete the proof of Theorem 4.1.6.

We use ideas from the exact learning algorithm of Angluin, Frazier and Pitt [AFP92] for conjunction of Horn clauses. The outline of our algorithm is as follows: Our aim is to construct a permutation preserving normal form for a given Horn-CNF T, which we will call the target. At each step of the algorithm we have a hypothesis H, which is a Horn-CNF, such that $T \Rightarrow H$, i.e. H is true whenever T is true. Then we find assignments $x \in \{0,1\}^n$ such that $T(x) = 0$ and $H(x) = 1$ in a canonical way. What we mean by

this is that for any T' which is equivalent to T, this procedure generates the same set of assignments x. Moreover if T' is isomorphic to T and π is an isomorphism, then the set of assignments for T' are isomorphic copies of the assignments for T under the same isomorphism π. From this set of assignments we modify the hypothesis. The procedure continues until the hypothesis is equivalent to the Horn-CNF T.

Notice that the algorithm is very similar to a learning algorithm where the assignments $x \in \{0,1\}^n$ such that $T(x) = 0$ and $H(x) = 1$ are the counter-examples of an equivalence query. But unlike in a learning algorithm, we are not handicapped by the fact that the function T is available only as a black-box. The place where we use the property of Horn-CNFs is when we construct the counter-examples. We will use the easily verifiable and well-known fact that Horn-CNFs have a unique minimal satisfying assignment.

We now describe the algorithm for constructing the permutation preserving normal form. Let $T = T_1 \wedge T_2 \wedge \cdots \wedge T_m$ be the given Horn-CNF.

Step 1: Let H be the constant function $\mathbf{1}$ and set $j = 1$.

Step 2: Compute the set of minimum weight satisfying assignments of $\neg T \wedge H$. Call it W^j. If W^j is empty, return H as the normal form N.

Step 3: Otherwise, for each assignment $a = (a_1, \ldots, a_n) \in W^j$, define

$$C'_a = \left\{ \left(\bigwedge_{i|a_i=1} x_i \right) \Rightarrow x_k : a_k = 0 \right\}.$$

Step 4: Remove clauses $C \in C'_a$ such that $T \wedge \neg C$ is satisfiable to obtain the set C_a. Define $H^j = \bigwedge_{a \in W^j} \bigwedge_{C \in C_a} C$. Set $H \leftarrow H \wedge H^j$ and $j \leftarrow j + 1$. Go to Step 2.

Let us look at an example Horn-CNF and see how the algorithm works on a Horn-CNF.

Example 4.4.1. *Consider the following Horn-CNF.*

$$T = (x_1 \to y_1) \wedge (y_1 \to x_1) \wedge (x_2 \to y_2) \wedge (y_2 \to x_2) \wedge (x_3 \to y_3) \wedge (y_3 \to x_3) \\ \wedge (y_1 \wedge y_2 \wedge y_3 \to z).$$

In the first iteration of the algorithm, in Step 2 the set W^1 will be the assignments that set exactly one of the x_i to 1 and all the other variables to 0 and exactly one of the y_i to 1 and all the other variables to 0. After removing the unwanted clauses in Step 4, this gives us the hypothesis $H = (x_1 \to y_1) \wedge (y_1 \to x_1) \wedge (x_2 \to y_2) \wedge (y_2 \to x_2) \wedge (x_3 \to y_3) \wedge (y_3 \to x_3)$.

In the second iteration, the set of minimum weight assignments that satisfy H and falsify T is exactly the one assignment that sets all the variables $x_1, x_2, x_3, y_1, y_2, y_3$ to 1 and z to 0. Thus we get the hypothesis $H = (x_1 \to y_1) \wedge (y_1 \to x_1) \wedge (x_2 \to y_2) \wedge (y_2 \to x_2) \wedge (x_3 \to y_3) \wedge (y_3 \to x_3) \wedge (x_1 \wedge x_2 \wedge x_3 \wedge y_1 \wedge y_2 \wedge y_3 \to z)$ which is equivalent to T and the algorithm stops.

We now proceed to the proof of the correctness of the algorithm. First we show that there are only polynomially many assignments in the set W^j in Step 2 of the algorithm.

Lemma 4.4.2. *Let a be a minimum weight satisfying assignment of $\neg T \wedge H$. Then there exists an i such that a is the minimum weight satisfying assignment of the Horn-CNF $\neg T_i \wedge H$.*

Proof. Any satisfying assignment for the formula $\neg T \wedge H$ satisfies a Horn-CNF $\neg T_i \wedge H$ for some $i \leq m$. Since each $\neg T_i \wedge H$ is a Horn-CNF, it has a unique minimum weight satisfying assignment. Hence, if a is a minimum weight satisfying assignment of $\neg T \wedge H$, then we know that it is the satisfying assignment for some $\neg T_i \wedge H$. Let a' be the minimal model for $\neg T_i \wedge H$, then a' is also a satisfying assignment for $\neg T \wedge H$. Hence $a' \leq a$ and consequently, $a' = a$. □

This lemma proves that we can execute Step 2 in the algorithm in time poly$(n)m$ to construct the set W^j of minimum weight negative counter-examples.

We will now prove that when the algorithm ends the value of j will be at most m and the hypothesis H will be the isomorphism preserving normal form for the Horn-CNF T. To that end, define the set

$$S_j = \begin{cases} \emptyset & \text{if } j = 0 \\ \left\{ T_i \mid \exists a \in W^j \text{ such that } T_i(a) = 0 \right\} \setminus \mathcal{S}_{j-1} & \text{otherwise} \end{cases}$$

We now prove the following lemma about the sets $\mathcal{S}_j$.

Lemma 4.4.3. *For any $j \leq m$ let H be the hypothesis before the execution of Step 4 of the j^{th} iteration of the algorithm. Then $H \equiv T^j$, where T^j is the conjunction of the clauses in $\bigcup_{i \leq j} \mathcal{S}_i$.*

Proof. We will prove this by induction on j. To prove the base case, assume that $j = 1$. For $T^1 = \bigwedge_{T_i \in \mathcal{S}_1} T_i$ and the hypothesis H, we will first show that for any assignment a which falsifies T^1 also falsifies H. If $T^1(a) = 0$, then for some $T_l \in \mathcal{S}_1$, $T_l(a) = 0$. This

implies that a sets all variables in antecedent(T_l) to 1 and consequent(T_l) to 0. Notice that $\bigwedge_{x_i \in \text{antecedent}(T_l)} x_i \Rightarrow \text{consequent}(T_l)$ is a clause in H due to Step 3 and this is not removed in Step 4 of the first iteration. Therefore $H(a) = 0$. Since the clause $\bigwedge_{x_i \in \text{antecedent}(T_l)} x_i \Rightarrow \text{consequent}(T_l)$ is not removed in Step 4 of the first iteration for any $T_l \in S_1$, $H(a) = 1$ whenever $T(a) = 1$. This proves the base case.

