ON CLASSICAL AND QUANTUM CONSTRAINT SATISFACTION PROBLEMS IN THE TRIAL AND ERROR MODEL

AARTHI SUNDARAM
(MSc. Mathematics and B.E. Computer Science)

A THESIS SUBMITTED
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
CENTRE FOR QUANTUM TECHNOLOGIES
NATIONAL UNIVERSITY OF SINGAPORE

2017

Supervisors:
Dr. Miklos Santha, Main Supervisor

Examiners:
Associate Professor Troy Lee, Centre for Quantum Technologies
Assistant Professor Xiaohui Bei, Nanyang Technological University
Dr. Ashely Montanaro, University of Bristol

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Aarthi Sundaram
August 5, 2017

Acknowledgments

As I take stock of the years spent in my PhD journey, I would like to acknowledge the many people who have been instrumental to the success of this journey.

First and foremost, I am deeply grateful for the support, guidance, generosity and encouragement of my adviser, Miklos Santha which has been indispensable to my undertaking and completing this thesis. Thank you for agreeing to take me on as your student and also for patiently and tirelessly striving to ensure I write proofs instead of veering into prose! The wonderful example you have set as a mathematician, researcher and mentor are lessons I will always carry with me.

I have had the privilege of interacting with different researchers during the course of my doctoral studies, chief among them being my co-authors. Each of their unique perspectives has helped deepen my own understanding. I would like to thank Itai Arad, Shengyu Zhang, Gabor Ivanyos, Youming Qiao, Raghav Kulkarni, Adam Bouland and Daniel Grier for their penetrating insight, constructive criticism, and fresh ideas that have refined my own approach to tackling research problems. I would also like to thank the members of my Thesis Advisory Committee, Rahul Jain and Frank Stephan.

I was introduced to the world of quantum information by one of my undergraduate professors, Dr. Radhika Vatsan, whose course offering ended up impacting the course of my evolving research interests and the choice of my academic path. I would like to thank her for encouraging me to actively explore this field of research by supporting me to pursue reading courses under her guidance and be her teaching assistant for later editions of her "Introduction to Quantum Computing" course.

My thanks to all the researchers (past and present) at the Centre for Quantum Technologies (CQT), and especially the CQT CS group, for fostering an environment where a free exchange of ideas (research related or not) is encouraged and where it has been a pleasure to learn to spread my academic wings. Additionally, my thanks to: my fellow PhD students and postdocs for the spirited discussions over coffee, chocolates, tea and more; my erstwhile office mates Penghui Yao, Ved Prakash, Attila Pereszlényi and Supartha Podder for the fun everyday banter; Jamie Sikora for his ever-available pointers on surviving a PhD and more; Jamie's better half Caitlin Cooke for her friendship and the willingness to put up with the quirks of all the quantum information scientists around her; Laura Mančinska, my go-to conference roommate, for providing thoughtful advise whenever it has been sought,

with the additional bonus of a feminine perspective; the growing numbers of CQT board-gamers who always know how to show you a good time on a Thursday night; and the Generation Q-Camp organisers who have allowed me to indulge in my love of teaching and cryptography. I would also like to thank the administrators at CQT for their efficiency and diligence in providing administrative support.

I believe these last few years would have been a far less enriching experience without the presence of my cheering (up) squad. My thanks to: Sarvagya Upadhyay and Neeti Kohli for effectively adopting me into the family; Nandhini Balaji and Saurabh Menon for being great company with fun, mirth, delicious Onam/Vishu food and a sprinkling of attacks on the squash court; my partners-in-crime these last few years Karthika Ramanathan and Mann Stha; my favourite lunch-buddy and sometimes only human contact during the last stages of my thesis-writing phase Chaitali Dighe; and a multitude of others whose path intersected mine in passing either during work or play. Thanks for always lending a ear anytime I wanted to get a load off my mind and the fun escapades ranging from concert halls to island getaways.

A heartfelt gratitude goes to my parents for being the constant in my life. Thank you Mom and Dad for always believing in me even if I didn't and being my true-north as I navigate the calm waters or stormy seas that life brings.

Finally, I would like to dedicate my thesis to my grandmother and the memory of my grandfather who was diagnosed with cancer mere days before I enrolled for my PhD and who passed away less than a year later. Grandma defied the norms to become the most educated woman of her generation in our family. As the first college graduate in our family, Grandpa placed immense value in providing an education both for his sons and daughters. Their views instilled in me a love for knowledge acquisition which, over time, blossomed into my desire to pursue a PhD and a career in research.

Summary

Following the introductory chapters (Chapters 1–3), the results in this thesis can be divided into two distinct parts. The first part studies a specific quantum constraint satisfaction problem – Quantum 2SAT (Q2SAT) – motivated by the desire to understand the connection between the computational power of quantum physics and the behaviour of physical systems.

- Chapter 4 presents a new linear time algorithm for Q2SAT designed in collaboration with Itai Arad, Miklos Santha and Shengyu Zhang [ASSZ16] which improves on the previous quartic time dependence on input size and also matches the best complexity of classical 2SAT. The techniques used hinge on extending Boolean unit propagation to encompass high rank and entangled constraints. This chapter also considers the behaviour of almost satisfiable Q2SAT instances i.e. those that are slightly distorted from satisfiable instances. As part of the same collaboration, in ongoing work [ASSZ17], we construct almost satisfiable Q2SAT instances where it is NP-hard to approximate a satisfying assignment.

The latter part of the thesis investigates the behaviour of classical constraint satisfaction problems (CSP) and quantum satisfiability (QSAT) in the trial and error model. The principal component in this setting involves an oracle which accepts assignments from the domain of the CSP as trials and reveals either that the trial is a valid solution or some information about the error that caused the assignment to fail.

- Chapter 5 provides a systematic framework, developed with Gabor Ivanyos, Raghav Kulkarni, Yourming Qiao and Miklos Santha [IKQ+14], to analyze how the complexity of a CSP with hidden inputs (H-CSPs) changes with respect to the information revealed by the oracle. This is done using transfer theorems that equate the complexity of a H-CSP to that of a potentially harder CSP by borrowing techniques from the world of relational algebra with the aim of classifying H-CSPs efficiently. The versatility of these transfer theorems is then demonstrated with their wide-ranging applications.
- Chapter 6 considers the minimum amount of information required to efficiently solve classical 1SAT and 2SAT when the input is unknown. Together with Itai Arad, Adam Bouland, Daniel Grier, Miklos Santha and Shengyu Zhang [ABG+16], we consider trials that are probability distributions over assignments to the variables. The oracle then returns the index of the constraint

that is most likely to be violated by this distribution. The information obtained this way is sufficient to solve 1SAT and 2SAT (without repeated clauses in the formula) in polynomial time. Extending these techniques to the quantum regime, we show that Q1SAT can be can be solved in polynomial time up to constant precision Q2SAT can be learned polynomial time up to inverse polynomial precision.

- Chapter 7 examines, with results from a manuscript [Sun17], the minimum amount of information required to determine if an unknown n-vertex graph has a monotone graph property. For this, the unknown graph is represented as a H-CSP where the constraints represent an edge missing from the input and the trials are certificates to the property in question. When the proposed certificate contains edges missing from the input graph, the oracle in Ivanyos et al. [IKQ+14] returned an arbitrary violated constraint index. Many properties with efficient algorithms in the standard setting cannot be solved in polynomial time with this oracle unless P = NP. To remedy this, the oracle is modified to return the smallest violated constraint index and this setting gives rise to polynomial time algorithms for a large class of monotone graph properties. Adapting this algorithm to binary clauses gives an efficient algorithm for hidden 2SAT formulas resolving the unsolved case of 2SAT with repetitions posed Arad et al. [ABG+16].

The results presented here utilise techniques from classical and quantum computational complexity, graph theory machine learning, relational algebra and first order logic.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

"If at first you don't succeed, try, try again" - an English proverb

Trial and Error - a historical perspective

This quote captures not just the essence of the process familiar to all researchers, but also the essence of the *trial and error* approach to problem solving. Trial and error is a fundamental heuristic problem solving method and the Merriam-Webster dictionary defines it as "a finding out of the best way to reach a desired result or a correct solution by trying out one or more ways or means and by noting and eliminating errors or causes of failure" [Dic17]. This highlights both its intuitive appeal and also its inherent connection to learning theories.

Generally, given a problem, one proposes various candidate solutions called *trials* and observes their validity. If the candidate satisfies the problem, the process aborts on having succeeded. If not, the characteristic of the problem would reveal some *error* associated with the trial. By repeating this process, one would like to minimise the number of trials used to reach a valid solution to the problem at hand. The goal here is finding a solution as opposed to developing a theory on why some particular solution works.

The term "trial and error" was first coined by 19th-century British psychologist C. Lloyd Morgan as part of his studies to explain animal behaviour and animal cognition [Tho83]. One of the first experiments with trial and error, which later pioneered the field of experimental analysis of behaviour, was conducted by Throndike [Tho98] where a cat confined to a puzzle-box with food on the outside, escaped the puzzle box by learning to solve the puzzle. The cat first exhibited random movements (i.e., errors) before opening the puzzle box. However, as the process was repeated, the cat gradually succeeded in opening the box with fewer errors and more deliberate movements. A theory prevalent at that time was to attribute many of the seemingly anthropomorphic behaviour of animals to trial and error learning. To this day, this method is used to analyze the problem solving behaviour of various mammals and insects [CJ15, SCT00]. In fact, biological evolution can be thought of as using trial and error over immense scales in terms of time or organisms involved. Over time, the trial and error learning theory has been the starting point from which more detailed learning theories both for humans and animals have been developed [HG96, SC03, MM14].

Beyond the field of comparative psychology, using trial and error to find real-world solutions to problems has extensive applications in an increasingly unpredictable and complex world [Har11]. These range from the simplicity of solving a sudoku puzzle to the highly complicated world of effective drug design. It is especially useful in cases where a lack of input information may make it hard to apply other targeted algorithms to solve the problem at hand as the following examples demonstrate:

(a) Puzzle Solving - Even without using the extensive literature in psychology to attest to, personal experience solving any one of your favourite puzzles will provide sufficient evidence for trial and error to be a significant tool in this process. This can be seen in the search for that one piece of the jigsaw to fit in somewhere or in systematically eliminating the possible choices when faced with a multiple choice question.

(b) Game Theory and the two-sided market place - Games modelling real world situations usually come with a large dose of unknowns: which strategies are available to a player and what are their payoffs exactly? Who are the other players and what actions are they taking? Or how do the actions of other players affect one's own payoff? Trading in the stock market or matching candidate preferences in a workforce to employer preferences, optimizing business strategy in a market with competing forces all fit this scenario. In recent times, game theorists and economists are exploring decentralized or adaptive learning techniques used by players to suitably modify their strategies and better their payoffs. One such technique is by using interactive trial and error, to find an efficient Nash equilibrium for a large class of games [You09, PY12, BCD+13].

(c) Policy Making and Product Development - S. Callander uses a novel technique of mapping the process of learning through trial and error to the mathematics of Brownian Motion. This has allowed for understanding when an agent tries a new trial as opposed to a previously tested method, how much risk is assumed while generating a new trial, and what is the efficacy of this method in solving hard problems with little or no information. This has been studied for entrepreneurs optimizing a product's innovative design throughout its life cycle [Cal11a] and how well policy makers are able to identify good public policies in a dynamic landscape [Cal11b]. At an organizational level, the pitfalls of singularly risk-averse practices [CM16a] and those governed by bounded rationality and cognitive maps [Nel08] are strongly tied to trial and error learning at the core.

(d) Drug Design - Classical pharmacology or phenotypic drug discovery has relied on using an empirical trial and error approach to identify key compounds that have therapeutic benefits by seeing their effects in cellular or animal disease models [LUM+12]. Only after this are efforts made to discern the biological action of the compound and understand its mechanism eventually leading to the creation of a new drug. This form of drug discovery still remains part of the industry standard as it helps find drugs with novel mechanisms and specifically, for diseases whose physiological mechanisms are not yet well understood [Kot12, ZTM13]. Recent advances also suggest combining both this more black box approach with other target-specific techniques to improve the timeline for developing new drugs.

Related Work

Although used as a tool in a wide variety of fields, the effectiveness of the trial and error method had not been analyzed from the computational perspective until it was recently formalized by Bei, Chen and Zhang [BCZ13]. In this work, they considered some specific problems where the input problem is hidden and accessed by what they term as a *verification oracle* V. One aims to solve the problem at hand by proposing possible solutions as trials to V. The oracle either informs that a solution has been successfully found or provides some information pertinent to the structure of the problem on why the trial failed as the example below indicates.

Example 1.1 (Boolean Satisfiability)**.** *Let the hidden problem be an instance Boolean Satisfiability (*SAT*) which is a Boolean formula $\phi = \bigwedge_{j=1}^{m} C_j$ on n variables given in conjunctive normal form (CNF) i.e. an AND of ORs where each C_j is the OR function acting on a subset of the variables. The verification oracle* V *accepts as trials an assignment to the n variables from the set $\{0,1\}^n$. A proposed trial succeeds when the OR function of* every *clause in the instance evaluates to true on it. When a trial fails, the oracle reveals the* index *j of some clause C_j such that the OR function on the variables in C_j evaluates to false on the proposed trial.*

The authors allow for the presence of a computational oracle C that can solve an instance of the original (i.e., un-hidden) problem. In the case of Example 1.1, C will be an oracle which, when queried with a Boolean CNF formula, will give an assignment such that all clauses of the formula evaluate to true. This makes sense when the focus is to measure the *extra difficulty* introduced by the lack of input information.

For their analysis, the authors of [BCZ13] considered several well-structured problems beyond Boolean Satisfiability (SAT) like Stable Matching [GS62], Sorting a list, Graph Isomorphism [GJ79] and Group Isomorphism [Her06]. They showed that while the difficulty of the first three problems in the hidden setting remains the same as the general setting, the latter two cases have their complexities magnified in the unknown input model. The common denominator among these problems is that each of them is a *constraint satisfaction problem*.

Constraint satisfaction problems (CSPs) are mathematical decision problems which are a cornerstone for research in theoretical computer science. Informally, they are a set of finite objects whose state, picked from a predefined domain, has to satisfy some finite set of constraints imposed on them. The uniformity of structure among CSPs allows them to describe a wide range of problems and promotes their study in a plethora of contexts. Some prominent examples include scheduling and asset allocation, circuit design, graph colouring and network flow related problems, subset sum, etc. Hence, they make excellent candidates to help formalise the trial and error model in different contexts.

One such context is that of determining the feasibility of a linear program,[1] when the constraints are unknown, as considered by Bei, Chen and Zhang in [BCZ15]. The verification oracle that

[1] A linear program is a mathematical optimization model whose constraints are given by linear functions over the variables that are to be optimized.

reveals an arbitrary violated constraint (i.e., half plane) in the linear program is shown to be inefficient – requiring exponentially many queries – to determine the feasibility of the program. However, the slightly modified oracle which reveals the index of the *worst violated* constraint (i.e., the half-plane that is furthest in Euclidean distance from the current trial), can determine the feasibility in polynomial time when used in conjunction with a variant of the ellipsoid method [LGS88].

Our Contributions

In most cases, bespoke methods give the best guarantees of solving a particular CSP but the regularity in their formulation and interchangeability admit a common thread in problem solving techniques for CSPs where domain specific knowledge of the problem is not a must. Among these techniques are heuristic based backtracking search of the solution space and resolution based constraint propagation. Sometimes, a CSP instance is accessed via an oracle and information about the instance may be gleaned by performing queries to this oracle. Examples include oracles with adjacency list queries or edge queries used to solve graph problems and decision tree models to analyze the complexity of Boolean functions. The trial and error model discussed in this thesis, recently formalized to the complexity framework, aims to combine attributes of the oracle access and heuristic methods.

Following the notion of bespoke methods, the previous works dealt with specific constraint satisfaction problems and discuss their behaviour in the trial and error model. What was lacking, however, was a *systematic framework* that could analyze a wide variety of CSPs paired with different verification oracles by taking advantage of the regularity in their formulation. This is the starting point of our work in [IKQ+14], which is presented in Chapter 5. As a first step, we adopt a more formal framework of CSPs which is detailed in Chapter 2. At this point, for the discussion to proceed, it suffices to view an instance of a CSP S as being defined on a set of n variables, $\mathcal{V}$ and a set of m constraints $\mathcal{C}$ involving those variables. Each constraint $C_j \in \mathcal{C}$ has arity q i.e. C_j acts on at most q variables from $\mathcal{V}$. Further, each constraint comes from a fixed set of r relations, $\mathcal{R}_{\mathrm{S}}$, that defines the type of the CSP S as explained in the example below.

Example 1.2 (Boolean Satisfiability (continued))**.** *When* S *is a Boolean CNF formula of arity at most* 2*,* $\mathcal{R}_{\mathrm{S}}$ *is the set of all possible OR functions on at most* 2 *variables, (say)* a *and* b

$$\mathcal{R}_{\mathrm{S}} = \{(a \vee b), (\bar{a} \vee b), (a \vee \bar{b}), (\bar{a} \vee \bar{b}), (a), (\bar{a})\}$$

where $\bar{a}$ *denotes the negation of* a*. Now, any instance* $\phi = \bigwedge_{j=1}^{m} C_j$ *of* S*, is such that each* $C_j \in \mathcal{R}_{\mathrm{S}}(a, b)$ *involves at most* 2 *of the* n *variables in* ϕ *with* $a, b \in \mathcal{V}$*.*

Considering an instance of a CSP S specified by the tuple $(\mathcal{C}, \mathcal{V}, \mathcal{R})$ allows for the presence of a larger variety of verification oracles. In the trial and error setting, the trials for this instance are assignments to the variables in $\mathcal{V}$ form an appropriate domain. The errors would be some information related to one of the constraints from $\mathcal{C}$ that is violated. As the purpose of verification oracles is to reveal some information pertaining to a violated constraint, they are also

referred to as *revealing oracles*. The oracle considered in Example 1.1 is a $\mathcal{C}$-revealing oracle as it reveals the index of a violated constraint. In this spirit, it is possible to consider any $\mathcal{U}$-revealing oracle (also denoted as $\mathcal{O}_{\mathcal{U}}$) for $\mathcal{U} \subseteq \{\mathcal{C}, \mathcal{R}, \mathcal{V}\}$. Suppose that one is interested in knowing the index of a violated constraint as well as the variables involved in this constraint, the pertinent oracle would be the $\{\mathcal{C}, \mathcal{V}\}$-revealing oracle. Or when one would like to know the index of a violated constraint and the index of the relation this constraint possesses, the $\{\mathcal{C}, \mathcal{R}\}$-revealing oracle will be used. Now, an instance of a *hidden* CSP (H–CSP) S where the constraints are unknown is given by $n = |\mathcal{V}|$, $m = |\mathcal{C}|$, and a corresponding $\mathcal{U}$-revealing oracle. To analyze the worst case complexity of a problem in this setting, when more than one constraint evaluates to false as in Example 1.1, V picks an index adversarially.

The next step is to use this framework and classify the complexity of a H–CSP combined with its revealing oracle. This is done by developing *transfer theorems* for each type of revealing oracle. For each hidden CSP, the transfer theorem associates a more complicated CSP in the un-hidden setting such that their difficulties are roughly the same. The more complicated CSP is obtained by performing specific *extension* operations on the relation sets $\mathcal{R}$ of the H–CSP that causes a blow up in their resulting size and complexity. This transfers the study of hidden CSPs to the study of these extended CSPs in the well-studied un-hidden setting. It is possible to view these transfer theorems as being in the spirit of structural classification theorems like Schaefer's Dichotomy Theorem [Sch78] for Boolean CSPs and Chen's generalisations using relational algebra [Che09]. The advantage of the transfer theorems is its suitability to study any H–CSP that can be represented using the $\{\mathcal{C}, \mathcal{R}, \mathcal{V}\}$ framework which conceivably covers every decision CSP. Our framework also works for hidden CSPs defined with some promise on their inputs, for example, a Boolean satisfiability instance which is promised to be *repetition free* i.e. where every clause in the instance is unique. An interesting class of problems that can also be suitably expressed is those determining if a hidden graph possesses a monotone graph property – graph properties preserved under the addition of edges. It bears mentioning that the $\{\mathcal{C}, \mathcal{V}\}$-revealing and $\{\mathcal{C}, \mathcal{R}\}$-revealing oracles along with the results involving them are considered for the first time in [IKQ+14] as [BCZ13] only considered the $\{\mathcal{C}\}$-revealing oracle in their setting. Additionally, the results for certain promise problems and monotone graph properties are studied here for the first time too. A side benefit of these transfer theorems is that, when applied to the examples considered in [BCZ13], they generally produces shorter and easier proofs of their complexity.

The intuition behind these extensions and the transfer theorems comes from the necessity for a methodical way to represent the information aggregated about the H–CSP instance after a series of trials and revealed violations. To accomplish this while still maintaining a link to the underlying structure of the instance, for every H–CSP instance H–S with m constraints, the extended CSP also possesses m constraints. So, intuitively, every constraint C'_j of the extended CSP, at any point, reflects the knowledge gained about the corresponding hidden constraint C_j. For this to be most effective, it is imperative that the revealing oracle always reveal the index of some violated constraint. Thereby, the revealing oracles that are of particular interest are

$\{\mathcal{C}, \mathcal{U}'\}$-revealing oracles for $\mathcal{U}' \subsetneq \{\mathcal{R}, \mathcal{V}\}$ where $\{\mathcal{C}, \mathcal{R}, \mathcal{V}\}$ is excluded as it can be shown to be polynomially equivalent to the un-hidden setting. Informally, the CSP extensions considered for some CSP S are: (a) *closure by union* ($\bigcup$S) whose relations could be an arbitrary union of the relations in the original problem; (b) *restricted union of arity extensions* (E–S) whose relations are unions of a relation in the original problem over different q-ary tuples of variable indices; and (c) *arbitrary union of arity extensions* ($\bigcup$E–S) whose relations are arbitrary unions of the relations from the previous extension. Using these, our first contribution can be stated below.

Result 1 (Transfer Theorems; cf. Section 5.2)**.** *Let* S *be a CSP whose parameters are "reasonable" with relations such that any assignment satisfying a relation can be determined in* P*. Then, the complexities of the following problems are polynomial time equivalent:*
(a) H–$\text{S}_{\{\mathcal{C},\mathcal{V}\}}$ *and* $\bigcup$S, *(b)* H–$\text{S}_{\{\mathcal{C},\mathcal{R}\}}$ *and* E–S, *(c)* H–$\text{S}_{\{\mathcal{C}\}}$ *and* $\bigcup$E–S.

What we observe is that the complexity of any hidden CSP is intimately connected with the complexity of the revealing oracle used. In the case of Boolean satisfiability or some simple monotone graph properties, among others, the $\{\mathcal{C}, \mathcal{V}\}$-revealing oracle allows the hidden instance of a CSP to be learned efficiently when an accompanying computational oracle C to solve the original CSP is provided. This reduces the complexity of the hidden CSP to that of the original CSP. Hence, this oracle may not blow up the complexity of the problem. The oracle that provides the least amount of information and heavily restricts access to the input is the $\{\mathcal{C}\}$-revealing oracle. Using this oracle, a problem's complexity in the hidden setting can be magnified to become NP-hard while the un-hidden version of the problem can be solved in polynomial time. For instance, 2SAT, which is the Boolean Satisfiability problem of arity 2, has a linear time algorithm [APT79, EIS76]. However, the hidden 2SAT problem (H–2SAT) with the $\{\mathcal{C}\}$-revealing oracle cannot be solved in polynomial time unless P = NP. Here P, NP refer to the polynomial time and nondeterministic polynomial time complexity classes respectively and H–$\text{2SAT}_{\{\mathcal{C}\}}$ is said to be NP-hard in shorthand. The same behaviour in the hidden setting is also reflected for many monotone graph properties like connectivity, bipartite perfect matching, etc., which have efficient algorithms in the un-hidden setting [Die12, Edm67, Tut54]. Some of the examples where transfer theorems were applied are summarised in Result 2.

With the $\{\mathcal{C}, \mathcal{V}\}$-revealing oracle being too powerful by allowing an instance to be learned completely and the $\{\mathcal{C}\}$-revealing oracle being extremely restrictive, as in the case of H–2SAT, it is natural to ask if there is some oracle of intermediate power that would allow one to solve a hidden instance without necessarily being able to completely learn an instance. A problem in the hidden setting only becomes more difficult to solve leading the focus to be on problems that can be solved efficiently when the input is completely known. As trials are proposed, the information gleaned from each violation could help one narrow down not just a satisfying assignment but also what the hidden instance could be as the example below shows.

Example 1.3 (Boolean Satisfiability (continued))**.** *Let the* SAT *formula* ϕ *on* n *variables be hidden by a* $\{\mathcal{C}\}$*-revealing oracle as in Example 1.1. Let the assignment* 0^n *fail on* ϕ *with the*

Result 2 (Applications of Transfer Theorems; cf. Sections 5.3, 5.4, 5.5, 5.6)**.** *Given revealing oracles* $\mathcal{O}_{\{\mathcal{C}\}}, \mathcal{O}_{\{\mathcal{C},\mathcal{V}\}}, \mathcal{O}_{\{\mathcal{C},\mathcal{R}\}}$*, the following hold:*

(a) *Boolean Satisfiability:* $\text{H–1SAT}_{\{\mathcal{C}\}}, \text{H–2SAT}_{\{\mathcal{C}\}}, \text{H–2SAT}^{\mathsf{RF}}_{\{\mathcal{C}\}}$ *are* NP*-hard but* $\text{H–1SAT}^{\mathsf{RF}}_{\{\mathcal{C}\}} \in \text{P}$ *where* RF *refers to the promise that an instance is repetition free.*

(b) *Unique Games (for definition cf. Section 5.3):* $\text{H–UG}[2]_{\{\mathcal{C},\mathcal{V}\}} \in P$ *but* $\text{H–UG}[k]_{\{\mathcal{C},\mathcal{V}\}}$ *is* NP*-hard for* $k \geq 3$ *where* $\mathsf{UG}[k]$ *denotes a unique game on an alphabet of size* k*.*

(c) *Monotone graph properties: Given a hidden graph* G *accessed using an* $\mathcal{O}_{\{\mathcal{C}\}}$ *oracle, it is* NP*-hard to determine if* G *has: (1) a spanning tree, (2) a perfect matching (if* G *is bipartite), (3) a cycle cover, (4) a path between two marked vertices* s *and* t*.*

(d) *Group Isomorphism (for definition cf. Section 5.6): Given a multiplication table* H *which is hidden by an* $\mathcal{O}_{\{\mathcal{C},\mathcal{V}\}}$ *oracle, and a family of groups,* $\mathcal{G}$ *whose multiplication table structure is known. It is* NP*-hard, in the trial and error setting to determine if* H *is equal to a multiplication table from the family* $\mathcal{G}$*.*

oracle returning the index j*. The all-*0 *assignment would have set any negated variable in* C_j *to true in which case it could not have caused a violation. So, it is possible to conclude that the clause* C_j *does not contain any negated variable. Similarly, if the all-*1 *assignment* 1^n *fails on* ϕ*, one can conclude that* ϕ *contains at least one clause made of only negated variables.*

So, the question on finding an oracle of intermediate power to solve a CSP S can be rephrased as: *What is the minimum amount of information required to solve a CSP* S *efficiently?*. This question is fielded in Chapter 6 (where the CSP is 2SAT) and Chapter 7 (where the problems are various monotone graph properties).

As evidenced by the discussion so far, a recurring CSP candidate throughout this work is Boolean Satisfiability (SAT). One reason for this is its simple structure as a Boolean formula in the form of a conjunction of *clauses*, where each clause is a disjunction of *literals* (variables or negated variables). This simple structure is still capable of expressing problems related to scheduling, circuit design, operations research and bioinformatics to name a few. The other reason is due to its pivotal position in theoretical computer science with the advent of the Cook-Levin Theorem [Coo71, Lev73] proving its NP-completeness. Moreover, it continues to be studied under various specialized models such as random-SAT and building efficient SAT-solvers for real-life scenarios. In the complexity theoretic setting, we know that while 3SAT is NP-complete [Coo71, Lev73], 2SAT can be solved in linear time [Kro67, EIS76, APT79]. kSAT indicates that each clause contains at most k literals.

Starting with the hidden versions of 1SAT and 2SAT, H–1SAT and H–2SAT respectively, our work in [ABG+16], presented in Chapter 6 aims to find a revealing oracle that can solve the problems without entirely learning them. As randomization sometimes provides speedups in query complexity and other models, the trial and error model considered here introduces randomization in form of *probabilistic assignments*. This means that a variable, instead of taking

Boolean assignments, is assigned a value in the interval $[0, 1]$ which signifies its probability of being true (or equivalently, 1).By assuming a product distribution over all the variables, it is possible to generate trials which are a probability distribution over Boolean assignments.

A probability distribution over assignments translates to the clauses too by signifying the probability with which they can be violated instead of a true or false value. It also induces a natural order on the extent to which a clause is violated. For instance, if $\Pr[a = 0] = 0.9, \Pr[b = 0] = 0.9$ for variables a, b, then for a clause $C = (a \vee b)$ that will be violated by 00, $\Pr[C(a, b) = 0] = 0.9^2 = 0.81$ and for a clause $C' = (a \vee \bar{b})$ that will be violated by by 01, $\Pr[C'(a, b) = 0] = 0.9 \times 0.1 = 0.09$ where $\Pr[\cdot]$ denotes the probability of an event happening. To take advantage of these probabilistic violations, the oracle, denoted $\mathcal{O}_{\text{MAX}}$, returns the index of the clause that has the highest probability of being violated by the current trial. This mirrors the *worst violated constraint* oracle used in [BCZ15] to efficiently solve linear programs with unknown constraints.

Using the $\mathcal{O}_{\text{MAX}}$ oracle, we obtain an algorithm to solve H–1SAT in polynomial time by using the trials to construct partial assignments which are then extended to full assignments. Any partial assignment that extends to a satisfying assignment is called *good*; otherwise, it is *bad*. At each point it is possible to separate the trials proposed into buckets which are labelled by the violated index $\mathcal{O}_{\text{MAX}}$ returns. The key idea is that all trials in a bucket are either good or all trials are bad. Now, one proceeds by picking one representative from each bucket to extend to a satisfying assignment. By ensuring that at least one bucket has good assignments, the algorithm guarantees a solution if the hidden instance is satisfiable.

We find that extending the same concept to H–2SAT poses a problem as the buckets are unable to separate the good and bad partial assignments. This can be understood by observing the nature of 1SAT and 2SAT formulas. In the former's case, a partial assignment on the first i variables is good if it satisfies all the clauses involving the first i variables. For instance, when $\phi = x_1 \wedge x_2 \wedge \overline{x}_3 \wedge \overline{x}_4$, the partial assignment $x_1 x_2 = 11$ is a good assignment. Any bad assignment here will not be placed in the same bucket as it would violate either the first or second clause by a maximum violation of 1. Consider now a 2SAT formula $\phi' = (x_1 \vee \overline{x}_2) \wedge (\overline{x}_2 \vee x_3) \wedge (\overline{x}_3)$. The two probabilistic assignments $x_1 = 1, x_2 = 1, x_3 = 0.5$ and $x_1 = 1, x_2 = 0, x_3 = 0.5$ both satisfy the first clause in ϕ'. They both also violate the third clause by 0.5 and could be placed in the same bucket during the algorithm. However, the partial assignment $x_1 x_2 = 11$ is bad while $x_1 x_2 = 10$ is good. This prompts a different approach for H–2SAT. We show that when a H–2SAT instance ϕ' is repetition-free, i.e., it is promised not to contain any repeated clauses, it is possible to find a 2SAT formula $\bar{\phi}$ such that ϕ' and $\bar{\phi}$ have the same set of satisfying assignments. Then solving $\bar{\phi}$ for a satisfying assignment, will provide us with a solution for ϕ' giving the following results.

It seems unlikely that the repetition-free condition for H–2SAT can be bypassed when the $\mathcal{O}_{\text{MAX}}$ oracle is involved[2]. On the other hand, a positive consequence of the probabilistic trials is that it

[2] A slightly different approach to remove the repetition-free condition is considered in Chapter 7.

acts as a stepping stone to deal with *quantum satisfiability* (QSAT) in the trial and error setting.

Quantum Satisfiability [Bra11] is a special case of the Local Hamiltonian problem [KSV02] and they are formally introduced in Chapter 3 along with some necessary quantum information concepts. The point to be made here is that the Local Hamiltonian problem, and by relation QSAT, is arguably as pivotal to studies in condensed matter physics and quantum complexity theory as SAT is to classical computational complexity. This makes QSAT a suitable candidate amongst quantum constraint satisfaction problems to be considered for any quantization of the trial and error model. A QSAT instance shares some structural resemblance to classical SAT. For example, QSAT would act on a system of n qubits with m quantum constraints. The space of n-qubit quantum states is the space of assignments for QSAT. Similar to kSAT, a QkSAT instance is such that each constraint acts on at most k qubits. The satisfying assignment of a k-qubit constraint is a k-qubit *ground state* and a ground state of the instance is also a ground state of each constraint in the instance.

One main difference in the quantum case is the structure of the constraints which also translates to the structure of the ground states. Each k-qubit QSAT constraint is a Hermitian operator with $0, 1$ eigenvalues on the space of k-qubits and a ground state for such a constraint is its 0-eigenvector. The other major difference is their continuous nature. Unlike a Boolean constraint which takes $\{0, 1\}$ values, a quantum constraint takes values in the interval $[0, 1]$ where 0 signifies that the constraint is completely satisfied and other values signify the degree to which a constraint is violated. Another consequence of the continuous nature is that the algorithms considered in Chapter 6 aim to find a state that is, in a suitably chosen metric, ϵ-far from a ground state. Here, ϵ could be bound by a function, $1/f(n)$, dependent on the number of qubits, n, where $f(n)$ is either a constant, polynomially bounded or an exponentially bounded function.

It is known that Q1SAT and Q2SAT can be solved in polynomial time [Bra11] with deterministic classical algorithms leading us to ask what the minimum information required to solve these problems may be. We use the $\mathcal{O}_{\text{MAX}}$ oracle in the trial and error setting to answer this question. The trials in this case are *density matrices* which can be viewed as a quantum equivalent of a probability distribution over quantum states. We show that given a H–Q1SAT instance and a precision ϵ, there is an algorithm requiring $\text{poly}(n^{\log \frac{1}{\epsilon}})$ time to output a state that is ϵ-far from the ground state in some suitably chosen metric. The algorithm generalises the algorithm used for H–1SAT and the exponential dependence on $\log \epsilon$ arises from the exponential blowup in the number of buckets required in the algorithm to pin down a suitable state for each qubit. When ϵ is a constant, this still gives us a polynomial time algorithm but inverse polynomial precision requires super-polynomial time. As in the classical case, the constraints themselves are not learned and remain hidden and the finite precision adds a further layer of obfuscation on the constraints present in the hidden instance.

The H–Q2SAT instance generalises the H–2SAT algorithm in that it attempts to find a Q2SAT instance that approximates the hidden instance. Specifically, given a H–Q2SAT instance H and a precision ϵ, the algorithm, in $\text{poly}(n)$ time, finds a Q2SAT instance H' such that each

constraint in H' is ϵ-far from the corresponding constraint in H in a suitably chosen metric. H' can be said to have the constraints of H *perturbed* by a factor of ϵ. At this juncture, it is suitable to highlight the crucial difference between the algorithm used for the classical and quantum settings. Essentially, in the classical case, the satisfiability of the H–2SAT formula ϕ is preserved in the equivalent 2SAT formula ϕ' learned by the algorithm. If ϕ is unsatisfiable, then so is ϕ' and vice-versa. However, this cannot be guaranteed for H and H'. It is possible that H is satisfiable (i.e., possesses a 0-eigenvector) but the perturbations in H' make the latter un-satisfiable or vice-versa. This means that running a Q2SAT algorithm on H' would not provide a reliable answer on the satisfiability of H unless the Q2SAT algorithm is robust to ϵ perturbations. This leads to an interesting set of questions on the robustness of Q2SAT algorithms in general[3]. Existing Q2SAT algorithms are robust when ϵ is exponentially small i.e. bounded by $\frac{1}{\exp(n)}$ but not in the inverse polynomial regime. The question of whether there exists any polynomial time algorithm to approximate the ground state in the inverse polynomial regime is left for future work. Hence, our H–Q2SAT algorithm presents an efficient way to learn a $\frac{1}{\text{poly}(n)}$-perturbed instance H' and the question of whether this also solves the H–Q2SAT instance is left open-ended. Now, it is possible to compare and contrast the classical and quantum behaviours with the $\mathcal{O}_{\text{MAX}}$ oracle.

Result 3 (SAT and QSAT with the $\mathcal{O}_{\text{MAX}}$ oracle; cf. Chapter 6)**.**

(a) *Given a* H–SAT *formula* Φ *on* n *variables and* m *clauses, accessed using the* $\mathcal{O}_{\text{MAX}}$ *oracle and probabilistic trials, it is possible to find a satisfying assignment for* Φ *in* $\text{poly}(n, m)$ *time if* Φ *is (1) a* 1SAT *formula or (2) a* 2SAT^{RF} *formula which is repetition free.*

(b) *Given a* H–QSAT *instance* H *on* n *qubits with* m *projectors and* $\epsilon > 0$*, it is possible to*

(i) solve H *to a precision* ϵ *in time* $O(n^{\log \frac{1}{\epsilon}})$ *if* H *is a* Q1SAT *instance or*

(ii) learn *each projector of* H *to a precision* ϵ *in time* $\text{poly}\left(n, \log \frac{1}{\epsilon}\right)$ *if* H *is a* Q2SAT *instance and there is no qubit,* i*, which is common to all the projectors in* H*.*

Another recurring set of problems in this work are *monotone graph properties*. A graph property, $\mathcal{P}$, is the property of a graph that is invariant under the renaming of vertices. The monotonicity comes into place when a graph property that is preserved under the addition of edges and some examples include the connectivity of a graph, a bipartite graph possessing a perfect matching between the vertices on each side of the bipartition and the $s-t$ connectivity for two marked vertices s, t in a given graph. A *certificate* for a monotone graph property is a smallest subgraph of the given graph that posses the property in question. In the case of connectivity, this would be a spanning tree of the given graph and the shortest path between the marked vertices s, t would certify the $s-t$ connectivity of the graph. A monotone graph property $\mathcal{P}$ on an n-vertex graph can be expressed as a monotone Boolean function, and thereby a Boolean CSP, on $\binom{n}{2}$ variables where each variable is an edge in the n-vertex graph. This Boolean function on a graph will evaluates to 1 if the graph possesses a certificate for the property $\mathcal{P}$. With this view, the monotone graph properties framework in the trial and error model introduced in [IKQ+14]

[3]This is discussed in Chapter 4 which presents the result on a new linear time Q2SAT algorithm

aims to find if a hidden graph possesses a property $\mathcal{P}$ by providing its certificates as trials. For instance, there are $n!$ possible spanning trees on n-vertices. These would be the trials made to find if a hidden graph is connected. The oracle used either informs us that the certificate is a subgraph of the input graph or reveals some information indicating why the input graph does not possess that certificate.

Characterizing graphs based on their properties has been the goal of algorithmic graph theory [BT97]. Ever since Euler first posed the question on the bridges of Königsberg in 1736, there have been numerous algorithms [Die12] to characterize when a given graph possesses a particular property [Edm67, Tut54] or test if a graph is close to possessing a property rather than far from it [AS08, GKNR09]. In a similar vein, Chapter 7 characterizes the monotone graph properties for which a certificate can be found efficiently in the trial and error setting. As the discussion previously indicates, this has to be achieved with an oracle of intermediate power between the $\{\mathcal{C}\}$-revealing and $\{\mathcal{C}, \mathcal{V}\}$-revealing oracles. In this model, the oracle, denoted $\mathcal{O}_{\text{FIRST}}$, returns the index of the lexicographically first constraint that is violated i.e., $\min_{1 \leq j \leq m}\{j | C_j \text{ is violated}\}$ where the CSP has m constraints $C_1, \ldots, C_m$. Effectively, the oracle returns the first error encountered when the constraints are checked for violation in the ascending order of their index.

Using the *lex-first* oracle $\mathcal{O}_{\text{FIRST}}$ on a hidden graph, as part of ongoing work [Sun17], we show that a certificate for property $\mathcal{P}$ can be found in polynomial time if an associated problem $\mathcal{P}^{\cap}$ can be solved in polynomial time. Informally, $\mathcal{P}^{\cap}$ refers to the problem of finding a certificate for $\mathcal{P}$ that additionally contains at least one edge from a given set of edges. This raises an interesting question on which properties have polynomial time algorithms to solve their $\mathcal{P}^{\cap}$ problem. First glance may suggest an equivalence between the certificates for $\mathcal{P}$ and $\mathcal{P}^{\cap}$ problems. However, finding an odd-length cycle in a directed graph provides a counter-example to the suggestion. Note that, so far, we are unaware of any property on an undirected graph that would be a suitable counter-example. One consequence of finding a counter example would be to elicit a finer structure amongst polynomial time solvable monotone graph properties. The properties used to illustrate the working of the lex-first oracle seem to be expressible through matroids and matroid intersection [Tut59]. Matroids are structures in combinatorics that abstract and generalize the notion of linear independence in vector spaces [Oxl06]. Hence, there may be a connection between graph matroids and the $\mathcal{P}^{\cap}$ problem but exploring this conjecture is left for future work.

The lex-first oracle, when applied to the case of H–2SAT removes the restriction of repetition-freeness that is present with the $\mathcal{O}_{\text{MAX}}$ oracle. Any instance of the H–2SAT problem oracle can be solved in polynomial time with the $\mathcal{O}_{\text{FIRST}}$. One reason for this is that the lex-first nature implies that when a repeated clause is violated, only the first occurrence of the repeated clause will ever be returned by $\mathcal{O}_{\text{FIRST}}$ as a violation. Breaking ties between repeated clauses is, by itself, insufficient to to make H–2SAT tractable in this setting. The other characteristic of the lex-first oracle is that it is able to identify the precise change of assignments that causes the

status of a particular clause to transition from being violated to being satisfied or vice-versa[4]. This cannot be guaranteed by the $\mathcal{C}$-revealing oracle which is assumed to be adversarial in nature. Consider the case when two successive assignments return the indices $j_1 < j_2$ respectively. With the lex-first oracle, it is possible to conclude that as j_2 is now the smallest clause index to be violated, the second assignment satisfied C_{j_1} and it is the difference between the two assignments that satisfied C_{j_1}. In the case of H–2SAT, the difference is the list of variables that do no have the same value across both assignments. The same conclusions cannot be made with the $\mathcal{C}$-revealing oracle. Incidentally, the above process of examining two successive trials and their violations j_1, j_2 to draw conclusions about what satisfied or violated the clause C_{j_1} is termed *candidate elimination*. This name is justified when one views the algorithm as maintaining a list of *candidates* for the literals that could be present in a clause of the hidden instance. The elimination helps rule out some of the candidates and move closer to the actual literals involved in the clause.

The polynomial time algorithms considered in Chapter 7 with the lex-first oracle are all made up of the same building blocks. Every lex-first algorithm can be broken up into *phases* which in turn is a repeated sequence of the following procedures: (1) the process of constructing the *next trial* $\mathbf{T}_2$ from the current trial $\mathbf{T}_1$, (2) getting a violation j_2 from proposing $\mathbf{T}_2$ to $\mathcal{O}_{\text{FIRST}}$ and (3) using $\mathbf{T}_1, \mathbf{T}_2, j_1, j_2$ to perform candidate elimination. This sequence ends when some previously unknown non-trivial information about a clause is eventually learned as a result of the candidate elimination. For any CSP, this information would amount to revealing the identity of one of the literals that is present in C_{j_1} or C_{j_2}. The next phase then proceeds by picking trials that satisfy the current knowledge the algorithm has about the hidden instance in an attempt to find a satisfying assignment. While the sequence of steps in a phase arguably shares similarities with H–$\mathsf{SAT}_{\{\mathcal{C}\}}$ given in [BCZ13] (cf. Algorithm 2.2), there are some key differences in implementation. For one thing, in [BCZ13], an arbitrary next trial is generated by using the assignment returned by querying a SAT oracle. For the examples using the lex-first algorithm, an explicit process to generate the next trial in polynomial time is described. The second difference is in the candidate elimination where the lex-first algorithm uses the difference between two successive trials and the two violations to determine which candidates to remove and which are to be kept. In contrast, the H–$\mathsf{SAT}_{\{\mathcal{C}\}}$ algorithm eliminates candidates solely on the basis of the current trial and violation. These differences necessitate the need for more involved data structures and book-keeping procedures to effectively minimize the number of trials used and the running time of the algorithms. The results with the lex-first oracle are summarised in Result 4.

The trial and error model is clearly not alone in dealing with unknown inputs through black box oracles. There are numerous such models studied in computational complexity. When determining the query complexity of a Boolean function, given an input Boolean string, the

[4]These conditions are also met by lex-last oracle – one that reveals the largest index among violated constraints making it capable to solve H–2SAT efficiently. The same does not hold for a lex-k^{th} oracle revealing the index of the k^{th} smallest violated constraint.

Result 4 (Graph properties and 2SAT with the $\mathcal{O}_{\text{FIRST}}$ oracle; cf. Chapter 7)**.**

(a) *Given an unknown graph G on n vertices accessed using the lex-first oracle $\mathcal{O}_{\text{FIRST}}$, it is possible in* poly(n) *time to determine if G has: (1) a spanning tree; (2) a perfect matching (when G is bipartite); (3) a cycle cover; (4) a path between marked vertices s, t.*

(b) *Given a* H–2SAT *formula Φ on n variables and m clauses, accessed using the $\mathcal{O}_{\text{FIRST}}$ oracle with Boolean assignments as trials, it is possible to find a satisfying assignment for Φ in* poly(n, m) *time if it exists or output* UNSAT*.*

decision tree model [BDW02] queries the bit at position i in the string. For the quantum query complexity, these queries are made in superposition. When testing if a graph has some property $\mathcal{P}$ or is far from having the property, the algorithms usually query if there is an edge between two vertices u, v or ask for the i^{th} neighbour of some vertex v [Gol11]. The common thread in these models is that it is the input that is queried in different forms. On the other hand, the trial and error model uses more general queries, in the form of assignments, as trials. Also, at no point does any trial query a constraint C_j in the hidden instance directly. It is the series of trials and violations observed that are used to accumulate information about which assignments may satisfy each constraint in the instance.

It is possible to view the candidate elimination used in the lex-first algorithms as being inspired from its namesake in concept learning with version spaces [Mit82]. This leads to questions about the relationship between existing machine learning techniques and the trial and error setting which has also been discussed by Bei, Chen and Zhang [BCZ13]. The main difference comes back to the fact that the goal in the model discussed here is, expressly, to find a solution for the problem at hand. The focus is not on explaining why that solution works or learning the details of the problem at hand in its entirety. For instance, one can compare concept learning with the trial and error setting. Suppose that the problem at hand is to identify a *bird*. Concept learning would proceed to learn all the characteristics that identify birds, say {has feathers, lays eggs, warm-blooded}. In the trial and error model, a series of animals could be proposed as trials and the first bird, say a penguin, in that list is accepted without any information revealed on whether the problem wanted an animal that laid eggs, lives in the cold or is a bird. In the probably approximately correct learning (PAC learning) framework [Val84], with sufficiently high probability, maybe the concept learned is {lays eggs, is warm-blooded}. Then, with some low probability, a duck-billed platypus could be termed as satisfying the concept by the PAC learner. In this case, the PAC learner has a non-zero probability of labelling an unsatisfying assignment as being a solution a scenario that will not occur with the trial and error oracles. So, while there is a chance for techniques from machine learning to get appropriated for use in the trial and error setting, algorithms in both models exhibit distinct goals.

Lastly, presented in Chapter 4 of this thesis, is the result that is independent of the trial and error model but pertains to quantum CSPs, namely Quantum Satisfiability. We show a classical, deterministic linear time algorithm to find the ground state for Q2SAT [ASSZ16] that improves on the previous best of an $O(n^4)$ time algorithm provided by Bravyi [Bra11] where the input

acts on n qubits. Our algorithm can be viewed as a quantum generalisation of the refined Davis-Putnam procedure [DP60, DLL62] used for 2SAT which relies on *unit propagation* and is a backtracking based search of the solution space. Unit Propagation acts on unary clauses, i.e., clauses with a single variable, and resolves the assignment to the variable so that the unary clause is satisfied. This is particularly useful for Boolean variables when there is only one possible assignment that can satisfy a unary clause. Backtracking involves building partial assignments or candidate solutions which are abandoned (by going back to a different assignment) as soon as it is determined that the candidate cannot be extended to a valid solution. This procedure, in the worst case, can take exponential time to find a solution for a general SAT instance but consumes quadratic time on a 2SAT formula. A tweak suggested by Even, Itai and Shamir [EIS76] for the Davis-Putnam procedure improves its running to a linear time on the input of a 2SAT formula. In the 50+ years since its inception, the Davis-Putnam procedure remains an integral part of modern SAT-solvers.

Given a 2SAT formula containing only two variable clauses, the Davis-Putnam algorithm does the following. It picks an unassigned variable, say x, and gives it an arbitrary assignment. The formula is then simplified by removing clauses that are already satisfied and resolving any remaining unary clauses generated during the course of the simplification. If no clause was violated in this process, we are left with a formula containing a smaller number of two variable clauses and the process can be repeated. If not, the violation occurs due to the current assignment given to x and the procedure backtracks to the point where x was assigned. This assignment is flipped to the other possibility and unit propagation is repeated. If this also leads to a contradiction, the input formula is unsatisfiable. If not, a partial assignment is built in a step-wise fashion. Generalizing this to deal with quantum states requires an efficient notion of unit propagation and showing that Q2SAT also requires only limited backtracking. Our result states the following.

Result 5 (Linear time Q2SAT algorithm; cf. Chapter 4)**.** *There exists a deterministic algorithm for* Q2SAT *whose running time is* $O(n+m)$ *where* n *is the number of qubits,* m *is the number of local terms in the instance and it is assumed that arithmetic operations on complex numbers consume unit time.*

The existence of classical algorithms for a quantum problem such as Q2SAT is possible due to the fact that every satisfiable Q2SAT instance has a ground state that with a polynomial sized classical description as shown through our *product state theorem*. The next consideration is expanding the notion of unit propagation involving quantum states and quantum constraints. The latter, as Hermitian operators, can be of rank 1, 2 or 3. Further, the rank-1 constraints come in two distinct forms: *product constraints* and *entangled constraints*. While the former bear a semblance to Boolean clauses except on a larger domain, the latter are intrinsically quantum in nature. Our definition of propagation deals with each of these cases such that for a constraint involving two qubits u, v, where qubit u is assigned some initial state, we can find

the unique state[5] that will be propagated across the constraint to qubit v with $O(1)$ operations on complex numbers. This can then be extended to show multi-step propagation across a series of constraints.

The Q2SAT algorithm can now be broken up into three distinct phases that acts on the input given as a constraint graph where a qubit pair share an edge if a constraint involves the pair: (a) Resolve rank-3 constraints; (b) resolve rank-1 product constraints and (c) resolve rank-1 entangled constraints. Each of these phases makes extensive use of propagation. While the first two phases bear a resemblance to the Boolean case adjusted to the larger domain size, the last step is solely created for the quantum instance. It borrows the idea of *sliding* a constraint across another from [JWZ11]. Limited backtracking is shown separately for each phase. Rank-3 constraints have unique satisfying assignment and cannot be backtracked. The satisfying assignment for rank-1 product constraints has one of two forms and so an assignment to them can be backtracked once before both the forms are tried for satisfaction. Repeated sliding in the last phase would produce a rank-2 constraint on some qubit pair. This is handled by showing that there exists a rank-1 product constraint in every rank-2 constraint's space and reduces to the problem of satisfying this rank-1 constraint which can be backtracked once.

A combination of factors lead to our algorithm having a linear time bound. One is that there is no global manipulation of the algorithm as is done by Bravyi's approach. Instead, parts of the instance are analyzed at any given time with local operations manipulating the partial assignments generated. Moreover, once a partial assignment has been found successfully, it can be decoupled from the rest of the system leaving a successively smaller instance that needs to be solved. Each propagation and sliding operation manipulates a constant sized matrix thereby requiring $O(1)$ operations on complex numbers. Additionally, repeated propagation and sliding act on the constraint graph using breath first traversals which can be executed in linear time [CLRS09]. Amortised over each phase, every edge occurs only in a constant number of breadth first traversals which justifies the bound. An underlying assumption used here is that operations over complex numbers consumes unit time i.e. we work in the algebraic computing model. For the sake of completeness, we can also consider the bit complexity of the algorithm – the complexity of the bitwise operations performed over the course of the algorithm. When $M(n)$ is the bitwise cost of multiplying two n-bit numbers, the algorithm has a bit complexity $O((m+n)M(n))$ where the input acts on n qubits and has m constraints. Independently, albeit simultaneously, de Beaudrap and Gharibian [dBG16] also presented a linear time algorithm for Q2SAT which largely differs in the way rank-1 entangled constraints are handled. Their algorithm can be considered a quantum analogue of Apsvall, Plass and Tarjan's algorithm for 2SAT [APT79] and matches the complexity of our algorithm in both the algebraic computing and bit complexity regimes.

Chapter 4 also presents as yet unpublished results that are part of ongoing work [ASSZ17]

[5] Up to some global phase and scaling factors.

related to the behaviour of specific *distorted* instances of Q2SAT. For some $\epsilon > 0$ and some Q2SAT instance H, these distortions generate an instance H' such that each local term of H' is ϵ-close to the corresponding local term of H. When $\epsilon = \frac{1}{\text{poly}(n)}$ for an instance on n qubits, and H is satisfiable H' is termed an *almost frustration free* Q2SAT instance. While this instance bears a striking resemblance to the Hamiltonian learned at the end of the H–Q2SAT algorithm, it is interesting in its own right. It provides a setting to understand if the behaviour of frustrated Q2SAT is similar to that of a frustrated version of 2SAT i.e. the MAX2SAT problem.

The MAX2SAT problem which tries to find an assignment that maximizes the number of satisfied clauses is known to be NP-hard due to which simple propagation techniques do not work for it. The decision version of MAX2SAT asks, given a 2SAT formula and a constant k, if there exists an assignment that violates at most k clauses or if for all assignments at leasts $c \cdot k$ clauses are violated for a constant c. The quantum analogue of this would be, given a 2-local Hamiltonian on n qubits and a threshold $a = \frac{1}{\text{poly}(n)}$, decide if there exists an assignment (i.e., a tensor product state) that has energy less than a or all assignments have energy at least $c' \cdot a$ for some constant c'. When the Hamiltonian is, projector-wise, close to a satisfiable Q2SAT instance, in the YES case, the hope is for this low energy state to *approximate* the ground state of the satisfiable instance. Hence, this problem is also termed Approx − Q2SAT. It is known that there are particular regimes (say, c = 1.0005 [Kar01]) where it is NP-hard to decided between the two cases of the MAX2SAT problem. We show that for a similar regime of parameters, the decision version of Approx − Q2SAT is also NP-hard to decide using a reduction from an NP-hard MAX2SAT instance. A complete understanding of this case is still unresolved as there do exist regimes where it is possible to *approximate* the optimal assignment to a MAX2SAT instance in polynomial time. Specifically, MAX2SAT has a robust semidefinite programming and rounding procedure that, in a particular regime, provides an assignment that satisfies almost all clauses [CMM09]. In this spirit, there may exist a similar rounding procedure which will make the Approx − Q2SAT problem easy to decide. This would, in a sense, make Q2SAT mirror 2SAT to a greater extent. If the negative is shown, a divergence in behaviour between 2SAT and Q2SAT will have been demonstrated. As part of ongoing work, this is left as an open question at the moment.

Organisation

The next chapter provides the notation used throughout this thesis pertaining to constraint satisfaction problems and the trial and error model. This is followed by a primer on some of the quantum information concepts essential to understanding the technicalities in this thesis and sets the notation related to quantum satisfiability and quantum constraint satisfaction problems.

The first result presented in Chapter 4 is the linear time algorithm for Quantum 2-SAT which is not connected the trial and error model. The trial and error results are presented in the remaining chapters starting with proving the various transfer theorems and showing their applications in Chapter 5. Following this, the power probabilistic of trials with the worst violated constraint oracle in the context of classical and quantum SAT is presented in Chapter 6. Lastly, the

construction of algorithms using the lex-first oracle for various monotone graph properties and 2-SAT is detailed in Chapter 7. At the beginning of each chapter that presents the various results is a question that captures the theme of the chapter and states the over-arching question the chapter hopes to answer.

Chapter 2

Constraint Satisfaction Problems

The study of constraint satisfaction problems (CSPs) is central to theoretical computer science. CSPs can express a wide and varied array of problems spanning graph theory, game theory and economics, set theory, cryptography, propositional logic, mathematical programming for optimzation problems and the construction of approximation algorithms, just to name a few. As mentioned previously, they are a set of constraints defined on a finite set of objects or variables which can take values from a pre-determined domain. A solution or assignment gives values to the variables in such a way that every constraint in the system is satisfied. This had lead to the development of increasingly efficient algorithms that can find a solution for various CSPs. These algorithms generally take advantage of the structure of the specific CSP they want to tackle and so, in many cases, the algorithms cannot be generalized to other CSPs. For the hard CSPs i.e. those presently without efficient algorithms, the attention shifts to analyzing their complexity. An example of this is their involvement in the study of NP-complete problems. At the core of the theory of NP-completeness is the design of various reductions which show that some candidate CSP A is NP-hard by reducing and instance of A to a known NP-complete problem B. In a sense, this shows some equivalence between different CSPs and highlights a regularity in their structure. This allows for CSPs to be denoted by a general framework and aids the development of theories on the behaviour of various problems by identifying their structural attributes in this framework. An example is Schaefer's Dichotomy Theorem[Sch78] used to classify Boolean formulas or the ongoing works in graph theory to classify hereditary graph properties[1] and the like [BT97].

A combination of both these aspects, namely, creating efficient custom-made algorithms for specific examples and developing more general tools of classification can be seen in the works covered in this thesis. The linear time algorithm for quantum 2SAT in Chapter 4 or the algorithms for the hidden versions of classical and quantum satisfiability with the probabilistic trials in Chapter 6 depict the first of these aspects. The latter can be seen in the transfer theorems of Chapter 5 and the lex-first algorithm for monotone graph properties in Chapter 7.

The framework used in this thesis to formally denote CSPs both in the hidden input case and

[1] A *hereditary property* of a graph is one which holds for all the induced subgraphs of the graph. An *induced subgraph* is a subset of vertices of a graph along with all the edges both of whose endpoints lies in this subset.

otherwise, follows standard notations for the most part. Any notable deviations are with the view of fitting better into the mould of the trial and error setting and will be suitably explained. For a positive integer k, the set of the first k numbers is given by $[k] := \{1, \ldots, k\}$, and an alphabet of size k is indexed with $[\![k]\!] := \{0, 1, \ldots, k-1\}$.A constraint satisfaction problem, S, is specified by its set of *parameters* and its *type* both of which are defined for every positive integer n.

The parameters. The parameters are functions of n for every $n \in \mathbb{N}$ where $\mathbb{N}$ is the set of natural numbers.

(a) the alphabet size $w(n)$ which enumerates the size of the domain from which the variables can take values. For instance, any Boolean CSP has $w(n) = 2$ and the problem of k-colouring a graph will have $w(n) = k$;

(b) the assignment length $\ell(n)$ which will be also be the length of the trials. Typically, the length of the trials would be n but the case of monotone graph properties presents a prominent exception to this rule where the length of the assignment is the possible number of edges in the graph which sets $\ell(n) = \binom{n}{2}$.

(c) the arity $q(n)$ which bounds the number of variables involved in each constraint. For the case of kSAT, the arity will be a constant independent of n with $q(n) = k$;

(d) the number of relations $s(n)$ which enumerates the different relations from which each constraint of the instance is picked. When S is a 2SAT formula, from Example 1.2, $s(n) = |\mathcal{R}_{\mathrm{S}}| = 6$;

(e) and the set of (admissible) assignments $W(n) \subseteq [\![w(n)]\!]^{\ell(n)}$ which defines the set from which a suitable trial is picked each time. This allows for more specificity in denoting trials which may need to satisfy certain additional conditions. This may exclude some of the strings from the set $[\![w(n)]\!]^{\ell(n)}$. In the example of spanning trees as trials, not every $v-1$ sized subgraph can be a suitable trial. A spanning tree poses the additional requirement that every vertex is covered by some edge and that there are no cycles in the subgraph.

The parameters, as functions of n, are assumed to be computable in time polynomial in n. For ease of notation, they will be generally denoted as w, ℓ, W, q and s with the dependence on n being implicit. The set of admissible assignments is a departure from standard notation but does play an interesting role in the context of the graph properties framework that will be detailed below.

The type. For a sequence $J = (j_1, \ldots, j_q)$ of q distinct indices W_J is the projection of W to the coordinates in J: $W_J = \{(v_1, \ldots, v_q) \in [\![w]\!]^q : \exists (w_1, \ldots, w_\ell) \in W \text{ with } w_{j_i} = v_i\}$. W_J is considered to be independent of the choice of J depending only on the number of indices in J i.e. for every J consisting of q distinct indices, $W_J = W_q := \{u \in [\![w]\!]^q \ : \ uv \in W \text{ for some } v \in [\![w]\!]^{\ell-q}\}$. This condition holds trivially for most cases, and for other cases (e.g. CSPs related to graphs), it holds due to the presence of certain symmetry conditions (e.g. graph properties are invariant for isomorphic graphs). Now, any q-ary *relation* is a set R such that $R \subseteq W_q$. For

b in W_q, $b \in R$ is sometimes written as $R(b) = \mathbb{T}$ where $\mathbb{T}$ denotes *true* and $b \notin R$ is written as $R(b) = \mathbb{F}$ and $\mathbb{F}$ denotes *false*. The *type* of S is a set of q-ary relations $\mathcal{R} = \{R_1, \ldots, R_s\}$, where $R_k \subseteq W_q$, for every $k \in [s]$. To put this in context, $\mathcal{R}_{\mathrm{S}}$ in Example 1.2 gives the type of 2SAT with a slight modification. The two unary relations are extended to binary relations by modifying $(a) \mapsto (a \vee \mathbb{F})$ and $(\bar{a}) \mapsto (\bar{a} \vee \mathbb{F})$ and $W_2 = \{00, 01, 10, 11\}$.

While allowing w, q and s to be functions of s is not standard and may look inconvenient, it does help to express certain examples within this framework like systems of linear equations over finite fields (where $q(n)$ and $s(n)$ are non-constant) and hyperplane non-cover (with non constant alphabet size). It can also lead to problems if there is no constraint posed on such functions. The first assumption towards this is to ensure that the functions can be computed in time polynomial in n. Next, the ability to efficiently represent and operate on the alphabet set and relation set forms the basis of any complexity consideration. Hence, these sets are assumed to be such that every letter and relation can be encoded by strings over $\{0, 1\}$ of length polynomial in n and given such a string, their validity can be decided in time polynomial in n. This translates to the existence of Turing machines that can, on being given n, or words $b \in [\![w]\!]^q$, $v \in [\![w]\!]^\ell$ and $k \in [s]$, can decide whether $v \in W$, $b \in W_q$ and compute $R_k(b)$ if $b \in W_q$ in poly(n) time. For a relation R, comp(R) is the time complexity of deciding the membership of a tuple in R, and for a set of relations $\mathcal{R}$, comp$(\mathcal{R}) := \max_{R \in \mathcal{R}} \mathrm{comp}(R)$. The *dimension* of $\mathcal{R}$, denoted as dim$(\mathcal{R})$, is defined as the length of the longest chain of relations (for inclusion) in $\mathcal{R}$. For instance, in the case of 2SAT in Example 1.2, taking a union of any two relations in $\mathcal{R}_{\mathrm{S}}$ results in a relation that is already in $\mathcal{R}_{\mathrm{S}}$ resulting in $\dim(\mathcal{R}_{(\mathrm{S})}) = 2$. Lastly, to avoid complications, it is also required that the membership in the set of admissible assignments W can be computed efficiently. All the CSP examples discussed in this section satisfy these conditions.

The instances. Let $[\ell]^{(q)}$ denote the set of distinct q-tuples from $[\ell]$. An *instance* of S is given by a set of m constraints $\mathcal{C} = \{C_1, \ldots, C_m\}$ over a sequence $\mathbf{x} = (x_1, \ldots, x_\ell)$ of variables, where the *constraint* C_j is $R_{k_j}(x_{j_1}, \ldots, x_{j_q})$ for some $k_j \in [s]$ and $(j_1, \ldots, j_q) \in [\ell]^{(q)}$ as illustrated in Figure 2.1. An assignment $\mathbf{a} = (a_1, \ldots, a_\ell) \in W$ denotes that the value assigned to variable x_i is given by a_i. The assignment $\mathbf{a}$ satisfies a constraint $C_j = R_{k_j}(x_{j_1}, \ldots, x_{j_q})$ if $R_{k_j}(a_{j_1}, \ldots, a_{j_q}) = \mathbb{T}$. An assignment *satisfies* $\mathcal{C}$ if it satisfies all its constraints. Note that the terms clause and constraint will be interchangeably used throughout this thesis.

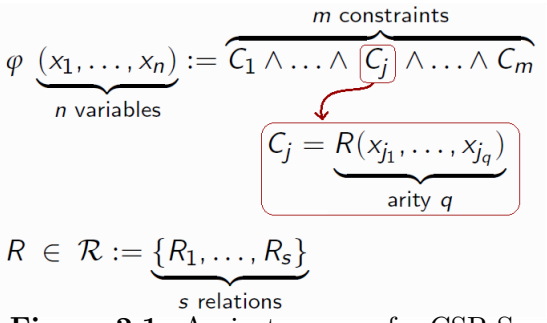


Figure 2.1: An instance φ of a CSP S

The *size* of an instance is $n + m(\log s + q \log \ell) + \ell \log w$ which includes the length of the description of $\mathcal{C}$ and the length of the assignments. A *solution* for $\mathcal{C}$ is any satisfying assignment if one exists or NO otherwise. For all applications considered, the size of the instance will be poly(n). Note that the size of an instance does not count the descriptions of the relations and the admissible

assignments. This is in keeping with the latter information being the meta data of a CSP, which will be known to the algorithm.

Following this rigorous setup, recurring examples of CSPs that are used throughout the extent of this thesis are presented. Boolean Satisfiability formulas and Monotone Graph Properties are of particular importance.

2.0.1 Boolean Satisfiability

The Boolean satisfiability problem, generally denoted as SAT, asks if there exists an assignment to the variables of a given Boolean formula such that the formula evaluates to $\mathbb{T}$ (i.e., `true`). Formally, it is a constraint satisfaction problem on n variables $\mathbf{x} = \{x_1, \ldots, x_n\}$ with m constraints or clauses and each clause is a disjunction of literals (variables x_j or their negations $\overline{x}_j$). Given a formula φ, the goal is to find an assignment $\mathbf{a} = a_1 \ldots a_n \in \{0,1\}^n$ where $\forall i$, the literal x_i takes the value $\mathbb{T}$ (resp. $\mathbb{F}$, i.e. `false`) if a_i is 1 (resp. 0) such that each clause of φ evaluates to $\mathbb{T}$. Whenever a literal $\ell = \mathbb{T}$, its negation $\bar{\ell} = \mathbb{F}$ and vice-versa. Note that, in accordance with the standard mapping of Boolean values to truth values, $0 \mapsto \mathbb{F}$ and $1 \mapsto \mathbb{T}$. If each clause involves at most k literals, then this problem is classified as kSAT. The parameters for kSAT are $w = 2, W = \{0,1\}^n, \ell = n$ and $q = k$. It is known that while $\mathsf{2SAT}$ can be solved in linear time [EIS76, Kro67, APT79], kSAT for $k \geq 3$ is NP-complete [Coo71, Kar72, Lev73].

Given a kSAT instance, a useful notion is that of a *clause tag* which is defined as the unordered set of literals present in the clause. Specifically, the clause tag for $C_j = (x_a \vee \overline{x}_b \vee x_c)$ is denoted by $T(C_j) = \{x_a, \overline{x}_b, x_c\}$. So, all possible clause tags for $\mathsf{2SAT}$ would be $\{\{x_a, x_b\}, \{x_a, \overline{x}_b\}, \{\overline{x}_a, x_b\}, \{\overline{x}_a, \overline{x}_b\}, \{x_a\}, \{\overline{x}_a\} \mid a, b \in [n] \text{ and } a \neq b\}$ where the last two tags will be used for clauses with only one literals. From this definition, it is clear that $\mathsf{2SAT}$ has $O(n^2)$ clause tags and similarly, kSAT would have $\binom{2n}{k} = O(n^k)$ clause tags. A SAT formula ϕ is *equivalent* to a SAT formula ϕ' if for all assignments $\mathbf{a} \in \{0,1\}^n$, $\mathbf{a}$ satisfies ϕ if and only if it satisfies ϕ'. For any formula ϕ, let $\mathsf{sat}(\phi) := \{\mathbf{a} \in \{0,1\}^n \mid \phi(\mathbf{a}) = 1\}$ implying that for equivalent formulas ϕ and ϕ', $\mathsf{sat}(\phi) = \mathsf{sat}(\phi')$. A partial assignment $p \in \{0,1\}^i$ to $x_1 \ldots x_i$ is considered to be *good* if there exists an assignment $p' \in \{0,1\}^{(n-i)}$ to the variables $x_{i+1} \ldots x_n$ such that the assignment pp' satisfies Φ. Correspondingly, call p *bad* if it cannot be extended to a satisfying assignment of Φ.

Below is an example of a $\mathsf{1SAT}$ instance which is just a disjunction of literals and is one of the simplest Boolean formulas.

Example 2.1 (Boolean SAT (continued))**.** *For* $\mathsf{1SAT}$*,* $w = 2$*,* $q = 1$*, and* $\mathcal{R}_{\mathsf{1SAT}} = \{\mathsf{Id}, \mathsf{Neg}\}$*, where* $\mathsf{Id} = \{1\}$ *is the identity relation, and* $\mathsf{Neg} = \{0\}$ *is its complement. Thus a constraint is a literal* x_i *or* $\bar{x}_i$*, and an instance is just a collection of literals. In case of* $\mathsf{2SAT}$*, the parameters are* $w = 2, q = 2$ *and* $|\mathcal{R}_{\mathsf{2SAT}}| = 6$ *from Example 1.2. In the case of* $\mathsf{3SAT}$ *the parameters are* $w = 2$*,* $q = 3$ *and* $|\mathcal{R}_{\mathsf{3SAT}}| = 14$ *as it contains all the OR functions on at most* 3 *variables*

$$\mathcal{R}_{\mathsf{3SAT}} = \{(a \vee b \vee c), (a \vee b \vee \bar{c}), (a \vee \bar{b} \vee \bar{c}), (\bar{a} \vee b \vee c),$$

$$(\bar{a} \vee b \vee \bar{c}), (\bar{a} \vee \bar{b} \vee c), (\bar{a} \vee \bar{b} \vee \bar{c})\} \cup \mathcal{R}_{\mathsf{2SAT}}.$$

A problem related to finding the satisfying assignment to a Boolean formula is to find an assignment which maximises the number of clauses satisfied when the underlying formula is unsatisfiable.

Definition 2.1 (Max-SAT (MAXSAT))**.** *Given a* SAT *formula φ with m clauses on n variables $x_1, \ldots, x_n$, find the maximum number clauses that are made true by assigning boolean values to the variables. If the maximum number of clauses is m, then φ has a satisfying assignment.*

It is known that MAX2SAT (where each clause has at most 2 literals) is NP-complete due to a reduction from 3SAT [GJS76]. The following version of MAX2SAT is a decision problem.

Definition 2.2 (MAX2SAT decision problem)**.** *Given an integer k and a 2-CNF Boolean formula φ with m clauses on n variables $x_1, \ldots, x_n$, distinguish between the two cases*

- *(*YES *instance) $\exists$ Boolean assignment on the n variables such that* UNSAT $\leq k$*;*
- *(*NO *instance) $\forall$ Boolean assignments on the n variables* UNSAT $\geq k+1$*;*

where UNSAT *is the number of* unsatisfied *clauses.*

2.0.2 Graphs and Graph Properties

Graphs are fundamental mathematical objects used in representing pairwise relations (as arcs or edges) between objects (as nodes or vertices) as shown in Figure 2.4. Their presence is ubiquitous in modern life whether it is through the mundane use of the maps that help our navigation on a daily basis or through their ability to model complex molecules making them an efficient visualisation tool in biochemistry and condensed matter physics. The study of these objects through graph theory has a rich history in mathematics and computer science being studied using combinatorial, algebraic, geometric and algorithmic approaches each of which now supports its own field of research.

In this thesis, graphs show up in two different contexts. The first is in the efficient representing a Q2SAT instance as input to the linear time algorithm in Chapter 4 which uses the capacity of the graph to effectively depict the relational structure between objects. This can be done either using directed or undirected graphs as defined below.

Definition 2.3 (Graph)**.** *A* graph *is given by an ordered pair of sets $G = (V, E)$ where V is the set of* vertices *or objects and E is the set of edges in the graph. An* undirected edge *is an unordered pair of vertices $e = \{u, v\}$ such that the endpoints of the edge are the vertices u and v. When e is a* directed edge *from u to v, e is given by the ordered pair (u, v). Now, G is an* undirected *(resp.* directed*) graph if E is a set of undirected (resp. directed) edges.*

To use a graph in algorithms, it is necessary to consider the data structures that will be used to store the graph data. While the vertices can be represented as an array or a list, the edges are commonly given as an adjacency list or adjacency matrix as defined below. Two vertices u, v are

said to be adjacent or neighbours if there is an edge $(u, v) \in E$. The list structure is defined here as it directly relates to the applications considered.

Definition 2.4 (Adjacency List)**.** *The* adjacency list *of a graph $G = (V, E)$ is a set of $|V|$ linked lists where each list describes the set of neighbours of a vertex in the graph.*

Another necessary aspect is the efficient manipulation of the graph structure in any algorithm that uses a graph as the underlying input. The one method relevant to the work covered here is the *breadth first traversal* of a graph. Starting from an arbitrary vertex , say v, this method traverses the graph by first exploring all the neighbours of v before exploring their neighbours in turn and so on till every vertex has been visited. Essentially, this traversal, starting from v covers the neighbours of v, followed by all vertices connected to v by a path of length 2, moving next to vertices at a distance of 3 edges from v and so on justifying the breadth first nature. For completeness, a pseudocode of this procedure is given in Algorithm 2.1 and it is possible to find numerous implementations in the programming language of one's choice. The complexity of the procedure is $O(|V| + |E|)$ as every edge in the graph may be explored in the worst case [CLRS09]. Note that, when the graph is given as an adjacency list, this process executes in time which is linear in the size of the input graph .

Algorithm 2.1: Breadth first Traversal of a graph $G = (V, E)$

```
1  Let all vertices in G be unmarked.
2  Pick a starting vertex u and mark u.
3  Add u to a list L and create a tree T whose root is set to u.
4  while L ≠ ∅ do
5      Pick the vertex v from the top of the list L.
6      Visit v. ▷ Add further processing here depending on what the traversal is used for.
7      for each unmarked neighbour w of v do
8          Mark w and add it to the end of list L
9          Add w and the edge {v, w} to the tree T.
```

The second context in which graphs appear in this work is through the monotone graph framework for the trial and error setting in Chapters 5 and 7. A graph property $\mathcal{P}$ is a family of graphs that is closed under isomorphism. In fact a graph property focuses only on the underlying structure of the graph largely ignoring the issue of how the graph itself is represented. Some terminology associated to various properties of graphs is formally defined below.

Definition 2.5 (Bipartite Graph)**.** *A graph $G = (V, E)$ is a bipartite graph if the vertex set is partitioned as $V = A \cup B$ with $A \cap B = \emptyset$ and $E = \{(u, v) \mid u \in A, v \in B \text{ or } u \in B, v \in A\}$. In other words, there is no edge with both endpoints in A or both endpoints in B.*

Definition 2.6 (Paths and Cycles)**.** *A* path *in a graph $G = (V, E)$ is a sequence of k vertices $(v_1, v_2, \ldots, v_k)$ where there is an edge between successive vertices in the sequence, i.e., for*

$1 \leq i < k$, $(v_i, v_{i+1}) \in E$. *A* cycle *is a closed path where* $v_1 = v_k$.

Definition 2.7 (Trees and Spanning Trees)**.** *A* tree $T = (V, E)$ *is an undirected graph that does not contain any cycles. A spanning tree* T *of a graph* $G = (V, E)$ *is an undirected subgraph that is a tree which is includes every vertex of* G *and uses the minimum possible number of edges.*

Definition 2.8 (Matching)**.** *A* matching *of a graph* $G = (V, E)$ *is a subset of the edges without any common vertices in the endpoints of the edges. A* maximum matching *refers to a matching that involves each vertex of the graph in exactly one of the edges in the matching.*

Definition 2.9 (Hamiltonian Paths and Cycles)**.** *A* Hamiltonian Path *in an* n *vertex graph* $G = (V, E)$ *is an* n*-length path in an undirected or directed graph such that every vertex in the graph visited exactly once. A* Hamiltonian cycle *in* G *is a closed Hamiltonian path which starts and ends at the same vertex and the remaining* $n - 1$ *vertices are visited exactly once.*

Definition 2.10 (k-Clique)**.** *A* clique *is a subset of vertices* V' *in a graph* $G = (V, E)$ *such that every vertex in* V' *is adjacent to all other vertices in* V'*. If* $|V'| = k$*, then this subset of vertices forms a* k*-clique. The problem of determining if a given graph has a* k*-clique is denoted as the* kCLQ *problem.*

Definition 2.11 (k-Colouring)**.** *Given a graph* $G = (V, E)$ *and a set of labels* $[\![k]\!]$*,* G *has a* k-colouring *if it is possible to assign labels* $\alpha_v \in [\![k]\!]$ *for every vertex* $v \in V$ *such that for every* $e = \{u, v\} \in E$*,* $\alpha_u \neq \alpha_v$.

The problem of determining if a given graph has a k-colouring is denoted as kCOL. It is known that kCOL is NP-complete for $k \geq 3$ with a reduction from 3SAT [Kar72] and 2COL is in P as it is equivalent to determining if a graph is bipartite which can be determined with a breadth first traversal.