Now assume $j > 1$. Let H' be the hypothesis after executing Step 4 of the $(j-1)^{th}$ iteration and by induction assumption we know that $H' \equiv T^{j-1}$. Let $T_l \in S_j$ be a clause such that for some $a \in W^j$, $T_l(a) = 0$ and $H'(a) = 1$. Since a falsifies T_l, a sets consequent(T_l) to false. The assignment a is a negative counter-example to the hypothesis H' and hence the set C_a at the j^{th} iteration contains a clause C such that consequent(C) = consequent(T_l) and antecedent(T_l) $\subseteq$ antecedent(C). Therefore, $T_l \Rightarrow C$ and hence is not removed in Step 4 of the j^{th} iteration. Also, for any a' such that $T_l(a') = 0$, we have $a' \geq a$ since a was the minimal model for the Horn-CNF $\neg T_l \wedge H'$. Therefore, $C(a') = 0$. Since we proved that the clause C is not removed by Step 4 of the j^{th} iteration and C is a clause in the hypothesis H, $H(a) = 0$. This completes the proof of the lemma. □

At the j^{th} iteration of the algorithm, if the set W^j is empty, then we know that the hypothesis H is equivalent to T. Note that at the j^{th} iteration, the formula $T^j = \bigwedge_{i \leq j} \bigwedge_{T_l \in S_i} T_l$ is a sub-formula of T. Since T has at most m clauses, within m steps we will compute the normal form H. We now prove two claims which show that the normal form that we constructed is in fact an isomorphism preserving normal form.

Lemma 4.4.4. *Let T and T' be equivalent Horn-CNFs. If H and H' are the normal forms computed for the formulas T and T' according to the algorithm, then $H = H'$.*

Proof. The proof is by induction. In the i^{th} iteration of the algorithm, let H and H' be the hypotheses. By induction assumption, they are identical. In Step 3, we compute the minimum weight satisfying assignments of $\neg T$ and $\neg T'$. Since $T \equiv T'$, these sets are identical. Hence the hypothesis constructed after Step 3 are identical. For any clause C in the hypothesis, C is not implied by the formula T if and only if C is not implied by the formula T'. Thus the hypotheses H and H' at the $(i+1)^{th}$ iteration of the algorithm are also identical. □

Lemma 4.4.5. *Let T and T' be isomorphic Horn-CNFs and let π be an isomorphism between them. Let H and H' be the normal forms computed by the algorithm for the formulas T and T'. Then $H^\pi = H'$, where H^π is the formula obtained from H by renaming the variables of H according to π.*

Proof. The proof is by induction. In the first iteration of the algorithm, let W^1 and W'^1 be the set of minimum weight counter-examples generated by the algorithm for T and T' respectively when the hypothesis is the constant function. If π is any isomorphism between T and T', then $\pi(W^1) = W'^1$. Therefore if H and H' are the hypotheses constructed at the first iteration, then $H^\pi = H'$.

Suppose at the i^{th} stage, we have hypotheses H_i and H'_i such that for any π which is an isomorphism between T and T', $H_i^\pi = H_i'^\pi$. Since $\neg T \wedge H_i$ and $\neg T \wedge H'_i$ are isomorphic, the sets W_i and W'_i of minimum weight satisfying assignments are also isomorphic. The lemma follows. □

Given an isomorphism preserving normal form H, we can construct a hypergraph $\mathcal{H}$ in the following way: the vertex set $V = \{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n\} \cup \{u_1, \ldots, u_n\}$ and the edge set $E = \{\ \mathrm{Var}(C) \mid \forall \text{ clauses } C \text{ in } H\} \cup \{(x_i, \overline{x}_i, u_i \mid 1 \leq i \leq n\}$ where $\mathrm{Var}(C)$ is the set of literals in the clause C.

This completes the proof of the Theorem 4.1.6.

4.4.1 GI-completeness of Horn-CNF isomorphism

We now outline a simple reduction from hypergraph isomorphism to the isomorphism of Horn-CNFs. Given a hypergraph $\mathcal{H}(V, \mathcal{E})$, we will construct a DNF $F(\mathcal{H})$ as follows:

$$F(\mathcal{H}) = \bigvee_{E \in \mathcal{E}} \left(\bigwedge_{v \in E} x_v \wedge x_E \right)$$

Observe that the minimal models of this DNF are precisely those assignments that satisfy the variables corresponding to an edge $E \in \mathcal{E}$. Since any isomorphism between two DNFs must map the minimal models of one to the minimal models of the other, we have the following lemma.

Lemma 4.4.6. *Let $\mathcal{H}_1$ and $\mathcal{H}_2$ be two hypergraphs and let $F(\mathcal{H}_1)$ and $F(\mathcal{H}_2)$ be the DNFs constructed as above. Then $\mathcal{H}_1 \cong \mathcal{H}_2$ if and only if $F(\mathcal{H}_1) \cong F(\mathcal{H}_2)$*

Clearly, two Boolean functions are isomorphic if and only if their complement functions are isomorphic. The complement of $F(\mathcal{H})$ is a Horn-CNF and this proves the following theorem.

Theorem 4.4.7. *Hypergraph isomorphism is polynomial-time reducible to Horn-CNF isomorphism.*

Representation	Complexity of Boolean Isomorphism
Truth-table	P
k-CNF, k-DNF, Monotone-DNF	GI-complete
Horn-CNF	GI-complete
Rank-1 decision trees	L-complete
Rank-r decision trees, $r \geq 2$	GI-complete, $2^{O(r^2\sqrt{n})}$-time isomorphism testing algorithm
$2\oplus$-decision lists	P
C-decision lists, where C is Horn, anti-Horn, bijunctive	GI-complete

4.5 Summary and Open Problems

In this chapter we looked at Boolean Isomorphism for classes of functions that have efficient satisfiability and equivalence testing algorithms. The results that we saw in the chapter use ideas that are similar to the ideas that were used in designing exact learning algorithms for these classes. Table 4.5 lists the complexity of the Boolean Isomorphism for various representations of the functions.

The algorithm for constructing permutation preserving normal forms for decision trees by looking at all possible assignments that fix the value of the function is similar to the learning algorithm for bounded rank decision trees proved by Simon [Sim95]. This remains the best known learning algorithm for decision trees in the exact learning model. Similarly in the case of Horn-CNFs, we used ideas of the learning algorithm of Angluin, Frazier and Pitt [AFP92] to construct the normal form.

It is worth wondering whether it is possible to use the learning algorithms to construct isomorphism preserving normal forms for other classes of functions. In the next chapter, we will see that this idea also works for decision lists. In [AT96], they used learning algorithms of Boolean formulas in a black-box manner to construct a normal form and design an IP protocol for Boolean non-isomorphism. Even though the normal form that they construct is not permutation preserving, it is not clear whether ideas from that learning algorithm can be used in a non black-box manner to construct permutation preserving normal forms.

Another class of functions which have efficient equivalence testing algorithms but we do not know any reduction from Boolean Isomorphism to graph isomorphism is the class of read-2 CNFs. The satisfiability and equivalence of read-2 CNFs can be checked efficiently using resolution. The complexity of Boolean Isomorphism for it remains open. On the other hand, it is known that equivalent read-1 formulas are syntactically identical and

hence Boolean Isomorphism is reducible to tree isomorphism.

Chapter 5

Isomorphism Testing via Satisfiability

In Chapter 4, we saw that Boolean Isomorphism for some representations is polynomial-time many-one reducible to Graph and Hypergraph Isomorphism. The main ingredient in the proofs is the computation of permutation preserving normal forms for the given representation. Recall that a permutation preserving normal form for a Boolean function f on n variables is a representation N_f such that (i) N_f is equivalent to f, (ii) if g is equivalent to f, N_g is identical to N_f, and (iii) for each permutation π we have $N_{f^\pi} = (N_f)^\pi$. Furthermore, the algorithms we described for computing permutation preserving normal form use the satisfiability testing algorithm for that representation as subroutine, and each satisfiability query made to the subroutine was for some n variable Boolean function (in the same representation).