A monotonic graph property is defined below.

Definition 2.12. *A* monotone graph property *of an* n*-vertex graph is a graph property* $\mathcal{P}$ *that is preserved under the addition of edges (also called monotone increasing property). Correspondingly, monotone decreasing properties are those which are preserved under the deletion of edges.*

Some of the monotone graph properties that will be considered in detail are defined below along with their notations.

Connectivity A graph G is said to be connected if there exists a path from a vertex to every other vertex in the graph. Also, G is connected if and only if G has a spanning tree. It is possible to find a spanning tree in a graph in linear time using breadth first search. Hence, it is also possible to determine if a graph is connected in linear time provided the vertex and edge set are known. For a directed graph, the goal is to find a spanning tree where every edge is directed towards from a root vertex. The problem is denoted as ST and its directed counterpart is DST.

Bipartite perfect matching A perfect matching of a bipartite graph $G = (V, E)$ with $V = A \cup B$ is maximum matching for G where each vertex from A is matched to exactly one vertex in B. Clearly, in this case, $|A| = |B|$. It is possible to find a maximum matching for a bipartite graph using the Ford-Fulkerson algorithm in $O(VE)$ time which is still

polynomial in the size of the input [FF56]. The problem is denoted as BPM.

Cycle covers A disjoint cycle cover of a graph G is a set of cycles which are subgraphs of G, contains all the vertices of G and each cycle in the cover has no vertex in common with any other cycle in the cover. It is possible to find a disjoint cycle cover in polynomial time by reducing it to finding a bipartite perfect matching in a larger graph [Tut54]. In a directed graph, the cycle is such that the edges are all oriented in the same direction giving a cycle that has either a clockwise or anticlockwise orientation. The problem is denoted as UCC and its directed counterpart is DCC.

$s-t$ connectivity A graph $G=(V,E)$ with two marked vertices s,t is $s-t$-connected if there exists a path between vertices s and t. Performing a breadth first search from s till the marked vertex t is discovered will provide a suitable path in linear time [Imm12]. In a directed graph, one aims to find a directed path where each edge is oriented away from s and towards t in a natural manner. The problem is denoted as UPATH and the directed version is DPATH.

To fit the setting of a CSP, a monotone graph property on an n-vertex graph can be expressed as a monotone Boolean function $\mathcal{P}$ on $\binom{n}{2}$ variables where each variable is an edge in the n-vertex graph indexed by $\{(i,j), 1 \leq i < j \leq n\}$. Now, given a graph $G=(V,E)$, the constraint satisfaction problem associated with $\mathcal{P}$, $\mathrm{S}_{\mathcal{P}}$, on this instance consists of m unary clauses of the form $(\overline{e})$ for each edge $e \notin E$ i.e. $m = \binom{n}{2} - |E|$. Note that it differs from a standard 1SAT instance in that the set of possible assignments is not $\{0,1\}^{\binom{n}{2}}$ but the set $W_{\mathcal{P}} = \{\mathbf{T} \mid \mathbf{T}$ is a graph with minimal number of edges satisfying $\mathcal{P}\}$. The goal is to decide, given a graph $G=(V,E)$, whether there exists a $\mathbf{T} \in W_{\mathcal{P}}$ such that $\mathbf{T}$ is a subgraph of G. Formally, the CSP $\mathrm{S}_{\mathcal{P}}$ associated with $\mathcal{P}$ has parameters $w=2$, $q=1$, $\ell = \binom{n}{2}$, $W_{\mathcal{P}}$ consisting of the minimal (for inclusion) graphs[2], satisfying $\mathcal{P}$, and $\mathcal{R} = \{\mathsf{Neg}\}$, where Neg is the negation function. Notice that the hidden instance can be constructed as a CSP irrespective of the property under consideration. In fact, specific part of this model related to the graph property is *fully* left to the specification of the set $W_{\mathcal{P}}$ of admissible assignments.

Following the jargon of Boolean functions, the graphs in $W_{\mathcal{P}}$ function as *witnesses* that certify the property $\mathcal{P}$ on some n-vertex graph just as a Boolean string that certifies the value of a Boolean function in the decision tree model. Consider the case when $\mathcal{P}$ refers to the connectivity of a graph. Then, $\mathbf{T}$ is appropriately a spanning tree and $W_{\mathcal{P}}$ is the set of all possible spanning trees in an n-vertex graph. Clearly, if the input graph G has more than one spanning tree, like the complete graph[3], more than one certificate exists to certify its connectivity. All of the examples considered above have certificates in the form of spanning trees, perfect matchings, cycle covers or shortest paths between two marked vertices. These are attributes of graphs which are well studied in their own right irrespective of this CSP framework [BT97]. However, well-studied or not, the notion of a certificate for any monotone increasing property still holds

[2]The graphs are given as the characteristic vector of the set of edges present in it

[3]The complete graph on n-vertices has n^{n-2} different spanning trees each of which is also present in $W_{\mathcal{P}}$.

as this condition ensures that either G is the only certificate or some subgraph of G satisfies the property and hence becomes the certificate.

This framework naturally extends to directed graphs by having a variable for each directed edge. This only increases the assignment length to $2\binom{n}{2}$ while all other parameters remain unchanged. Similarly, monotone decreasing properties (preserved under deletion of edges) can be treated by changing the clauses to reflect the edges present in G instead of those absent in it and suitably modifying the idea of certificates.

Other examples of constraint satisfaction problems considered in this thesis are defined, for the sake of continuity, as and when required.

2.1 The Trial and Error Model

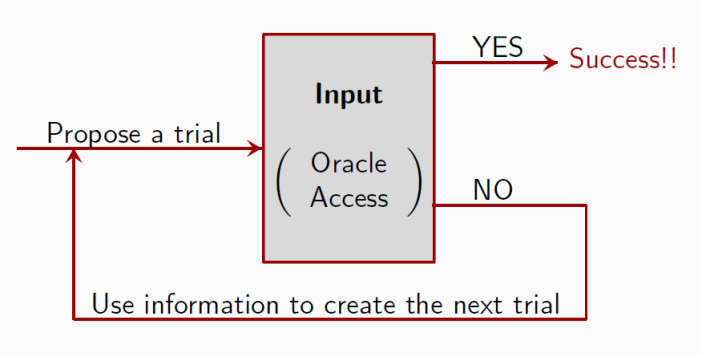
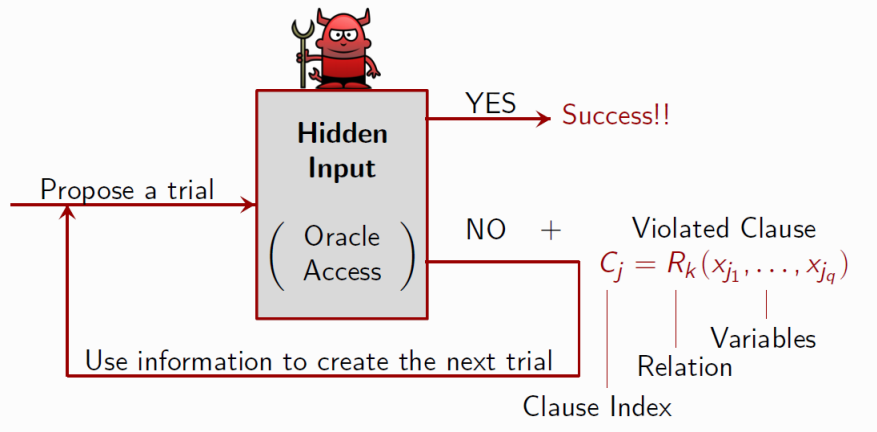


Figure 2.2: The trial and error method

Figure 2.2 captures the essence of the trial and error approach to problem solving. Clearly, this method is useful only when input information is missing as otherwise, any bespoke method to solve the problem at hand can be used. However, there is some way to interact with the input and check if an assignment works or not. This is considered as equivalent to having some oracle access to the problem. With such limited access, the only possibility is to propose trials and receive some information about an unsuccessful trial. Since blindly trying trials is not going to be helpful, one would like to harness all the information received from the violations to adaptively propose the next trial.

From the computational perspective, this approach was applied by Bei, Chen and Zhang [BCZ13] to the scenario of solving constraint satisfaction problems in the "unknown input" setting. Specifically, the CSPs with *hidden inputs* (H-CSPs) are accessed by an oracle to which one proposes candidate solutions for the CSP i.e. trials. If the trial is not satisfying, then the oracle reveals some information about the cause of failure i.e. errors. In this case, it would be the index of some violated constraint. For example, if the CSP instance is a SAT formula, on proposing a satisfying assignment to the formula, the oracle would return YES. After an unsuccessful trial, the oracle reveals that "clause 5 is violated" and nothing more. If there are

several violated clauses, to analyze worst case complexity, it is assumed that the oracle picks one clause *adversarially*.



Note: We assume that the oracle always reveals the violated clause's index

Information Revealed	Oracle Type	Problem Notation
Only j	$\{\mathcal{C}\}$-revealing	H–S$_{\{\mathcal{C}\}}$
j and R_k	$\{\mathcal{C},\mathcal{R}\}$-revealing	H–S$_{\{\mathcal{C},\mathcal{R}\}}$
j and $(x_{j_1},\ldots,x_{j_q})$	$\{\mathcal{C},\mathcal{V}\}$-revealing	H–S$_{\{\mathcal{C},\mathcal{V}\}}$
j, R_k and $(x_{j_1},\ldots,x_{j_q})$	$\{\mathcal{T}\}$-revealing	H–S$_{\{\mathcal{T}\}}$

Figure 2.3: The trial and error method for H-CSPs with oracle and problem types

However, as seen before, a CSP's constraint (or clause) is made up of three key parts: a constraint index, the relation the constraint refers to and the set of variables it acts on. For our purposes, the violation information can be some combination of these parts as shown in Figure 2.3 leading to a larger variety of oracles dependent on the violation information they reveal. As mentioned in Chapter 1, to effectively use the information revealed by the oracle to create the next trial, it is necessary that, at the very least, the constraint index be revealed by the oracles.

Definition 2.13 ($\mathcal{U}$-revealing oracles)**.** *Let $\mathcal{C}$ refer to the constraint set, $\mathcal{V}$ the variables set and $\mathcal{R}$ the set of relations of a hidden instance of some* CSP S*. Then, a $\mathcal{U}$-revealing oracle, denoted $\mathcal{O}_{\mathcal{U}}$, for $\mathcal{U} \subseteq \{\mathcal{C},\mathcal{R},\mathcal{V}\}$ on encountering a violated clause $C_j = R_{k_j}(x_{j_1},\ldots x_{j_q})$ of parity q reveals the following information:*

- *the constraint index j if $\mathcal{C} \in \mathcal{U}$*
- *the relation R_{k_j} if $\mathcal{R} \in \mathcal{U}$*
- *the tuple of variables $(x_{j_1},\ldots x_{j_q})$ if $\mathcal{V} \in \mathcal{U}$*

When $\mathcal{U} = \{\mathcal{C},\mathcal{R},\mathcal{V}\}$, the violated clause is totally *revealed i.e. this oracle is $\mathcal{T}$-revealing.*

Analogously, for every CSP S, and for every $\mathcal{U} \subseteq \{\mathcal{C},\mathcal{R},\mathcal{V}\}$, we define the *hidden constraint satisfaction problem* (H–CSP) with the *$\mathcal{U}$-revealing oracle*, H–S$_{\mathcal{U}}$.

Definition 2.14 (Hidden CSPs with $\mathcal{U}$-revealing oracles)**.** *Consider a* CSP S *defined on n variables from the set $\mathcal{V}$, with a relation set $\mathcal{R}$, and an instance of* S *with constraints $\mathcal{C} = \{C_1,\ldots,C_m\}$. If the input instance of* S *is unknown and can be accessible only by a revealing*

oracle $\mathcal{O}_{\mathcal{U}}$, the corresponding problem is denoted H–S$_{\mathcal{U}}$*. In this setting, the parameters and type of* S *namely, $\mathcal{R}, n, m$ and $\mathcal{V}$ are considered to be part of the problem input (i.e. known) and it is only the constraints $C_1, \ldots, C_m$ that are hidden by the $\mathcal{U}$-revealing oracle.*

Now, consider the behaviour of the revealing oracles in the context of a 1SAT formula.

Example 2.2 (Hidden-SAT)**.** *Let us suppose that we present an assignment $\mathbf{a} \in \{0,1\}^{\ell}$ for an instance of the hidden version* H–1SAT$_{\mathcal{U}}$ *of* 1SAT *to the $\mathcal{U}$-revealing oracle. If $\mathcal{U} = \{\mathcal{C}, \mathcal{V}\}$ and the oracle reveals j and i respectively for the violated constraint and the variable in it, then we learn $C_j = (x_i)$ if $a_i = 0$, and $C_j = (\bar{x}_i)$ otherwise. If $\mathcal{U} = \{\mathcal{C}, \mathcal{R}\}$ and say the oracle reveals j and* Id *then we learn that C_j is a positive literal. If $\mathcal{U} = \{\mathcal{C}\}$ and the oracle reveals j then we only learn that C_j is either a positive literal corresponding to one of the indices where $\mathbf{a}$ is* 0*, or a negative literal corresponding to an index where $\mathbf{a}$ is* 1*.*

An algorithm *solves* the problem H–S$_{\mathcal{U}}$ if for all n, m, and for every instance $\mathcal{C}$ for S, specified by any $\mathcal{U}$-revealing oracle for $\mathcal{C}$, it outputs a satisfying assignment if there exists any, and NO otherwise. The complexity of an algorithm for H–S$_{\mathcal{U}}$ is the number of steps in the worst case over all inputs and all $\mathcal{U}$-revealing oracles, where a query to the oracle is counted as one step. The trial and error model is characterized by the immediacy of finding a solution as opposed to developing a full fledged notion of why a solution works compounded by incomplete input information for the problem one wants to solve. So, any algorithm in this setting aims to effectively use the errors to minimize the number of trials needed to find a solution while also keeping in mind the cost of proposing a trial.

Sometimes, setting aside the cost of proposing each new trial, one would prefer to analyze just the number of trials needed to find a solution (i.e., query complexity). This can be useful in different ways. While in the complexity setting we attribute the cost of a trial to be constant, in many scenarios, like drug testing or adopting new management strategies and observing their effects, the trials themselves can be very costly. For such scenarios, the main goal would be to optimize the query complexity irrespective of the cost of making generating a new trial. Beyond these, query complexity plays a prominent role in cases where the original problem is already hard, say NP-complete like SAT. In this case, it is possible to allow for the presence of a computational oracle C that can solve this hard problem in the un-hidden setting. The complexity of the computational oracle is the bottleneck for the cost of proposing the trial and the algorithm is then constructed to focus on minimising the query complexity with respect to the revealing oracle. When the question at hand is only the *extra difficulty* introduced to the problem due to a lack of input information, this is a reasonable assumption to use. For instance, there is an algorithm [BCZ13] with query complexity polynomial in the number of variable in a SAT formula to solve H–SAT. However, the algorithm requires access to a computational SAT oracle implying that each trial reveals enough information to solve the hidden instance but decoding this information is costly and requires access to a SAT oracle. To provide a clear picture on how an algorithm in the trial and error setting proceeds, the algorithm for H–SAT as described in [BCZ13] is given below.

Algorithm 2.2: Trial and error algorithm for H–SAT$_{\{\mathcal{C}\}}$

1 **Input:** A H–SAT formula ϕ on n variables with m clauses.
2 **Oracles:** A computational oracle SAT, a verification oracle $\mathcal{O}_{\{\mathcal{C}\}}$.
3 Initialize $\hat{C}_1 = \hat{C}_2 = \ldots = \hat{C}_m = \{x_1, \overline{x}_1, \ldots x_n, \overline{x}_n\}$.
4 **repeat**
5 Let $\Phi = \bigwedge_{j=1}^{m} (\vee_{\ell \in \hat{C}_j} \ell)$, propose ϕ to the SAT oracle and get back **a**.
6 **if a** = UNSAT **then output** UNSAT.
7 **else**
8 Propose **a** to $\mathcal{O}_{\{\mathcal{C}\}}$ and get back j. ▷ As **a** is a satisfying assignment for Φ.
9 **if** j = YES **then output a**.
10 **else**
11 j is a violated constraint index
12 Update $\hat{C}_j \leftarrow \hat{C}_j \cap \{x_i | a_i = 0\} \cup \{\overline{x}_i | a_i = 1\}$
13 ▷ the literals in $\hat{C}_j$ are those which are falsified by **a**.
14 **until** $\bigcup_{j=1}^{m} \hat{C}_j = \emptyset$
15 **output** UNSAT.

The intuition behind Algorithm 2.2 is a standard candidate elimination approach used in learning theories [Mit82]. The formula Φ represents the current knowledge consistent with the trials proposed and violations received so far. The current trial **a** is generated by using the SAT oracle to get a satisfying assignment for Φ if one exists (Line 5). The candidates, for each clause C_j, refers to the $2n$ possible literals initially present in $\hat{C}_j$ and the elimination results in progressively removing all literals that cannot be present in C_j. The candidates to be eliminated are determined by the current trial **a** which generates a violation j from $\mathcal{O}_{\{\mathcal{C}\}}$. Specifically, the literals in C_j are set to $\mathbb{F}$ by the current assignment and $\hat{C}_j$ is updated to reflect this fact (Line 12). A key fact used to show the correctness of the algorithm is that if ϕ is satisfiable, then so is Φ and conversely, if Φ is unsatisfiable then so is ϕ. The complexity of the algorithm is shown to be $O(nm)$ by showing that $2mn$ repetitions are sufficient to either find a satisfying assignment or make some $\hat{C}_j$ empty so that Φ is unsatisfiable. This results in $O(mn)$ calls to the SAT oracle and $O(mn)$ trials proposed to the $\{\mathcal{C}\}$-revealing oracle. At this point, it is imperative to point out the difference between learning the underlying instance and finding a solution for it. The above algorithm only determines if the hidden instance is unsatisfiable or finds a satisfying assignment for it. The goal is not to learn what the hidden formula ϕ is. In fact, [BCZ13] show the existence of instances[4] that cannot be learned even when allowed an exponential number of trials. The lex-first algorithms presented in Chapter 7 bear some resemblance to this algorithm in that they also use (a) candidate elimination and (b) clause lists $\hat{C}_j$s and Φ to represent the

[4]There are unsatisfiable formulas where the arity of each clause is $O(n)$

current knowledge about the clauses of the hidden instance.

The other oracle types considered in this work besides the $\mathcal{U}$-revealing oracles include the *max-violation oracle* $\mathcal{O}_{\text{MAX}}$ and the *lex-first oracle* $\mathcal{O}_{\text{FIRST}}$.

Definition 2.15 (Max-violation oracle $\mathcal{O}_{\text{MAX}}$)**.** *Let* H–S *be a* H–CSP *whose input instance is accessed by the max-violation oracle* $\mathcal{O}_{\text{MAX}}$ *and let* $\mathbf{a} \in W$ *be an admissible assignment for* S*. Consider a function* $\mathsf{viol} : (C, \mathbf{a}) \to [0, 1]$ *that calculates the violation of a constraint* C *on an assignment. When* $\mathbf{a}$ *is proposed to* $\mathcal{O}_{\text{MAX}}$*, it returns* YES *if the instance is satisfied by* $\mathbf{a}$ *or returns an index* $j = \arg\max_j \mathsf{viol}(C_j, \mathbf{a})$ *i.e.* C_j *has the* maximum violation *amongst all the clauses with respect to the assignment* $\mathbf{a}$*. If more than one clause has the same maximum violation, the tie is broken adversarially. The corresponding problem is denoted as* H–S_{MAX}*.*

Note that the violation function is entirely dependent on the problem S and its corresponding assignments. The max-violation oracle was first considered in [BCZ15] as the authors tried to use the trial and error setting to determine the feasibility of a linear program when the constraints are unknown. In this case, a natural violation function emerged in the form of the Euclidean distance a proposed solution has to the half-planes represented by the constraints. Then, the maximum violation is given by the maximum distance. Clearly, the continuous nature of linear programming helps in this case as opposed to the binary valued SAT. Here, the violation function would map only to 0 or 1 as a clause is either fully violated (violation 1) or fully satisfied (violation 0) by an assignment. As every violated constraint is then treated equally to break ties, the H–SAT problem with $\mathcal{O}_{\text{MAX}}$ is identical to the problem with $\mathcal{O}_{\mathcal{C}}$ providing no advantage. In order to change this, in Chapter 6, the set of admissible assignments is changed to accept a probability distribution over assignments so that the violation function could represent the probability with which a clause can be violated.

Example 2.3 (H–1SAT and the $\mathcal{O}_{\text{MAX}}$ oracle)**.** *Consider the* H–1SAT *formula* $\varphi = x_1 \wedge x_2 \wedge x_3 \wedge \overline{x}_4 \wedge x_3$*. Let the first trial be the probabilistic assignment* $\mathbf{a} = (a_1, a_2, a_3, a_4) = (0.3, 0.7, 0.9, 0.3)$ *where* $\forall i,\ \Pr[x_i = 1] = a_i$*. The violations for a clause* C_j *is given by* $1 - \Pr[C_j(\mathbf{a}) = 1]$ *resulting in the clause wise violations* $(0.7, 0.3, 0.1, 0.3, 0.1)$*. Clearly,* C_1 *has the maximum violation in this case and* $\mathcal{O}_{\text{MAX}}$ *would return* $j = 1$*. Suppose the assignment* $\mathbf{a}' = (0.8, 0.7, 0.9, 0.3)$ *is proposed, the clause wise violations would be* $(0.2, 0.3, 0.1, 0.3, 0.1)$*. As the maximum violation is* 0.3*, the oracle will adversarially pick either* C_2 *or* C_4 *to send as a violation.*

Finally, the last oracle type, the lex-first oracle is defined below.

Definition 2.16 (Lex-first oracle $\mathcal{O}_{\text{FIRST}}$)**.** *Let* H–S *be a* H–CSP *whose input instance is accessed by the lex-first oracle* $\mathcal{O}_{\text{FIRST}}$ *and let* $\mathbf{a} \in W$ *be an admissible assignment for* S*. If* $\mathbf{a}$ *satisfies the hidden instance, then the oracle returns* YES*. Otherwise,* $\mathcal{O}_{\text{FIRST}}$ *returns the index of the lexicographically first violated clause, i.e.,* $\min\{j | C_j(\mathbf{a})$ *is a violation*$\}$*. The corresponding problem is denoted as* H–S_{FIRST}*.*

Example 2.4 (H–1SAT and the $\mathcal{O}_{\text{FIRST}}$ oracle)**.** *Consider again the* H–1SAT *formula* $\varphi = x_1 \wedge x_2 \wedge x_3 \wedge \overline{x}_4 \wedge x_3$*. Let the trial be* $\mathbf{a} = 1011$*. Clearly,* C_2 *and* C_4 *are violated by this assignment but the lex-first oracle, in keeping with its definition, returns* $j = 2$ *as the violation.*

For a CSP S, any algorithm for the H–S_{MAX} and H–S_{FIRST} is in P if it can, in polynomial time find a satisfying assignment for the hidden instance or output UNSAT otherwise. A natural way to do this is to partially determine the clauses in the hidden instance consistent with the series of violations observed.

2.1.1 Monotone Graph Properties Framework

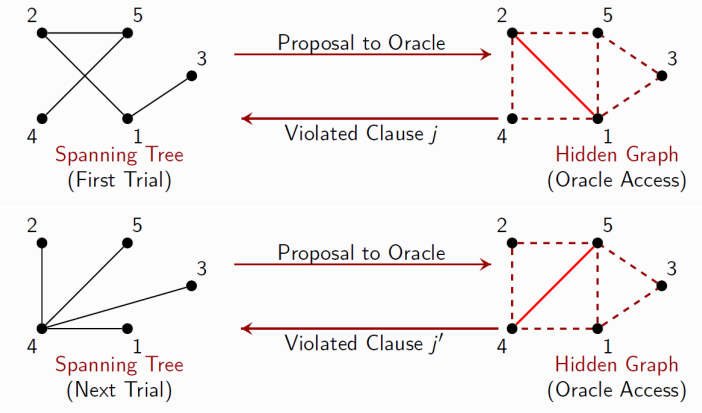


Figure 2.4: Checking if a hidden graph is connected using an $\mathcal{O}_{\mathcal{U}}$ oracle where the violated clause picked by the oracle in the trials is C_j and $C_{j'}$ respectively.

Figure 2.4 depicts how an algorithm in the trial and error model interacts with the oracles to determine if the hidden graph is connected. The spanning trees in a 5-vertex graph form the set of admissible assignments $W_{\mathcal{P}}$ for this instance. Each trial $\mathbf{T} \in W_{\mathcal{P}}$ is represented as a set of edges that make up the certificate. If $\mathbf{T}$ is a valid certificate for the hidden graph, the oracle returns YES. Otherwise, the error returned by the oracle when $\mathcal{T}$ is not a certificate is the information of a clause $(\bar{e})$ such that $e \in T$ but $e \notin E$. In the figure, we have $C_j = (\bar{e}_{1,2})$ and $C_{j'} = (\bar{e}_{4,5})$, where $e_{u,v}$ is the undirected edge $\{u, v\}$, but this information is hidden by the oracle. For an $\mathcal{O}_{\mathcal{U}}$ oracle,

- If $\mathcal{U} = \{\mathcal{C}, \mathcal{V}\}$, then the information returned for the two trials respectively would be $(j, e_{1,2})$ and $(j', e_{4,5})$. By revealing a missing edge, it is possible to effectively learn all the edges absent from the graph.
- If $\mathcal{U} = \{\mathcal{C}\}$, then the information returned is only j and j' respectively which does not provide much insight into which clause refers to which edge of the graph.
- If $\mathcal{U} =$ FIRST, then information returned is still only j and j' respectively except the additional conclusion is that the edges in the first spanning tree do not intersect the clauses $C_1, \ldots, C_{j-1}$ and the second spanning tree's edges does not intersect the clauses $C_1, \ldots, C_{j'-1}$.

This covers all the requisite notations involving classical CSPs and the trial and error model. The next chapter introduces the basics of quantum computing.

Chapter 3

Quantum Satisfiability

Chapter 1 introduced classical constraint satisfaction problems and a central problem in classical complexity theory – the Boolean SAT problem. Continuing on this theme, one would hope that a quantum generalization of SAT would play a central role in Quantum Complexity Theory. This is definitively the case as quantum SAT (QSAT) is known to be a restriction of the Local Hamiltonian Problem which is itself a principal object in condensed matter physics. Informally, a k-local Hamiltonian on n qubits can be viewed as a set of constraints each of which affects at most k qubits thereby being local constraints. Then, given an instance of a k-local Hamiltonian and two thresholds a, b such that $b - a \geq \frac{1}{\text{poly}(n)}$, the goal is to determine if the expected number of violated constraints is below a or at least b. In fact, this is more in keeping with the spirit of the MAXkSAT problem.

Physically, the local constraints model the local interactions between quantum spins and the expected number of violated constraints is intimately linked to the low temperature physics of the system, such as quantum phase transitions and collective quantum phenomena [Sac07, VLRK03]. This expected number of violated constraints is commonly referred to as the ground energy and the states which achieve this expectation form the set of ground states of the given k-local Hamiltonian. The intensive research in quantum complexity theory and quantum information theory over the last two decades or so has also revealed connections between the complexity of finding the ground state and its entanglement structure. While the notion of entanglement is formally defined later, it can be thought of as correlations exhibited by two or more qubits that are governed by quantum mechanics and cannot be reproduced by classical probability distributions. Entanglement is viewed as a key quantum resource in quantum communication and cryptographic protocols [BS16, LS09], in defining quantum strategies for nonlocal games [CHTW04], and in harnessing speedups using quantum algorithms [Mon16]. In recent years, the efforts to better understand entanglement structure has revealed intricate behaviours such as area laws [ECP10] and topological order [Kit03]. The former shows that a certain measure of entanglement grows proportional to the area of a region of interest defying the expectation that it would scale like the volume of the region. The latter has implications in the construction of toric codes – a fault tolerant quantum error correction code.

3.1 Basics of Quantum Computation

Before formally defining the QSAT problem, it is useful to delve into some of the objects and basic concepts commonly used in the quantum realm and relevant to the results discussed here[1] The fundamental unit of computation in a classical computer is a bit which takes Boolean values. The analogous fundamental unit in the quantum world is a two dimensional quantum system called a qubit. The values of interest are the state of this qubit which can take a value comprised by two basic states denoted as the zero and one states. Physically, this could correspond to the up-spin or down-spin of a quantum particle or some other physical parameter. What makes the qubit state different from the classical value possessed by any bit is that the two states only form a basis of the possible states a qubit could be it. The whole range of possible states for a qubit includes any arbitrary linear combination (with complex coefficients) of the two basis states. Formally, using Dirac's bra-ket notation, a single qubit state is described as follows

Definition 3.1 (Single Qubit State)**.** *A qubit has a state* $|\alpha\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$ *depicted as a unit vector in the complex Hilbert space* $\mathbb{C}^2$ *where* $\{|0\rangle, |1\rangle\}$ *is the standard computational basis with*

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ and } |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \text{ and } |\alpha\rangle = \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix}.$$

For $i = 0, 1$ the coefficients α_i are termed the amplitudes of the state with respect to this chosen basis. Physically, this is interpreted as: the probability for the qubit to be in state $|i\rangle$ is $|\alpha_i|^2$, where $|\cdot|$ gives the absolute value. Now, since the amplitudes are linked to a probability distribution, it follows that $|\alpha_0|^2 + |\alpha_1|^2 = 1$ thereby making the requirement for a unit vector clear. As the qubit lives in a Hilbert space, there is an inner product between states that has to be defined. To enable that, the dual for a state $|\alpha\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$ is given by its adjoint $\langle\alpha| := (|\alpha\rangle)^\dagger = \alpha_0^*\langle 0| + \alpha_1^*\langle 1| = [\alpha_0^* \ \alpha_1^*]$ where the $*$ denotes the complex conjugate. The inner product between any two states $|\alpha\rangle$ and $|\beta\rangle$ is just the dot product of $|\alpha\rangle$ with the dual of $|\beta\rangle$.

Definition 3.2 (Inner Product)**.** *The inner product of two states* $|\alpha\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$ *and* $|\beta\rangle = \beta_0|0\rangle + \beta_1|1\rangle$ *is given by* $\langle\beta|\alpha\rangle = \langle\alpha|\beta\rangle = \langle\beta| \cdot |\alpha\rangle = \beta_0^*\alpha_0 + \beta_1^*\alpha_1$.

The inner product induces the norm $\||\alpha\rangle\|^2 = \langle\alpha|\alpha\rangle$ which is clearly 1 for each state. Every single-qubit state $|\alpha\rangle$ has a corresponding orthogonal state $|\alpha^\perp\rangle$ with which it has inner product 0 and it has a succinct expression $|\alpha^\perp\rangle := \alpha_1|0\rangle - \alpha_0|1\rangle$. Adding a global phase $e^{i\theta}$ ($\theta \in \mathbb{R}$) to $|\alpha\rangle$ leaves it computationally invariant. Hence, all notations will be considered equivalent up to a global phase. The following example illustrates the fact that the amplitudes of a state depend on the chosen basis.

Example 3.1 (Basis dependency of amplitudes)**.** *Consider the state* $|\alpha\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$ *given in the standard computational basis and consider a different basis* $\{|+\rangle, |-\rangle\}$ *where* $|+\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$

[1] For more information, please refer to any of the introductory textbooks on Quantum Information and Quantum Computing [KSV02, NC00].

and $|-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$*. Then, the state can also be written as* $|\alpha\rangle = \alpha_0'|+\rangle + \alpha_1'|-\rangle$ *since* $\{|+\rangle, |-\rangle\}$ *is also a 2-dimensional orthonormal basis. Then,*

$$|\alpha\rangle = \alpha_0'|+\rangle + \alpha_1'|-\rangle = \alpha_0'\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right) + \alpha_1'\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) = \left(\frac{\alpha_0' + \alpha_1'}{\sqrt{2}}\right)|0\rangle + \left(\frac{\alpha_0' - \alpha_1'}{\sqrt{2}}\right)|1\rangle.$$

$$\Rightarrow \quad \frac{\alpha_0' + \alpha_1'}{\sqrt{2}} = \alpha_0 \quad \text{and} \quad \frac{\alpha_0' - \alpha_1'}{\sqrt{2}} = \alpha_1$$

$$\Rightarrow \quad \alpha_0' = \frac{\alpha_0 + \alpha_1}{\sqrt{2}} \quad \text{and} \quad \alpha_1' = \frac{\alpha_0 - \alpha_1}{\sqrt{2}}.$$

Definition 3.3 (Multi-qubit states)**.** *An n-qubit state* $|\phi\rangle$ *is a unit vector in the Hilbert space* $\mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2 \otimes \cdots \otimes \mathcal{H}_n$*, where* $\mathcal{H}_i = \mathbb{C}^2$ *is the space of the* i^{th} *qubit.*

Denoting that the single qubit state $|\alpha\rangle$ lives in $\mathcal{H}_i$ is explicitly done as $|\alpha\rangle_i$ and correspondingly $|\psi\rangle_{ij}$ signifies that the 2-qubit state $|\psi\rangle$ lives in $\mathcal{H}_i \otimes \mathcal{H}_j$. The standard computational basis becomes $\{|0\rangle, |1\rangle\}^{\otimes n} = \{|i\rangle \mid i \in \{0,1\}^n\}$. Then it is possible to represent $|\phi\rangle$ as $|\phi\rangle = \sum_{i\in\{0,1\}^n} \alpha_i|i\rangle$ where $\forall i$, $\alpha_i \in \mathbb{C}$ and $\sum_{i\in\{0,1\}^n} |\alpha_i|^2 = 1$ in keeping with the probabilities. Viewing $|\phi\rangle$ as a complex, unit vector of dimension 2^n, it is clear that the inner product and norm extend to the n-qubit setting too. Given another n-qubit state $|\psi\rangle = \sum_{i\in\{0,1\}^n} \beta_i|i\rangle$, the inner product is given by $\langle\phi|\psi\rangle = \sum_{i\in\{0,1\}^n} \alpha_i^*\beta_i$.

The qubit states discussed so far are called *pure states*. The simplest kind of multi-qubit pure states to think of would be ones which can be represented as a tensor product of their individual components. For example, a 2-qubit state $|\psi\rangle_{ij} = |\psi_1\rangle^i \otimes |\psi_2\rangle^j$ is the tensor product of the single qubit states $|\psi_1\rangle$ on qubit i and $|\psi_2\rangle_j$ on qubit j. These kinds of states are called product states.
Definition 3.4 (Product State)**.** *An n-qubit state* $|\phi\rangle$ *is called a product state if it can be expressed as a tensor product of its individual components as*

$$|\phi\rangle = |\psi_1\rangle_1 \otimes |\psi_2\rangle_2 \otimes \ldots \otimes |\psi_n\rangle_n \quad \text{where } \forall i \in [n],\ |\psi_i\rangle \in \mathbb{C}^2$$

By the principle of superposition, it is true that any linear combination of pure states can be normalized to represent a pure state. Normalization just refers to the operation of writing out the resulting state in some basis and having all the probabilities summing to 1. However, not all these linear combinations remain product in nature. Such states i.e. non-product pure states are what are known as *entangled states*. Physically, this implies that the state of each particle in the system cannot be described independently of the other particles but can only be expressed as a whole. Moreover, this holds even if the particles are spatially separated.
Example 3.2 (Entangled state)**.** *The 2-qubit state* $|\Psi^-\rangle = \frac{1}{\sqrt{2}}|0\rangle \otimes |1\rangle - \frac{1}{\sqrt{2}}|1\rangle \otimes |0\rangle$ *is an entangled state as it cannot be written as* $|\psi_1\rangle \otimes |\psi_2\rangle$ *for any choice of single qubit states* $|\psi_1\rangle, |\psi_2\rangle$*.*

Given an entangled state that is known to be bipartite i.e. exists in the space $\mathcal{H}_A \otimes \mathcal{H}_B$, the *Schmidt decomposition* gives a useful way to represent this state as a sum of terms each of which is orthonormal to other terms. In fact, it helps to characterize the entanglement structure for

such bipartite systems.

Theorem 3.1 (Schmidt Decomposition)**.** *Consider the state of a bipartite system* $|\Psi\rangle_{AB} \in \mathcal{H}_A \otimes \mathcal{H}_B$*, where the* $\dim(\mathcal{H}_A) = n$ *and* $\dim(\mathcal{H}_B) = m$*. There exist orthonormal basis* $\{|\alpha_1\rangle, \ldots, |\alpha_n\rangle\}$ *of* $\mathcal{H}_A$ *and* $\{|\beta_1\rangle \ldots, |\beta_m\rangle\}$ *of* $\mathcal{H}_B$ *such that* $|\Psi\rangle$ *can be written as*

$$|\Psi\rangle = \sum_{k=1}^{d=\min\{m,n\}} c_k |\alpha_k\rangle \otimes |\beta_k\rangle.$$

Moreover, the Schmidt coefficients, c_k*, can be chosen to be real, non-negative with* $c_1 \geq \ldots \geq c_d \geq 0$ *and uniquely determine the state.*

The number of Schmidt coefficients, d, is called the Schmidt rank of the state. From this, one can conclude that a pure bipartite state is entangled if and only if the Schmidt rank of the state is > 1. To find the Schmidt decomposition of a state, rewrite the state vector as a matrix and find the singular value decomposition of this matrix. The singular values give the Schmidt coefficients and the singular vectors provide the Schmidt basis vectors. Going beyond bipartite systems, multi-partite systems could exhibit varying kinds of entanglement. The first is to possess a bipartite subsystem that is entangled but the rest of the system is in product form as in $|\Psi\rangle_{AB}|1\rangle_C|0\rangle_D$. At the other end of the spectrum is a *genuinely entangled state.*

Definition 3.5 (Genuinely entangled states)**.** *A state* $|\psi\rangle$ *over* n *qubits is genuinely entangled if for any bi-partition of the qubits into two subsets* A, B*, it cannot be written as a product state* $|\psi\rangle = |\psi_A\rangle \otimes |\psi_B\rangle$*, where* $|\psi_A\rangle$*,* $|\psi_B\rangle$ *are defined on the qubits of* A *and* B *respectively.*

So, a genuinely entangled state is a pure state on n-qubits that is not product with respect to any bi-partition of the system. In fact, it is not possible to factor out any qubit's state in the system. Genuine multi-partite entanglement is understood to a lesser degree as most of the tools from understanding bipartite entanglement cannot be extended to multi-partite settings. For instance, generalizing the Schmidt decomposition to $N > 2$ systems poses difficulties as it becomes a formidable task to write any state as the sum of terms each of which is the product of states from N-orthogonal bases.