This motivates the following natural question: If there is an α^n-time satisfiability algorithm for a representation of Boolean functions, say CNF, is there an $O^*(\alpha^{n+o(n)})$-time Boolean Isomorphism algorithm for that representation? Proving this statement for arbitrary representations of Boolean functions seems difficult. We saw in Chapter 2 that Hypergraph Isomorphism is reducible to Boolean Isomorphism for monotone DNFs in which n-vertex hypergraphs are encoded as monotone DNFs on $2n + 1$ variables. But the best known algorithm for Hypergraph Isomorphism has a running time of c^n for a large constant c [Luk99]. However, in this chapter, we will see some interesting examples of representations of Boolean functions which have $O^*(\alpha^n)$-time satisfiability algorithm (for some $\alpha < 2$) which can be used to obtain an $O^*(\alpha^{n+o(n)})$-time algorithm for Boolean Isomorphism.

We show that if f and g are k-CNF formulas there is an isomorphism test with running time $O^*(\alpha^{n+o(n)})$, where $O^*(\alpha^n)$ is the running time bound for the fastest k-CNF SAT algorithm. This result can also be viewed as a SERF reduction [IPZ01] from k-CNF isomor-

phism to k-CNF satisfiability. We further observe that this is true for the representation class of k-decision lists. Recall that k-CNFs and k-DNFs can be encoded as k-decision lists and Rivest [Riv87] showed that there are example of functions that have polynomial-sized k-decision lists but do not have polynomial-sized k-CNFs and k-DNFs.

Motivated by the above connection, we turn back to the graph-theoretic setting and introduce a *generalized graph isomorphism problem*. In this problem, the input instance is two graphs $G = ([n], E)$ and $G' = ([n], E')$ and we seek a bijection $\pi : [n] \to [n]$ as usual. However, we do not demand that π preserves all adjacencies between vertices. We only require that π preserves a specified graph property. As illustration, consider the graph 3-colorability property. For a graph G, let $W(G)$ denote the set of all proper 3-colorings of G. The generalized isomorphism problem corresponding to 3-colorings is to check if there is a bijection π from $V(G)$ to $V(G')$ such that $\pi(W(G)) = W(G')$.

More generally, for a large class of graph properties $\mathcal{P}$ we can similarly define and study the $\mathcal{P}$-isomorphism problem. The question we study is whether these *generalized* isomorphism problems have algorithms which are as fast as algorithms for testing the graph property $\mathcal{P}$. Apart from setting up a framework for these questions we give positive answers for natural properties like 3-coloring and Hamiltonian paths.

Another motivation for defining generalized Graph Isomorphism is that it is related to an interesting class of constraint satisfaction problems. The k-CNF satisfiability is a special case of Γ-CSP satisfiability where Γ is a k-ary relation over some alphabet Σ. Given a CSP instance S with k-ary constraints over n variables, the CSP is satisfiable if there is a assignment $a : [n] \to \Sigma$ to the variables in S, such that all constraints in S are satisfied by the assignment. Furthermore, the Γ-CSP satisfiability problem has been shown to be polynomial-time equivalent to the H-coloring problem for directed graphs H ([FV98]). We recall some definitions at this point.

For a fixed graph H, let $\mathcal{P}$ be the set of graphs G which are *homomorphic* to the graph H: A *homomorphism* is a map $\phi : V(G) \to V(H)$ such that ϕ maps edges in G to edges in H.[1] Then we define the corresponding $\mathcal{P}$-isomorphism problem as follows. Given a pair of n-vertex graphs G and G' as input check if there is a bijection $\pi : V(G) \to V(G')$ such that the set of homomorphisms from $\pi(G)$ to H is *identical* to the set of homomorphism from G' to H. We refer to this as *H-coloring Isomorphism*. Indeed, notice that if we let $H = K_3$ then we get the 3-coloring Isomorphism already defined.

[1] A graph homomorphism ϕ is not a bijection in general and it needs only to map edges to edges.

5.1 Boolean Isomorphism for k-CNF

In this section we show that given an algorithm for k-CNF-satisfiability in time α^n, we can solve Boolean Isomorphism for k-CNF in time $\alpha^{n+o(n)}$. We do this by constructing an permutation preserving normal form for k-CNF formulas in time $n^{O(k)}\alpha^n$ and then we can encode the normal form as a bounded rank hypergraph to which we can apply the bounded-rank Hypergraph Isomorphism algorithm [BC08].

For a k-CNF formula $F = C_1 \wedge C_2 \wedge \ldots C_m$, we describe an algorithm to construct a hypergraph H_F of rank k (i.e. with hyperedges of size at most k) with the following two properties.

1. If F and F' are identical Boolean functions given as k-CNF, then the hypergraphs H_F and $H_{F'}$ are identical.
2. Let F and F' be Boolean functions represented as k-CNF. Then the hypergraph H_F is isomorphic to $H_{F'}$ if and only if F and F' are isomorphic as Boolean functions. Furthermore, given an isomorphism between H_F and $H_{F'}$ in polynomial-time we can compute an isomorphism between F and F'.

Let $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ be a k-CNF defined over the variables $x_1, \ldots, x_n$. In order to construct the hypergraph H_F we will cycle through all the $2^k\binom{n}{k}$ clauses C of size k and pick C if and only if $F \to C$. Note that $F \to C$ is easy to check with a k-CNF satisfiability algorithm because $F \wedge \neg C$ is satisfiable if and only if $F \not\to C$. Furthermore, $F \wedge \neg C$ is an OR of k instances of k-CNF satisfiability.

Let S_F be the set of all clauses implied by F. For each Boolean variable x_i in F we introduce two vertices y_i and z_i, where y_i stands for x_i and z_i for $\overline{x_i}$. We also include another vertex t in H_F. These $2n + 1$ vertices form the vertex set of H_F. We now define the edge set. We include all pairs $\{(y_i, z_i) \mid 1 \leq i \leq n\}$ and all pairs $\{(t, y_i) \mid 1 \leq i \leq n\}$ as edges. For each clause $C \in S_F$ we include a hyperedge e_C of size k, where e_C contains y_i if x_i is in C and contains z_i if $\overline{x_i}$ is in C. Notice that t is the unique vertex that is *not* in any hyperedge of size k and is adjacent to all the y_i.

Lemma 5.1.1. *Let F and F' be Boolean functions given as k-CNF. Then F is isomorphic to F' if and only if the hypergraphs H_F and $H_{F'}$ are isomorphic.*

Proof. Suppose F and F' are isomorphic as Boolean functions via the permutation π on the variable set $x_1, x_2, \ldots, x_n$. Clearly, since F is logically equivalent to $\hat{F} = \wedge_{C \in S_F} C$ and

F' is logically equivalent to $\hat{F}' = \wedge_{C \in S_{F'}} C$ it follows that π is an isomorphism from $\hat{F}$ to $\hat{F}'$.

Now, for each clause C of size k, we know that C is implied by F if and only if the clause $\pi(C)$ is implied by F'. Hence π maps the set S_F to $S_{F'}$. Define the permutation ψ on the vertex set of H_F as follows: if $\pi(x_i) = x_j$ then $\psi(y_i) = y_j$ and $\psi(z_i) = z_j$ and $\psi(t) = t$.

Conversely, suppose the hypergraphs H_F and $H_{F'}$ are isomorphic via the isomorphism ψ. Clearly, by the uniqueness of the special vertex t, ψ maps t to t. Hence ψ induces a permutation on the y_i's and on the z_i's such that $\psi(y_i) = y_j$ iff $\psi(z_i) = z_j$. We define $\pi(x_i) = x_j$) iff $\psi(y_i) = y_j$. It follows that a k-clause C is in S_F if and only if the k-clause $\pi(C)$ is in $S_{F'}$. Furthermore, for each clause $C \in S_F$, note that a truth assignment $(x_1, x_2, \ldots, x_n)$ satisfies C iff $(\pi(x_1), \pi(x_2), \ldots, \pi(x_n))$ satisfies $\pi(C)$. Therefore, a truth assignment $(x_1, x_2, \ldots, x_n)$ satisfies $\hat{F} = \wedge_{C \in S_F} C$ if and only if $(\pi(x_1), \pi(x_2), \ldots, \pi(x_n))$ satisfies $\hat{F}' = \wedge_{C \in S_{F'}} C$. □

We now state the main theorem of this section.

Theorem 5.1.2. *Given an $O^*(\alpha^n)$-time satisfiability algorithm for k-CNF formulas, there is an $O^*(\alpha^{n+o(n)})$-time algorithm for Boolean Isomorphism for k-CNF.*

Proof. Let F and F' be the k-CNF formulas as input instance for Boolean Isomorphism. Since we have already described the algorithm and argued its correctness in Lemma 5.1.1, it suffices to do a running time analysis. Since there are $2^k\binom{n}{k}$ k-clauses we can compute the sets of k-clauses S_F and $S_{F'}$ implied by F and F' respectively, in time $O^*(2^k\binom{n}{k}\alpha^n)$. The resulting hypergraphs H_F and $H_{F'}$ have vertex sets of size $2n + 1$ and $n^{O(k)}$ edges. Since the hypergraphs have hyperedges of size at most k, we can use the algorithm of Babai and Codenotti [BC08] to test if H_F and $H_{F'}$ are isomorphic and, if so, to compute an isomorphism ψ in time $2^{k^2(\log n)^{O(1)}\sqrt{n}}$. As explained in Lemma 5.1.1, fromψ we can recover an isomorphism between F and F' in polynomial time. Thus, for $k = o(\sqrt{n})$ the overall running time of our algorithm is $O^*(\alpha^{n+o(n)})$. □

Remark. We note here that the fastest known algorithm for k-CNF satisfiability has running time α^n for $\alpha = 1 - \Omega(\frac{1}{k})$.

The algorithm as an SERF reduction

We now briefly describe how the above $O^*(\alpha^{n+o(n)})$-time algorithm for Boolean Isomorphism for k-CNF can be viewed as a *SERF reduction* from Boolean Isomorphism to k-CNF-Satisfiability. We briefly recall the definition given in [IPZ01]. A language A is

SERF-reducible to a language B, denoted by $A \leq_{SERF} B$, if there is a constant c such that for every $\varepsilon > 0$, there is a Turing machine $M = M_\varepsilon$ with the following properties:

1. M^B solves A,
2. M runs in time $2^{\varepsilon|x|_A}$ for the input x, and
3. Every query x' to B made by M satisfies $|x'|_B \leq c|x|_A$, where $|\cdot|$ is the size of the chosen *complexity parameter* for the language.

In other words, suppose $A \leq_{SERF} B$ and for every ε there is an algorithm for deciding B that runs in time $2^{\varepsilon n}$, then for every ε' there is an algorithm for deciding A that runs in time $2^{\varepsilon' n}$. SERF reductions were introduced in [IPZ01] to study the exact exponential-time complexity of NP-hard problems. For a given complexity parameter $|\cdot|$ suppose we hypothesize that a language A (for example, satisfiable k-CNF formulas) does not have algorithms with running time $O^*(2^{o(|x|)})$. Now, if A is SERF reducible to B then assumed hypothesis implies that B too does not have algorithms with running time $O^*(2^{o(|x|)})$ (where $|\cdot|$ here stands for the complexity parameter of B). Thus, SERF reducibility gives a notion of computational hardness in the context of subexponential-time algorithms, just as NP-hardness in the context of polynomial-time computation.

Our algorithm for k-CNF Isomorphism can be seen as a SERF reduction of k-CNF Isomorphism to k-CNF satisfiability. In the case of k-CNF Isomorphism and k-CNFSAT the complexity parameter for both problems in the definition of SERF reductions is the number of variables n.

Theorem 5.1.3. *k-CNF-Iso $\leq_{SERF}$ k-CNF-SAT.*

Proof. Let the constant c in the definition be 1. The isomorphism testing algorithm for k-CNF that we described above is a SERF reduction. For k-CNF isomorphism, the complexity parameter is the number of variables in the CNF instance. The algorithm at each stage queries the k-CNF satisfiability algorithm with a CNF with the same number of variables. Finally, we use the algorithm of Babai and Codenotti [BC08] to test the isomorphism of rank-k hypergraphs and this algorithm runs in time $2^{\widetilde{O}(k^2\sqrt{n})}$. □

Boolean Isomorphism for k-DL

We now briefly describe how the isomorphism algorithm for k-CNFs can be extended to the representation class of k-decision lists. Recall that a k-decision list (k-DL) is a function

f that can be expressed as a decision lists $\langle C_1, b_1\rangle, \ldots, \langle C_m, b_m\rangle, \langle 1, b_{m+1}\rangle$ where C_is are conjunctions on at most k literals. We explain an algorithm for Boolean Isomorphism for k-DLs that is nearly as fast as any satisfiability algorithm for k-DLs. Firstly, observe that the k-DL satisfiability has an $O^*(\alpha^n)$ time algorithm where α^n is the running time of the satisfiability algorithm for k-CNFs, for constant k.

Proposition 5.1.4. *Suppose there is an α^n time satisfiability algorithm for k-CNFs, then there is a $O(n^k\alpha^n)$ time algorithm for the satisfiability problem for k-DLs.*

Proof. Let f be the decision lists $\langle C_1, b_1\rangle, \ldots, \langle C_m, b_m\rangle, \langle 1, 1\rangle$. Let $C_{i_1}, \cdots, C_{i_r}$ be the conjunctions such that $i_1 < i_2 < \cdots < i_r$ and $\langle C_{i_j}, 1\rangle$ is a tuple in the decision list for each $j \leq r$. Then f can be represented as the following Boolean formula:

$$f = \bigvee_{s=1}^{r} \bigwedge_{j<i_s} \neg C_j \wedge C_{i_s}.$$

Thus, f is a disjunction of $r \leq n^{O(k)}$, k-CNFs. Therefore, in time $n^{O(k)}\alpha^n$, we can check if f is satisfiable. □

To construct a permutation preserving normal form for a given k-DL L computing a function f, we find all conjunctions C of at most k literals, such that all assignments that satisfy C force f to the same truth-value, and include them in T_1. In general, T_i consists of all conjunctions C of at most k literals, such that all assignments that satisfy $C \wedge \bigwedge_{C' \in T_j, j<i} \neg C'$ force f to the same truth-value. Checking this amounts to checking if $C \wedge \bigwedge_{C' \in T_j, j<i} \neg C' \Rightarrow f$ is a tautology. This formula is a tautology if and only if $C \wedge \bigwedge_{C' \in T_j, j<i} \neg C' \wedge \neg f$ is not satisfiable. Since f can be written as a disjunction of CNFs, using Proposition 5.1.4, we get the following theorem for Boolean Isomorphism for k-DLs. The sets $T_1, \ldots, T_r$ form a permutation preserving normal form and we can encode it as hypergraph of rank $2k$. Now, using the algorithm for bounded rank Hypergraph Isomorphism of Babai and Codenotti [BC08], we obtain the following theorem.