While one way to consider a composite system is to consider a larger state space, another way is to consider possessing a statistical ensemble of quantum states i.e. a large number of copies of the same system which, when considered all at once, exhibits the probability distribution of the states that the system could be in. In other words, it is a mixture of all the possible states the system could be in. The state of this mixture is called a *mixed state* and is not represented in vector form but by a *density matrix.*

Definition 3.6 (Density Matrices)**.** *A density matrix* ρ *on an* n*-qubit system is a positive semidefinite matrix of the form*

$$\rho = \sum_i p_i |\phi_i\rangle\langle\phi_i| \quad \textit{where} \quad \sum_i p_i = 1 \quad \textit{and} \quad \forall i,\ \phi_i \in \mathbb{C}^{\otimes n}$$

with p_i *being interpreted as the fraction of the ensemble that is in the state* $|\psi_i\rangle$*.*

Note that the p_is in the density matrix are not amplitudes but probabilities which are generated classically unlike quantum superpositions. $|\phi\rangle\langle\phi|$ is the projection onto the subspace spanned by $|\phi\rangle$. A physical way to prepare a mixed state would go as follows. Think of a source that is capable of producing particles which could be in state $|\phi_1\rangle$, $|\phi_2\rangle$ and so on but the exact state of the particle is not known. However, it is known that the source produces a particle in $|\phi_1\rangle$ with p_1 probability, in $|\phi_2\rangle$ with p_2 probability and so on. Then, the density matrix defined above completely captures this scenario.

A density matrix ρ represents a pure state if and only if $\mathrm{Tr}(\rho^2) = 1$ and if $\mathrm{Tr}(\rho^2) < 1$, it represents a mixed state. Here Tr is the trace operator acting on a matrix. In the former case, the density matrix also uniquely determines the pure state but this doesn't hold for a mixed state as there could be infinitely many ways to decompose a density matrix as a probabilistic mixture of pure states. For instance, the density matrix given by $\frac{1}{2}\mathbb{I}$ can be decomposed as

$$\frac{1}{2}\mathbb{I} \;=\; \frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1| \;=\; \frac{1}{2}|\alpha\rangle\langle\alpha| + \frac{1}{2}|\alpha^{\perp}\rangle\langle\alpha^{\perp}| \quad \forall\, |\alpha\rangle \in \mathbb{C}^2$$

which can be verified with a little calculation. A more non-trivial example is the following

Example 3.3 (Ensembles of pure states)**.** *Consider an ensemble that is made up of the state* $|0\rangle$ *with probability* $1/2$ *and the state* $|+\rangle = (|0\rangle + |1\rangle)/2$ *with probability* $1/2$*. The density matrix for this in the standard basis is given by*

$$\begin{aligned}\rho &= \frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|+\rangle\langle +| \\ &= \frac{1}{2}\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + \frac{1}{2}\begin{bmatrix} 1/2 & 1/2 \\ 1/2 & 1/2 \end{bmatrix} = \begin{bmatrix} 3/4 & 1/4 \\ 1/4 & 1/4 \end{bmatrix}\end{aligned}$$

However, the eigenvalues of ρ *are*

$$\lambda_+ = 1/2 + \sqrt{2}/2 \qquad \textit{and} \qquad \lambda_- = 1/2 - \sqrt{2}/2$$

with the corresponding eigenvectors

$$|\phi_+\rangle = \sqrt{\lambda_+}|0\rangle - \sqrt{1-\lambda_+}|1\rangle \qquad \textit{and} \qquad |\phi_-\rangle = \sqrt{\lambda_-}|0\rangle + \sqrt{1-\lambda_-}|1\rangle.$$

Then, it is also possible to view ρ *as an ensemble of mixture of the states* $|\phi_\pm\rangle$ *with probability* $\lambda_\pm$ *respectively. This shows that any mixed state density matrix could have multiple decompositions as an ensemble of pure states.*

This property also makes the identification of a density matrix as being entangled or not a more complicated task. The equivalent of a product state in the density matrix setting is called a separable state.

Definition 3.7 (Separable State)**.** *Given two sub-systems A, B and a density matrix* ρ_{AB}

defined on the joint system $\mathcal{H}_A \otimes \mathcal{H}_B$. ρ_{AB} describes a separable state if it can be written as

$$\rho_{AB} = \sum_i c_i \sigma_i^A \otimes \tau_i^B \quad \text{with } c_i \geq 0 \text{ for each } i,$$

where each σ_i^A is a density matrix in $\mathcal{H}_A$, each τ_i^B is a density matrix in $\mathcal{H}_B$ and $\sum_i c_i = 1$.

Since the same density matrix represents different mixtures of states, it is enough that one mixture of product states exists for a density matrix to be separable. A density matrix which is not separable is then called entangled. More specifically, an entangled state was required to produce the mixture. While in the case of pure states it is easy to determine if a given state is product or not, it is considered to be a hard problem even in the case of certain bipartite systems [Gha10]. For the simple case of the 2-qubit state, the Peres-Horodecki condition of having [Per96, HHH96] a Partial Positive Transpose is both necessary and sufficient. However, this criterion becomes inconclusive for larger systems. Hence, the problem of finding if a generic density matrix is entangled remains a hard problem.

The main goal in quantum computation is to manipulate quantum states by performing operations on them. For this, one uses objects from linear algebra namely, linear operators.

Definition 3.8 (Linear Operators)**.** *A linear operator on a vector space $\mathcal{H}$ is a linear transformation $T : \mathcal{H} \to \mathcal{H}$ that maps vectors in $\mathcal{H}$ to vectors in $\mathcal{H}$.*

One of the simplest examples of an operator is the outer product of two states $|\alpha\rangle$, $|\beta\rangle$ given by $|\alpha\rangle\langle\beta|$. Clearly, applying this operator to any state $|\psi\rangle$ results in

$$|\alpha\rangle\langle\beta||\psi\rangle = |\alpha\rangle\langle\beta|\psi\rangle = (\langle\beta|\psi\rangle)|\alpha\rangle$$

which could be considered an un-normalized state. Recall that in this sense, the orthogonal projection $|\alpha\rangle\langle\alpha|$ is an operator that projects the state it is applied to the 1-dimensional space spanned by $|\alpha\rangle$. Any convex combination of the projectors would not affect the linearity of the transformation. Hence, the density matrix can also be viewed as an operator which is termed the *density operator*.

There is an efficient way to express the action of every linear operator in terms of how it acts on the basis states of the Hilbert space it acts on.

Theorem 3.2. *Given an orthonormal basis for a Hilbert space $\mathcal{H}$, $\mathcal{B} = \{|b_i\rangle\}$, every linear operator T on $\mathcal{H}$ can be written as*

$$T = \sum_{b_i, b_j \in \mathcal{B}} T_{i,j} |b_i\rangle\langle b_j| \quad \text{where } T_{i,j} = \langle b_i|T|b_j\rangle$$

Now the action of T on a state $|\phi\rangle$ is expressed as

$$T|\phi\rangle = \left(\sum_{b_i, b_j \in \mathcal{B}} T_{i,j} |b_i\rangle\langle b_j| \right) |\phi\rangle = \sum_{b_i, b_j \in \mathcal{B}} \langle b_j|\phi\rangle |b_i\rangle.$$

Example 3.4. *Consider the classical bit flip operation X. Consider it's action on the basis states $\{|0\rangle, |1\rangle\}$*

$$|0\rangle \to |1\rangle \quad \textit{and} \quad |1\rangle \to |0\rangle$$

allowing the operator X to be written as

$$X = |0\rangle\langle 1| + |1\rangle\langle 0| = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Since operators are matrices, they can also be classified as being self-adjoint, unitary or Hermitian as seen with this sequence of definitions

Definition 3.9. *For any operator T on $\mathcal{H}$, its adjoint operator $T^\dagger$ is defined as an operator satisfying the condition*

$$(\langle\alpha|T^\dagger|\beta\rangle)^* = \langle\beta|T|\alpha\rangle \quad \forall|\alpha\rangle, |\beta\rangle \in \mathcal{H}$$

Note that this satisfies the condition that the matrix form of $T^\dagger$ is just the complex conjugate transpose of the matrix representation of T.

Definition 3.10. *An operator U is called unitary if $U^\dagger = U^{-1}$ where U^{-1} is the matrix inverse of U.*

Unitary operators help to represent the consistent evolution of quantum systems i.e. one where the sum of probabilities of all events remains 1. This class of operators also encompass quantum logic gates which are building blocks for quantum circuits and quantum algorithms. They preserve the inner product and the norm between vectors.

Claim 3.1. *Given two states $|\psi\rangle$ and $|\phi\rangle$, any unitary operator U preserves the inner product between $(|\psi\rangle, |\phi\rangle)$ and $(U|\psi\rangle, U|\phi\rangle)$.*

Proof. Using the fact that $U^\dagger U = \mathbb{I}$ and $(AB)^\dagger = B^\dagger A^\dagger$,

$$\begin{aligned}(U|\psi\rangle)^\dagger U|\phi\rangle &= \langle\psi|U^\dagger U|\phi\rangle \\ &= \langle\psi|\mathbb{I}|\phi\rangle = \langle\psi|\phi\rangle\end{aligned}$$

□

Another important set of operators is the set of Hermitian operators which also correspond to *observables* – operators that can be used to measure some property of a quantum system i.e. a measurement. An operator T is called Hermitian (or self-adjoint) if $T^\dagger = T$. A useful property of Hermitian operators is that they have real eigenvalues. In the case of observables, these eigenvalues correspond to the possible outcomes of a measurement. Given such an observable M, and an n-qubit state $|\psi\rangle$, the *expectation value* of M with respect to $|\psi\rangle$ is given by the expression $\langle\psi|M|\psi\rangle$. An expectation corresponds not to a single measurement performed on one

copy of the state but an average of the measurement performed over may copies of the state $|\psi\rangle$ while collecting the statistics of each measurement. Considering each measurement as an independent operation, one can use Chernoff bounds to determine that, using poly(n) copies to measure will allow for the approximation of the expectation to within an additive error of $\frac{1}{\text{poly}(n)}$. This can also be generalized to using any density matrix in the place of the pure state $|\psi\rangle$ except that in this case, the expectation value is given by $\text{Tr}(M\rho)$.

A special type of observable is the projector defined as

Definition 3.11 (Projector)**.** *A linear operator P acting on a Hilbert space $\mathcal{H}$ is a projector if it satisfies the condition that $P^2 = P$. A projector is considered an orthogonal projector if it is self-adjoint i.e. $P^\dagger = P$.*

Clearly, the previously mentioned projectors $P_\psi = |\psi\rangle\langle\psi|$ satisfy the above definition and are in fact orthogonal projectors. While the trace of an operator T is the same as the trace of its matrix form, it can be defined with respect to an orthonormal basis $\{|b_i\rangle\}$ of the corresponding Hilbert space as

$$\text{Tr}(T) = \sum_{b_i} \langle b_i|T|b_i\rangle.$$

While the trace is a scalar-value function, the *partial trace* is an operator-valued function defined on a bipartite quantum system existing in $\mathcal{H}_A \otimes \mathcal{H}_B$. The partial trace is used to "trace out" one of the systems (say, B) and generate a reduced density matrix to describe the residual state of the remaining system (in this case, A). This operation is termed as taking a partial trace with respect to B, Tr_B or as tracing out B. It can be symmetrically defined with respect A using Tr_A too. Formally,

Definition 3.12 (Partial Trace)**.** *Given a bipartite quantum state $\rho_{AB} \in \mathcal{D}(\mathcal{H}_A \otimes \mathcal{H}_B)$ where $\mathcal{D}(\cdot)$ is the set of density operators in the given Hilbert space, the partial trace with respect to B, $\text{Tr}_B : \mathcal{D}(\mathcal{H}_A \otimes \mathcal{H}_B) \to \mathcal{D}(\mathcal{H}_A)$ generates a reduced density matrix ρ_A given by*

$$\rho_A = \text{Tr}_B(\rho_{AB}) = \sum_{b_i} \langle b_i|\rho_{AB}|b_i\rangle \quad \textit{where } \{|b_i\rangle\} \textit{ is an orthonormal basis for } \mathcal{H}_B.$$

Bloch Sphere

The Bloch sphere is a geometrical representation of the state space of a single qubit and as the name suggests is a sphere [NC00]. The antipodal points of the Bloch sphere corresponds to orthogonal states and in general, the north and south poles are usually indexed as $|0\rangle$ and $|1\rangle$ respectively as shown in Figure (3.1). The points on the surface of the sphere correspond to pure states and the interior of the sphere corresponds to mixed states – probabilistic mixtures of pure states. The center of the sphere is the completely mixed state $(\mathbb{I}/2)$ which can be interpreted as an equal parts mixture of any state and its orthogonal state.

The exact correspondence between a single qubit state and the Bloch Sphere can be understood

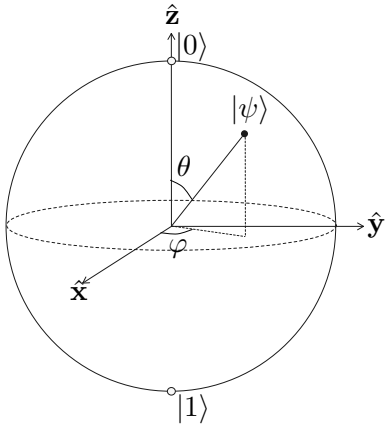


Figure 3.1: The Bloch sphere representing $|\psi\rangle$

with the following relations. Let the Cartesian coordinates of any point on the surface of the sphere correspond to (x, y, z). In the polar coordinate form this is equivalent to

$$(x, y, z) = (\sin\theta\cos\varphi, \sin\theta\sin\varphi, \cos\theta)$$

and the (θ, φ) couple maps to single qubit states as

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + \mathrm{e}^{i\varphi}\sin\left(\frac{\theta}{2}\right)|1\rangle.$$

This can be verified easily since $|0\rangle$ has $\theta = 0, \varphi = 0$, $|1\rangle$ has $\theta = \pi, \varphi = 0$ and for $|-\rangle = \frac{(|0\rangle - |1\rangle)}{\sqrt{2}}$ one finds that $\theta = \frac{\pi}{2}, \varphi = \pi$.

3.2 Local Hamiltonian Problem

The Hamiltonian of a system is an operator that captures the total energy of the system in terms of the interactions between the particles in the system and their kinetic and potential energies. Mathematically, Hamiltonians are expressed as a Hermitian operator (or a sum of Hermitian operators) in the Hilbert space defined by the system of particles. The eigenvectors of this operator define an orthonormal basis for this space and the eigenvalues, being real, also correspond to the energy levels occupied by the system. One of the goals to accomplish using a Hamiltonian is to understand the minimum energy that a system of particles and interactions could possess and also, find the state that possess this minimum energy.

Definition 3.13 (Ground energy and ground states)**.** *The minimum eigenvalue of a given Hamiltonian H is called its ground energy. The corresponding eigenvector is called the ground state of the Hamiltonian. If more than one ground state exists, then the space formed by the span of all the ground states is called the ground space of the Hamiltonian.*

A more structured Hamiltonian is the k-Local Hamiltonian which is a sum of Hermitian operators each of which acts non-trivially on at most k qubits in the system. For instance, each term in the

Local Hamiltonian could quantify the interactions between at most k qubits in the system. Using this structure, it is possible to formally define the k-Local Hamiltonian problem as introduced by Kitaev [KSV02].

Definition 3.14 (k-Local Hamiltonian Problem)**.** *Given a k-Local Hamiltonian $H = \sum_{j=1}^{m} H_{(j)}$ defined on n qubits with $m = \text{poly}(n)$ and a constant k. Let each $H^{(j)}$ have a bounded operator norm $\|H^{(j)}\| \leq \text{poly}(n)$ with entries specified by $\text{poly}(n)$ bits. Additionally, given two constants $0 \leq a < b$ such that $b - a > \frac{1}{\text{poly}(n)}$, the aim is to distinguish between the following cases*

YES*: The ground energy of H is at most a;*

NO*: The ground energy of H is at least b.*

Since each term $H^{(j)}$ in the Hamiltonian is also an observable, given a quantum state $|\psi\rangle$, it is natural to consider the expectation of the Hamiltonian: $\langle\psi|H|\psi\rangle = \sum_j \langle\psi|H^{(j)}|\psi\rangle$. Here, finding the ground energy of H is equivalent to finding the state $|\Gamma\rangle$ that minimizes the expectation value for H. In this respect, given any state $|\psi\rangle$, its expectation value with an observable $H^{(j)}$ can be considered as a measure of how much the state *violates* the observable $H^{(j)}$. This analogy would naturally extend to considering each local term in the Hamiltonian as a constraint on k-qubits where the goal now is to minimize some global objective function. Clearly, this gives the k-Local Hamiltonian problem the flavour of being quantum constraint satisfaction problem. In fact, it is fair to argue that it is a quantum extension of the MAXkCSP problem which, given a CSP instance with k-ary constraints, asks which assignment satisfies the largest number of constraints.

Many examples of the MAXkCSP problem turn out to be NP-complete, like MAXkSAT which is NP-complete for $k \geq 2$. Correspondingly, Kitaev [KSV02], resolved the question of the complexity of the k-Local Hamiltonian problem by showing that the 5-Local Hamiltonian problem is QMA-complete. QMA (Quantum Merlin-Arthur) is considered to be a quantum analogue of the NP class and is defined here for completeness.

Definition 3.15 (Quantum Merlin-Arthur (QMA))**.** *A language L is said to be in $\text{QMA}(c, s)$ if there exists polynomial time quantum verifier V and polynomials p, p' such that for all inputs x,*

$x \in L \Rightarrow$ there exists a state $|\psi\rangle$ such that V accepts $(x, |\psi\rangle)$ with probability at least c;
$x \notin L \Rightarrow$ for all states $|\psi\rangle$, V accepts $(x, |\psi\rangle)$ with probability at most s

where $|\psi\rangle$ is a quantum state on at most $p(|x|)$ qubits. Further, $\text{QMA}(c, s)$ is in QMA when $c - s \geq \frac{1}{p'(|x|)}$.

Kempe and Regev [KR03] reduced the locality to show that even 3-Local Hamiltonian is QMA-complete and this was equivalently shown with a different construction by Mozes and Nagaj [NM07]. Since the 1-Local Hamiltonian problem is in P, the remaining question of the 2-Local Hamiltonian was resolved by Kempe, Kitaev and Regev [KKR06] by showing its QMA-completeness. One sub-class of the Local Hamiltonian problem considered here is that of *frustration-free* Local Hamiltonians.

Definition 3.16. *A frustration free Local Hamiltonian $H = \sum_j H^{(j)}$ is a Hamiltonian whose global groundstate is also a groundstate of each of its local terms $H^{(j)}$.*

This class does encompass many interesting commuting Hamiltonians like the toric code which is useful for topological quantum computing [Kit03], general quantum error correcting codes [Got97], as well as non-commuting Hamiltonians like the AKLT model [AKLT87] which are used in numerical techniques to approximate the ground state of various Hamiltonians.

3.3 Quantum SAT

With the k-local Hamiltonian problem extending MAXkCSPs, the candidate for an extension of MAXkSAT would be QkSAT. As expected, it is restriction of the k-local Hamiltonian, specifically to the frustration free case with each local term being a k-local projector.

Definition 3.17 (Local Projector)**.** *Given an n-qubit system and a constant k that signifies the locality, a k-local projector is an n-qubit projector of the form $\Pi = \Pi_J \otimes \mathbb{I}_{rest}$ where $J = (j_1, \ldots, j_k)$ is a tuple of k qubits and Π_J is a projector that works in the 2^k-dimensional Hilbert space $\mathcal{H}_{j_1} \otimes \ldots \otimes \mathcal{H}_{j_k}$ and $\mathbb{I}_{rest}$ is the identity operator on the rest of the system.*

Example 3.5. *Given an n qubit system, $\Pi_{ij} = (|00\rangle\langle 00| + |11\rangle\langle 11|)_{ij} \otimes \mathbb{I}_{rest}$ is a 2-local projector acting on qubits (i, j) with the identity acting on the remaining $n-2$ qubits. Similarly, a 1-local projector on qubit i would be written as $\Pi_i = |\psi\rangle\langle\psi|_i \otimes \mathbb{I}_{rest}$ where $|\psi\rangle$ is some single qubit state.*

In general, the *rank* of a k-local projector $\Pi = \Pi_J \otimes \mathbb{I}_{rest}$ is given by the rank of its non-trivial part i.e. the rank of Π_J and in a slight abuse of notation, this would be denoted as $\texttt{rank}(\Pi)$. A k-local projector Π with $\texttt{rank}(\Pi) = 2^k - 1$ is said to have *maximal* rank and is said to be *unsatisfiable* if it has full rank i.e. $\texttt{rank}(\Pi) = 2^k$. Projectors being non-negative operators, their minimum eigenvalue is 0 implying that the ground state of every local projector has energy 0. Now given a set of these projectors as a QkSAT instance, it is possible that the ground energy of the sum of these projectors is also 0. In that case, the ground state achieving 0 energy also has 0 energy with each local projector. Equivalently, the QkSAT instance in frustration-free and this also extends the classical notion of perfectly satisfying a given formula. Then, the QkSAT problem can be stated as the problem of deciding if an instance is frustration free or not.

Definition 3.18. *Given a k-local Hamiltonian defined on n qubits as a sum of m k-local projectors $H = \sum_{i=1}^{m} \Pi^{(i)}$ with $k = O(1)$ and a constant $b > 1/n^c$ for some constant c, decide which of the following cases holds*

> YES*: The ground energy of H is 0;*
> NO*: The ground energy of H is at least b.*

Since the YES case can be determined without error, it follows that QkSAT can be considered as belonging to the class QMA_1 – the one sided error version of QMA. Bravyi [Bra11] showed that QkSAT is QMA_1-complete for $k \geq 4$ along with the interesting result that Q2SAT has a polynomial time algorithm to decide it. It was recently shown by Gosset and Nagaj [GN13] that

even Q3SAT is QMA_1-complete.

3.3.1 Q1SAT and Q2SAT

As the focus would be more on the tractable parts of the QSAT landscape – Q1SAT and Q2SAT, they are explicitly discussed here. Q1SAT like its classical counterpart, 1SAT can be treated as a read-once formula which contributes to it being in P. Then, any Q1SAT instance is satisfiable if each qubit is constrained to be orthogonal to at most one single qubit state as the maximal rank is $2^1 - 1 = 1$. Formally,

Definition 3.19. *A* Q1SAT *Hamiltonian on n qubits is Hermitian operator $H = \sum_{\{i \in I\}} \Pi_i$ for some $I \subseteq [n]$ where $\Pi_i = |\psi_i\rangle\langle\psi_i|_i \otimes \mathbb{I}_{[n]\setminus\{i\}}$ for some $|\psi_i\rangle \in \mathcal{H}_i$.*

In the case of Q2SAT, the 2-local projector Π could have $1 \leq \texttt{rank}(\Pi) \leq 3$ in a frustration free instance with the maximal rank being $2^2 - 1 = 3$. Of course, a rank-4 2-local projector being full rank cannot have 0 ground energy. Formally

Definition 3.20. *A* Q2SAT *Hamiltonian on n qubits with m local terms is a Hermitian operator $H = \sum_{\{e \in I\}} \Pi_e$ for some $I \subseteq \{(i,j) \in [n] \times [n] : 1 \leq i \leq j \leq n\}$ where for $i < j$, $\Pi_{ij} = \hat{\Pi}_{ij} \otimes \mathbb{I}_{rest}$ with $\hat{\Pi}_{ij}$ defined on $\mathcal{H}_i \otimes \mathcal{H}_j$. If $i = j$, $\Pi_{ii} = \hat{\Pi}_i \otimes \mathbb{I}_{rest}$ with $\hat{\Pi}_i$ defined on $\mathcal{H}_i$.*

It will be a common practice to use Π_{ij} instead of $\hat{\Pi}_{ij}$ to refer to the action of the 2-local projector on $\mathcal{H}_i \otimes \mathcal{H}_j$. A rank-1 projector on 2 qubits $\Pi_{ij} = |\psi\rangle\langle\psi|$ is referred to as *entangled* if $|\psi\rangle$ is an entangled state and *product* when $|\psi\rangle$ is a product state.

A Q2SAT Hamiltonian H is said to have a *Star-like* configuration if there exists a pair of qubits u, v with $\Pi_{u,v} \neq 0$ such that *all* projectors involve either u or v. Finally, we say that H has *no repetitions* if there does not exist any pair of different projectors $\Pi_e, \Pi_{e'}$ which act non-trivially on the same set of qubits.

3.3.2 Energies and distances

Clearly, a crucial part of dealing with Local Hamiltonians involves finding energies of states with respect to a given Hamiltonian. To denote those energies more in terms of constraint satisfaction terminology, energies are mapped to the extent to which a state satisfies or violates a local term. For any projector Π and a state $|\psi\rangle$, $|\psi\rangle$ is said to *satisfy* Π up to ϵ if $E_\psi(\Pi) := \langle\psi|\Pi|\psi\rangle \leq \epsilon^2$. The energy $E_\psi(\Pi)$ is the *violation energy* of $|\psi\rangle$ with respect to the projector Π. When the state of the system is described by a density matrix, ρ, its violation energy with respect to Π is given by $E_\rho := \mathrm{Tr}(\rho\Pi)$.

Another metric useful in understanding the precision of a solution is a distance measure between quantum states. It complements the use of violation energies and it is possible to switch between the distances of 2-qubit states and their violation energies quite easily. Their relation is described below.

Given a projector $\Pi_\psi = |\psi\rangle\langle\psi|$ and a state $\rho_\alpha := |\alpha\rangle\langle\alpha|$, the violation energy is given by

$\mathrm{Tr}(\Pi_\psi \rho_\alpha) = |\langle\alpha|\psi\rangle|^2$. On the other hand, the norm or *Frobenius distance* between the states $|\alpha\rangle$ and $|\psi\rangle$ is defined as

$$\|\alpha - \psi\| := \sqrt{\mathrm{Tr}[(|\alpha\rangle\langle\alpha| - |\psi\rangle\langle\psi|)(|\alpha\rangle\langle\alpha| - |\psi\rangle\langle\psi|)^\dagger]}.$$

It now follows that $\|\alpha - \psi\|^2 = 2 - 2|\langle\alpha|\psi\rangle|^2 = 2 - 2\,\mathrm{Tr}(\Pi_\psi \rho_\alpha)$. Therefore, the violation energy is related to the Frobenius distance as

$$\mathrm{Tr}(\Pi_\psi \rho_\alpha) = 1 - \frac{1}{2}\|\alpha - \psi\|^2 . \tag{3.1}$$

This completes the discussion on all the preliminaries required for the results that will be presented in the following chapters.

Chapter 4

A linear time quantum 2SAT algorithm

Question: *Can quantum 2SAT be solved in linear time, similar to its classical counterpart?*

This chapter has results on a linear time algorithm for the quantum-2SAT problem, its robustness to frustration-freeness and dealing with almost frustration free 2-Local Hamiltonian problems. While these results can be appreciated independently of the trial and error model, the robustness and notion of almost-free 2-Local Hamiltonians has a direct bearing on solving the unknown input version of quantum 2SAT in the trial and error model.

4.1 Algorithms for Boolean 2SAT

A first step to understanding both Bravyi's algorithm and the linear time algorithm for Q2SAT is understanding the classical algorithms that served as a basis for them – Krom's algorithm and the Davis-Putnam procedure respectively.

4.1.1 Krom's Algorithm

Krom's algorithm [Kro67] proceeds by finding the transitive closure of the clauses in the given 2SAT formula. For any triple of variables $\{a, b, c\}$, if the formula contains the clauses $(a \vee b)$ and $(\overline{b} \vee c)$, they can be viewed as the implications $\overline{a} \Rightarrow b$ and $b \Rightarrow c$. Then, by transitivity of the implications, the implication $\overline{a} \Rightarrow c$ can be inferred which translates to adding the clause $(a \vee c)$ to the formula. After repeating this process till no more implications can be added, the formula is considered consistent if both $(z \vee z)$ and $(\overline{z} \vee \overline{z})$ are not generated simultaneously. Moreover, a formula is satisfiable if and only if it is consistent. The transitive closure procedure would consume $O(n^3)$ time to check for every triple but would only be able to check if the formula is consistent or not. To extract a satisfying assignment requires running $O(n)$ consistency checks – one for each variable in the formula and so requires $O(n^4)$ time to run in total.

4.1.2 Davis-Putnam Procedure

The Davis-Putnam Procedure [DP60, DLL62] based on unit propagation and resolution was conceived to check the validity of any first-order logic formula. While the generic procedure also

dealt with quantifiers, the version sketched here focuses on a quantifier free 2SAT formula. In the worst case, it could have exponential complexity on general SAT formulas but has a polynomial complexity for 2SAT formulas.

It can be assumed without loss of generality that the formula contains only two literal clauses since there is only one way to satisfy a single literal clause and so, simplify the formula. The procedure does the following. Pick a variable that is currently unassigned, say x, set it to `true` and simplify the formula: Any clause of the form $(x \vee y)$ is also set to `true` and can be removed from the formula. Any clause of the $(\overline{x} \vee z)$ will be simplified to the single literal clause (z) since $\overline{x}$ is `false`. This forcibly sets z to `true` to satisfy the formula and this value could propagate to other clauses that contain z and so on. The latter step defines the unit propagation that could cascade throughout the formula until no more variables are forcibly assigned values. If no contradiction arose up to this point, then the simplified formula now contains only two literal clauses and the whole process is repeated. If not, the contradiction can be traced to setting x to `true` as all other variables assigned after that were set forcibly. So, set x to `false` and propagate again. If a contradiction is faced again, then it can be concluded that there is no valid assignment for x and hence, the formula is unsatisfiable.

There are instances where the above procedure can take $\Omega(n^2)$ time to execute. A simplification suggested by Even, Itai and Shamir [EIS76] was to propagate the two values for x in parallel and accepting the assignment of the first branch that stops without facing a contradiction thereby consuming only $O(n + m)$ time.

4.2 2-Local Hamiltonian and Quantum 2SAT

To put the result presented in this chapter into context, it helps to recall what is known about the 2-local Hamiltonian. Wocjana and Beth [WB03] initially showed that NP-complete problems like finding a max-cut in a graph or finding the largest independent set in a graph can be formulated as a 2-local Hamiltonian implying that the complexity of the 2-local Hamiltonian is NP-hard at the very least. Bravyi and Vyalyi [BV05] considered the special case where the local terms of the Hamiltonian commute putting the 2-local Hamiltonian with commuting terms firmly in NP. However, the general case was resolved by Kempe, Kitaev and Regev [KKR06] when they showed the QMA-completeness of the 2-local Hamiltonian. A full classification of 2-local qubit Hamiltonians was undertaken by Cubitt and Montanaro [CM16b] in the spirit of Schaefer's classification of Boolean CSPs [Sch78]. Depending on the fixed set $\mathcal{S}$ from which the local terms of the Hamiltonian are picked, the corresponding 2-local qubit $\mathcal{S}$-Hamiltonian is either in P, NP-complete, QMA-complete or falls in a class which lies between MA and AM. Here MA and AM refer to the private coin and public coin interactive proof systems respectively and the former is also viewed as a probabilistic analogue of NP. Our result zeros in on the case where the 2-local qubit Hamiltonian is in P, specifically, when $\mathcal{S}$ is the set of 2-local projectors. Additionally, as the goal is to find a ground state when these Hamiltonians are frustration free,

our result focuses on the Quantum 2SAT problem. The first algorithm for this problem, proposed by Bravyi, had a running time of $O(n^4)$ when arithmetic operations on complex numbers were assumed to take unit time.

4.2.1 Bravyi's Algorithm

Bravyi's algorithm [Bra11] could be viewed as analogous to Krom's transitive closure algorithm and showed that the frustration freeness of a Q2SAT instance could be decided by a deterministic algorithm in $O(n^3)$ time by checking if the instance had a complete, equivalently consistent, set of constraints. Moreover, the algorithm by running $O(n)$ consistency checks could extract a ground state having a polynomial sized classical description.

The quantum constraints being projectors on a 4-dimensional system could have rank ≥ 1 and the first part of the algorithm performs rank reduction to modify the instance to an equivalent one with just rank 1 constraints. A rank 4 constraint would render the instance unsatisfiable and a rank 3 constraint has only one possible satisfying assignment thereby reducing the size of the instance. A rank 2 constraint on Π_{ab} on qubits (a, b) can be viewed as an isometry $V : \mathbb{C} \longrightarrow \mathbb{C} \otimes \mathbb{C}$ where $\Pi_{ab} = \mathbb{I} - (VV^\dagger)$. The isometry can be used to merge qubits (a, b) into a single qubit c. Once the instance possesses only rank 1 constraints, the algorithm proceeds as in the classical case. Each triple is examined to add a constraint on qubits (a, c) if there exists constraints on qubits (a, b) and (a, c). Any increase in rank of the constraints is reduced again until no more changes can be made. Of course, generating the transitive closure and the assignment is not as straightforward as in the classical case and requires viewing the constraints as tensors and performing operations on them.

4.3 Generalizing the Davis-Putnam Procedure

The linear time algorithm in this chapter can be viewed as a suitable modification of the Davis-Putnam procedure so as to be able to handle (a) high entanglement structure in the ground state (b) projectors of rank ≥ 1 and (c) the increase in domain size for a qubit from Boolean values to an arbitrary quantum state. Moreover, all this has to be handled while still maintaining the deterministic classical properties of the algorithm. At this juncture, to get a clear picture of the techniques used, it is helpful to consider that the algorithm works in the algebraic complexity scenario where it is assumed that arithmetic operations on complex numbers consume unit time.

4.3.1 Simple ground states

A ground state with high entanglement structure cannot be described with a polynomial sized classical description and so poses the biggest threat to the classical nature of the algorithm. While the idea of propagating assignments in a bigger domain is still reasonable, propagating an entangled state naively becomes an unclear concept. Indeed, it is also not possible to escape

from the possibility that the qubits may be entangled if a rank 3 constraint forces them to be in an entangled state. The product state theorem helps to curb the amount of entanglement in the ground state. It effectively states that every frustration free Q2SAT instance has a ground state made up of a tensor product of 1-qubit and 2-qubit states with the 2-qubit states occurring only in the support of certain rank 3 constraints. This structural theorem enables the algorithm to pick an arbitrary qubit assignment and perform operations akin to the unit propagation over Boolean constraints. Additionally, in the case of a contradiction, it indicates two candidate assignments – $|\psi_i\rangle$ to qubit i or $|\psi_j\rangle$ to qubit j – such that at least one of them will propagate successfully in a frustration free instance.

A slightly weaker claim has already appeared in Theorem 2 of [CCD+11]. The main difference is that here the 2-qubit states are specifically attributed to rank-3 projectors. Just as in [CCD+11], our derivation relies on the notion of a *genuinely entangled state* using which Theorem 1 of [CCD+11] is restated below.

Proposition 4.1. *Any 2-local frustration-free Hamiltonian on $n \geq 3$ qubits that has a genuinely entangled ground state also has another ground state, which is a product of one-qubit and two-qubits states.*

Along with that, the following fact about 2-dimensional subspaces in $\mathbb{C}^2 \otimes \mathbb{C}^2$ is also required.

Proposition 4.2. *Any 2-dimensional subspace V of the 2-qubit space $\mathbb{C}^2 \otimes \mathbb{C}^2$ contains at least one product state, which can be found in constant time.*

Proof. Take a basis $\{|\psi\rangle, |\phi\rangle\}$ of the two-dimensional subspace $V^\perp$, the orthogonal complement of V where $|\psi\rangle$ and $|\phi\rangle$ are 2-qubit states. The goal is to find a product state $|\alpha\rangle \otimes |\beta\rangle$ such that $\langle\psi|(|\alpha\rangle \otimes |\beta\rangle) = \langle\phi|(|\alpha\rangle \otimes |\beta\rangle) = 0$. First, express all the states in the standard computational basis as

$$|\psi\rangle = \sum_{ij\in\{0,1\}^2} \psi_{ij}|ij\rangle \quad \text{and} \quad |\phi\rangle = \sum_{ij\in\{0,1\}^2} \phi_{ij}|ij\rangle; |\alpha\rangle = \sum_{i\in\{0,1\}} \alpha_i|i\rangle \quad \text{and} \quad |\beta\rangle = \sum_{i\in\{0,1\}} \beta_i|i\rangle$$

$$\Rightarrow \quad |\alpha\rangle \otimes |\beta\rangle = \sum_{ij\in\{0,1\}^2} \alpha_i\beta_j|ij\rangle$$

The orthogonality conditions translate to

$$\langle\psi|(|\alpha\rangle \otimes |\beta\rangle) = 0 \Rightarrow \sum_{ij} \psi^*_{ij}\alpha_i\beta_j = 0 \quad \text{and} \quad \langle\phi|(|\alpha\rangle \otimes |\beta\rangle) = 0 \Rightarrow \sum_{ij} \phi^*_{ij}\alpha_i\beta_j = 0$$

and now the goal is to find the complex coefficients α_i's and β_j's that satisfy these conditions. Grouping the coefficients into 2×2 matrices and 2×1 vectors as

$$\Psi = \begin{pmatrix} \psi^*_{00} & \psi^*_{01} \\ \psi^*_{10} & \psi^*_{11} \end{pmatrix}, \quad \Phi = \begin{pmatrix} \phi^*_{00} & \phi^*_{01} \\ \phi^*_{10} & \phi^*_{11} \end{pmatrix}, \quad v_\alpha = \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix}, \quad \text{and} \quad v_\beta = \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix}$$

the idea is to find vectors v_α, v_β such that

$$v_\alpha^T \Psi v_\beta = v_\alpha^T \Phi v_\beta = 0\,. \tag{4.1}$$

If there was a v_β such that $\Psi v_\beta \propto \Phi v_\beta$, then one could easily pick a single appropriate v_α to satisfy Equation (4.1). In the case that Φ is a singular matrix, it is possible to choose v_β that is in the null space of Φ. Then, correspondingly it is possible to pick a v_α that is in the null space of Ψ. On the other hand, when Φ is not a singular matrix, choose v_β such that it is an eigenvector of $\Phi^{-1}\Psi$ so that $\Phi^{-1}\Psi v_\beta = c v_\beta$ for some eigenvalue c. This satisfies $\Psi v_\beta \propto \Phi v_\beta$ and an appropriate v_α can now be chosen. □

Note that, as the proof is constructive, it also shows that the product state can be found in constant time. The product state theorem is stated as follows.

Theorem 4.1. *Any frustration-free* Q2SAT *instance has a ground state which is a tensor product of one qubit and two-qubit states, where two-qubit states only appear in the support of rank-*3 *projectors.*

Proof. Consider a frustration-free Q2SAT instance H and let $|\Gamma\rangle$ be its ground state. Much like any number that can be decomposed as a product of prime numbers, an n-qubit state can be written as the tensor product of one or more genuinely entangled states. Generally, $|\Gamma\rangle$ can be written as the tensor product

$$|\Gamma\rangle = |\alpha^{(1)}\rangle \otimes |\alpha^{(2)}\rangle \otimes \cdots \otimes |\alpha^{(r)}\rangle\,,$$

where each $|\alpha^{(i)}\rangle$ is a genuinely entangled state defined on a subset $S^{(i)}$ of qubits. Note that the $S^{(i)}$s form a partition of the set of n qubits. If $\Pi_{jk} = \mathbb{I} - |\psi\rangle\langle\psi|_{jk}$ is a rank-3 projector then, the only way to satisfy Π_{jk} is to set qubits (j,k) to $|\psi\rangle_{jk}$. Hence, *every* ground state of H will contain $|\psi\rangle_{jk}$ as a tensor product with the rest of the system. Now, if $|\psi\rangle_{jk}$ is an entangled state, there would necessarily be a subset $S^{(i)} = \{j,k\}$ in the above decomposition where $|\alpha^{(i)}\rangle = |\psi\rangle_{jk}$. On the other hand, if $|\psi\rangle_{jk}$ is a product state, then there would be two subsets $S^{(i_1)} = \{j\}$, and $S^{(i_2)} = \{k\}$. Therefore, when a rank-3 projector is involved in some partition, $|S^{(i)}| \leq 2$ for that partition.