Theorem 5.1.5. *Let f and g be two k-decision lists on n variables. If there is an α^n time algorithm for k-CNF satisfiability, then there is an $O(n^k\alpha^n + 2^{O(k^2\sqrt{n})})$ time algorithm for k-DL Isomorphism.*

5.2 A Generalized Graph Isomorphism Problem

We define *generalized graph isomorphism* motivated by the connection between the isomorphism problem and satisfiability problem for Boolean functions that we have already

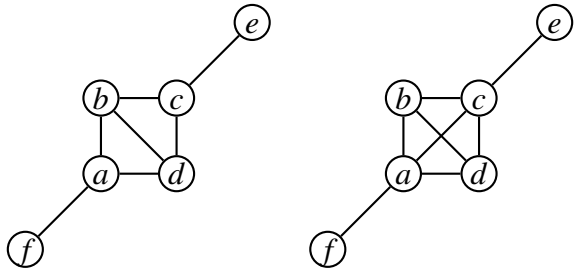

Figure 5.1: Two $\mathcal{P}^+$-equivalent graphs that have the same set of Hamiltonian paths: $f - a - b - d - c - e, f - a - d - b - c - e, e - c - d - b - a - f, e - c - b - d - a - f$

seen.

Let $\mathcal{P}$ be any graph property that is closed under permutations $\pi \in S_n$. More precisely, $\mathcal{P}$ is a subset of undirected graphs such that for any graph G and any permutation π of its vertices $G \in \mathcal{P}$ if and only if $\pi(G) \in \mathcal{P}$.

We denote $G \supset H$ to mean that the graph G is obtained by adding edges to H. From $\mathcal{P}$ we can define a *monotone graph property* $\mathcal{P}^+$ and an *antimonotone graph property* $\mathcal{P}^-$:

$$\mathcal{P}^+ = \{G \mid G \supset H \text{ for some } H \in \mathcal{P}\},$$
$$\mathcal{P}^- = \{G \mid G \subset H \text{ for some } H \in \mathcal{P}\}.$$

The following proposition is immediate from the definition.

Proposition 5.2.1. *For any $\mathcal{P}$ both $\mathcal{P}^+$ and $\mathcal{P}^-$ are in $\mathsf{NP}^{\mathcal{P}}$. In particular, if $\mathcal{P}$ is polynomial-time recognizable, then $\mathcal{P}^+$ and $\mathcal{P}^-$ are in* NP.

Definition 5.2.2. *We say two graphs G_1 and G_2 are $\mathcal{P}^+$-equivalent if for every $H \in \mathcal{P}$,: $G_1 \supset H \iff G_2 \supset H$. Similarly, G_1 and G_2 are $\mathcal{P}^-$-equivalent if for every $H \in \mathcal{P}$: $G_1 \subset H \iff G_2 \subset H$.*

Consider the following example: If $\mathcal{P}$ is the set of all simple paths of length n on n vertices. Then $\mathcal{P}^+$ are precisely graphs that have Hamiltonian paths. Applying the definition in this setting, two graphs are $\mathcal{P}^+$ equivalent precisely when they have the same set of Hamiltonian paths. Figure 5.1 shows two graphs that are $\mathcal{P}^+$ equivalent for this property.

We now define a notion of generalized isomorphism with respect to a graph property $\mathcal{P}$.

Definition 5.2.3. *We say two n-vertex graphs G_1 and G_2 are* $\mathcal{P}^+$-isomorphic *if there is a permutation $\pi \in S_n$ such that $\pi(G_1)$ and G_2 are $\mathcal{P}^+$-equivalent. Similarly, G_1 and G_2 are* $\mathcal{P}^-$-isomorphic *if $\pi(G_1)$ and G_2 are $\mathcal{P}^-$-equivalent for some permutation π.*

For a graph property $\mathcal{P}$, the $\mathcal{P}^+$ Isomorphism problem ($\mathcal{P}^-$ Isomorphism) is to decide if G_1 and G_2 are $\mathcal{P}$-isomorphic (respectively, $\mathcal{P}^-$ isomorphic).

For each graph $G \in \mathcal{P}^+$ we have an edge-minimal subgraph $G_* \subset G$ such that G_* and G are $\mathcal{P}^+$-equivalent. I.e. G_* and G are $\mathcal{P}^+$-equivalent but for each edge $e \in G_*$ the graphs $G_* \setminus e$ and G are not $\mathcal{P}^+$-equivalent. Moreover, computing G_* from G is polynomial-time reducible to testing $\mathcal{P}^+$ equivalence. For each edge $e \in G$, if $G \setminus e$ and G are $\mathcal{P}^+$-equivalent we discard e from G. Finally, we are left with the subgraph G_* consisting of edges that cannot be discarded maintaining $\mathcal{P}^+$ equivalence with G.

Likewise, for each G we have an edge-maximal graph G^* such that G and G^* are $\mathcal{P}^-$-equivalent but G and $G^* + e$ are not for any new edge e. Similar to the above we can see that computing G^* from G is polynomial-time reducible to testing $\mathcal{P}^-$ equivalence. We summarize this below.

Lemma 5.2.4. *Let $\mathcal{P}$ be a graph property. If there is an α^n-time algorithm for deciding $\mathcal{P}^+$ equivalence ($\mathcal{P}^-$ equivalence) then for any given graph G, the graph G_* (respectively G^*) can be computed from G in time $O^*(\alpha^n)$.*

Lemma 5.2.5. *For any two graphs G and G':*

1. *G and G' are $\mathcal{P}^+$-equivalent if and only if $G_* = G'_*$.*
2. *G and G' are $\mathcal{P}^-$-equivalent if and only if $G^* = G'^*$.*

Proof. Suppose $G_* = G'_*$. Since G and G_* are $\mathcal{P}^+$-equivalent, and G' and G'_* are $\mathcal{P}^+$-equivalent. It follows that G and G' are $\mathcal{P}^+$-equivalent. Conversely, suppose G and G' are $\mathcal{P}^+$ equivalent. We know that G_* and G'_* are $\mathcal{P}^+$-equivalent. Suppose e is an edge in G_* and not in G'_*. Since $e \notin G'_*$ it means there is a subgraph $H \in \mathcal{P}$ such that $e \in H$ and $H \subset G'_*$ (otherwise, we could delete e from G'_* preserving $\mathcal{P}^+$ equivalence). However, $H \not\subset G_*$ as $e \notin G_*$. This contradicts the $\mathcal{P}^+$ equivalence of G_* and G'_*. The second part can be proved similarly. □

We now extend the above lemma to give a characterization of $\mathcal{P}^+$ Isomorphism and $\mathcal{P}^-$ Isomorphism in terms of Graph Isomorphism.