Let $H^{(i)}$ be the Hamiltonian that is the sum of all the projectors whose support is in $S^{(i)}$. By definition, $|\alpha^{(i)}\rangle$ is also a ground state of $H^{(i)}$. As discussed above, any subset $S^{(i)}$ with exactly 2 qubits either corresponds to the support of a rank-3 projector or corresponds to an $H^{(i)}$ consisting of rank-1 and rank-2 projectors only. Finally, the remaining case of subsets $S^{(i)}$ with more than 2 qubits, corresponds to $H^{(i)}$ consisting only of rank-1 and rank-2 projectors. Using Proposition 4.2, in conjunction with Proposition 4.1, it is known that any $H^{(i)}$ involving multiple qubits but containing no rank-3 projectors has a ground state $|\beta^{(i)}\rangle$ which is a product

state of one qubit states:

$$|\beta^{(i)}\rangle = |\beta_1^{(i)}\rangle \otimes |\beta_2^{(i)}\rangle \otimes \cdots \tag{4.2}$$

For the cases when the $S^{(i)}$ correspond either to one qubit subsets, or to 2-qubits subsets of entangled rank-3 projectors, set $|\beta^{(i)}\rangle = |\alpha^{(i)}\rangle$.

Now the claim is that the product of one-qubit and two-qubit states, $|\beta\rangle = |\beta^{(1)}\rangle \otimes \cdots \otimes |\beta^{(r)}\rangle$, is a ground state of H. This would hold if $|\beta\rangle$ were in the ground space of each projector Π_e. If the support of Π_e is inside one of the S_i subsets, then by definition $\Pi_e|\beta^{(i)}\rangle = 0 \Rightarrow \Pi_e|\beta\rangle = 0$. Assume then that Π_e is supported on a qubit from $S^{(i)}$ and a qubit from $S^{(j)}$ with $i \neq j$. Consider the 3 cases:

1. $|S^{(i)}| = |S^{(j)}| = 1$: Then, $\Pi_e|\beta^{(i)}\rangle \otimes |\beta^{(j)}\rangle = \Pi_e|\alpha^{(i)}\rangle \otimes |\alpha^{(j)}\rangle = 0$.
2. $|S^{(i)}| = 1$ and $|S^{(j)}| \geq 2$: Then, expand $|\alpha^{(j)}\rangle = \lambda_0|0\rangle_u \otimes |y_0\rangle_{S^{(j)}\setminus\{u\}} + \lambda_1|1\rangle_u \otimes |y_1\rangle_{S^{(j)}\setminus\{u\}}$. Here u is the qubit in $S^{(j)}$ which is in the support of Π_e. The vectors $\lambda_0|y_0\rangle, \lambda_1|y_1\rangle$ are, by assumption, linearly independent since $|\alpha_j\rangle$ is assumed to be genuinely entangled. Now, the condition

   $$\begin{aligned} & \Pi_e|\alpha^{(i)}\rangle \otimes |\alpha^{(j)}\rangle = 0 \\ \Rightarrow\ & \lambda_0|y_0\rangle \otimes (\Pi_e|\alpha^{(i)}\rangle \otimes |0\rangle) + \lambda_1|y_1\rangle \otimes (\Pi_e|\alpha^{(i)}\rangle \otimes |1\rangle) = 0 \\ \Rightarrow\ & \Pi_e|\alpha^{(i)}\rangle \otimes |0\rangle = \Pi_e|\alpha^{(i)}\rangle \otimes |1\rangle = 0 \quad \text{(By the linear independence of } |y_0\rangle, |y_1\rangle\text{)}. \end{aligned}$$

   Therefore, Π_e is orthogonal to the subspace $|\alpha^{(i)}\rangle \otimes \mathbb{C}^2$ of the two qubits that it acts on, and in particular it's also orthogonal to $|\beta^{(i)}\rangle \otimes |\beta^{(j)}\rangle$ since $|\beta^{(i)}\rangle = |\alpha^{(i)}\rangle$.
3. $|S^{(i)}|, |S^{(j)}| \geq 2$: This case does not occur as can be shown by contradiction. By writing both $|\alpha^{(i)}\rangle, |\alpha^{(j)}\rangle$ in their respective Schmidt decompositions, and using a similar argument as above, it can be concluded that Π_e must be orthogonal to 4 independent vectors. It therefore cannot be a rank-1 or a rank-2 projector. □

4.3.2 The Constraint Graph

For a classical 2SAT instance, the notion of a constraint graph is straight forward: considering each variable as a vertex, add an edge between vertices (u, v) if there is a 2SAT clause containing a literal of u and v; each edge is labeled by the clause it refers to and hence the constraint graph is a multigraph. Each clause is like a rank-1 projector in the standard computational basis making the representation simple. For Q2SAT, the need to deal with projectors of rank ≥ 1 arises. So, for an efficient representation of the instance, rank-2 projectors are decomposed into a sum of their constituent rank-1 projectors. This is called the rank-1 decomposition of the instance.

Definition 4.1 (Rank-1 Decomposition)**.** *Let H be a* Q2SAT *Hamiltonian. Consider every $(i, j) \in I$ such that* `rank`$(\Pi_{ij}) = 2$ *and decompose Π_{ij} as sum of rank-1 components as $\Pi_{ij} =$*

$\Pi_{ij,1} + \Pi_{ij,2}$. *Then, the rank-*1 *decomposition of* H *is given by*

$$H = \sum_{\texttt{rank}(\Pi_{ij})\neq 2} \Pi_{ij} + \sum_{\texttt{rank}(\Pi_{ij})=2} (\Pi_{ij,1} + \Pi_{ij,2}) \,,$$

Corresponding to this decomposition, it is possible to associate a constraint graph where each qubit is a vertex and each projector term corresponds to a specific directed edge in the graph. Note that if there is a single qubit projector, then it maps to a self loop in the graph. Rank-1 and rank-3 projectors are treated equivalently and there is a directed edge between the 2-qubits involved in the projector. In fact, if qubits (u, v) have a rank-1 projector, there is an edge both from u to v and from v to u. For rank-2 projectors, the qubit pair shares 2 parallel edges in each direction – one for each rank-1 component.

To distinguish between the parallel edges of this multigraph, it is necessary to label them uniquely. For every $u < v$ with $\Pi_{uv} \neq 0$, labelling the edge from u to v with Π_{uv} works well. However, since the same projector also induces an edge from v to u, an appropriate label for these reverse edges is needed. To overcome this, the *reverse* projector is defined as

Definition 4.2 (Reverse Projector)**.** *For a projector* Π *acting on two qubits, its* reverse *projector* Π^{rev} *is given by* $\Pi^{\mathsf{rev}}|\alpha\rangle|\beta\rangle = \Pi|\beta\rangle|\alpha\rangle$*, and for* $i \leq j$ *and* $b \in [2]$*,* $\Pi_{ji} = \Pi_{ij}^{\mathsf{rev}}$ *and* $\Pi_{ji,b} = \Pi_{ij,b}^{\mathsf{rev}}$*.*

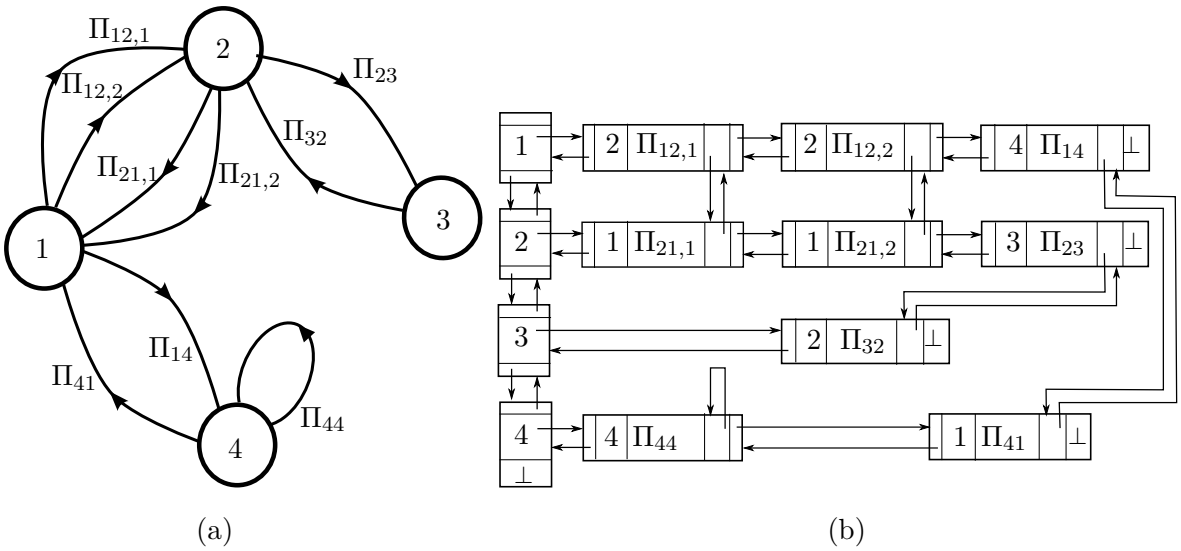
Now, each edge (u, v) is labeled by Π_{uv} and each edge (u, v, b) is labeled with $\Pi_{uv,b}$. Using these concepts, the constraint graph is defined as

Definition 4.3 (Constraint Graph)**.** *Let* H *be a* Q2SAT *given in its rank-*1 *decomposition. The constraint graph of* H *is a directed, labeled, graph with self loops* $G(H) = (V, E, \ell)$ *where* $V = \{i \in [n] : \exists j \in [n] \text{ such that } (i, j) \in I \text{ or } (j, i) \in I\}$*. The edge set* $E = E_1 \cup E_2$ *where*

$$\begin{aligned} E_1 &= \{(i,j) \in [n] \times [n] : (i,j) \in I \text{ and } \texttt{rank}(\Pi_{ij}) \in \{1,3\}, \text{ or } (j,i) \in I \text{ and } \texttt{rank}(\Pi_{ji}) \in \{1,3\}\}; \\ E_2 &= \{(i,j,b) \in [n] \times [n] \times [2] : (i,j) \in I \text{ and } \texttt{rank}(\Pi_{ij}) = 2, \text{ or } (j,i) \in I \text{ and } \texttt{rank}(\Pi_{ji}) = 2\}. \end{aligned}$$

An edge e is said to go from u to v if $e \in \{(u, v), (u, v, 1), (u, v, 2)\}$. Correspondingly, e^{rev} is the reverse edge of e where $(i, j)^{\mathsf{rev}} = (j, i), (i, j, 1)^{\mathsf{rev}} = (j, i, 1)$ and $(i, j, 2)^{\mathsf{rev}} = (j, i, 2)$ respectively. The input to the algorithm is the constraint graph $G(H)$, given in the standard adjacency list representation of weighted graphs. It is suitably modified to deal with parallel edges as shown in Figure (4.1).

The doubly linked list of size at most n has an element for each qubit in the system. The i^{th} element in this list points to a doubly linked list containing the edges that leave vertex i with an element for each (i, j) or (i, j, b). Each element (i, j) stores j, b (if it is an (i, j, b) edge), the label Π_{ij} (or $\Pi_{ij,b}$) and a pointer to the next and previous edges in the list. For the ease of performing quick updates during the course of the algorithm, there is also a double link between the elements representing e and e^{rev} for each edge e. In a slight abuse of notation, for a graph $G = (V, E)$, given a subset of vertices $U \subseteq V$, $G(U)$ refers to the subgraph induced by U. This



(a) (b)

Figure 4.1: (a) The constraint graph for Hamiltonian $H = \Pi_{12} + \Pi_{14} + \Pi_{23} + \Pi_{44}$ where $\texttt{rank}(\Pi_{44}) = \texttt{rank}(\Pi_{23}) = 1, \texttt{rank}(\Pi_{12}) = 2$ and $\texttt{rank}(\Pi_{14}) = 3$ using its rank-1 decomposition (b) The adjacency list representation for the constraint graph $G(H)$.

is not to be confused with the constraint graph notation $G(H)$ for a Hamiltonian H and the context would be clear from usage.

4.3.3 Assignments

Following the classical Davis-Putnam procedure, the ground state is built up in a step-wise manner using partial assignments that are extended one qubit at a time, or simply assignments in short.

Definition 4.4. *A partial assignment, or simply* assignment*, s is a mapping from $[n]$ to 1-qubit or 2-qubit states. For every $i \in [n]$, the value $s(i)$ is one of the following: (a) a 1-qubit state $|\alpha\rangle_i$, (b) a 2-qubit entangled state $|\gamma\rangle_{ij}$ for some $j \neq i$ and the entangled state is shared with j or (c) a symbol $\ominus$ which is used to signify an unassigned variable.*

It is common practice to consider normalized quantum states, that is, states $|\alpha\rangle$ such that $\langle\alpha|\alpha\rangle = 1$. However, this algorithm will deal with and assign un-normalized states to the variables. This does not affect the accuracy of the algorithm but can result in an un-normalized ground state as the output. It is possible to additionally normalize the ground state without affecting the running time of the algorithm. A partial assignment has various properties which are listed below:

- The *support* of s is the set of qubits that have been assigned values in the partial assignment and is given by $\mathsf{supp}(s) = \{i \in [n] : s(i) \neq \ominus\}$. The assignment s is *empty* if $\mathsf{supp}(s) = \emptyset$. For clarity, the empty assignment is denoted as $s_\ominus$.
- A natural ordering can be imposed on assignments. For 2 assignments s and s', s' is an *extension* of s, if for every i, such that $s(i) \neq \ominus$, $s'(i) = s(i)$.
- An assignment is *total* if $s(i) \neq \ominus$, for all i.

- Every assignment defines a product of 1-qubit and 2-qubit states on the support of s denoted as $|s\rangle$.
- An assignment s *satisfies* a projector Π_e (or the edge e) if for any total extension s' of s, $\Pi_e|s'\rangle = 0$.
- For a Hamiltonian H in its rank-1 decomposition and an assignment s, the *reduced Hamiltonian* H_s of s is

$$H_s = H - \sum_{s \text{ satisfies } e} \Pi_e.$$

  The constraint graph $G(H_s)$ of the reduced Hamiltonian H_s is $G_s = (V_s, E_s)$.
- An assignment s is a *pre-solution* if it has a total extension s' satisfying every constraint in H. s is a *solution* if s itself satisfies every constraint in H. Obviously, an assignment is a solution if and only if G_s is the empty graph.
- An assignment s is *closed* if $\mathsf{supp}(s) \cap V_s = \emptyset$.

4.3.4 Propagation

Recalling the original Davis-Putnam Procedure (DP-Procedure), the building block that allows one to determine the assignment for a cluster of variables is the unit propagation process. Essentially, this process involves *unit clauses* i.e. clauses containing just one literal. The only way to satisfy a unit clause containing a literal ℓ is to set ℓ to `true`. This assignment propagates to other clauses in two ways. Any clause which also contains ℓ is automatically satisfied and can be simplified. In any clause where $\bar{\ell}$ occurs, this literal is deleted reducing the size of the clause. This leads to a simplified instance which is still equivalent to the previous set of clauses. In the case of 2SAT this means that once one of the literals in the clause is set to `false`, a unit clause is obtained which can be satisfied according to the rules of unit propagation. Repeated applications of this propagation helps to decide the assignment of a sequence of propagated literals.

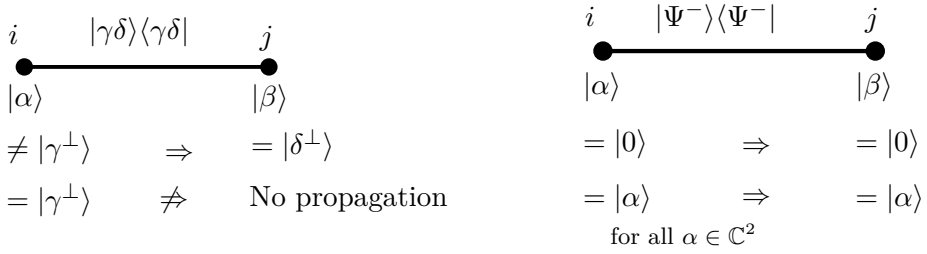
Here, the idea is to extend this notion of propagation to the case of Q2SAT by propagating quantum states across rank-1 constraints in the Hamiltonian. A related notion of using transfer matrices can be traced back to Bravyi's algorithm [Bra11] and Laumann et al. [LMSS10] studying the satisfiability of random instances of QkSAT. Since the product state theorem guarantees a product state solution over rank-1 constraints, propagation should help achieve the following: Given a rank-1 constraint $\Pi_{12} = |\psi\rangle\langle\psi|$, find a state $|\alpha\rangle_1 \otimes |\beta\rangle_2$ such that

$$(\langle\alpha| \otimes \langle\beta|)\Pi_{12}(|\alpha\rangle \otimes |\beta\rangle) = 0 \Rightarrow \langle\alpha| \otimes \langle\beta| \cdot |\psi\rangle = 0.$$

Definition 4.5 (Propagation)**.** *Let $\Pi_e = |\psi\rangle\langle\psi|$ be a rank-1 projector acting on variables i, j, and let $|\alpha\rangle$ be either a 1-qubit state assigned to variable i, or a 2-qubit entangled state assigned to variables k, i for some $k \neq j$. We say that* Π_e propagates $|\alpha\rangle$ *if, up to an arbitrary complex phase, there exists a unique 1-qubit state $|\beta\rangle$ such that $\Pi_e|\alpha\rangle \otimes |\beta\rangle_j = 0$. In such case we say*

that $|\alpha\rangle$ *is propagated to* $|\beta\rangle$ *along* Π_e*, or that* Π_e *propagated* $|\alpha\rangle$ *to* $|\beta\rangle$*.*

When qubit 1 is already assigned the state $|\alpha\rangle$, then analogous to unit propagation, two cases are possible: *(a)* $|\alpha\rangle$ satisfies Π_{12} and so, any $|\beta\rangle$ suffices or *(b)* There exists a unique $|\beta\rangle$ that can satisfy the constraint Π_{12}. More specifically, when $|\psi\rangle$ is a product state, any one of the two cases could occur but if $|\psi\rangle$ is an entangled state then only the second case occurs as illustrated in Figure (4.2) and proved with Lemma 4.1 below.



(a) Product State Constraint (b) Entangled State Constraint

Figure 4.2: Propagation for Quantum 2SAT when $|\psi\rangle$ is a product state or is entangled.

Lemma 4.1. *Consider the rank-*1 *projector* $\Pi_e = |\psi\rangle\langle\psi|$*, defined on qubits* i, j*. If* $|\psi\rangle$ *is entangled, it propagates* every 1*-qubit state* $|\alpha\rangle_i$ *to a state* $|\beta(\alpha)\rangle_j$ *such that if* $|\alpha\rangle_i \neq |\alpha'\rangle_i$ *then* $|\beta(\alpha)\rangle_j \neq |\beta(\alpha')\rangle_j$*. This propagation can be calculated in constant time. When* $|\psi\rangle$ *is a product state* $|\psi\rangle = |x\rangle_i \otimes |y\rangle_j$*, the projector* Π_e *does not propagate states that are proportional to* $|x^\perp\rangle_i$*, while all other states are propagated to* $|y^\perp\rangle_j$*.*

Proof. Let $|\psi\rangle$ be entangled and consider the state $|\alpha\rangle$. Expanding $|\psi\rangle, |\alpha\rangle, |\beta\rangle$ in the standard computational basis,

$$|\psi\rangle = \sum_{i,j} \psi_{ij}|i\rangle \otimes |j\rangle; \quad |\alpha\rangle = \sum_i \alpha_i|i\rangle; \quad |\beta\rangle = \sum_j \beta_j|j\rangle. \tag{4.3}$$

$$\text{Then, } \Pi_e(|\alpha\rangle \otimes |\beta\rangle) = 0 \Rightarrow \sum_{i,j} \psi^*_{ij}\alpha_i\beta_j = 0. \tag{4.4}$$

Since $|\psi\rangle$ is entangled, the 2×2 matrix $\Psi = [\psi^*_{ij}]$ is non-singular. Then, $([\alpha^*_0\ \alpha^*_1]\Psi)^T$ is a 2-dimensional complex vector. Now, using the fact that in a 2-dimensional space, every vector has a unique orthogonal vector (up to scaling factors), it is possible to find $|\beta\rangle$ orthogonal to this vector that would satisfy Equation (4.3). Moreover, this can be found in constant time and a different $|\alpha\rangle$ necessarily maps to a different $|\beta\rangle$.

When $|\psi\rangle$ is a product state, the reasoning is straightforward and follows the same reasoning as the classical case. That is, either the assignment to qubit i is already orthogonal to the constraint's state on qubit i and this single-handedly satisfies the constraint. In this case, there is no propagation as qubit j can be assigned any state and the constraint is still satisfied. However, if qubit i's assignment isn't sufficient, the only other possibility is to satisfy the constraint from qubit j's assignment thereby propagating that assignment across the edge (i, j). □

Propagation will be used, as in the Davis-Putnam procedure, to generate some partial assignment as a tensor product of 1 and 2-qubit states or to find out if the system is unsatisfiable. The following two lemmas highlight this capacity of the propagation sub-routine.

Lemma 4.2 (Single qubit propagation)**.** *Consider a frustration-free* Q2SAT *system* $H = \sum_{e\in I}\Pi_e$ *with a rank-*1 *projector* $\Pi_e = |\psi\rangle\langle\psi|$ *between qubits* i, j*, and assume that* H *has a ground state of the form* $|\Gamma\rangle = |\alpha\rangle_i \otimes |rest\rangle_{[n]\setminus\{i\}}$*. If* Π_e *propagates* $|\alpha\rangle_i$ *to* $|\beta\rangle_j$ *then necessarily* $|rest\rangle_{[n]\setminus\{i\}} = |\beta\rangle_j \otimes |rest'\rangle_{[n]\setminus\{i,j\}}$*.*

Proof. For the first claim assume that Π_e propagates $|\alpha\rangle_i$ to $|\beta\rangle_j$. Without loss of generality,

$$|\mathrm{rest}\rangle_{[n]\setminus\{i\}} = |\beta\rangle_j \otimes |\mathrm{rest}_1\rangle + |\beta^\perp\rangle_j \otimes |\mathrm{rest}_2\rangle \ ,$$

where the states $|\mathrm{rest}_1\rangle, |\mathrm{rest}_2\rangle$ are defined on all the qubits of the system except for (i, j), and are not necessarily normalized. Plugging this expansion into the condition $\Pi_e|\Gamma\rangle = 0$ provides the equation

$$(\Pi_e|\alpha\rangle_i|\beta\rangle_j) \otimes |\mathrm{rest}_1\rangle + (\Pi_e|\alpha\rangle_i|\beta^\perp\rangle_j) \otimes |\mathrm{rest}_2\rangle = 0\,.$$

Since Π_e propagates $|\alpha\rangle_i$ to $|\beta\rangle_j$, one finds that $\Pi_e|\alpha\rangle_i|\beta\rangle_j = 0$ and $\Pi_e|\alpha\rangle_i|\beta^\perp\rangle_j \neq 0$. Therefore, the above equation implies that $|\mathrm{rest}_2\rangle = 0$ and $|\mathrm{rest}'\rangle = |\mathrm{rest}_1\rangle$. □

Lemma 4.3 (Entangled 2-qubits propagation)**.** *Consider a frustration-free* Q2SAT *system* H *with a rank-*1 *projector* $\Pi_e = |\psi\rangle\langle\psi|$ *between qubits* i, j*. Assume that* H *has a ground state of the form* $|\Gamma\rangle = |\phi\rangle_{ik} \otimes |rest\rangle_{[n]\setminus\{i,k\}}$*, where* $|\phi\rangle$ *is an entangled state on qubits* i, k *with* $k \neq j$*. The following facts hold:*

1. $|\psi\rangle$ *is a product state* $|\psi\rangle = |x\rangle_i|y\rangle_j$*.*
2. Π_e *propagates* $|\phi\rangle$ *to* $|y^\perp\rangle$ *and necessarily* $|rest\rangle_{[n]\setminus\{i,k\}} = |y^\perp\rangle_j \otimes |rest'\rangle_{[n]\setminus\{i,j,k\}}$*.*

Proof. Write $|\phi\rangle_{ik}$ in its Schmidt decomposition $|\phi\rangle = \lambda_1|\alpha\rangle \otimes |\beta\rangle + \lambda_2|\alpha^\perp\rangle \otimes |\beta^\perp\rangle$, and note that both $\lambda_1, \lambda_2 \neq 0$, since $|\phi\rangle$ is entangled. Plugging this into the condition $\Pi_e|\Gamma\rangle = 0$,

$$\Pi_e|\Gamma\rangle = \lambda_1|\beta\rangle_k \otimes \Pi_e\big(|\alpha\rangle_i \otimes |\mathrm{rest}\rangle\big) + \lambda_2|\beta^\perp\rangle_k \otimes \Pi_e\big(|\alpha^\perp\rangle_i \otimes |\mathrm{rest}\rangle\big) = 0\,.$$

Since $|\beta\rangle$ is linearly independent of $|\beta^\perp\rangle$, each term above has to independently be 0 i.e. $\Pi_e\big(|\alpha\rangle_i \otimes |\mathrm{rest}\rangle\big) = \Pi_e\big(|\alpha^\perp\rangle_i \otimes |\mathrm{rest}\rangle\big) = 0$.

For the first claim, assume that $|\psi\rangle$ is entangled. Then by Lemma 4.1, Π_e propagates $|\alpha\rangle$ and $|\alpha^\perp\rangle$ to two *different* states, say, $|\gamma_1\rangle \neq |\gamma_2\rangle$. But then by Lemma 4.2, it follows that $|\mathrm{rest}\rangle$ must be both in the form $|\gamma_1\rangle_j \otimes |\mathrm{rest}'\rangle$ and $|\gamma_2\rangle_j \otimes |\mathrm{rest}'\rangle$ and this leads to a contradiction.

For the second claim, assume that $|\psi\rangle = |x\rangle \otimes |y\rangle$ is a product state. Since $\Pi_e\big(|\alpha\rangle_i \otimes |\mathrm{rest}\rangle\big) = \Pi_e\big(|\alpha^\perp\rangle_i \otimes |\mathrm{rest}\rangle\big) = 0$, both states $|\alpha\rangle_i \otimes |\mathrm{rest}\rangle, |\alpha^\perp\rangle_i \otimes |\mathrm{rest}\rangle$ are ground states of the single projector Hamiltonian $\tilde{H} = \Pi_e$. Using Lemma 4.1 and Lemma 4.2, together with the fact

that at least one of the states $|\alpha\rangle, |\alpha^{\perp}\rangle$ is different from $|x^{\perp}\rangle$, the conclusion is that $|\text{rest}\rangle = |y^{\perp}\rangle_j \otimes |\text{rest}'\rangle$. □

The Propagation Procedure

Since propagation is a crucial sub-routine that would be called multiple times in the main algorithm, it would help to understand how it is executed. To that end, some notation to help understand the pseudocode is introduced. Given a Q2SAT instance H consider an assignment s where the graph $G_s = (V_s, E_s)$ is the graph of the reduced Hamiltonian H_s. The goal is to describe the result of a multi-step propagation in the graph G_s until no further propagation is possible. Assume that this propagation starts from some qubit i and can get started when $s(i) = \ominus$ for which i is started with an assignment $|\alpha\rangle \in \mathbb{C}^2$. The other scenario is when qubit i is already assigned some 1 or 2-qubit state in s i.e. $s(i) = |\delta\rangle$ for $|\delta\rangle \in \{|\alpha\rangle_i, |\gamma\rangle_{ij}\}$.

Let s, i and $|\delta\rangle$ be such that $s(i) \in \{\ominus, |\delta\rangle\}$. An edge $e \in E_s$ from i to j *propagates* $|\delta\rangle$ if Π_e propagates it, and $\mathsf{prop}(s, e, |\delta\rangle)$ denotes the state $|\delta\rangle$ is propagated to. Since one is interested in multi-step propagation, the notion is now extended to one over multiple connected edges, specifically, paths. Let $i_0 = i, i_1, \ldots i_k = j$ be vertices in V_s, and let e_r be an edge from i_r to i_{r+1}, for $r = 0, \ldots, k-1$. Set set $|\alpha_0\rangle = |\delta\rangle$ and let $|\alpha_1\rangle, \ldots, |\alpha_k\rangle$ be states such that the propagation of $|\alpha_r\rangle$ along e_r is $|\alpha_{j+1}\rangle$, for $r = 0, \ldots, k-1$. Then, the path $p = (e_0, \ldots, e_{k-1})$ from i_0 to i_k *propagates* $|\delta\rangle$, and we set $\mathsf{prop}(s, p, |\delta\rangle) = |\alpha_k\rangle$. A vertex $j \in V_s$ is *accessible* by propagating $|\delta\rangle$ from i if either $j = i$ or there is a path from i to j that propagates $|\delta\rangle$. $V_s^{\mathsf{prop}}(i, |\delta\rangle)$ denotes the set of such vertices, and by $\mathsf{ext}_s^{\mathsf{prop}}(i, |\delta\rangle)$ the extension of s by the values given to the vertices in $V_s^{\mathsf{prop}}(i, |\delta\rangle)$ by iterated propagation.

The set $V_s^{\mathsf{prop}}(i, |\delta\rangle)$ divides the edges E_s into three disjoint subsets: the edges E_1 of the induced subgraph $G(V_s^{\mathsf{prop}}(i, |\delta\rangle))$, the edges E_2 between the induced subgraphs $G(V_s^{\mathsf{prop}}(i, |\delta\rangle))$ and $G(V_s \setminus V_s^{\mathsf{prop}}(i, |\delta\rangle))$, and the edges E_3 of the induced subgraph $G(V_s \setminus V_s^{\mathsf{prop}}(i, |\delta\rangle))$. While the edges in $E_1 \cup E_2$ are satisfied by s', none of the edges in E_3 are satisfied. Therefore $G_{s'}$ is nothing but $G(V_s \setminus V_s^{\mathsf{prop}}(i, |\delta\rangle))$ without the isolated vertices, and it can be constructed by the following process. Given s and i, the edges in $E_1 \cup E_2$ can be traversed via a breadth first search rooted at i. The levels of the tree are decided dynamically: at any level the next level is composed of those vertices whose value is propagated from the current level. The leaves of the tree are vertices in $V_s \setminus V_s^{\mathsf{prop}}(i, |\delta\rangle)$. The sub-routine Propagation uses a temporary queue Q to implement this process as shown in Algorithm 4.1 and illustrated in Figure (4.3).

Lemma 4.4 (Propagation Lemma)**.** *Let* $\mathsf{Propagation}(s, G_s, i, |\delta\rangle)$ *be called when* H_s *does not have rank-*3 *constraints, and* $s(i) \in \{\ominus, |\delta\rangle\}$*. Let* s' *and* $G' = (V', E')$ *be the outcome of the procedure. The following hold true:*

1. *If* $\mathsf{Propagation}(s, G_s, i, |\delta\rangle)$ *doesn't return "unsuccessful" then* $s' = \mathsf{ext}_s^{\mathsf{prop}}(i, |\delta\rangle)$ *and* $G' = G_{s'}$*. Moreover, if* s *is a pre-solution then* s' *is a pre-solution, and if* s *is closed then* s' *is also closed.*

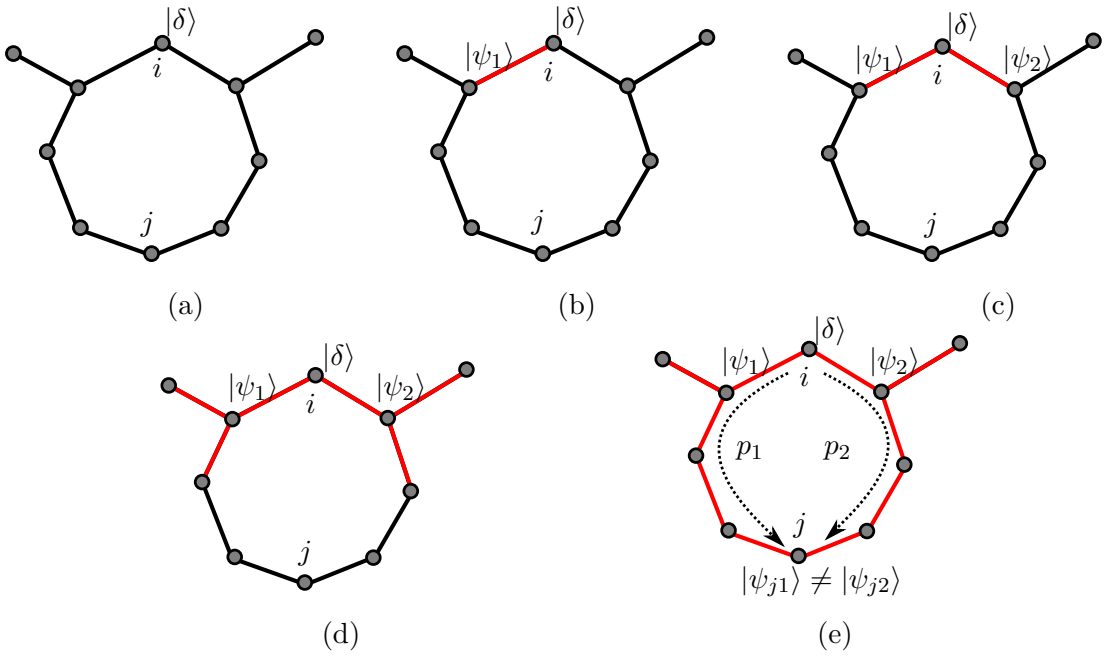


Figure 4.3: ***Multi-step Propagation.*** (a) The propagation starts from i with $|\delta\rangle$; (b) Propagate across any edge going from i; (c) Propagate across all edges going out from i; (d) Propagate to all vertices at a distance of 2 from i in a BFS fashion; (e) j receives two assignments along two different path p_1 and p_2 giving a contradiction if they aren't equal.

Algorithm 4.1: Propagation$(s, G_s, i, |\delta\rangle)$

```
1  s(i) := |δ⟩
2  create queue Q, and put i into Q
3  while Q is not empty do
4      remove the head j of Q
5      foreach edge e from j to k do
6          if e propagates s(j) then
7              if s(k) ∉ {⊖, prop(s, e, s(j))} then abort and return "unsuccessful"
8              if s(k) = ⊖ then s(k) := prop(s, e, s(j))
9              enqueue k
10         remove e and e^rev from E_s
11         if the list pointed to by k is empty then remove k from V_s
12     remove j from V_s
```

---

2. *If* Propagation(s, G_s, i) *returns "unsuccessful" then there is no solution* z *of which is an extension of* s *and for which* $z(i) = |\delta\rangle$.
3. *The complexity of the procedure is* $O\left(|E_s| - |E_{s'}|\right)$.

Proof. The assignments made during the breadth first search correspond exactly to the paths that propagate $|\delta\rangle$ from i. Therefore the extension of s created by the process is indeed $s' = \mathsf{ext}_s^{\mathsf{prop}}(i, |\delta\rangle)$. The while loop removes the edges between vertices in $V_s^{\mathsf{prop}}(i, |\delta\rangle)$ and the edges which go from $V_s^{\mathsf{prop}}(i, |\delta\rangle)$ to $V_s \setminus V_s^{\mathsf{prop}}(i, |\delta\rangle)$, as well as the vertices in $V_s^{\mathsf{prop}}(i, |\delta\rangle)$. Then the edges from $V_s \setminus V_s^{\mathsf{prop}}(i, |\delta\rangle)$ to $V_s^{\mathsf{prop}}(i, |\delta\rangle)$ are removed, as well as the remaining vertices without outgoing (and incoming) edges. resulting in $G' = G_{s'}$.

Suppose that s is a pre-solution, and let z be an extension of s which is a solution and which is a product state on the vertices in V_s. By Theorem 4.1 there exists such a solution since H_s doesn't have rank-3 constraints. Define the assignment z' by

$$z'(j) = \begin{cases} s'(j) & \text{if } \ j \in \mathsf{supp}(s') \\ z(j) & \text{otherwise.} \end{cases}$$

Then z' is a solution which is an extension of s', and therefore s' is a pre-solution. If s is closed then so is s' since only the vertices in $V_s^{\mathsf{prop}}(i, |\delta\rangle)$ get assigned during the process, and they are not included into $V_{s'}$.

Suppose that the procedure returns "unsuccessful". Then there is a vertex $k \in V_s^{\mathsf{prop}}(i, |\delta\rangle)$, and two paths p and p' in G_s from i to k such that $\mathsf{prop}(s, p, |\delta\rangle) = |\beta\rangle$, $\mathsf{prop}(s, p', |\delta\rangle) = |\beta'\rangle$ and $|\beta\rangle \neq |\beta'\rangle$. Suppose also that there exists a solution z which is an extension of s and for which $z(i) = |\delta\rangle$. Then by the repeated use of Lemma 4.2, and also by using Lemma 4.3 when $|\delta\rangle$ is a 2-qubit entangled state, one concludes that $z(k)$ is simultaneously equal to $|\beta\rangle$ and to $|\beta'\rangle$, which is a contradiction.

Finally Statement 3 follows since every step of the procedure can be naturally charged to an edge in $E_s \setminus E_{s'}$, and every edge is charge only a constant number of times. □

4.4 The main algorithm

Having discussed the precise nature of the input and output of the problem the goal of the Q2SAT problem can be formally stated as

> Q2SAT
>
> **Input:** The constraint graph $G(H)$ of a 2-local Hamiltonian H, given in the adjacency list representation.
>
> **Output:** A product state solution if H is frustration free, or "H is unsatisfiable" if it is not.

4.4.1 Algorithm Sketch

An overview of the algorithm can be obtained by understanding the Q2SATSolver procedure. The algorithm can be divided into four phases, each made up of multiple stages and each stage

can be seen as corresponding to a single Propagation sub-routine. The algorithm also identifies an unsatisfiable instance and stops at some point in its execution. This could either happen when the rank-3 constraints in the instance cannot be satisfied or it is found that a contradiction cannot be avoided. Here, contradiction refers to a single qubit being simultaneously assigned multiple non-empty values through the course of the algorithm. The global variables used are two assignments s_0 and s_1 both initialized to $s_\ominus$ and two corresponding constraint graphs G_0 and G_1, in the adjacency list representation both initialized to $G(H)$.

While running on a frustration free instance, the following invariants are maintained at the end of each stage: $s_0 = s_1$ and $G_0 = G_1 = G_{s_0}$. In the first two phases, modifications are only made to s_0 and G_0 and are synchronized with s_1 and G_1 respectively. In the latter two stages, s_0 and s_1 (along with the corresponding graphs) develop independently. However, at the end of each stage, the state of only one of the two i.e. (s_0, G_0) or (s_1, G_1) is retained and synchronized with the other. This reflects the notion of parallel propagation suggested in the improvements for the Davis Putnam procedure and is essential for complexity considerations ensuring that the useless work done is proportional to the useful work.

The first thing to do is to deal with maximal rank or rank-3 constraints since there is only one assignment to satisfy them which is what the first phase MaxRankRemoval does. The second phase checks if these assignments already lead to any contradictions by propagation them. If not, one is left with a closed coherent assignment s and a reduced instance H_s made up of just rank-1 constraints. The next phase tries to satisfy product constraints and propagate them. There are two possible ways to satisfy a product state and each of these choices is efficiently tried in ParallelPropagation. This leaves a constraint graph containing only rank-1 entangled constraints and these are dealt with one by one. Trying an arbitrary value on a qubit and propagating it would reveal a contradiction if any. However, this also provides a simple way to find a product state implied by the entangled constraints. This is used as in the previous phase to satisfy and propagate assignments. A successful run of the solver returns a satisfying assignment and an empty constraint graph. A crucial reason for the effectiveness of this algorithm is the fact that once an edge is checked for a potential successful propagation, then irrespective of whether propagation actually happens across the edge or not, it can be removed from the instance without changing its satisfiability.

Theorem 4.2. *Let $G(H) = (V, E)$ be the constraint graph of a 2-local Hamiltonian. Then:*

1. *If H is frustration-free, the algorithm* Q2SATSolver$(G(H))$ *outputs a ground state $|s\rangle$.*
2. *If H is not frustration-free, the algorithm* Q2SATSolver$(G(H))$ *outputs "H is unsatisfiable".*
3. *The running time of the algorithm is $O(|V| + |E|)$.*

To prove Theorem 4.2, the subroutines used are discussed before the main solver analyzed.

Algorithm 4.2: Q2SATSolver($G(H)$)

$s_0 = s_1 := \ominus$, $G_0 = G_1 := G(H)$ ▷ Initialize global variables
`MaxRankRemoval()` ▷ Phase 1: Remove maximal rank constraints
while $\exists\, i \in V_0$ such that $s(i) \neq \ominus$ **do**
    `Propagation` $(s_0, G_0, i, s_0(i))$ ▷ Phase 2: Propagate all assigned values
    if the propagation returns "unsuccessful" **then output** "H is unsatisfiable"
    $s_1 := s_0, G_1 := G_0$
while $\exists$ a product constraint $\Pi_{i_0 i_1} = |\alpha_0^\perp\rangle\langle\alpha_0^\perp|_{i_0} \otimes |\alpha_1^\perp\rangle\langle\alpha_1^\perp|_{i_1}$ **do**
    `ParallelPropagation`$(i_0, |\alpha_0\rangle, i_1, |\alpha_1\rangle)$ ▷ Phase 3: Remove product constraints
while G_0 is not empty **do**
    `ProbePropagation(`i`)` for some vertex i ▷ Phase 4: Remove entangled constraints
output $|s\rangle$ for any total extension s of s_0.

4.4.2 Max rank removal

As every maximal rank constraint has a unique solution (up to a global phase), MaxRankRemoval uses this assignment for each of these constraints and checks for globally consistency.

Algorithm 4.3: MaxRankRemoval()

foreach $i \in V_0$ such that `rank`$(\Pi_{ii}) = 1$ and $|\phi\rangle$ uniquely satisfies Π_{ii} **do**
    $s_0(i) := |\phi\rangle$
foreach $i \in V_0$, $\forall\, e \in E_0$ from i to j such that `rank`$(\Pi_e) = 3$ **do**
    let $|\gamma\rangle$ be the unique state satisfying Π_e
    if $|\gamma\rangle = |\alpha\rangle_i|\beta\rangle_j$ is a product state **then**
        if $s_0(i) \notin \{\ominus, |\alpha\rangle\}$ **then output** "H is unsatisfiable"
        if $s_0(i) = \ominus$ **then** $s_0(i) := |\alpha\rangle$
    else if $|\gamma\rangle$ an entangled state **then**
        if $s_0(i) \notin \{\ominus, |\gamma\rangle\}$ **then output** "H is unsatisfiable"
        if $s_0(i) = \ominus$ **then** $s_0(i) := |\gamma\rangle$
remove from E_0 every edge e such that Π_e is satisfied by s_0.
remove every isolated vertex from G_0
$s_1 := s_0$, $G_1 := G_0$

Lemma 4.5. *Let s_0, G_0, s_1, G_1 be the outcome of* MaxRankRemoval*. It holds that:*

1. *If* MaxRankRemoval *does not output "H is unsatisfiable" then s_0 satisfies every maximal rank constraint, $G_0 = G(H_{s_0})$ and $s_0 = s_1, G_0 = G_1$. Moreover, if H is satisfiable then s_0*

is a pre-solution.

2. *If* MaxRankRemoval *outputs "H is unsatisfiable" then H is unsatisfiable.*
3. *The complexity of the procedure is $O(|V| + |E|)|$.*

Proof. If the procedure doesn't output "H is unsatisfiable" then indeed s_0 satisfies all maximal rank constraints. The removal of the necessary edges and vertices insures that $G_0 = G(H_{s_0})$, and obviously $s_0 = s_1, G_0 = G_1$. If H is satisfiable, then it has a ground state for some total assignment s. This s is an extension of s_0 because there is a unique way to satisfy the maximal rank constraints.

Maximal rank projectors are such that there is a unique assignment for their qubits which satisfies them. The first part of the procedure creates the assignment which assigns these necessary values. If this assignments is not coherent then H is unsatisfiable. Similarly, if s_0 assigns an entangled 2-qubit state between variables i and k, and there is an entangled rank-1 constraint between i and j, then by Lemma 4.3 it is impossible to extend s_0 into a satisfying assignment, and therefore H is unsatisfiable. This proves the second statement.

The procedure can be executed by a constant number of vertex and edge traversals for s_0, and similarly for s_1 thereby satisfying the complexity of the third statement. □

4.4.3 Parallel Propagation

The procedure ParallelPropagation is called when s_0 is a closed assignment, G_{s_0} contains only rank-1 constraints and there also exists an edge with a product constraint on some qubits (i, j) as $\Pi_{ij} = |\alpha^\perp\rangle\langle\alpha^\perp|_i \otimes |\beta^\perp\rangle\langle\beta^\perp|_j$. The possible satisfying satisfying assignments for this constraint either assign $|\alpha\rangle$ to i or $|\beta\rangle$ to j. These are tried in parallel where one copy progresses with (s_0, G_{s_0}) and the other with (s_1, G_{s_1}). The propagation proceeds in parallel by traversing one edge in each copy at a time. The propagation that is the first to terminate successfully determines the progress made as illustrated in Figure (4.4)

Algorithm 4.4: ParallelPropagation$(i_0, |\alpha_0\rangle, i_1, |\alpha_1\rangle)$

```
1  Run in parallel Propagation (s_0, i_0, |α_0⟩) and Propagation (s_1, i_1, |α_1⟩)
2  until one of them terminates successfully or both terminate unsuccessfully
3  if both propagations terminate unsuccessfully then output "H is unsatisfiable"
4  else
5      let Propagation (s_0, G_0, i_0, |α_0⟩) terminate first   ▷ The other case is symmetric
6      undo Propagation(s_1, G_1, i_1, |α_1⟩)
7      s_1 := s_0,  G_1 := G_0
```

Lemma 4.6. *Let* ParallelPropagation *be called when s_0 is closed, H_{s_0} doesn't have rank-3 constraints, $G_0 = G_{s_0}$, there exists a product edge in G_0 from i_0 to i_1 with $\Pi_{i_0 i_1} = |\alpha_0^\perp\rangle\langle\alpha_0^\perp|_{i_0} \otimes$*

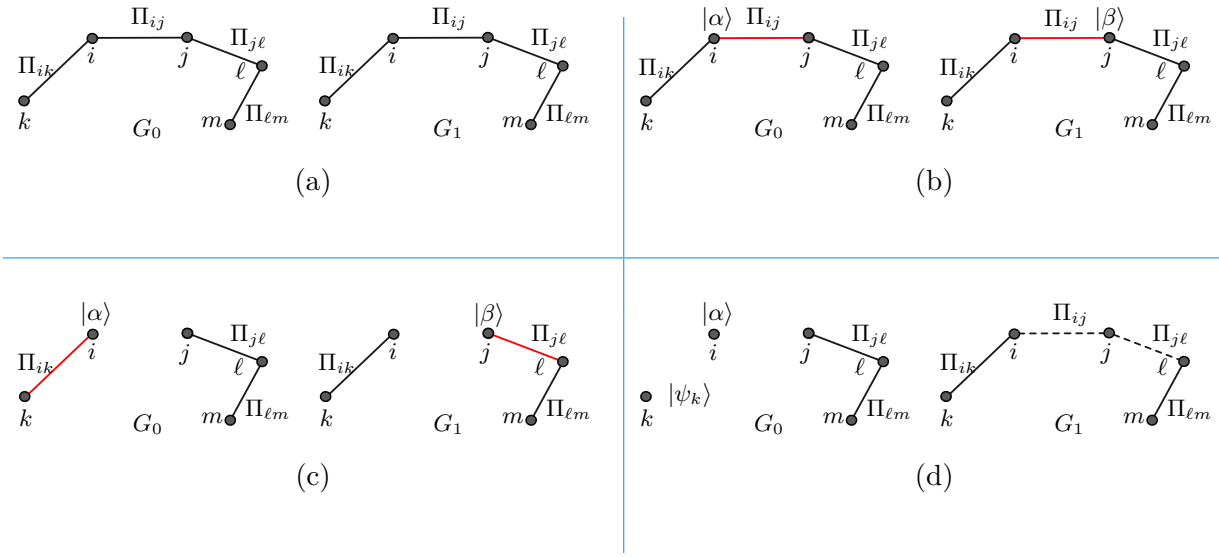


Figure 4.4: ***Parallel Propagation.*** (a) Copy G_0 into G_1; (b) Assign $s_0(i) = |\alpha\rangle$ and $s_1(j) = |\beta\rangle$; (c) Propagate across 1 edge in G_0 and G_1; (d) G_0's propagation terminates successfully and G_1's propagation is undone.

$|\alpha_1^\perp\rangle\langle\alpha_1^\perp|_{i_1}$, *and* $s_1 = s_0, G_1 = G_0$. *Let* s_0', s_1', G_0', G_1' *be the outcome of the procedure. The following hold true:*

1. *If* ParallelPropagation *does not output "H is unsatisfiable" then* s_0' *is a proper closed extension of* s_0, $G_0' = G_{s_0'}$, *and* $s_1' = s_0'$, $G_1' = G_0'$. *Moreover, if* s *is a pre-solution then* s_0' *is a pre-solution.*
2. *If* ParallelPropagation *outputs "H is unsatisfiable" then H is unsatisfiable.*
3. *The complexity of the procedure is* $O(|E_{s_0}| - |E_{s_0'}|)$.

Proof. If the procedure does not output "H is unsatisfiable", then there was at least one propagation that terminated successfully, say $\mathsf{Propagation}(s_0, G_0, i_0, |\alpha_0\rangle)$. Then, s_0' is a proper extension of s_0 since s_0 was a closed assignment with $s(i_0) = \ominus$. All other claims of the first statement are straightforward and follow from Lemma 4.4.

As H_{s_0} does not contain any rank-3 constraints, its ground state is will be a product of 1-qubit states in case it is satisfiable. Since H_{s_0} contains a product state constraint $\Pi_{i_0 i_1} = |\psi\rangle\langle\psi|$ where $|\psi\rangle = |\alpha_0^\perp\rangle_{i_0} \otimes |\alpha_1^\perp\rangle_{i_1}$, the two possible product state assignments that would satisfy $\Pi_{i_0 i_1}$ would either have $|\alpha_0\rangle$ assigned to i_0 or $|\alpha_1\rangle$ to i_1. With both of the propagations failing, Lemma 4.4 implies that there is no way to satisfy $\Pi_{i_0 i_1}$ and by translation, H_{s_0} and H. Hence, H is not frustration free.

For the complexity analysis, in the case of a successful propagation run, the unterminated or unsuccessful run performs at most the same amount of work as the successful run. Undoing the propagation can be done in time proportional to the number of edges along which propagation has occurred so far possibly by tracking removed edges in temporary lists. The complexity of a successful propagation run follows from Lemma 4.4. □

4.4.4 Probe Propagation

The most interesting case of the algorithm happens when the instance has only rank-1 entangled constraints. For one thing, this is not a case that occurs classically in any form and so requires some inherently quantum arguments to solve. For any form of propagation to occur, one needs an initial state. One way to proceed is to start with an arbitrary state (say, $|0\rangle$) on an unassigned qubit i. The next obvious step is to propagate this assignment to see if it terminates successfully. If so, another unassigned qubit can be picked and the same process repeated, If not, a contradiction arises: $\exists j \in V_s$ such that two propagating paths reaching j assign different values to it. To deal with this contradiction, observations of the "Sliding Lemma" which initially appeared in Ji, Wei, Zeng [JWZ11] are essential. If the propagating path containing rank-1 entangled constraints is of the form $i \Rightarrow i_1 \Rightarrow \ldots \Rightarrow j$, then, the Sliding Lemma states that the ground space of the Hamiltonian $\Pi_{ii_1} + \Pi_{i_1 i_2} + \ldots + \Pi_{i_{k-1} j}$ is equivalent to the ground space of the Hamiltonian $\Pi_{ij} + \Pi_{i_1 i_2} + \ldots + \Pi_{i_{k-1} j}$ where Π_{ij} is a new projector defined on qubits (i, j) that is replacing Π_{ii_1}. The reason it is called sliding arises from the fact that it can be graphically viewed as sliding the constraint $\Pi_{i_0 i_1}$ along the edges of the path $i_1 \Rightarrow \ldots \Rightarrow i_k$ as shown in Figure (4.5).

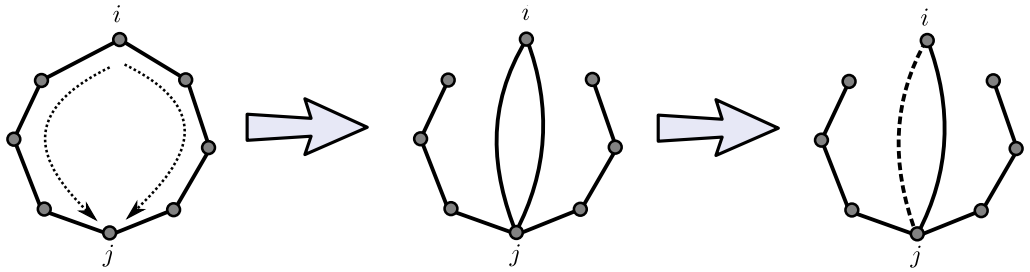


Figure 4.5: Handling a contradicting cycle: we slide edges that touch i along the two paths to j until we get a double edge with a 'tail'. We then use a structure lemma to deduce that at least one of these edges can be written as a product projector.

Now, if there are two paths such from i to j, then one ends up with 2 projectors between qubits (i, j). For a contradiction to arise, it is obvious that the two projectors are different. Proposition 4.2 then implies that this two dimensional space spanned by the two projectors contains a product state which can be used as the basis of a call to ParallelPropagation.

Lemma 4.7 (Sliding Lemma)**.** *Consider a system on* 3 *qubits* i, j *and* k*. Suppose that we have two rank-*1 *constraints* $\Pi_1 = |\psi_1\rangle\langle\psi_1|_{ij}$ *on qubits* (i, j) *and* $\Pi_2 = |\psi_2\rangle\langle\psi_2|_{jk}$ *on qubits* (j, k)*. If* $|\psi_2\rangle$ *is entangled, then there is another rank-*1 *constraint* $\Pi_3 = |\psi_3\rangle\langle\psi_3|_{ik}$ *on qubits* (i, k) *such that the ground space of* $\Pi_1 + \Pi_2$ *is identical to the ground space of* $\Pi_2 + \Pi_3$*. In addition, if a single qubit state* $|\alpha\rangle_i$ *is propagated by* $\Pi_1 + \Pi_2$ *to* $|\beta\rangle_k$*, then it is also propagated to* $|\beta\rangle_k$ *directly via* Π_3*.*

Proof. Expand $|\psi_2\rangle_{jk}$ in terms of the standard basis on k as $|\psi_2\rangle_{jk} = |x\rangle_j|0\rangle_k + |y\rangle_j|1\rangle_k$, where the states $|x\rangle$ and $|y\rangle$ are neither necessarily normalized nor orthogonal but are linearly independent of each other as $|\psi_2\rangle$ is entangled. Consequently, using a Gaussian elimination, one can find

a unique non-singular transformation T on qubit j such that $T|x\rangle = |1\rangle$ and $T|y\rangle = -|0\rangle$. Then, $T|\psi_2\rangle_{jk} = |1\rangle_j|0\rangle_k - |0\rangle_j|1\rangle_k$ is the anti-symmetric state. Let $|\tilde{\psi}_1\rangle_{ij} = T|\psi_1\rangle_{ij}$ and $|\tilde{\psi}_2\rangle_{jk} = T|\psi_2\rangle_{jk}$ respectively and use them to define the rank-1 projectors $\tilde{\Pi}_1 = |\tilde{\psi}_1\rangle\langle\tilde{\psi}_1|_{ij}$, $\tilde{\Pi}_2 = |\tilde{\psi}_2\rangle\langle\tilde{\psi}_2|_{jk}$. As $\tilde{\Pi}_2$ projects into the anti-symmetric subspace, any state in the ground space of $\tilde{\Pi}_1 + \tilde{\Pi}_2$ must be invariant under a swapping of qubits j, k. Therefore, defining $|\psi_3\rangle_{ik} = |\tilde{\psi}_1\rangle_{ik}$, and $\Pi_3 = |\psi_3\rangle\langle\psi_3|_{ik}$, the ground space of $\tilde{\Pi}_1 + \tilde{\Pi}_2$ is identical to the ground space of $\Pi_3 + \tilde{\Pi}_2$. Now, applying the inverse transformation T^{-1} on qubit j, the projector $\tilde{\Pi}_2$ returns to Π_2, while Π_3 remains unchanged. Since both T, T^{-1} are non-singular, it follows that ground space of $\Pi_1 + \Pi_2$ is identical to the ground space to $\Pi_2 + \Pi_3$.

For the second claim, assume by way of contradiction that Π_3 does not propagate $|\alpha\rangle_i$ to $|\beta\rangle_k$. Then there is a 1-qubit state $|\gamma\rangle \neq |\beta\rangle$, such that $\Pi_3(|\alpha\rangle_i|\gamma\rangle_k) = 0$. Since Π_2 is a rank-1 entangled projector, by Lemma 4.1, it propagates $|\gamma\rangle_k$ to some state $|\delta\rangle_j$, and therefore the state $|\alpha\rangle_i|\delta\rangle_j|\gamma\rangle_k$ is a ground state of $\Pi_2 + \Pi_3$, as well as of $\Pi_1 + \Pi_2$. However, this contradicts the assumption that latter propagates $|\alpha\rangle_i$ to $|\beta\rangle_k$. □

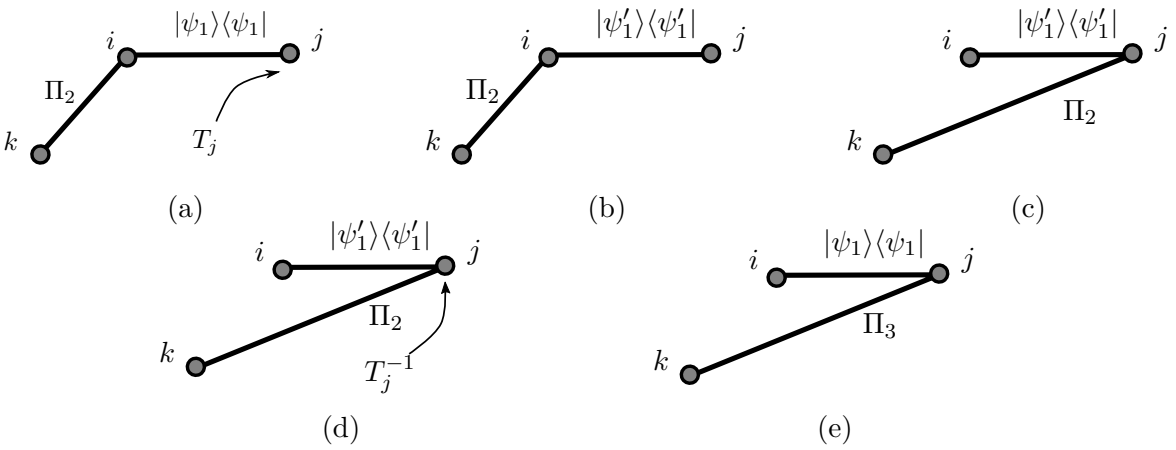


Figure 4.6: ***Sliding.*** (a) Apply the non-singular transformation T_j to qubit j; (b) $|\psi_1'\rangle = T_j|\psi_1\rangle = |y_1\rangle|y_2\rangle - |y_2\rangle|y_1\rangle$ for $|y_1\rangle \neq |y_2\rangle$; (c) Slide Π_2 along $|\psi_1'\rangle\langle\psi_1'|$ as it projects onto the symmetric subspace; (d) Apply the inverse transformation to qubit j; (e) Constraint Π_{ik} changes to Π_3 on applying the transformation.

Applying Lemma 4.7 iteratively along a path $p := i_0 \to i_1 \to \ldots \to i_k$ made of entangled rank-1 projectors $\Pi_{i_0i_1}, \ldots, \Pi_{i_{k-1}i_k}$, it is possible to transform $\Pi_{i_0i_1}$ to the projector $\Pi_{i_0i_2}$ and so on till $\Pi_{i_0i_k}$ is obtained as illustrated in Figure (4.5). Let the underlying 2-qubit state in $\Pi_{i_0i_k}$ be denoted as $|\mathsf{slide}(p)\rangle$ thereby setting $\Pi_{i_0i_k} = |\mathsf{slide}(p)\rangle\langle\mathsf{slide}(p)|$. Then, the following corollary arises.

Corollary 4.1. *Given a path $p := i_0 \to i_1 \to \ldots \to i_k$ of entangled rank-1 projectors $\Pi_{i_0i_1}, \ldots, \Pi_{i_{k-1}i_k}$, apply the sliding lemma iteratively to obtain $|\mathsf{slide}(p)\rangle_{i_0i_k}$. Then the ground space of $\Pi_{i_0i_1} + \Pi_{i_1i_2} + \ldots + \Pi_{i_{k-1}i_k}$ is equal to the ground space of $\Pi_{i_1i_2} + \ldots + \Pi_{i_{k-1}i_k} + |\mathsf{slide}(p)\rangle\langle\mathsf{slide}(p)|$. Moreover, if $|\alpha\rangle_{i_0}$ is propagated to $|\beta\rangle_{i_k}$ along p, then it is also propagated directly by $|\mathsf{slide}(p)\rangle\langle\mathsf{slide}(p)|$.*

The ProbePropagation procedure can now be stated and analyzed.

Algorithm 4.5: ProbePropagation(i)

```
1  Propagation (s_0, G_0, i, |0⟩)
2  if the propagation is successful then  s_1 := s_0, G_1 := G_0
3  else
4      Pick a j such that |s_0(j) > 1|
5      find two paths p_1, p_2 in G_0 from i to j such that prop(s_0, p_1, |0⟩) ≠ prop(s_0, p_2, |0⟩)
6      find a state |α^⊥⟩_i ⊗ |β^⊥⟩_j in the 2-dimensional subspace: span{slide(p_1), slide(p_2)}
7      undo Propagation (s_0, G_0, i, |0⟩)
8      ParallelPropagation(i, |α⟩, j, |β⟩)
```

Lemma 4.8. *Let* ProbePropagation *be called when s_0 is closed, H_{s_0} has only rank-1 entangled constraints, $G_0 = G_{s_0}$, and $s_1 = s_0, G_1 = G_0$. Let s'_0, s'_1, G'_0, G'_1 be the outcome of the procedure. Then the following holds:*

1. *If* ProbePropagation *does not output "H is unsatisfiable" then s'_0 is a proper closed extension of s_0, $G'_0 = G_{s'_0}$, and $s'_1 = s'_0$, $G'_1 = G'_0$. Moreover, if s is a pre-solution then s'_0 is a pre-solution.*
2. *If* ParallelPropagation *outputs "H is unsatisfiable" then H is unsatisfiable.*
3. *The complexity of the procedure is $O(|E_{s_0}| - |E_{s'_0}|)$.*

Proof. If the procedure does not output "H is unsatisfiable" then either Propagation$(s_0, G_0, i, |0\rangle)$ or one of the parallel propagations (say Propagation$(s_0, G_0, i, |\alpha\rangle)$) terminates successfully. Then s'_0 is a proper extension of s_0 since s_0 is closed and therefore $s_0(i_0) = \ominus$. Obviously $s'_1 = s'_0$ and $G'_1 = G'_0$, and all other claims follow from the Propagation Lemma.

Suppose that all three propagations are unsuccessful. By Corollary 4.1, any solution for H_{s_0} also satisfies the system obtained after sliding the constraints from i along paths p_1 and p_2, with the new constraints $|\mathsf{slide}(p_1)\rangle\langle\mathsf{slide}(p_1)|_{ij}$ and $|\mathsf{slide}(p_2)\rangle\langle\mathsf{slide}(p_2)|_{ij}$. Using Proposition 4.2, as $|\alpha^\perp\rangle_i \otimes |\beta^\perp\rangle_j$ lies in $\mathsf{span}\{|\mathsf{slide}(p_1)\rangle, |\mathsf{slide}(p_2)\rangle\}$, any state orthogonal to the latter subspace will also be orthogonal to the former. Hence, any solution for H_{s_0} should also satisfy the product constraint $|\alpha^\perp\rangle\langle\alpha^\perp|_i \otimes |\beta^\perp\rangle\langle\beta^\perp|_j$. Then, Lemma 4.6 implies that H_{s_0}, and by extension H is unsatisfiable if the call to ParallelPropagation, made to satisfy $|\alpha^\perp\rangle\langle\alpha^\perp|_i \otimes |\beta^\perp\rangle\langle\beta^\perp|_j$, fails.

For the complexity analysis the interesting case is when the first propagation, that we call $\texttt{Prop}_{\text{failure}}$, is unsuccessful but one of the two parallel propagations is successful. Let's call the successful one $\texttt{Prop}_{\text{success}}$. The main observation here is that every propagating edge in $\texttt{Prop}_{\text{failure}}$ will also be propagating in $\texttt{Prop}_{\text{success}}$, since by Lemma 4.1 entangled edges always propagate. The paths p_1 and p_2 can be found in time proportional to the size of the subgraph visited by $\texttt{Prop}_{\text{failure}}$. Indeed, observe that the edges of the two paths, except the last edge of

one of the two, are edges in the BFS tree underlying $\texttt{Prop}_{\text{failure}}$. The way from a vertex to the root of the tree can be then found, for example, by maintaining for each vertex in the tree, a pointer towards its parent. The product state $|\alpha\rangle \otimes |\beta\rangle$ can be found in constant time by Proposition 4.2. Therefore, by Lemma 4.6, the complexity is indeed $O(|E_{s_0}| - |E_{s'_0}|)$. □

4.4.5 Analysis of the algorithm

Proof of Theorem 4.2. If H is frustration free then by Lemma 4.5 MaxRankRemoval outputs a pre-solution s_0 that satisfies every maximal rank constraint. By the Propagation Lemma, at the end of Phase 2, s_0 is additionally a closed solution. By Lemma 4.6 ParallelPropagation outputs s_0 such that in addition in H_s there are only entangled constraints. By Lemma 4.8 at the end of the algorithm in addition H_s is empty, and therefore s is a solution.

If the algorithm doesn't output "H is unsatisfiable" then by Lemma 4.5, by the Propagation Lemma, and by Lemmas 4.6 and 4.8 it outputs a coherent assignment s such that G_s is the empty graph, and therefore s is a solution.

The complexity of MaxRankRemoval by Lemma 4.5 is $O(|E|)$. After the second phase, the propagation of the assigned values during MaxRankRemoval, the copying of s_0 and G_0 into s_1 and G_1 respectively can be done by executing the same propagation steps this time with s_1 and G_1. The complexity of the rest of the algorithm by the Propagation Lemma, and Lemmas 4.6 and 4.8 is a telescopic sum which sums up to also $O(|E|)$. Observe that the constants hidden in the big-O notation in the terms of this sum are all the same, they are equal to the absolute constant in the complexity analysis of the algorithm Propagation referenced in the Propagation Lemma. Therefore the telescopic sum evaluates to $O(|E|)$, and the overall complexity of the algorithm is $O(|E|)$. □

4.5 Bit Complexity of Q2SATSolver

As mentioned at the beginning of this chapter, the algorithm Q2SATSolver has been analyzed in the algebraic model of computation, where arithmetic operations on complex numbers are assumed to consume unit time. This was done mainly in order to simplify the presentation. However, in order to assess the running time of the algorithm in any realistic scenario, we must also take into account the actual cost of the arithmetic operations. This is not a completely trivial task: one hand, the continuous nature of a Q2SAT Hamiltonian implies that it should be represented using complex numbers with an exponentially high accuracy. But on the other hand, we also want the representation to be as efficient as possible, in order not to degrade the running time of the algorithm too much. One way to approach this problem is to consider the input in the framework of *bounded algebraic numbers* and analyze the complexity of the algorithm in this setting. Essentially, this would require calculating the bit-wise cost of representing the input in this framework and performing the arithmetic operations of the algorithm on it. In other words,

we need to calculate the *bit complexity* of the algorithm. The techniques used in this section are based on standard methods in algebraic computational complexity, and further details can be found in [Coh93].

Representing the input and the output

As we would like to work in the framework of algebraic numbers, it helps to first list the kinds of number fields that will be used to represent the input and output. The simplest field to consider is that of rational numbers, $\mathbb{Q}$. To bound the size of the entries from $\mathbb{Q}$, a *$k_{\mathbb{Q}}$-number*, for some integer $k > 0$, is defined below.

Definition 4.6 ($k_{\mathbb{Q}}$-number)**.** *A rational number u is a $k_{\mathbb{Q}}$-number if there exists integers u_1 and u_2 of at most k-bits such that $u = \frac{u_1}{u_2}$.*

A $k_{\mathbb{Q}}$-number $\frac{u_1}{u_2}$ will be represented as the tuple (u_1, u_2) requiring $2k$ bits. As quantum projectors and states use complex entries, we also require numbers from $\mathbb{Q}(\mathrm{i})$, the extension field of $\mathbb{Q}$ with $\mathrm{i} = \sqrt{-1}$. This leads to the following definition of a *$k_{\mathbb{Q}(\mathrm{i})}$-number*:

Definition 4.7 ($k_{\mathbb{Q}(\mathrm{i})}$-number)**.** *A complex number $a + b\mathrm{i} \in \mathbb{Q}(\mathrm{i})$ is a $k_{\mathbb{Q}(\mathrm{i})}$-number if its coefficients a and b are $k_{\mathbb{Q}}$-numbers.*

A $k_{\mathbb{Q}(\mathrm{i})}$-number $a + b\mathrm{i}$ will be represented as the tuple (a, b) for $k_{\mathbb{Q}}$-numbers a and b, thereby requiring $4k$ bits in total. When the square root, $\sqrt{r}$, of a square-free integer t (that is, an integer with a non-integer square root) is generated during the course of the algorithm, we shall consider the field extension $\mathbb{Q}(\mathrm{i}, \sqrt{t})$ of $\mathbb{Q}(\mathrm{i})$ and the corresponding *$k_{\mathbb{Q}(\mathrm{i}, \sqrt{t})}$-numbers*, defined below.

Definition 4.8 ($k_{\mathbb{Q}(\mathrm{i}, \sqrt{t})}$-number)**.** *The number $a_1 + a_2\sqrt{t} + a_3\mathrm{i} + a_4\sqrt{t}\mathrm{i} \in \mathbb{Q}(\mathrm{i}, \sqrt{t})$ is a $k_{\mathbb{Q}(\mathrm{i}, \sqrt{t})}$-number if its coefficients, a_1, a_2, a_3 and a_4, are $k_{\mathbb{Q}}$-numbers.*

Clearly, when the integer t can be represented as a k'-bit integer, a $k_{\mathbb{Q}(\mathrm{i}, \sqrt{t})}$-number can be represented using $8k + k'$ bits by the tuple (a_1, a_2, a_3, a_4, t) for $k_{\mathbb{Q}}$-numbers a_1, a_2, a_3, a_4.

The input Hamiltonian is a set of m different 2-qubit projectors whose non-trivial parts are 4×4 complex matrices. As mentioned in Section 3.3.1, for a 2-qubit projector $\Pi = \hat{\Pi} \otimes \mathbb{I}_{\text{rest}}$, the input specifies the non-trivial part $\hat{\Pi}$ which is uniquely determined by its image subspace, $\mathsf{img}(\hat{\Pi})$. We consider each projector is given as a *$k_{\mathbb{Q}(\mathrm{i})}$-projector* which is defined below for some constant $k > 0$.

Definition 4.9 ($k_{\mathbb{Q}(\mathrm{i})}$-projector)**.** *Consider a 2-qubit projector $\hat{\Pi}$ of rank-r to be specified by r independent, but not necessarily orthogonal, 4×1 vectors $\{v_1, \ldots, v_r\}$ such that $\mathsf{img}(\hat{\Pi}) = \mathsf{span}\{v_1, \ldots, v_r\}$. $\hat{\Pi}$ is a $k_{\mathbb{Q}(\mathrm{i})}$-projector if $v_1, v_2, \ldots, v_r$ are given in the standard basis with $k_{\mathbb{Q}(\mathrm{i})}$-number coefficients.*

Then, the Hamiltonian can be represented by m different $k_{\mathbb{Q}(\mathrm{i})}$-projectors, leading to a total space consumption of at most $m \times 4 \times 4 \times 1 \times 4k = 64mk$ bits. The ground state output will be a tensor product of single qubit and 2-qubit states which are length 2 and 4 complex vectors, respectively. Each entry of these vectors could belong to $\mathbb{Q}, \mathbb{Q}(\mathrm{i})$ or $\mathbb{Q}(\mathrm{i}, \sqrt{t})$, for different square

free integers t. We will show below that there can be at most n different integers t, and that the state assigned to each variable consumes $O(n)$ bits. Recall from Section 4.3.3 that the algorithm proceeds by assigning un-normalized states without affecting its accuracy. We also suppose that the basis vectors describing the image subspace of an input projector need not be normalized.

Cost of arithmetic operations

A useful fact on the bit complexity of basic arithmetic operations that will be used recurrently is stated below. Let $M(k, k')$ be the time required to multiply a k-bit integer with a k'-bit integer and let $M(k) = M(k, k)$. The currently known most efficient algorithm for integer multiplication uses Fourier transforms bounding $M(k) = k \log k 8^{\Theta(\log^* k)}$ [HvdHL16] where $\log^* k = \min\{w \in \mathbb{N} : \log \log \overset{w \text{ times}}{\ldots} \log k \leq 1\}$.

Fact 4.1. *Let a be a $k_{\mathbb{Q}}$-number and b a $k'_{\mathbb{Q}}$-number. Adding, subtracting, multiplying or dividing a and b can be performed in time $O(M(k, k'))$, and the result is an $O(k + k')_{\mathbb{Q}}$-number.*

Now we consider the bit complexity incurred during the course of the Q2SATSolver algorithm.

Theorem 4.3. *Let H be a* Q2SAT *Hamiltonian on n qubits consisting of m projectors with complex entries, each given as two $k_{\mathbb{Q}(\mathrm{i})}$-numbers, for some constant $k > 0$. Then the bit complexity of* Q2SATSolver(H) *is $O((n + m)\ M(n))$.*

Proof. The straightforward approach is to calculate the bit complexity of each operation that manipulates the projectors and assignments. In the first phase of MaxRankRemoval, the unique state satisfying each 2-qubit projector of rank-3 can be found using Gaussian elimination which for an $O(1)$-sized matrix is equivalent to a constant number of multiplications. Using Fact 4.1, the 2-qubit state found will hence be an $O(k)_{\mathbb{Q}(\mathrm{i})}$-number.

The second and third phases of the algorithm repeatedly propagate values across rank-1 constraints. The bit complexity of propagation is now analyzed. Given a product constraint $|\alpha\rangle\langle\alpha| \otimes |\beta\rangle\langle\beta|$, it requires finding $|\alpha^{\perp}\rangle$ and $|\beta^{\perp}\rangle$ using complex conjugates. Given an entangled constraint and a state assigned to one end of the constraint, the propagated state can be found by solving a system of linear equations. Assuming the initial state and the projector use $k_{\mathbb{Q}(\mathrm{i})}$-numbers, the propagated state will be represented using $2k_{\mathbb{Q}(\mathrm{i})}$-numbers. To propagate ℓ steps, each step using $k_{\mathbb{Q}(\mathrm{i})}$-number projectors, starting with a $k'_{\mathbb{Q}(\mathrm{i})}$-number state, the final propagated state will use $(k' + \ell k)_{\mathbb{Q}(\mathrm{i})}$-numbers. Hence, an $O(n)$ step propagation will result in a final state represented with $O(n)_{\mathbb{Q}(\mathrm{i})}$-numbers when $k = O(1)$ and $k' = O(n)$. This is linear in the number of qubits and takes at most $O(n)M(n, k)$ time. This covers the second and third phases of the algorithm.

The last phase with ProbePropagation will deal with one or more disconnected components, each made of only entangled constraints. The complexity of this phase is of interest when a contradiction arises during propagation in a component and a satisfying assignment has to be found by sliding constraints and using Proposition 4.2. Consider the way sliding across a constraint is done as described in Lemma 4.7. Finding the transformation and its inverse both

require a constant number of multiplications and the new constraint after sliding will be given using $(ck)_{\mathbb{Q}(\mathrm{i})}$-numbers for some constant c. Sliding along $O(n)$ constraints will finally result in constraints represented by $O(n)_{\mathbb{Q}(\mathrm{i})}$-numbers and consumes $O(n)M(n,k)$ time.

Computing the product constraint in the resulting 2-dimensional subspace requires computing an eigenvector as per Proposition 4.2 leading to the possibility of an irrational square root $\sqrt{r}$ being introduced into the representation, for some square-free integer r. This highlights the necessity of moving to the field extension $\mathbb{Q}(\mathrm{i},\sqrt{r})$. Assuming the two parallel rank-1 projectors were initially given using $k'_{\mathbb{Q}(\mathrm{i})}$-numbers, the product constraint will require $O(k')_{\mathbb{Q}(\mathrm{i},\sqrt{r})}$-numbers to describe it. Also, $\sqrt{r}$ is generated as the irrational part of the eigenvalue of a 2×2 matrix whose entries are $O(k')_{\mathbb{Q}(\mathrm{i})}$-numbers due to which t will be an $O(k')$-bit integer. Hence, the representation of each $O(k')_{\mathbb{Q}(\mathrm{i},\sqrt{t})}$-number will require $O(k')$ bits. Any propagation after this point does not introduce new irrational square roots. The component, if satisfiable after ℓ propagation steps, will have assignments given by $O(k'+\ell k)_{\mathbb{Q}(\mathrm{i},\sqrt{t})}$-numbers. An $O(n)$ step propagation in this setting will result in this component using $O(n)_{\mathbb{Q}(\mathrm{i},\sqrt{t})}$-numbers for the assignment with the operations bounded by $O(nM(n,n)) = O(nM(n))$ time when $k' = O(nk) = O(n)$. Note that each disconnected component at the beginning of this phase may introduce different irrational square roots into their partial assignments.