Lemma 5.2.6. *For any two graphs G and G':*

1. *G and G' are $\mathcal{P}^+$-isomorphic if and only if the graphs G_* and G'_* are isomorphic.*
2. *G and G' are $\mathcal{P}^-$-isomorphic if and only if the graphs G^* and G'^* are isomorphic.*

Proof. We will prove only the first part as the second part is similar. For any permutation π notice that $\pi(G_*) = \pi(G)_*$. Suppose π is a $\mathcal{P}^+$ isomorphism from G to G'. Then $\pi(G)$ and G' are $\mathcal{P}^+$-equivalent which implies $\pi(G_*) = G'_*$. Hence π is an isomorphism between the graphs G_* and G'_*.

Conversely, if π is an isomorphism between the graphs G_* and G'_* we have $\pi(G_*) = G'_*$ which implies $\pi(G)$ and G' are $\mathcal{P}^+$-equivalent by Lemma 5.2.5. □

We now discuss three examples of generalized graph isomorphism problems.

5.2.1 Hamiltonian-Path Isomorphism

Let the graph property $\mathcal{P}$ consist of the set of all simple paths of length n on n vertices. Thus, for each n there are $n!$ labeled paths of length n which are members of $\mathcal{P}$ and $\mathcal{P}$ is a polynomial-time decidable property. The graph property $\mathcal{P}^+$ is the NP-complete language consisting of all graphs with Hamiltonian paths.

Now, $\mathcal{P}^+$ Isomorphism is the problem of checking for two given graphs G_1 and G_2 on vertex set $[n]$ if there is a bijection $\pi : [n] \to [n]$ such that $\pi(G_1)$ and G_2 have identical sets of Hamiltonian paths. We terms this problem as Hamiltonian-Path Isomorphism. Using Lemmas 5.2.6 and 5.2.4 we show the following algorithm for Hamiltonian-Path Isomorphism.

Theorem 5.2.7. *There is a randomized $O^*(1.657^{n+o(n)})$ time algorithm that takes as input two graphs G and G' and accepts with probability more than $2/3$ if they are Hamiltonian-Path isomorphic and rejects with probability $2/3$ if they are not Hamiltonian-Path isomorphic.*

Proof. By Lemma 5.2.6 we can check if G and G' are Hamiltonian-Path isomorphic by first computing G_* and G'_* and then checking if G_* and G'_* are isomorphic graphs. Since graph isomorphism can be solved in time $2^{O(\sqrt{n \lg n})}$, it suffices to bound the running time of the algorithm for computing the graph G_* from G for the Hamiltonian Path property.

By Lemma 5.2.4 it suffices to give an $O^*(\alpha^n)$ time algorithm for checking if two graphs G_1 and G_2 on vertex set $[n]$ are Hamiltonian-Path equivalent. Note that G_1 and G_2 are not Hamiltonian-Path equivalent iff either there is an edge e in $G_1 \setminus G_2$ such that e lies on a Hamiltonian path in G_1 or there is an edge e in $G_2 \setminus G_1$ such that e lies on a Hamiltonian path in G_2. But this condition is easy to check using an algorithm for the Hamiltonian Path

problem: for each $e \in G_1 \setminus G_2$ check if the graph $G_{1,e}$, obtained from G_1 by subdividing the edge e by adding a new vertex u_e, is Hamiltonian.

Now, we can apply the important result of Björklund [Bjö10] that the Hamiltonian Path problem has an $O^*(1.657^n)$ time randomized algorithm with constant success probability and one-sided error. We can reduce the error probability to a suitable inverse polynomial and use it as oracle for Hamiltonian Path in the above computation. It now follows from standard arguments for randomized computation that Hamiltonian-Path isomorphism has an $O^*(\alpha^{n+o(n)})$ time randomized algorithm with success probability $2/3$. □

5.2.2 3-Coloring Isomorphism

We now discuss an example of $\mathcal{P}^-$ Isomorphism. We will consider the property $\mathcal{P}^-$ to be the set of all 3-colorable graphs G. The underlying property $\mathcal{P}$ consists of all complete tripartite graphs (which are maximally 3-colorable under edge addition). Notice that each complete tripartite graph is uniquely 3-colorable and hence we observe the following.

Proposition 5.2.8. *Two graphs G and G' on vertex set $[n]$ are 3-colorable isomorphic precisely when there is a permutation $\pi : [n] \to [n]$ such that for every 3-coloring $f : [n] \to \{1,2,3\}$, f is a proper 3-coloring of the graph $\pi(G)$ if and only if it is a proper 3-coloring of the graph G'.*

There is a deterministic algorithm for checking 3-colorability of graphs that runs in time $O^*(1.3289^n)$ due to Beigel and Eppstein [BE05]. We show that it can be used to obtain a 3-coloring isomorphism test with a similar running time.

Theorem 5.2.9. *There is a $O^*(1.3289^{n+o(n)})$ time algorithm for 3-coloring Isomorphism.*

Proof. By Lemmas 5.2.6, 5.2.5 and 5.2.4 it suffices to show an $O^*(1.3289^n)$ time algorithm for testing 3-coloring equivalence of the input graphs G_1 and G_2. The graphs G_1 and G_2 are 3-coloring *inequivalent* if there is a 3-coloring $f : [n] \to \{1,2,3\}$ which is proper for one of them and not proper for the other. Consider the edges e in the symmetric difference $G_1 \triangle G_2$ one by one. If $e = (u,v) \in G_1 \setminus G_2$, then we contract the pair (u,v) in the graph G_2 (drop any multiple edges) to obtain an $n-1$ vertex graph $G_{2,e}$. The graph $G_{2,e}$ is 3-colorable if and only if G_2 has a 3-coloring that gives the same color to both u and v which would make G_1 and G_2 inequivalent. It follows that G_1 and G_2 are 3-coloring inequivalent if and only if either for some edge $e \in G_1 \setminus G_2$ $G_{2,e}$ is 3-colorable or for an $e \in G_2 \setminus G_1$ the graph $G_{1,e}$ is 3-colorable. Putting it together, 3-coloring equivalence can be tested in time $O^*(1.3289^n)$ which proves the theorem. □

5.2.3 H-coloring Isomorphism

Finally, we briefly discuss a generalization of 3-coloring Isomorphism called H-coloring Isomorphism.

Definition 5.2.10. *For two graphs G and G', a homomorphism $\phi : G \to G'$ is a map such that $\forall (u,v) \in E(G)$, $(\phi(u), \phi(v)) \in E(G')$. Let $HOM(G, G')$ denote the set of all homomorphisms from G to G'.*

An *H-coloring* of a graph G is a homomorphism from the graph G to the graph H. The *H-coloring* problem is: Given a graph G, check if $HOM(G, H)$ is empty or not. When H is the complete graph K_k, this is the standard k-coloring problem. The trivial algorithm for testing if a graph is H-colorable is to list down all the H-colorings and this runs in time $|H|^n$.

Definition 5.2.11. *The decision problem* H-coloring Isomorphism *is to test, given two graphs G and G', whether there exists a bijection $\pi : V(G) \to V(G')$ such that $HOM(\pi(G), H) = HOM(G', H)$.*

For arbitrary H we have a simple $|H|^{n+o(n)}$-time algorithm for H-coloring Isomorphism. We first express H-coloring Isomorphism in the language of $\mathcal{P}^-$-isomorphisms. The property $\mathcal{P}$ is the set of graphs G such that $HOM(G, H) \neq \emptyset$ and for all $(u,v) \notin E(G)$ there exists some homomorphism $\phi \in HOM(G, H)$ such that the image $(\phi(u), \phi(v)) \notin E(H)$. In other words, $\mathcal{P}$ is the set of all graphs G with the maximum number of edges such that adding any new edge to G kills some homomorphism in $HOM(G, H)$. For instance, observe that when $H = K_k$ then $\mathcal{P}$ is the set of all complete k-partite graphs.