Recall that since propagation is carried forward with unnormalised states, this is the only phase that may introduce square roots in the final assignment. The final output keeps all of these separate representations and does not homogenize them into a single representation. To conclude, each propagating, sliding or eigenvector finding step adds an overhead of at most $M(n,k) < M(n)$ where bit complexity is concerned. Using Theorem 4.2, the bit complexity of Q2SATSolver is therefore $O((n+m)M(n))$. □

Remark 4.1. *The above explanation started by assuming the base representation field as that of the Gaussian rationals, $\mathbb{Q}(\mathrm{i})$, and extends to the appropriate $\mathbb{Q}(\mathrm{i},\sqrt{r})$ field for some square-free integer r when required. It is also possible to consider any algebraic number field as the base field and extend it appropriately using techniques from computational algebraic number theory* [Coh93]. *Of course, any overheads in performing arithmetic operations and calculating field extensions will have to be accounted for accordingly.*

4.6 On approximate Quantum 2SAT

This section deals with previously unpublished results from ongoing work [ASSZ17]. The linear time algorithm for Q2SAT matches the linear time behaviour of classical 2SAT. This prompts one to wonder if there exist other parallels in the behaviour of the quantum and classical versions of 2SAT, say for instances which are *almost satisfiable*. Of course, the first step here is to define a suitable notion of what entails almost satisfiable instances in either the classical or quantum regime. Informally speaking, a natural transition to almost satisfiable instances in the classical case could come from unsatisfiable 2SAT formulas where removing at least a small fraction, say

some $\epsilon > 0$, of the clauses would make the resulting formula a satisfiable 2SAT formula. Such a formula would be an ϵ-satisfiable formula. In fact, this is precisely a MAX2SAT formula where any assignment satisfies at most a $1 - \epsilon$ fraction of clauses.

Given the discreteness of Boolean clauses, it makes sense to denote distortions to a satisfiable 2SAT formula by the addition of clauses. However, with the continuous nature of Q2SAT, an alternative idea for distortion exists. It is possible to consider instances where each projector of a satisfiable Q2SAT instance is modified to a projector which is at most ϵ away in operator norm distance from the original. Depending on the distance ϵ this would produce *almost frustration free* Q2SAT instances as defined below.

Definition 4.10 (Almost frustration free Q2SAT)**.** *An n-qubit* Q2SAT *instance $H = \sum_j \Pi_{(j)}$ with m local projectors is* ϵ-close *to an n-qubit* Q2SAT *Hamiltonian $H' = \sum_j \Pi'_{(j)}$ with m local terms if, for each j, $\Pi_{(j)}$ and $\Pi'_{(j)}$ act on the same pairs of qubits and $\|\Pi_{(j)} - \Pi'_{(j)}\| \leq \epsilon$. When H' is a frustration free n-qubit* Q2SAT *Hamiltonian, H is also said to $(1-\epsilon)$-satisfiable. Additionally, if there is a polynomial p such that $\epsilon \leq \frac{1}{p(n)}$, H is said to be* almost frustration free*.*

Of course, it may happen that $(1 - \epsilon)$-satisfiable H is *frustrated* (i.e., with non-zero ground energy). Then, finding a ground state for H could be QMA-hard [BV05, CM16b] (i.e., as hard as the general 2-local Hamiltonian case). Instead, consider the following restricted case of the 2-local Hamiltonian problem where the goal is to find a tensor product state having low energy.

Definition 4.11 (Approximate Q2SAT ($\mathsf{Approx} - \mathsf{Q2SAT}$))**.** *Given an n-qubit* Q2SAT *instance $H = \sum_j \Pi_{(j)}$ and two thresholds a and b such that $b - a > \frac{1}{\mathrm{poly}(n)}$, distinguish between the cases*

- *(*YES *instance) $\exists$ an n qubit state $|\psi\rangle$ which is a tensor product of 1-qubit and 2-qubit states such that $\langle\psi|H|\psi\rangle \leq a$*
- *(*NO *instance) $\forall$ n-qubit states as described above, $\langle\psi|H|\psi\rangle \geq b$*

Consider a $(1 - \epsilon)$-satisfiable H which is close to a frustration free H'. The ground state for H' would violate each term of H by at most ϵ. Summing this violation over the m terms of H' and setting $a = m\epsilon$, it can be concluded that the ground state for H' is a suitable witness for the YES case of the $\mathsf{Approx} - \mathsf{Q2SAT}$ problem. This highlights the reason to look for a tensor product of 1-qubit and 2-qubit states: every frustration free Q2SAT instance has a ground state exhibiting this structure. Additionally, the interest in almost frustration free Q2SAT instances, (i.e., $\epsilon = \frac{1}{\mathrm{poly}(n)}$) is for the following reason. For 2-local Hamiltonians, deciding if they are frustration free or not, seems to be lie in P. However, so far, the complexity of trying to determine the extent to which the system is frustrated, for general instances seems to be QMA. By trying to better understand almost frustration free instances, the aim is to throw some light on whether this shift in complexity from P to QMA is a sharp increase as the instance moves away from being frustration-free or if the increase in complexity is gradual.

The first step in understanding the $\mathsf{Approx} - \mathsf{Q2SAT}$ scenario is to question the hardness of the problem. The problem is definitely in NP. The low energy state, as a tensor product of 1-qubit and 2-qubit states, has an efficient (i.e, polynomial sized) classical description and will be a

suitable witness. The more pertinent question is looking for specific cases which may be NP-hard. As mentioned previously, the almost satisfiable version of 2SAT leads to the MAX2SAT problem which exhibits the following behaviour. Recall that for a Boolean formula φ, and an assignment $\mathbf{a}$, $\mathsf{unsat}(\varphi, \mathbf{a})$ is the number of clauses of φ that are not satisfied by $\mathbf{a}$.

Theorem 4.4 (Theorem 6 from [Kar01])**.** *For constant $c = 1.0005$, given an integer k and a* 2SAT *formula φ with m clauses on n variables it is* NP*-hard to distinguish between the two cases*

- *(*YES *instance) $\exists$ a Boolean assignment $\mathbf{a}$ on the n variables such that $\mathsf{unsat}(\varphi, \mathbf{a}) \leq k$*
- *(*NO *instance) $\forall$ Boolean assignments $\mathbf{a}$ on the n variables $\mathsf{unsat}(\varphi, \mathbf{a}) \geq c \cdot k$*

The constant $c = 1.0005$ is termed the inapproximability constant for MAX2SAT. Essentially, the MAX2SAT problem is known to be NP-hard, but the above result also shows that it is NP-hard to distinguish if certain instances have a minimal 1 or 2 violations. Drawing parallels between 2SAT and Q2SAT, a good starting point to understand the hardness of $\mathsf{Approx} - \mathsf{Q2SAT}$ would be to extend the above result about MAX2SAT, to the frustrated Q2SAT setting. In fact, the following result demonstrates that this behaviour persists in the quantum case as well even when restricted to product-state solutions. This is shown by reducing a MAX2SAT formula to a frustrated Q2SAT instance.

Theorem 4.5. *Let $c = 1.0005$ be the inapproximability constant for* MAX2SAT*. Then, for every constant $1 < c' \leq (c)^{\frac{1}{6}} < c$, given a* Q2SAT *Hamiltonian H on N qubits and a threshold $a = \frac{1}{\text{poly}(N)}$, it is* NP*-hard to distinguish between the following cases*[1]*:*

- *(*YES *instance) $\exists$ an N-qubit* product state $|\psi\rangle$*, such that $\langle\psi|H|\psi\rangle \leq a$*
- *(*NO *instance) $\forall$ N-qubit* product states $|\psi\rangle$*, $\langle\psi|H|\psi\rangle \geq c' \cdot a$*

Proof. The proof is a reduction from a MAX2SAT instance which is given by the formula φ on n variables and the integer k and it is broken up into the following steps. First, the formula ϕ on n variables is converted to a Q2SAT Hamiltonian H on $N = O(n)$ qubits. Also, using a parameter $\varepsilon = \frac{1}{\text{poly}(N)}$, k is converted to the threshold $a = k\frac{\varepsilon^4}{(1+\varepsilon^2)^2}$. Using this, the soundness and completeness of the reduction is proved.

Before delving into the description of the reduction, observe that it would be straightforward to map a 2SAT clause to a projector made up of $\{|0\rangle\langle 0|, |1\rangle\langle 1|\}$ and map a Boolean assignment to the states $\{|0\rangle, |1\rangle\}$. With the operator norm distance between these projectors (i.e., $\||0\rangle\langle 0| - |1\rangle\langle 1|\|$) being large (i.e., a constant), the violation of any Boolean state on these projectors is a constant (i.e., 0 or 1). However, the energy threshold, a, required for the N-qubit Hamiltonian is $\frac{1}{\text{poly}(N)}$. So, to shrink the energy to this bound, the classical projectors in the Hamiltonian and the states are *skewed*. That is, the classical projectors made up of $\{|0\rangle\langle 0|, |1\rangle\langle 1|\}$ are skewed to

[1]The requirement that $c' \leq (c)^{\frac{1}{6}}$ comes from the poof for the soundness of the reduction that transforms a MAX2SAT formula to a Hamiltonian.

$\{|0\rangle\langle 0|, |1_\varepsilon\rangle\langle 1_\varepsilon|\}$ where

$$|1_\varepsilon\rangle := \frac{1}{\sqrt{1+\varepsilon^2}}(|0\rangle + \varepsilon|1\rangle).$$

This skewing ensures that the classical projectors are always close to $|0\rangle\langle 0|$. Correspondingly, it would help to have the Boolean assignments mapped to states almost orthogonal to the projectors i.e. close to $|1\rangle$. This is done by skewing the $\{|0\rangle, |1\rangle\}$ assignments to $\{|0_\varepsilon\rangle, |1\rangle\}$ respectively where

$$|0_\varepsilon\rangle := \frac{1}{\sqrt{1+\varepsilon^2}}(|1\rangle - \varepsilon|0\rangle).$$

Now, any classical assignment is mapped to a state with the energy shrunk by a factor of $O(\varepsilon^4) = O\left(\frac{1}{\text{poly}(N)}\right)$. With this in mind, the way to convert a MAX2SAT formula to an appropriate Q2SAT Hamiltonian is described below.

Mapping Instances: The MAX2SAT formula φ is converted to a Q2SAT Hamiltonian $H = H_\varphi + H_G$ in two steps. Here H_φ acting on n qubits depends on the formula φ. and H_G acting on $O(n)$ qubits arises from the addition of some gadgets that help to maintain the soundness of the reduction. Table 4.1 shows how each clause of φ is converted to a projector in H_φ.

MAX2SAT clause	Forbidden state	Q2SAT Projector	Skewed Projector
$x_i \vee x_j$	00	$\|00\rangle_{ij}$	$\|00\rangle_{ij}$
$\bar{x}_i \vee x_j$	10	$\|10\rangle_{ij}$	$\|1_\varepsilon 0\rangle_{ij}$
$x_i \vee \bar{x}_j$	01	$\|01\rangle_{ij}$	$\|01_\varepsilon\rangle_{ij}$
$\bar{x}_i \vee \bar{x}_j$	11	$\|11\rangle_{ij}$	$\|1_\varepsilon 1_\varepsilon\rangle_{ij}$

Table 4.1: From MAX2SAT clauses to skewed Q2SAT Projectors

So, a clause acting on variables x_i, x_j is mapped to a 2-local projector on qubits i, j by setting the projector to the unsatisfying assignment of the clause. For example, if the clause is $(x_i \vee x_j)$, the projector on the two qubits would be $|00\rangle\langle 00|_{ij}$. Then, projector is skewed by applying the following transformation to each qubit in the projector:

$$|0\rangle \longmapsto |0\rangle \qquad |1\rangle \longmapsto |1_\varepsilon\rangle.$$

In this way, the boolean formula φ is converted to a Hamiltonian $H^{(\varphi)}$.

As mentioned previously, Boolean assignments will be mapped to states that are close to $|1\rangle$. So, to maintain the soundness of the reduction, one would like to enforce the condition that product states where the qubits are assigned states far from $|1\rangle$ have high energy with respect to H. This is done by adding $O(k)$ dummy qubits with appropriate gadgets on them for every one of the original n qubits in H_φ. The role of these gadgets is to have 0 energy when the qubits in

H_φ are assigned values from $\{|1\rangle, |0_\varepsilon\rangle\}$ and to have *high* energy when the values assigned are far from these states. This will be seen while proving the completeness and soundness of the reduction. Specifically, for an integer $f > \left(\frac{c'}{\left(1-\frac{1}{c'}\right)^4}\right) \cdot k$, add a *flower* gadget G_i corresponding to each qubit $i \in H_\varphi$ in the manner described below:

- Add $2f$ dummy qubits labeled from $i_0, \ldots i_{2f-1}$;
- Add the constraints $|1_\varepsilon 0\rangle\langle 1_\varepsilon 0|i, i_{(2\ell)}$ for $0 \leq \ell < f$;
- Add the constraints $|1_\varepsilon 0\rangle\langle 1_\varepsilon 0|i_{(2\ell)}, i_{(2\ell+1)}$ for $0 \leq \ell < f$;
- Add the constraints $|1_\varepsilon 0\rangle\langle 1_\varepsilon 0|i_{(2\ell+1)}, i$ for $0 \leq \ell < f$.

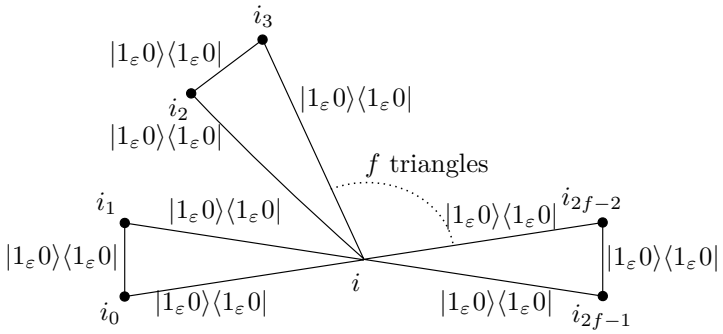


Figure 4.7: The flower gadget G_i for a qubit i.

The constraint graph for G_i, illustrated in Figure 4.7 shows that it is a collection of triangles added with their intersection at qubit i and every edge of the triangle has the constraints $|1_\varepsilon 0\rangle\langle 1_\varepsilon 0|$ on it. Now, setting $H_G = \sum_{i=1}^{n} G_i$, the Hamiltonian of interest is:

$$H = H_\varphi + H_G$$

and the size of the frustrated instance has now blown up to $N = n + f \cdot n = O(n)$ qubits.

Completeness of the reduction: The completeness of the reduction is shown by describing how to map the best classical assignment on the n variables to a low energy product state on N qubits. Map the Boolean assignment $\{0, 1\}$ on a variable x_i to the states $\{|0_\varepsilon\rangle, |1\rangle\}$ on qubit i so that:

$$1 \longmapsto |1\rangle \qquad\qquad 0 \longmapsto |0_\varepsilon\rangle.$$

After a qubit by qubit transformation, the assignment $\mathbf{a} \in \{0,1\}^n$ is mapped to a state $|\psi_{\mathbf{a}}\rangle \in \{|0_\varepsilon\rangle, |1\rangle\}^{\otimes n}$. The inner products between the four states $\{|0\rangle, |1\rangle, |1_\varepsilon\rangle, |0_\varepsilon\rangle\}$ is useful to calculate the energy of a state with respect H_φ:

$$\langle 1_\varepsilon | 0_\varepsilon \rangle = 0, \qquad \langle 1 | 1_\varepsilon \rangle = \frac{\varepsilon}{\sqrt{1+\varepsilon^2}}, \qquad \langle 0 | 0_\varepsilon \rangle = \frac{-\varepsilon}{\sqrt{1+\varepsilon^2}}.$$

Suppose, without loss of generality, that the projector in H_φ is $|01_\varepsilon\rangle\langle 01_\varepsilon|$ for which the energy

of the skewed states is:

$$|\langle 0_\varepsilon 0_\varepsilon|01_\varepsilon\rangle|^2 \quad = \quad |\langle 10_\varepsilon|01_\varepsilon\rangle|^2 \quad = \quad |\langle 11|01_\varepsilon\rangle|^2 \quad = \quad 0, \tag{4.5}$$

$$|\langle 0_\varepsilon 1|01_\varepsilon\rangle|^2 \quad = \quad |\langle 0_\varepsilon|0\rangle\ \langle 1|1_\varepsilon\rangle|^2 \quad = \quad \left|\frac{-\varepsilon}{\sqrt{1+\varepsilon^2}}\frac{\varepsilon}{\sqrt{1+\varepsilon^2}}\right|^2 \quad = \quad \frac{\varepsilon^4}{(1+\varepsilon^2)^2}. \tag{4.6}$$

The energy for other projectors behaves similarly implying that for the forbidden state (in this case, $|0_\varepsilon 1\rangle$), the energy is $\frac{\varepsilon^4}{(1+\varepsilon^2)^2}$ and for the other assignments from $\{|0_\varepsilon\rangle, |1\rangle\}^{\otimes 2}$ it is 0.

Now consider the flower gadgets added on each qubit. As the case is symmetric for each qubit, and for each triangle of the flower, consider the triangle $\{i, i_a, i_b\} \in G_i$. The constraints on the triangle will be satisfied if $|\psi_i\rangle = |\psi_{i_a}\rangle = |\psi_{i_b}\rangle$ whenever $|\psi_i\rangle \in \{|0_\varepsilon\rangle, |1\rangle\}$. Hence, any assignment where $\{i, i_a, i_b\}$ have the same state assigned to them has 0 energy. This can be extended to all the flowers resulting in 0 energy on the gadget H_G. Hence, a boolean assignment $\mathbf{a}$ which has $\mathsf{unsat}(\varphi, \mathbf{a}) \leq k$ gives a product state $|\psi\rangle \in \{|0_\varepsilon\rangle, |1\rangle\}^{\otimes N}$ such that $\langle\psi|\tilde{H}|\psi\rangle \leq k\frac{\varepsilon^4}{(1+\varepsilon^2)^2} = a$.

Soundness of the reduction: To show that the reduction is sound, the following statement is proved: when H has a low energy product state $|\phi\rangle$ i.e., $\langle\phi|H|\phi\rangle \leq a = k\frac{\varepsilon^4}{(1+\varepsilon^2)^2}$, it is possible to extract a classical assignment $\mathbf{a}$ from $|\phi\rangle$ such that $\mathsf{unsat}(\varphi, \mathbf{a}) \leq k$. Let this N-qubit product state $|\phi\rangle$ be expressed as $|\phi\rangle = |\phi_1\rangle \ldots |\phi_N\rangle$ where

$$\forall i, \quad \varepsilon_i \in [0,1] \quad \text{and} \quad |\phi_i\rangle = \frac{1}{\sqrt{1+|\varepsilon_i|^2}}(|1\rangle - \varepsilon_i|0\rangle) \tag{4.7}$$

$$\text{Then,} \qquad |\langle\phi_i|1_\varepsilon\rangle|^2 = \frac{|\varepsilon-\varepsilon_i|^2}{(1+\varepsilon^2)(1+|\varepsilon_i|^2)} \tag{4.8}$$

$$\text{and} \qquad |\langle\phi_i|0\rangle|^2 = \frac{|\varepsilon_i|^2}{(1+|\varepsilon_i|^2)}. \tag{4.9}$$

The proof for soundness is done in two parts. First, by setting $\delta = (1 - \frac{1}{c'})\varepsilon$, it is deduced that $\forall i$, ε_i is in the *good range*, i.e., the interval $(-\delta, \delta) \cup (\varepsilon - \delta, \varepsilon + \delta)$, for a low energy state on H. Essentially, the good range implies that the single-qubit state is either close to 0 or close to ε . The second part extracts a good classical assignment for φ from a low energy state $|\phi\rangle$.

For the first part, assume toward a contradiction that there exists a qubit i in the low energy state $|\phi\rangle$ such that $\varepsilon_i \notin (-\delta, \delta) \cup (\varepsilon - \delta, \varepsilon + \delta)$. By showing that the energy of the flower gadget $G_i \geq c'k\frac{\varepsilon^4}{(1+\varepsilon^2)^2}$, the low energy assumption is contradicted. Going back to the triangle formed by $\{i, i_a, i_b\}$ for qubit i with the constraints $\{|1_\varepsilon 0\rangle\langle 1_\varepsilon 0|_{ii_a}, |1_\varepsilon 0\rangle\langle 1_\varepsilon 0|_{i_a i_b}, |1_\varepsilon 0\rangle\langle 1_\varepsilon 0|_{i_b i}\}$, let the states in the assignment to these qubits be $|\phi_i\rangle, |\phi_{i_a}\rangle, |\phi_{i_b}\rangle$ respectively. When the states have the structure in equation (4.7), using equations (4.8) and (4.9), the energy of the triangle is

$$\frac{1}{(1+\varepsilon^2)}\left(\frac{|\varepsilon-\varepsilon_i|^2|\varepsilon_{i_a}|^2}{(1+|\varepsilon_i|^2)(1+|\varepsilon_{i_a}|^2)} + \frac{|\varepsilon-\varepsilon_{i_a}|^2|\varepsilon_{i_b}|^2}{(1+|\varepsilon_{i_a}|^2)(1+|\varepsilon_{i_b}|^2)} + \frac{|\varepsilon-\varepsilon_{i_b}|^2|\varepsilon_i|^2}{(1+|\varepsilon_{i_b}|^2)(1+|\varepsilon_i|^2)}\right). \tag{4.10}$$

To lower bound this energy, observe that $a + b + c \geq 3\min\{a, b, c\}$ when $a, b, c \geq 0$. Let $I = \{(i, i_a), (i_a, i_b), (i_b, i)\}$. Then, as energy terms are always at least 0, the triangle's energy is

lower bounded by

$$\begin{aligned}\frac{3}{(1+\varepsilon^2)}\min_{(j,j')\in I}\left\{\frac{|\varepsilon-\varepsilon_j|^2|\varepsilon_{j'}|^2}{(1+|\varepsilon_j|^2)(1+|\varepsilon_{j'}|^2)}\right\} &\geq \frac{3}{2(1+\varepsilon^2)^2}\min_{(j,j')\in I}\{|\varepsilon-\varepsilon_j|^2|\varepsilon_{j'}|^2\} \quad (\text{Since, } \varepsilon_i \leq 1.)\\ &\geq \frac{3\delta^4}{2(1+\varepsilon^2)^2} \quad (\text{Since, } \min\{\varepsilon_i,(\varepsilon-\varepsilon_i)\}\geq\delta.)\\ &\geq \frac{3(1-\frac{1}{c'})^4\varepsilon^4}{2(1+\varepsilon^2)^2}. \quad (\text{Since, } \delta=(1-\frac{1}{c'})\varepsilon.)\end{aligned}$$

Given f triangles, the energy of the flower gadget G_i is

$$f\frac{3(1-\frac{1}{c'})^4\varepsilon^4}{2(1+\varepsilon^2)^2} \;>\; \frac{c'\cdot k}{(1-\frac{1}{c'})^4}\frac{3(1-\frac{1}{c'})^4\varepsilon^4}{2(1+\varepsilon^2)^2} \;>\; c'\;k\;\frac{\varepsilon^4}{(1+\varepsilon^2)^2}.$$

Hence, if ε_i is far from the good range, then energy contributed from the flower gadget G_i is high implying that these states will never have low energy product with respect to H.

Moving to the second part, it is known that the low energy product sate $|\phi\rangle$ is such that for all i, $\varepsilon_i \in (0,\delta)\cup(\varepsilon-\delta,\varepsilon+\delta)$ and $\langle\phi|H|\phi\rangle \leq k\frac{\varepsilon^4}{(1+\varepsilon^2)^2}$. Recall that H is a combination of the flower gadgets H_G and the Hamiltonian H_φ which has the skewed projectors corresponding to the formula φ. Then for $|\phi\rangle = |z\rangle_\varphi \otimes |\phi\rangle_G$, where $|z\rangle_\varphi$ acts on the n qubits of H_φ,

$$\langle\phi|H|\phi\rangle \leq k\frac{\varepsilon^4}{(1+\varepsilon^2)^2} \;\Rightarrow \langle z|H_\varphi|z\rangle \leq k\frac{\varepsilon^4}{(1+\varepsilon^2)^2}. \tag{4.11}$$

The implication holds as H_φ is one of the components of H and $|z\rangle$ is just a restriction of $|\phi\rangle$ to the qubits H_φ acts on. Due to the intervals in which the ε_is can be found, the states for each of the n qubits is close to the states $\{|1\rangle,|0_\varepsilon\rangle\}$. A straightforward way to extract a classical assignment $\mathbf{a}$ from $|z\rangle$ is, for all $1\leq i\leq n$, set $a_i=1$ when $\varepsilon_i\in(-\delta,\delta)$ and $a_i=0$ when $\varepsilon_i\in(\varepsilon-\delta,\varepsilon+\delta)$. To calculate $\mathsf{unsat}(\varphi,\mathbf{a})$, consider a clause C from φ involving the variables x_i,x_j that is violated by the assignment $\mathbf{a}$. Let the corresponding projector in H_φ be Π and the energy of $|z\rangle$ on this clause is $\langle z_iz_j|\Pi|z_iz_j\rangle$. Now, $\mathsf{unsat}(\varphi,\mathbf{a})$ is bounded by

$$\mathsf{unsat}(\varphi,\mathbf{a}) \leq \frac{\text{Energy of } |z\rangle \text{ on } H_\varphi}{\min_{i,j}\{\langle z_iz_j|\Pi|z_iz_j\rangle \mid \mathbf{a} \text{ violates the clause } C\}}$$

Suppose that $C=(\bar{x}_i\vee\bar{x}_j)$ which, using Table 4.1, becomes the projector $\Pi=|1_\varepsilon1_\varepsilon\rangle\langle1_\varepsilon1_\varepsilon|$ in H_φ. C is violated when $a_i=a_j=1$ and in this case,

$$\begin{aligned}\langle z_iz_j|\Pi|z_iz_j\rangle = |\langle1_\varepsilon1_\varepsilon|z_iz_j\rangle|^2 &= \frac{|\varepsilon-\varepsilon_i|^2|\varepsilon-\varepsilon_j|^2}{(1+\varepsilon^2)^2(1+|\varepsilon_i|^2)(1+|\varepsilon_j^2|)} \quad (\text{Using Equation 4.8})\\ &\geq \frac{(\varepsilon-\delta)^4}{(1+\varepsilon^2)^2(1+\delta^2)^2} \quad (\text{As } \varepsilon_i,\varepsilon_j\in(-\delta,\delta) \text{ when } a_i=a_j=1)\\ &> \frac{(\varepsilon-\delta)^4}{(1+\varepsilon^2)^2(1+\delta)^2} \quad (\text{As } \delta<1\Rightarrow\delta^2<\delta)\end{aligned}$$

$$
\begin{aligned}
&> \frac{(\varepsilon-\delta)^4}{(1+\varepsilon^2)^2(c')^2} \quad \text{(By definition, } \delta < c'-1\text{)} \\
&> \frac{1}{(c')^6}\frac{\varepsilon^4}{(1+\varepsilon^2)^2} \quad \text{(Using } \delta = (1-\frac{1}{c'})\varepsilon\text{)} \\
&> \frac{1}{c}\frac{\varepsilon^4}{(1+\varepsilon^2)^2} \quad \text{(Using } c' \leq (c)^{\frac{1}{6}}\text{).}
\end{aligned}
$$

Performing similar calculations for the other projectors (i.e., $\{|01_\varepsilon\rangle\langle 01_\varepsilon|, |1_\varepsilon 0\rangle\langle 1_\varepsilon 0|, |00\rangle\langle 00|\}$), one can conclude that

$$
\begin{aligned}
&\min\{\langle z_i z_j|\Pi|z_i z_j\rangle \mid \mathbf{a} \text{ violates the clause } C\} > \frac{1}{c}\frac{\varepsilon^4}{(1+\varepsilon^2)^2} \\
&\Rightarrow \quad \mathsf{unsat}(\varphi,\mathbf{a}) < k\frac{\varepsilon^4}{(1+\varepsilon^2)^2}\frac{1}{\frac{1}{c}\frac{\varepsilon^4}{(1+\varepsilon^2)^2}} < c\cdot k.
\end{aligned}
$$

A key observation at this point is that the inapproximability constant c for MAX2SAT has been picked for the instance in Theorem 4.4 in such a way that $c \cdot k < k+1$. Then,

$$
\mathsf{unsat}(\varphi,\mathbf{a}) < c\cdot k < k+1 \Rightarrow \mathsf{unsat}(\varphi,\mathbf{a}) \leq k.
$$

This proves that the reduction is sound. □

Theorem 4.5 demonstrates, that in this particular regime, differentiating between the two cases of the $\mathsf{Approx}-\mathsf{Q2SAT}$ problem is NP-hard providing more parallels in the behaviour of Q2SAT and 2SAT. This allows one to hope that the more specific $\mathsf{Approx}-\mathsf{Q2SAT}$ problem on almost frustration free instances, which is similar to a *robust* instance of MAX2SAT[2], may still have a polynomial time approximation algorithm. This rides on the result by Charrikar et. al. [CMM09] where they construct an algorithm using semidefinite programming and rounding the solution to find a *near optimal* solution. That is, for a robust instance on n variables where at least $1-\epsilon$ fraction of clauses could be satisfied, the rounding generates an assignment that satisfies at least $1-\sqrt{\epsilon}$ fraction of the clauses in polynomial time if $\epsilon = \frac{1}{\mathrm{poly}(n)}$. If this can be extended to the $\mathsf{Approx}-\mathsf{Q2SAT}$ setting for almost frustration free instances, it could enable the finding of a low energy tensor product state. It could also provide some insight into the construction of approximation algorithms for quantum CSPs. However, a complete solution to this problem is left for future work.

[2]For a robust MAX2SAT formula on n variables, the optimal assignment is promised to satisfy at least $1-\frac{1}{\mathrm{poly}(n)}$ fraction of clauses

Chapter 5

Trial and error for constraint satisfaction problems

Question *Is there a single framework to determine the complexity of a hidden-*CSP*?*

This chapter deals with the development of tools that help to determine the complexity of a CSP in the trial and error setting when the properties of the base CSP are given. This is done with the use of *transfer theorems* that provide operations which, when performed on the *type* and *parameters* of a hidden CSP, transfers the study of hidden CSPs to the more commonly studied setting of CSPs with full access to the input. Beyond covering all the $\mathcal{U}$-revealing oracles for $\mathcal{U} \subseteq \{\mathcal{C}, \mathcal{R}, \mathcal{V}\}$, these transfer theorems can also accommodate CSPs where the input instance is required to adhere to some *promise* PROM.

As a means to motivate the operations which create a richer set of CSPs, a key observation is the polynomial time equivalence between H–$\mathrm{S}_{\{\mathcal{C},\mathcal{R},\mathcal{V}\}}$ and S for some CSP S for which $\mathrm{comp}(\mathcal{R}_{\mathrm{S}}) \in \mathrm{P}$. Clearly, any algorithm for H–$\mathrm{S}_{\{\mathcal{C},\mathcal{R},\mathcal{V}\}}$ can solve S as the answers of the oracle will attest to a proposed assignment satisfying the instance. Now, for the converse, the idea is to propose successive trials to the oracle $\mathcal{O}_{\{\mathcal{C},\mathcal{R},\mathcal{V}\}}$. The oracle, for each violation, completely reveals a violated constraint. As a subset of constraints becomes known, the next trial is then constructed so as to satisfy all the constraints known so far using the algorithm for S. With a weaker oracle, the procedure is not so straightforward anymore and the requirement for more complex CSPs arises. Fortunately, as hinted to in the above case, these complex CSPs for any H–S are related to the base problem S and the revealing oracle in question. Hence, knowing the type of S and the oracle, it becomes a fairly straightforward process to construct these CSP extensions with the goal of determining their complexity.

5.1 CSP extensions

The CSP extensions provide a richer sets of relations that can be derived from a given CSP S. These extensions generally turn out to be harder than the base problem with the added bonus of being equivalent to various hidden variants of S. The first operation connects to the $\{\mathcal{C}, \mathcal{V}\}$-revealing oracle. In this case, the oracle's answer reveals the indices of the variables involved in some clause C_j. This would help to exclude all the relations that are violated by the assignment to those indices. But then, C_j is necessarily one of the relations which was satisfied

by the assignment at those indices. In order to gain more information about satisfying the input, the trials proposed henceforth should satisfy at least one of the satisfied relations, i.e., a union of the satisfied relations, leading to the *closure by union* operation.

Definition 5.1 (Closure by Union)**.** *Given a* CSP S *with the relation set* $\mathcal{R}$*, the closure by union of* $\mathcal{R}$ *gives the relation set* $\bigcup\mathcal{R} = \{\bigcup_{R\in\mathcal{R}'} R \ : \ \mathcal{R}' \subseteq \mathcal{R}\}$*. Then the closure by union of* S *is the* CSP $\bigcup$S *whose type is* $\bigcup\mathcal{R}$ *and whose other parameters are the same as those of* S*.*

A relation $R \in \bigcup\mathcal{R}$ is represented as a list of indices from $\mathcal{R}$ whose union gives R. For instance, in the case of 1SAT, given $\mathcal{R}_{\mathsf{1SAT}} = \{\mathsf{Id}, \mathsf{Neg}\}$, $\bigcup\mathcal{R}_{\mathsf{1SAT}} = \{\mathsf{Id}, \mathsf{Neg}, R' = \{0, 1\}\}$ as $\mathsf{Id} \cup \mathsf{Id} = \mathsf{Id}$, $\mathsf{Neg} \cup \mathsf{Neg} = \mathsf{Neg}$ and $R' = R_1 \cup R_2 = \mathsf{Id} \cup \mathsf{Neg} = \{0, 1\}$. Note that $\dim(\bigcup\mathcal{R}_{\mathsf{1SAT}}) \leq |\mathcal{R}_{\mathsf{1SAT}}|$.

With respect to the $\{\mathcal{C}, \mathcal{R}\}$-revealing oracle, the situation requires compensating with a stronger CSP so as to cover for the lack of information about the variables the violated clause C_j works on. As the q-ary relation involved in C_j is revealed, the trials, henceforth, should try to satisfy that relation on at least one of the possible q-tuple of indices. This, as a first step, leads to the *arity extension* operation that produces relations whose arities are as large as the assignment length.

Definition 5.2 (Arity Extension)**.** *Given a* CSP S*, for a relation* $R \in \mathcal{R}$ *and a* q*-tuple* $(j_1, \ldots, j_q) \in [\ell]^{(q)}$*, the* ℓ*-ary relation* $R^{(j_1,\ldots,j_q)} = \{a \in W \ : \ (a_{j_1}, \ldots, a_{j_q}) \in R\}$ *and the arity extension of* R *is given by* X–$\mathcal{R} = \{R^{(j_1,\ldots,j_q)} \ : \ R \in \mathcal{R}$ *and* $(j_1, \ldots, j_q) \in [\ell]^{(q)}\}$*. The arity extension of* S *is, correspondingly, the problem* X–S *whose type is* X–$\mathcal{R}$*, arity is equal to the assignment length* ℓ *and whose other parameters are the same as* S*'s.*

Essentially, X–$\mathcal{R}$ contains natural extensions of the relations in $\mathcal{R}$ to ℓ-ary relations where each extension is distinguished by the choice of the q-tuple. A relation R in X–$\mathcal{R}$ is represented by the index of a relation in R and a tuple from $[\ell]^{(q)}$. Going back to the case of 1SAT, X–$\mathcal{R}_{\mathsf{1SAT}} = \{(\mathsf{Id}, 1), \ldots, (\mathsf{Id}, n), (\mathsf{Neg}, 1), \ldots, (\mathsf{Neg}, n)\}$. However, this does not completely reflect the situation with the $\mathcal{O}_{\{\mathcal{C},\mathcal{R}\}}$ oracle as the information obtained regarding C_j only reveals that it is the relation R acting on one of several possible tuples in $[\ell]^{(q)}$. This leads to considering the restricted union of these arity extended relations coming from the same base relation R.

Definition 5.3 (Restricted union of arity extensions)**.** *Given a* CSP S*, for a relation* $R \in \mathcal{R}$ *and a set of* q*-tuples* $I \subseteq [\ell]^{(q)}$*, let* $R^I = \bigcup_{(j_1,\ldots,j_q)\in I} R^{(j_1,\ldots,j_q)}$ *and* E–$\mathcal{R} = \{R^I \ : \ R \in \mathcal{R}$ and $I \subseteq [\ell]^{(q)}\}$*. Then the restricted union of the arity extension of* S *is the* CSP E–S *with type* E–$\mathcal{R}$*, parity* ℓ*, and other parameters the same as* S*.*

Since, each extension tries to correspond to each of the revealing oracles from Figure 2.3, the final extension corresponds to the $\mathcal{O}_{\{\mathcal{C}\}}$ oracle which combines the first two extensions to account for the extremely restrictive nature of this oracle. Essentially, the information revealed by this oracle is related to the unrestricted union of arity extensions of the base problem.

Definition 5.4 (Union of arity extensions)**.** *Given a* CSP S *with relation set* $\mathcal{R}$*, the* $\bigcup$X–$\mathcal{R}$ *is*

the arbitrary union of arity extensions of $\mathcal{R}$ with

$$\bigcup \text{X–}\mathcal{R} = \left\{ \bigcup_{1 \le i \le |\mathcal{R}'|, R_i \in \mathcal{R}', (i_1,\ldots,i_q) \in I} R_i^{(i_1,\ldots,i_q)} \;:\; \mathcal{R}' \subseteq \mathcal{R}, I \subseteq [\ell]^{(q)}, |\mathcal{R}'| = |I| \right\}.$$

So, each relation from $\bigcup$X–$\mathcal{R}$ is assumed to be represented as a sequence of pairs, each consisting of an index of a relation in $\mathcal{R}$ and an element of $[\ell]^{(q)}$. If the difference between the last two operations is not so clear, note that E–$\mathcal{R} \subseteq \bigcup$ X–$\mathcal{R} = \bigcup$ E–$\mathcal{R}$. In the context of 1SAT, given a set of indices $I = \{j_1, j_2, \ldots, j_{|I|}\}$, a relation from E–$\mathcal{R}_{\mathsf{1SAT}}$ will be of the form $(\mathsf{Id}, I) = \mathsf{Id}^{(j_1)} \cup \ldots \cup \mathsf{Id}^{(j_{|I|})}$ while a relation from $\bigcup$X–$\mathcal{R}_{\mathsf{1SAT}}$ could be of the form $\Big((\mathsf{Id}, j_1), (\mathsf{Neg}, j_2), (\mathsf{Neg}, j_3), \ldots, (\mathsf{Id}, j_{|I|})\Big) = \mathsf{Id}^{(j_1)} \cup \mathsf{Neg}^{(j_2)} \cup \mathsf{Neg}^{(j_3)} \ldots \cup \mathsf{Id}^{(j_{|I|})}$.