The corresponding antimonotone property $\mathcal{P}^-$ consists of all graphs obtained by edge deletion from graphs in $\mathcal{P}$. Two graphs G and G' are $\mathcal{P}^-$-equivalent (i.e. H-coloring equivalent) precisely when $HOM(G, H) = HOM(G', H)$.

Let (G, G') be an instance of H-coloring isomorphism. By Lemma 5.2.6, it suffices to test if G^* and G'^* are isomorphic graphs. Therefore, we need an algorithm to compute G^* from G. Following this approach, we first obtain a simple $|H|^{n+o(n)}$ time algorithm for the H-coloring Isomorphism for any graph H.

Proposition 5.2.12. *For any graph H there is an $|H|^{n+o(n)}$ time algorithm for the H-coloring Isomorphism.*

Proof. We first construct the graphs G^* as follows: For each pair of vertices $(u,v) \notin E(G)$, we check if there is a homomorphism ϕ from G to H such that $(\phi(u), \phi(v)) \notin E(H)$. Since

there are at most $|H|^n$ many H-colorings for G, this can be done in time $|H|^n$. If there are no such homomorphisms, then we can add the edge (u, v) to the graph G. Iterating over all such pairs we can construct the graph G^* in time $|H|^n \operatorname{poly}(n)$. Similarly, we construct the graph G'^* from G. Now, by Lemma 5.2.6, it is sufficient to test if G^* and G'^* are isomorphic which can be done in time $2^{O(\sqrt{n \log n})}$. □

5.3 Summary and Open Problems

The last result is really a brute-force algorithm that works for any graph H. In keeping with the theme of this chapter, the question that remains open is the following: if $\mathcal{H}$ is a class of graphs such that for any $H \in \mathcal{H}$, H-coloring has a c^n time algorithm for $c < |H|$, then is there a $c^{n+o(n)}$ time algorithm for H-coloring Isomorphism for any $H \in \mathcal{H}$?

Overall, in this chapter we explored the question whether faster satisfiability and equivalence testing yields faster isomorphism algorithms for the corresponding problems. It would be nice to syntactically classify which representation classes of Boolean functions have such isomorphism algorithms.

In the last two chapters the reductions from Boolean isomorphism to graph isomorphism used ideas from learning theory to construct *permutation preserving normal forms*. An interesting question is whether a relationship between learning algorithms and normal forms can be formally expressed. Exact learning algorithms use counterexamples to construct a new hypothesis after each equivalence query. A question to explore would be to see for which representation classes of Boolean functions is there a way to construct counterexamples so that we manage to obtain permutation preserving normal forms using the learning algorithm.

Bibliography

[AB10] Noga Alon and Eric Blais, *Testing Boolean function isomorphism*, APPROX-RANDOM, 2010, pp. 394–405.

[ABSS97] Sanjeev Arora, László Babai, Jacques Stern, and Z. Sweedyk, *The hardness of approximate optima in lattices, codes, and systems of linear equations*, J. Comput. Syst. Sci **54** (1997), no. 2, 317–331.

[AFK02] Sanjeev Arora, Alan M. Frieze, and Haim Kaplan, *A new rounding procedure for the assignment problem with applications to dense graph arrangement problems*, Math. Program. **92** (2002), no. 1, 1–36.

[AFP92] Dana Angluin, Michael Frazier, and Leonard Pitt, *Learning conjunctions of Horn clauses*, Machine Learning **9** (1992), 147–164.

[AKKV12] Vikraman Arvind, Johannes Köbler, Sebastian Kuhnert, and Yadu Vasudev, *Approximate graph isomorphism*, MFCS (Branislav Rovan, Vladimiro Sassone, and Peter Widmayer, eds.), Lecture Notes in Computer Science, vol. 7464, Springer, 2012, pp. 100–111.

[AT96] Manindra Agrawal and Thomas Thierauf, *The Boolean isomorphism problem*, FOCS, 1996, pp. 422–430.

[Bab79] László Babai, *Monte-Carlo algorithms in graph isomorphism testing.*, Tech. Report 79-10, Univ. de Montréal, Dép. de mathématiques et de statistique, 1979.

[BC08] László Babai and Paolo Codenotti, *Isomorhism of hypergraphs of low rank in moderately exponential time*, FOCS, 2008, pp. 667–676.

[BCS+13] László Babai, Xi Chen, Xiaorui Sun, Shang-Hua Teng, and John Wilmes, *Faster canonical forms for strongly regular graphs*, FOCS, IEEE Computer Society, 2013, pp. 157–166.

[BdW02] Harry Buhrman and Ronald de Wolf, *Complexity measures and decision tree complexity: a survey*, Theor. Comput. Sci. **288** (2002), no. 1, 21–43.

[BE05] Richard Beigel and David Eppstein, *3-coloring in time O(1.3289^n)*, J. Algorithms **54** (2005), no. 2, 168–204.

[BGM82] László Babai, D. Yu. Grigoryev, and David M. Mount, *Isomorphism of graphs with bounded eigenvalue multiplicity*, STOC, 1982, pp. 310–324.

[BHRV04] Elmar Böhler, Edith Hemaspaandra, Steffen Reith, and Heribert Vollmer, *The complexity of Boolean constraint isomorphism*, STACS, 2004, pp. 164–175.

[BHZ87] Ravi B. Boppana, Johan Håstad, and Stathis Zachos, *Does co-NP have short interactive proofs?*, Inf. Process. Lett. **25** (1987), no. 2, 127–132.

[Bjö10] Andreas Björklund, *Determinant sums for undirected hamiltonicity*, FOCS, 2010, pp. 173–182.

[BL83] László Babai and Eugene M. Luks, *Canonical labeling of graphs*, STOC, 1983, pp. 171–183.

[Blu92] Avrim Blum, *Rank-r decision trees are a subclass of r-decision lists*, Inf. Process. Lett. **42** (1992), no. 4, 183–185.

[BO10] Eric Blais and Ryan O'Donnell, *Lower bounds for testing function isomorphism*, IEEE Conference on Computational Complexity, 2010, pp. 235–246.

[CFI92] Jin Yi Cai, Martin Fürer, and Neil Immerman, *An optimal lower bound on the number of variables for graph identifications*, Combinatorica **12** (1992), no. 4, 389–410.

[CGSM11] Sourav Chakraborty, David García-Soriano, and Arie Matsliah, *Nearly tight bounds for testing function isomorphism*, SODA, 2011, pp. 1683–1702.

[Cob65] Alan Cobham, *The intrinsic computational difficulty of functions*, Proceedings of the 1964 Congress for Logic, Methodology, and the Philosophy of Science, 1965, pp. 24–30.

[Coo71] Stephen A. Cook, *The complexity of theorem-proving procedures*, STOC (Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, eds.), ACM, 1971, pp. 151–158.

[CR02] Alberto Caprara and Romeo Rizzi, *Packing triangles in bounded degree graphs*, Inf. Process. Lett. **84** (2002), no. 4, 175–180.

[Edm65] Jack Edmonds, *Paths, trees, and flowers*, Canadian Journal of mathematics **17** (1965), no. 3, 449–467.

[EH89] Andrzej Ehrenfeucht and David Haussler, *Learning decision trees from random examples*, Inf. Comput. **82** (1989), no. 3, 231–246.

[Ete97] Kousha Etessami, *Counting quantifiers, successor relations, and logarithmic space*, J. Comput. Syst. Sci. **54** (1997), no. 3, 400–411.

[Fei02] Uriel Feige, *Relations between average case complexity and approximation complexity*, STOC (John H. Reif, ed.), ACM, 2002, pp. 534–543.

[FHL80] Merrick L. Furst, John E. Hopcroft, and Eugene M. Luks, *Polynomial-time algorithms for permutation groups*, FOCS, 1980, pp. 36–41.

[FM06] Eldar Fischer and Arie Matsliah, *Testing graph isomorphism*, SODA, ACM Press, 2006, pp. 299–308.

[FSS81] Merrick L. Furst, James B. Saxe, and Michael Sipser, *Parity, circuits, and the polynomial-time hierarchy*, FOCS, 1981, pp. 260–270.

[FV98] Tomás Feder and Moshe Y. Vardi, *The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory*, SIAM J. Comput. **28** (1998), no. 1, 57–104.

[GI03] Venkatesan Guruswami and Piotr Indyk, *Embeddings and non-approximability of geometric problems*, SODA, 2003, pp. 537–538.

[GJ79] Michael R. Garey and David S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, W. H. Freeman & Co., New York, NY, USA, 1979.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson, *How to play any mental game or a completeness theorem for protocols with honest majority*, STOC (Alfred V. Aho, ed.), ACM, 1987, pp. 218–229.

[GR08] Venkatesan Guruswami and Prasad Raghavendra, *Constraint satisfaction over a non-Boolean domain: Approximation algorithms and unique-games hardness*, APPROX-RANDOM, 2008, pp. 77–90.

[GXTL10] Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li, *A survey of graph edit distance*, Pattern Anal. Appl. **13** (2010), no. 1, 113–129.

[Hås86] Johan Håstad, *Almost optimal lower bounds for small depth circuits*, STOC, 1986, pp. 6–20.

[Hås01] ———, *Some optimal inapproximability results*, J. ACM **48** (2001), no. 4, 798–859.

[IPZ01] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane, *Which problems have strongly exponential complexity?*, J. Comput. Syst. Sci. **63** (2001), no. 4, 512–530.

[Kar72] Richard M. Karp, *Reducibility among combinatorial problems*, Complexity of Computer Computations, 1972, pp. 85–103.

[Kho02] Subhash Khot, *On the power of unique 2-prover 1-round games*, STOC, 2002, pp. 767–775.

[KKL88] J. Kahn, G. Kalai, and N. Linial, *The influence of variables on Boolean functions*, Proceedings of the 29th Annual Symposium on Foundations of Computer Science (Washington, DC, USA), SFCS '88, IEEE Computer Society, 1988, pp. 68–80.

[KKMO04] Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell, *Optimal inapproximability results for max-cut and other 2-variable csps?*, FOCS, 2004, pp. 146–154.

[KM93] Eyal Kushilevitz and Yishay Mansour, *Learning decision trees using the Fourier spectrum*, SIAM J. Comput. **22** (1993), no. 6, 1331–1348.

[KOS04] Adam R. Klivans, Ryan O'Donnell, and Rocco A. Servedio, *Learning intersections and thresholds of halfspaces*, J. Comput. Syst. Sci. **68** (2004), no. 4, 808–840.

[KSTW00] Sanjeev Khanna, Madhu Sudan, Luca Trevisan, and David P. Williamson, *The approximability of constraint satisfaction problems*, SIAM J. Comput. **30** (2000), no. 6, 1863–1920.

[LLZ02] Michael Lewin, Dror Livnat, and Uri Zwick, *Improved rounding techniques for the max 2-sat and max di-cut problems*, IPCO, 2002, pp. 67–82.

[LMN93] Nathan Linial, Yishay Mansour, and Noam Nisan, *Constant depth circuits, Fourier transform, and learnability*, J. ACM **40** (1993), no. 3, 607–620.

[LR13] Lorenzo Livi and Antonello Rizzi, *The graph matching problem*, Pattern Analysis and Applications **16** (2013), no. 3, 253–283 (English).

[LRS06] Michael Langberg, Yuval Rabani, and Chaitanya Swamy, *Approximation algorithms for graph homomorphism problems*, APPROX-RANDOM, 2006, pp. 176–187.

[Luk82] Eugene M. Luks, *Isomorphism of graphs of bounded valence can be tested in polynomial time*, J. Comput. Syst. Sci. **25** (1982), no. 1, 42–65.

[Luk86] ———, *Parallel algorithms for permutation groups and graph isomorphism*, FOCS, 1986, pp. 292–302.

[Luk99] ———, *Hypergraph isomorphism and structural equivalence of Boolean functions*, STOC, 1999, pp. 652–658.

[Mil83] Gary L. Miller, *Isomorphism of k-contractible graphs: A generalization of bounded valence and bounded genus*, Information and Control **56** (1983), no. 1/2, 1–20.

[OWWZ14] Ryan O'Donnell, John Wright, Chenggang Wu, and Yuan Zhou, *Hardness of robust graph isomorphism, Lassere gaps, and asymmetry of random graphs*, SODA, 2014, pp. 1659–1677.

[Pet94] Erez Petrank, *The hardness of approximation: Gap location*, Computational Complexity **4** (1994), 133–157.

[Raz87] Alexander A. Razborov, *Lower bounds for the size of circuits of bounded depth with basis* $\{\wedge, \oplus\}$, Math. notes of the Academy of Sciences of the USSR **41** (1987), no. 4, 333–338.

[Riv87] Ronald L. Rivest, *Learning decision lists*, Machine Learning **2** (1987), no. 3, 229–246.

[RZ00] Sarnath Ramnath and Peiyi Zhao, *On the isomorphism of expressions*, Inf. Process. Lett. **74** (2000), no. 3-4, 97–102.

[Sch78] Thomas J. Schaefer, *The complexity of satisfiability problems*, STOC, 1978, pp. 216–226.

[Sch99] Uwe Schöning, *A probabilistic algorithm for k-sat and constraint satisfaction problems*, FOCS, IEEE Computer Society, 1999, pp. 410–414.

[Sim95] Hans-Ulrich Simon, *Learning decision lists and trees with equivalence-queries*, EuroCOLT, 1995, pp. 322–336.

[Smo87] Roman Smolensky, *Algebraic methods in the theory of lower bounds for Boolean circuit complexity*, STOC, 1987, pp. 77–82.

[Spi96] Daniel A. Spielman, *Faster isomorphism testing of strongly regular graphs*, STOC, 1996, pp. 576–584.

[Tar93] Jun Tarui, *Probablistic polynomials, AC^0 functions, and the polynomial-time hierarchy*, Theor. Comput. Sci. **113** (1993), no. 1, 167–183.

[Thi00] Thomas Thierauf, *The computational complexity of equivalence and isomorphism problems*, LNCS, vol. 1852, Springer, 2000.

[Tor04] Jacobo Torán, *On the hardness of graph isomorphism*, SIAM J. Comput **33** (2004), no. 5, 1093–1108.

[ZKT85] V.N. Zemlyachenko, N.M. Korneenko, and R.I. Tyshkevich, *Graph isomorphism problem*, Journal of Soviet Mathematics **29** (1985), 1426–1482.