The arity extension operation justifies the consideration of CSPs with arities, number of relations and alphabet size which are functions of n. Though such a dependence may be unfamiliar notationally (in most cases, these parameters are a constant), it does capture some natural instances like the system of linear inequalities. Moreover, the arity extension also results in the number of relations depending on n since $|\text{X–}\mathcal{R}| = O(s \times \ell^q)$ and the assignment length ℓ is a function of n. However, as mentioned above, the relations arising as a result of these extensions can be represented succinctly and their number can be computed easily. In fact, starting with a set of relations $\mathcal{R}$ with comp($\mathcal{R}$) and considering the most complicated extension $\bigcup$X–$\mathcal{R}$, it is possible to determine whether an admissible assignment $v \in W$ satisfies a relation in $\bigcup$X–$\mathcal{R}$ in time $O(|I| \cdot |\mathcal{R}| \cdot \text{comp}(\mathcal{R}))$ for $I \subseteq [\ell]^{(q)}$.

Now, that the extensions are formally defined, it is possible to enter the nitty gritty of the transfer theorems themselves.

5.2 Transfer Theorems

There are three different transfer theorems, one for each type of revealing oracle and they show that for any CSP S, using $\bigcup$S, E–S and $\bigcup$E–S, it is possible to compensate for the information hidden by the $\mathcal{O}_{\{\mathcal{C},\mathcal{V}\}}$, $\mathcal{O}_{\{\mathcal{C},\mathcal{R}\}}$ and $\mathcal{O}_{\{\mathcal{C}\}}$ oracles respectively. It is perhaps more surprising to also notice that these statements hold in the reverse direction: if H–CSP with some revealing oracle can be solved, then the corresponding CSP extension can also be solved. Note that, for each transfer theorem, H–CSP instance with m constraints corresponds to a CSP extension also containing m constraints.

Theorem 5.1. *(a) If* $\bigcup$S *is solvable in time T then* H–S$_{\{\mathcal{C},\mathcal{V}\}}$ *is solvable in time* $O((T + s \times \text{comp}(\mathcal{R})) \times m \times \min\{\dim(\bigcup\mathcal{R}), |W_q|\})$.
(b) If H–S$_{\{\mathcal{C},\mathcal{V}\}}$ *is solvable in time T then* $\bigcup$S *is solvable in time* $O(T \times m \times \text{comp}(\bigcup\mathcal{R}))$.

Proof. (*a*) Let $\mathcal{A}$ be an algorithm which solves $\bigcup$S in time T. It is possible to construct an algorithm $\mathcal{B}$ for H–S$_{\{\mathcal{C},\mathcal{V}\}}$ which will repeatedly call $\mathcal{A}$ and query $\mathcal{O}_{\{\mathcal{C},\mathcal{V}\}}$, until it finds a

satisfying assignment or reaches the conclusion NO. The instance used to query $\mathcal{A}$ at the t^{th} call, $\mathcal{C}^t = \{C_1^t, \ldots, C_m^t\}$, is defined as $C_j^t = \bigcup_{R \in \mathcal{R}: R \cap A_j^t = \emptyset} R(x_{j_1^t}, \ldots, x_{j_q^t})$ where $A_j^t \subseteq W_q$ and $(j_1^t, \ldots, j_q^t) \in [\ell]^{(q)}$, for $j \in [m]$, are determined successively by $\mathcal{B}$. The set A_j^t reflects the algorithm $\mathcal{B}$'s knowledge about C_j after t steps in that it contains those q-tuple assignments which, at that instant, are known to violate C_j. Initially $A_j^1 = \emptyset$ and $(j_1^1, \ldots, j_q^1)$ is arbitrary. If $\mathcal{A}$'s output for $\mathcal{C}^t$ is NO then $\mathcal{B}$ outputs NO. On the other hand, if $\mathcal{A}$ returns an assignment $\mathbf{a} \in W$, then $\mathcal{B}$ proposes $\mathbf{a}$ to $\mathcal{O}_{\{\mathcal{C},\mathcal{V}\}}$. When the oracle returns YES, then $\mathcal{B}$ outputs $\mathbf{a}$ and concludes. Otherwise, the oracle reveals a violated constraint j and the q-tuple $(j_1, \ldots, j_q)$ that C_j acts on. Then, $\mathcal{B}$ updates $A_j^{t+1} = A_j^t \bigcup \{(a_{j_1}, \ldots, a_{j_q})\}$, and $(j_1^{t+1}, \ldots, j_q^{t+1}) = (j_1, \ldots, j_q)$ while keeping these sets unchanged for all other indices: $A_i^{t+1} = A_i^t$ and $(i_1^{t+1}, \ldots, i_q^{t+1}) = (i_1^t, \ldots, i_q^t)$ for $i \neq j$. The q-tuple of indices is changed at most once for each C_j – the first time when the oracle returns C_j as a violation.

To prove that $\mathcal{B}$ correctly solves H–S$_{\{\mathcal{C},\mathcal{V}\}}$, let $\mathcal{C} = \{C_1, \ldots, C_m\}$ be the hidden instance of S accessed using $\mathcal{O}_{\{\mathcal{C},\mathcal{V}\}}$. It needs to be shown that if $\mathcal{B}$ answers NO then $\mathcal{C}$ is unsatisfiable. $\mathcal{B}$ answers NO when, for some t, the t^{th} call to $\mathcal{A}$ resulted in the output NO. By construction, for every $j \in [m]$, A_j^t has the list of assignments to the indices $(j_1^t, \ldots, j_q^t)$ that violates C_j implying that if $R \cap A_j^t \neq \emptyset$, then $C_j \neq R(x_{j_1^t}, \ldots, x_{j_q^t})$. Clearly, supposing C_j was $R(x_{j_1^t}, \ldots, x_{j_q^t})$ and $b \in R \cap A_j^t$, then the $\mathcal{O}_{\{\mathcal{C},\mathcal{V}\}}$'s answer at the trial where b was added to A_j^t would be incorrect. Therefore, as C_j^t, for every j, contains the possible relations for C_j, when $\mathcal{C}^t$ is unsatisfiable, the hidden instance $\mathcal{C}$ also unsatisfiable.

For the complexity of the algorithm, observe that if for some j and t, the constraint C_j^t is the empty relation then $\mathcal{B}$ stops since $\mathcal{C}^t$ becomes unsatisfiable. This happens in particular if $A_j^t = W_q$. Since every call to $\mathcal{A}$ adds one new element to one of the A_j^ts and at least one new relation in $\mathcal{R}$ is excluded from the C_j^ts, the number of calls is upper bounded by $m \times \min\{\dim(\mathcal{R}), |W_q|\}$. To compute a new constraint, some number of relations in $\mathcal{R}$ have to be computed on a new argument, which can be done in time $s \times \mathrm{comp}(\mathcal{R})$.

(b) Let $\mathcal{A}$ be an algorithm which solves H–S$_{\{\mathcal{C},\mathcal{V}\}}$ in time T and assume without loss of generality that $\mathcal{A}$ outputs a satisfying assignment $\mathbf{a}$ only after submitting it to the verifying oracle $\mathcal{O}_{\{\mathcal{C},\mathcal{V}\}}$. An algorithm $\mathcal{B}$ for $\bigcup$S is constructed as follows. Let $\mathcal{C} = \{C_1, \ldots, C_m\}$ be an instance of $\bigcup$S where, for $j \in [m]$, $C_j = \bigcup_{R \in \mathcal{R}_j} R(x_{j_1}, \ldots, x_{j_q})$, for some $\mathcal{R}_j \subseteq \mathcal{R}$ and $(j_1, \ldots, j_q) \in [\ell]^{(q)}$. The algorithm $\mathcal{B}$ calls $\mathcal{A}$, and outputs NO whenever $\mathcal{A}$ outputs NO. During $\mathcal{A}$'s run $\mathcal{B}$ simulates the $\mathcal{O}_{\{\mathcal{C},\mathcal{V}\}}$ for $\mathcal{A}$ as described below. Along with the behaviour of $\mathcal{O}_{\{\mathcal{C},\mathcal{V}\}}$, for $t \geq 1$, instances of $\bigcup$S $\mathcal{C}^t = \{C_1^t, \ldots, C_m^t\}$ are specified so as to be used in the proof of correctness of $\mathcal{B}$. For $j \in [m]$, the constraints of $\mathcal{C}^t$ are defined as $C_j^t = \bigcup_{R \in \mathcal{R}: R \cap A_j^t = \emptyset} R(x_{j_1}, \ldots, x_{j_q})$, where the sets $A_j^t \subseteq W_q$ are determined by the result of the t^{th} call to $\mathcal{O}_{\{\mathcal{C},\mathcal{V}\}}$. Initially $A_j^0 = \emptyset$. For the t^{th} trial $\mathbf{a} \in W$, the algorithm $\mathcal{B}$ checks if $\mathbf{a}$ satisfies $\mathcal{C}$. If this is the case, then $\mathcal{O}_{\{\mathcal{C},\mathcal{V}\}}$ returns YES and $\mathcal{B}$ outputs $\mathbf{a}$. Otherwise there exists $j \in [m]$ such that $\mathbf{a}$ violates C_j, and $\mathcal{O}_{\{\mathcal{C},\mathcal{V}\}}$'s answer is j and $(j_1, \ldots, j_q)$ (where j is chosen arbitrarily among the violated constraints, if there are several). Observe that this is a legitimate oracle for any instance of H–S$_{\{\mathcal{C},\mathcal{V}\}}$ whose jth constraint is arbitrarily chosen

from $\mathcal{R}_j$. A_j^t is updated as $A_j^t = A_j^{t-1} \bigcup \{(a_{j_1}, \ldots, a_{j_q})\}$, and for $i \neq j$, $A_i^t = A_i^{t-1}$.

Showing that the instance $\mathcal{C}$ is unsatisfiable whenever $\mathcal{A}$ outputs NO would prove the correctness of $\mathcal{B}$. Suppose $\mathcal{A}$ made t queries before outputting NO. An algorithm for H–S$_{\{\mathcal{C},\mathcal{V}\}}$ outputs NO only if all possible instances of S which are compatible with the answers received from the oracle are unsatisfiable. In such an instance, the relation in C_j necessarily has an empty intersection with A_j^t, therefore implying that the $\bigcup$S instance $\mathcal{C}^t$ is unsatisfiable. It also holds that $A_j^t \bigcap (\bigcup_{R \in \mathcal{R}_j} R) = \emptyset$ for every $j \in [m]$, since if $b \in A_j^t \bigcap (\bigcup_{R \in \mathcal{R}_j} R)$ then the trial which added b to A_j^t wouldn't violate C_j. Thus $\bigcup_{R \in \mathcal{R}_j} R \subseteq \bigcup_{R \in \mathcal{R}: R \cap A_j^t = \emptyset} R$, and $\mathcal{C}$ is unsatisfiable.

For the complexity analysis, observe that during the algorithm, for every query to the oracle and for every constraint, one relation in $\bigcup \mathcal{R}$ is evaluated. □

Theorem 5.2. *(a) If* E–S *is solvable in time* T *then* H–S$_{\{\mathcal{C},\mathcal{R}\}}$ *is solvable in time* $O((T + |[\ell]^{(q)}| \times \text{comp}(\mathcal{R})) \times m \times |[\ell]^{(q)}|)$.
(b) If H–S$_{\{\mathcal{C},\mathcal{R}\}}$ *is solvable in time* T *then* E–S *is solvable in time* $O(T \times m \times \text{comp}(\text{E–}\mathcal{R}))$.

Proof. This proof follows the proof of Theorem 5.1 with some appropriate changes.
(a) Let $\mathcal{A}$ be an algorithm which solves E–S in time T. An algorithm $\mathcal{B}$ for H–S$_{\{\mathcal{C},\mathcal{R}\}}$ is constructed which will repeatedly call $\mathcal{A}$, until it finds a satisfying assignment or reaches the conclusion NO. Since each constraint of E–S is an ℓ-ary relation, it is identified with the relation itself. For the first call $\mathcal{C}^1 = \{C_1^1, \ldots, C_m^1\}$ and for all j, set $C_j^1 = W$. For $t > 1$, the instance $\mathcal{C}^t$ of the t^{th} call will be recursively defined via $A_j^t \subseteq W$ and $I_j^t \subseteq [\ell]^{(q)}$, for $j \in [m]$, where initially $A_1^1 = \ldots = A_m^1 = \emptyset$ and $I_1^1 = \ldots = I_m^1 = [\ell]^{(q)}$. Here the set I_j^t reflects $\mathcal{B}$'s knowledge of the hidden instance after t steps in that it contains those q-tuples of indices which, at that instant, could still be the variable indices on which C_j acts. If $\mathcal{A}$'s output for $\mathcal{C}^t$ is NO, then $\mathcal{B}$ outputs NO. On the other hand, if $\mathcal{A}$ outputs $\mathbf{a} \in W$, then $\mathcal{B}$ proposed $\mathbf{a}$ as a trial to $\mathcal{O}_{\{\mathcal{C},\mathcal{R}\}}$. When the oracle returns YES, $\mathcal{B}$ can output $\mathbf{a}$ and conclude. Otherwise, $\mathcal{O}_{\{\mathcal{C},\mathcal{R}\}}$ returns a violated constraint's index j and relation $R \in \mathcal{R}$. $\mathcal{B}$ updates $A_j^{t+1} = A_j^t \bigcup \{\mathbf{a}\}$ and $I_j^{t+1} = \{(j_1, \ldots, j_q) \ : \ A_j^{t+1} \bigcap R^{(j_1,\ldots,j_q)} = \emptyset\}$ and $C_j^{t+1} = R^{I_j^{t+1}}$. For $i \neq j$, $A_i^{t+1} = A_i^t$, $I_i^{t+1} = I_i^t$ and $C_i^{t+1} = C_i^t$.

To prove that $\mathcal{B}$ correctly solves H–S$_{\{\mathcal{C},\mathcal{R}\}}$, let $\mathcal{C} = \{C_1, \ldots, C_m\}$ be an instance of S accessed using the $\mathcal{O}_{\{\mathcal{C},\mathcal{R}\}}$ oracle. It has to be shown that $\mathcal{C}$ is unsatisfiable when $\mathcal{B}$ answers NO. If $\mathcal{B}$ answers NO, then for some t, the t^{th} call to $\mathcal{A}$ resulted in the output NO. The claim is that for every constraint C_j whose relation R has already been revealed, if $R^{(j_1,\ldots,j_q)} \cap A_j^t \neq \emptyset$ then C_j cannot be $R(x_{j_1}, \ldots, x_{j_q})$. Supposing $C_j = R^{(j_1,\ldots,j_q)}(x_{j_1}, \ldots, x_{j_q})$ and $\mathbf{a} \in R \cap A_j^t$, then $\mathcal{O}_{\{\mathcal{C},\mathcal{R}\}}$'s answer for the trial when $\mathbf{a}$ was added to A_j^t would be incorrect. Therefore, as C_j^t, for every j, is the union of $R^{(j_1,\ldots,j_q)}$ over all the still possible q-tuple of indices $(j_1, \ldots, j_q)$, when $\mathcal{C}^t$ is unsatisfiable, so is the hidden instance $\mathcal{C}$.

For the complexity of $\mathcal{B}$, note that if the constraint C_j^t, for some j and t, is the empty relation then $\mathcal{B}$ stops since $\mathcal{C}^t$ becomes unsatisfiable. This happens in particular if $I_j^t = \emptyset$. Since for every call to $\mathcal{A}$, for some j, the size of I_j^t decreases by at least one, the total number of calls is

upper bounded by $m \times |[\ell]^{(q)}|$. To compute new constraints, at most $|[\ell]^{(q)}|$ relations from $\mathcal{R}$ are evaluated in a new argument. Therefore the overall complexity is as claimed.

(b) Let $\mathcal{A}$ be an algorithm which solves H–S$_{\{\mathcal{C},\mathcal{R}\}}$ in time T and without loss of generality suppose that $\mathcal{A}$ outputs a satisfying assignment $\mathbf{a}$ only after submitting it to the verifying oracle. The algorithm $\mathcal{B}$ for E–S is constructed as follows. Let $\mathcal{C} = \{C_1, \ldots, C_m\}$ be an instance of E–S where for $j \in [m]$, $C_j = R_{k_j}^{I_j}$ for some $R_{k_j} \in \mathcal{R}$ and $I_j \subseteq [\ell]^{(q)}$. $\mathcal{B}$ calls $\mathcal{A}$, outputs NO whenever $\mathcal{A}$ outputs NO and simulates the $\mathcal{O}_{\{\mathcal{C},\mathcal{R}\}}$ oracle during $\mathcal{A}$'s run in the following fashion. Along with the description of $\mathcal{O}_{\{\mathcal{C},\mathcal{R}\}}$'s behaviour, for $t \geq 1$, the instances $\mathcal{C}^t = \{C_1^t, \ldots, C_m^t\}$ of E–S are specified which help to show the correctness of $\mathcal{B}$. ℓ-ary constraints are identified by the relations they represent. The constraints of $\mathcal{C}^t$ are set to be $C_j^t = R_{k_j}^{I_j^t}$, where the sets $I_j^t \subseteq [\ell]^{(q)}$ are defined as $I_j^t = \{(j_1, \ldots, j_q) \ : \ A_j^t \bigcap R_{k_j}^{(j_1,\ldots,j_q)} = \emptyset\}$, and the sets $A_j^t \subseteq W$ are determined by the result of the t^{th} trial proposed to $\mathcal{O}_{\{\mathcal{C},\mathcal{R}\}}$. Initially $A_j^0 = \emptyset$. For the t^{th} trial $\mathbf{a} \in W$, $\mathcal{B}$ checks if $\mathbf{a}$ satisfies $\mathcal{C}$. If so, then $\mathcal{O}_{\{\mathcal{C},\mathcal{R}\}}$ returns YES and $\mathcal{B}$ outputs $\mathbf{a}$. Otherwise, there exists $j \in [m]$ such that $\mathbf{a}$ violates C_j in which case, the $\mathcal{O}_{\{\mathcal{C},\mathcal{R}\}}$ returns j and R_{k_j}. This depicts the behaviour of a legitimate oracle for any instance of H–S$_{\{\mathcal{C},\mathcal{R}\}}$ where C_j is arbitrarily chosen from $\{R_{k_j}(x_{j_1}, \ldots, x_{j_q}) \ : \ (j_1, \ldots, j_q) \in I_j\}$. A_j^t is updated as $A_j^t = A_j^{t-1} \bigcup \{a\}$, and for $i \neq j$, $A_i^t = A_i^{t-1}$.

Showing that $\mathcal{C}$ is unsatisfiable whenever $\mathcal{A}$ outputs NO suffices to prove the correctness of $\mathcal{B}$. Suppose that $\mathcal{A}$ made t queries before outputting NO. An algorithm for H–S$_{\{\mathcal{C},\mathcal{R}\}}$ outputs NO only when all possible instances of S which are compatible with the oracle's answers are unsatisfiable. In such an instance C_j necessarily does not intersect A_j^t implying that the E–S instance $\mathcal{C}^t$ is unsatisfiable. For every $j \in [m]$, it also holds that $A_j^t \bigcap C_j = \emptyset$, since the supposition that $\mathbf{a} \in A_j^t \bigcap C_j$ implies that the trial which added a to A_j^t wouldn't violate C_j. Thus $C_j \subseteq C_j^t$, and $\mathcal{C}$ is unsatisfiable.

For the complexity analysis, observe that during the algorithm, for every query to the oracle and for every constraint, one relation in E–$\mathcal{R}$ is evaluated. □

Theorem 5.3. *(a) If* $\bigcup$X–S *is solvable in time* T *then* H–S$_{\{\mathcal{C}\}}$ *is solvable in time* $O((T + s \times \frac{\ell!}{(\ell-q)!} \times \mathrm{comp}(\mathcal{R})) \times m \times \dim(\bigcup \text{X–}\mathcal{R}))$.
(b) If H–S$_{\{\mathcal{C}\}}$ *is solvable in time* T *then* $\bigcup$X–S *is solvable in time* $O(T \times m \times \mathrm{comp}(\bigcup \text{X–}\mathcal{R}))$.

Proof. Apply Theorem 5.1 to X–S and observe that H–X–S$_{\{\mathcal{C},\mathcal{V}\}}$ and H–S$_{\{\mathcal{C}\}}$ are essentially the same in the sense that an algorithm solving one of the problems also solves the other one. Indeed, the variable index disclosure of the $\{\mathcal{C},\mathcal{V}\}$-revealing oracle is pointless since the relations in X–S involve all the variables. Moreover, the map sending a constraint $R(x_{j_1}, \ldots x_{j_q})$ of S to the constraint $R^{(j_1,\ldots,j_q)}(x_1, \ldots x_\ell)$ of X–S is a bijection which preserves satisfying assignments. □

Corollary 5.1. *Let* $\mathrm{comp}(\mathcal{R})$ *be polynomial. Then the complexities of the following problems are polynomial time equivalent: (a)* H–S$_{\{\mathcal{C},\mathcal{V}\}}$ *and* $\bigcup$S *if the number of relations* s *is constant; (b)* H–S$_{\{\mathcal{C},\mathcal{R}\}}$ *and* E–S *if the arity* q *is constant and (c)* H–S$_{\{\mathcal{C}\}}$ *and* $\bigcup$X–S *if both* s *and* q *are constant.*

The polynomial time equivalence of Theorems 5.1, 5.2, 5.3 and Corollary 5.1 remain true when the algorithms have access to the same computational oracle. Therefore, we get generic easiness results for H–CSPs under an NP oracle. An H–CSP is said to be NP-hard essentially when it demonstrates an equivalence to a CSP extension which is known to be NP-hard.

Example 5.1 (Hidden 1SAT (continued))**.** *Since* $\bigcup\{\mathsf{Id}, \mathsf{Neg}\} = \{\emptyset, \mathsf{Id}, \mathsf{Neg}, \{0,1\}\}$, $\bigcup$1SAT *has only the two trivial (always false or always true) relations in addition to the relations in* 1SAT*. Therefore it can be solved in polynomial time, and by the Transfer Theorem* H–1SAT$_{\{\mathcal{C},\mathcal{V}\}}$ *is also in* P*. On the other hand, for any index set* $I \subseteq [\ell]$, Id^I *is a disjunct of positive literals with variables from* I*, and similarly* Neg^I *is a disjunct of negative literals with variables from* I*. Thus* E–1SAT *includes Monotone SAT (*MONSAT*). The problem* MONSAT *is a special case of* SAT *where every clause has only positive or only negated literals. It is known to be* NP*-hard by Schaefer's characterization [Sch78], and therefore the Transfer Theorems imply that* H–1SAT$_{\{\mathcal{C},\mathcal{R}\}}$ *and* H–1SAT$_{\{\mathcal{C}\}}$ *are also* NP*-hard.*

As NP-hard problems obviously remain NP-hard in the hidden setting (without access to an NP oracle), it is arguably more interesting to investigate the complexity of various polynomial-time solvable CSPs. The Transfer Theorems are first applied when there is no promise on the input instances and the hidden CSPs are categorised depending on the type of the revealing oracle used.

5.3 Constraint index and variables revealing oracle

With the $\{\mathcal{C},\mathcal{V}\}$-revealing oracle, $\mathcal{O}_{\{\mathcal{C},\mathcal{V}\}}$, there are several interesting families of CSPs that are considered including the exact-Unique Games Problem and equality to a member of a fixed class of graphs. Many of the monotone graph properties, with this oracle, show a polynomial-time equivalence to their counterparts in the un-hidden setting and remain in P as do some other Boolean CSPs like 2SAT. The hidden version of the exact-Unique game problem with $\mathcal{O}_{\{\mathcal{C},\mathcal{V}\}}$ shows some split behaviour with the problem on an alphabet of size 2 remaining in P but becoming NP-hard for alphabet size ≥ 3. Finding if a hidden graph is equal to a k-clique is NP-hard in this setting and is equivalent to the Graph Isomorphism problem considered in [BCZ13, Theorem 13], whose proof, with the help of the Transfer Theorem, becomes very simple. For the sake of completeness, the CSPs considered in this section are described below under the assumption that $W = [\![w]\!]^\ell$ unless explicitly specified.

1. Deltas on Triplets (Δ): This problem arises by flipping the truth table for 3SAT i.e. there is exactly one string satisfying each relation instead of just one string that falsifies it. Formally, $w = 2$, $q = 3$, and $\mathcal{R} = \{R_{abc} : \{0,1\}^3 \to \{\mathbb{T},\mathbb{F}\} \mid a,b,c \in \{0,1\}\}$,where $R_{abc}(x,y,z) := (x = a) \wedge (y = b) \wedge (z = c)$.
2. Hyperplane Non-Cover (HYP−NC): Let p be a prime, and F_p be the field of size p. Denote $V = F_p^N$, and $S = \{\text{all hyperplanes in } F_p^N\}$. Informally, given a set of hyperplanes $S' \subseteq S$, the problem asks to decide if there exists $v \in F_p^N$ not covered by these hyperplanes.

Formally, $\ell = 1,\ q = 1,\ w = p^N,\ W = V$ and $\mathcal{R}_S = \{R_H \mid H \in S\}$ where $R_H(a)$ evaluates to $\mathbb{T}$ if and only if $a \notin H$.

3. Arbitrary sets of binary relations on Boolean alphabet, (in particular, 2SAT): Recall that for 2SAT, we have $w = 2,\ q = 2$, and $\mathcal{R} = \{R_T, R_F, (a), (\bar{a}), (a \vee b), (a \vee \bar{b}), (\bar{a} \vee b), (\bar{a} \vee \bar{b})\}$, where $R_T(a, b) = \mathbb{T}$ is the trivially true function, $R_F(a, b) := \mathbb{F}$ is the function that is always false and $\bar{a}$ denotes the negation of a.
4. Exact-Unique Game Problem (UG[k]): Given an undirected graph, $G = (V, E)$, and a permutation $\pi_e : [\![k]\!] \to [\![k]\!]$ for every edge $e \in E$, the goal is to decide if one can assign labels $\alpha_v \in [\![k]\!]$ for every vertex $v \in V$ such that for every edge $e = \{u, v\} \in E$ with $u < v$ we have $\pi_e(\alpha_u) = \alpha_v$. Formally: $w = k,\ q = 2$ and $\mathcal{R} = \{\pi : [\![k]\!] \to [\![k]\!] \mid \pi \text{ is a permutation}\}$.

5. k-Clique Isomorphism (kCLQ–ISO): Given an undirected graph $G = (V, E)$, determine if there exists a permutation π on $[n]$ such that
   (1) $\forall (i, j) \in E,\ \ R_{\leq k}(\pi(i), \pi(j))$;
   (2) $\forall (i, j) \notin E,\ \ \neg R_{\leq k}(\pi(i), \pi(j))$.
   Formally, $w = n,\ q = 2,\ \ell = 1$, W is the set of n-tuples of integers from $[n]$ which define permutations on $[n]$, and $\mathcal{R} = \{R_{\leq k}, \neg R_{\leq k}\}$, where $R_{\leq k}(\alpha, \beta) := \mathbb{T} \iff \alpha \leq k\ \&\ \beta \leq k$.
6. Equality to some member in a fixed class of graphs ($\mathsf{EQ}_{\mathcal{K}}$): For a fixed class $\mathcal{K}$ of graphs on n vertices variables, we denote by $\mathcal{P}_{\mathcal{K}} : \{0,1\}^{\binom{n}{2}} \to \{\mathbb{T}, \mathbb{F}\}$ the property of being equal to a graph from $\mathcal{K}$. Formally, $W = \mathcal{K},\ w = 2,\ q = 1,\ \ell = \binom{n}{2}$, and $\mathcal{R} = \{\mathsf{Id}, \mathsf{Neg}\}$ under the assumption is that membership in $\mathcal{K}$ can be tested in polynomial time. The following special cases will be considered:
   - Equality to k-Clique ($\mathsf{EQ}_{k\mathsf{CLQ}}$): Given a graph, decide if it is equal to a k-clique.
   - Equality to Hamiltonian Cycle ($\mathsf{EQ}_{\mathsf{HAMC}}$): Decide if G is a cycle on all n vertices.
   - Equality to Spanning Tree ($\mathsf{EQ}_{\mathsf{ST}}$): Given a graph, decide if it is a spanning tree.

First, the problems in P are illustrated after which the NP-hard problems are discussed.

Theorem 5.4. *The following are in* P *(a)* H–2SAT$_{\{\mathcal{C},\mathcal{V}\}}$*, (b)* H–UG[2]$_{\{\mathcal{C},\mathcal{V}\}}$*, (c)* H–EQ$_{\mathsf{ST}\{\mathcal{C},\mathcal{V}\}}$.

Proof. The following problems in the hidden input setting with the constraint and variable index revealing oracle are solvable in polynomial time.

(a) Arbitrary binary Boolean relations (H–2SAT$_{\{\mathcal{C},\mathcal{V}\}}$)

In the case of 2SAT, taking the union of any two relations in $\mathcal{R}_{\mathsf{2SAT}}$ is equivalent to the disjunction of the two boolean expressions the relations signify. For example, $R_a \bigcup R_b = R_{a \vee b}$ and the union remains in $\mathcal{R}_{\mathsf{2SAT}}$. Hence, $\bigcup \mathsf{2SAT} = \mathsf{2SAT}$, which is in P. Therefore, from Theorem 5.1(a), H–2SAT$_{\{\mathcal{C},\mathcal{V}\}}$ is also in P.

The above statement can be extended to an *arbitrary* set $\mathcal{R}'$ of binary relations as follows. Let $\mathcal{R}''$ stand for the set of *all* binary relations in Boolean variables. We trivially have $\bigcup \mathcal{R}' \subseteq \mathcal{R}''$, therefore an instance of H–2SAT$_{\{\mathcal{C},\mathcal{V}\}}$ can actually be described by a conjunction of the form $\bigwedge_{k=1}^{m} R_k(x_{i_k}, x_{j_k})$ where R_k is a binary relation. Expressing each R_k by a Boolean formula in conjunctive normal form, we obtain an instance of 2SAT consisting of

$O(m)$ clauses, which can be solved in polynomial time.

(b) Unique Games with alphabet size 2 (H–$\mathsf{UG}[2]_{\{\mathcal{C},\mathcal{V}\}}$)

$\mathsf{UG}[2]$ is a CSP with $w = 2$, $q = 2$, and $\mathcal{R} = \{\pi : [\![2]\!] \to [\![2]\!] \mid \pi \text{ is a permutation}\}$. The only permutations in $\mathcal{R}_{\mathsf{UG}[2]}$ enforce either that $\alpha_u = \alpha_v$ or that $\alpha_u = \alpha_v \oplus 1$ for an edge $e = (u, v)$. Both of these relations can be represented as binary boolean relations. Hence, $\mathsf{UG}[2]$ is an instance of $\mathsf{2SAT}$ and from Theorem 5.4(a), H–$\mathsf{UG}[2]_{\{\mathcal{C},\mathcal{V}\}}$ is in P.

(c) Equality/Isomorphism to a member in a fixed class of graphs

We consider the H–$\mathsf{EQ}_{\mathcal{K}\{\mathcal{C},\mathcal{V}\}}$ problem in more detail. Given a graph instance $G = (V, E)$ in this model, the $\binom{n}{2}$ constraints for G_2 are such that $C_e = \mathbb{I}(\alpha_e)$ for $e \in E$ and $C_e = \neg(\alpha_e)$ otherwise. This implies that $\bigcup\mathcal{R} = \{\mathsf{Id}, \mathsf{Neg}, \mathbb{T}\}$ and instances of $\bigcup\mathsf{EQ}_{\mathcal{K}}$ are parametrized with graphs (sets of edges) $E_1 \subseteq E_2$. Here E_2 is the set of edges for which the constraint is either Id or $\mathbb{T}$ while E_1 consists of pairs for which the constraint is Id. The $\bigcup\mathsf{EQ}_{\mathcal{K}}$-problem then becomes: given sets E_1, E_2 such that $E_1 \subseteq E_2$, does there exist a graph $G' = (V, E') \in \mathcal{K}$ such that $E_1 \subseteq E' \subseteq E_2$?

From Theorem 5.1, the complexity of H–$\mathsf{EQ}_{\mathcal{K}}$, can be analyzed by considering the complexity of $\bigcup\mathsf{EQ}_{\mathcal{K}}$. Below, we analyze the complexity of $\bigcup\mathsf{EQ}_{\mathcal{K}}$ when $\mathcal{K}$ is the class of spanning trees on n vertices.

Remark 5.1. *For any $\mathcal{K}$, if we take $E_1 = \emptyset$, then solving $\mathsf{EQ}_{\mathcal{K}}$ becomes equivalent to finding out if there exists $G \in \mathcal{K}$ which is a subgraph of E_2.*

Remark 5.2. *Note that if we assume that $\mathcal{K}$ is the set of all graphs isomorphic to some G_0 and $E_1 = E_2$ as arbitrary graphs on n vertices, then solving $\mathsf{EQ}_{\mathcal{K}}$ becomes equivalent to finding out if E_2 is isomorphic to G_0.*

Proof for Equality to a Spanning Tree (H–$\mathsf{EQ}_{\mathsf{ST}\{\mathcal{C},\mathcal{V}\}}$): Here, $\mathcal{K}$ is the set of all possible spanning trees on n vertices and E_1 without loss of generality is a forest F. E_2 is any arbitrary graph on n vertices containing E_1. In this case, the $\bigcup\mathsf{EQ}_{\mathcal{K}}$ problem becomes equivalent to finding a spanning tree on E_2 which also contains the forest F. This problem is in P which implies that the H–$\mathsf{EQ}_{\mathsf{ST}\{\mathcal{C},\mathcal{V}\}}$ problem is also in P. □

Theorem 5.5. *The following are* NP*-hard:(a)* H–$\Delta_{\{\mathcal{C},\mathcal{V}\}}$*, (b)* H–$\mathsf{HYP-NC}_{\{\mathcal{C},\mathcal{V}\}}$*, (c)* H–$\mathsf{UG}[k]_{\{\mathcal{C},\mathcal{V}\}}$ *for $k \geq 3$, (d)* H–$k\mathsf{CLQ}$–$\mathsf{ISO}_{\{\mathcal{C},\mathcal{V}\}}$*, (e)* H–$\mathsf{EQ}_{k\mathsf{CLQ}\{\mathcal{C},\mathcal{V}\}}$*, (f)* H–$\mathsf{EQ}_{\mathsf{HAMC}\{\mathcal{C},\mathcal{V}\}}$*.*

Proof. The following problems in the hidden model with the constraint and variable index revealing oracle are shown to be NP-hard. Using Theorem 5.1, the complexity of each H–$\mathrm{S}_{\{\mathcal{C},\mathcal{V}\}}$ is analyzed by considering the complexity of $\bigcup\mathrm{S}$.

(a) Deltas on Triplets (H–$\Delta_{\{\mathcal{C},\mathcal{V}\}}$)

By definition, each relation in $\mathcal{R}_\Delta$ identifies a boolean string on 3 variables. This implies that $\bigcup\mathcal{R}_\Delta$ forms the set of all Boolean predicates on 3 variables. Thus, $\mathsf{3SAT}$ can be expressed as the $\bigcup\Delta$ problem. Hence, from Theorem 5.1(b), H–$\Delta_{\{\mathcal{C},\mathcal{V}\}}$ is NP-hard.

(b) Hyperplane Non-Cover (H–$\mathsf{HYP-NC}_{\{\mathcal{C},\mathcal{V}\}}$)

The Hyperplane Non-Cover problem ($\mathsf{HYP-NC}$) is the solvability of homogeneous linear in-equations in F_p^N. The $\mathsf{HYP-NC}$ problem over Z_p^N for $p \geq 3$ includes the $\mathsf{3COL}$ problem

and is already NP-hard. To see this, let E be a graph on vertex set $[N]$ and consider the in-equations $x_i \neq x_j$ for indices $i, j \in [N]$ such that $\{i, j\} \in E$. If $p = 3$ that's all we need. Otherwise, add a variable y together with the in-equations $y \neq 0$, $x_i \neq ky$ for $i \in [N]$ and $k \in [p-3]$.

Hence, it remains to consider the H–$\mathsf{HYP-NC}_{\{\mathcal{C},\mathcal{V}\}}$ problem over F_2^N, which we need to examine $\bigcup\mathsf{HYP-NC}$ by Theorem 5.1. In this setting, let T be the set of all subspaces (not necessarily hyperplanes) of F_2^N. Then the set of constraints of $\bigcup\mathsf{HYP-NC}$ consists of $\{R_P \mid P \in T\}$ where $R_P(a)$ evaluates to $\mathbb{T}$ if and only if $a \notin P$. This problem is NP-hard, as it includes Subgroup Non-Cover by subgroups of index 4 which encompasses the 4COL problem. Hence, the former will be NP-hard using Theorem 5.1.

(c) Unique games (H–$\mathsf{UG}[k]_{\{\mathcal{C},\mathcal{V}\}}$ for $k \geq 3$)

$\mathsf{UG}[3]$ is a CSP with $w = 3$, $q = 2$, and $\mathcal{R} = \{\pi : [\![3]\!] \to [\![3]\!] \mid \pi \text{ is a permutation}\}$. Let

$$R^{\circ} := \bigcup_{\pi:(\forall i)(\pi(i)\neq i)} \pi.$$

Note that $R^{\circ} \in \bigcup\mathcal{R}$. Choosing R° as the constraint for every edge gives us the 3COL problem. Hence, from Theorem 5.1(b), H–$\mathsf{UG}[3]_{\{\mathcal{C},\mathcal{V}\}}$ is NP-hard.

Remark 5.3. *Our proof method also shows that* H–$\mathsf{UG}[k]_{\{\mathcal{C},\mathcal{V}\}}$ *is* NP*-hard for any* $k > 2$.

(d) k-Clique Isomorphism (H–$k\mathsf{CLQ}$–$\mathsf{ISO}_{\{\mathcal{C},\mathcal{V}\}}$ for $k \geq 3$)

Obviously, replacing the constraints of type R_k by $R_k \cup \neg R_k$ in an instance of H–$k\mathsf{CLQ}$–ISO we obtain an instance of $\bigcup$H–$k\mathsf{CLQ}$–ISO. This however just means omitting Constraints (1), and we obtain the $k\mathsf{CLQ}$ problem (deciding whether the graph contains a k-clique) which is NP-hard. Hence from Theorem 5.1(b), the H–$k\mathsf{CLQ}$–$\mathsf{ISO}_{\{\mathcal{C},\mathcal{V}\}}$ problem is NP-hard.

(e) Equality to a k-Clique (H–$\mathsf{EQ}_{k\mathsf{CLQ}\{\mathcal{C},\mathcal{V}\}}$)

We use the framework defined in the previous proof for the H–$\mathsf{EQ}_{\mathsf{ST}\{\mathcal{C},\mathcal{V}\}}$ problem. As mentioned in Remark 5.1, given a graph E_2, consider $\mathcal{K}$ to be the set of all possible k-cliques on n vertices and $E_1 = \emptyset$. In this setting, the $\bigcup\mathsf{EQ}_{\mathcal{K}}$ problem is equivalent to finding a k-clique on E_2 which is NP-hard.

Remark 5.4. *The above proof could also serve as an alternate proof for Theorem 5.5(d).*

(f) Equality to a Hamiltonian Cycle (H–$\mathsf{EQ}_{\mathsf{HAMC}\{\mathcal{C},\mathcal{V}\}}$)

We use the framework defined in the previous proof for the H–$\mathsf{EQ}_{\mathsf{ST}\{\mathcal{C},\mathcal{V}\}}$ problem. Here, $\mathcal{K}$ is the set of all possible Hamiltonian cycles on n vertices and $E_1 = \emptyset$. For an arbitrary graph E_2. the $\bigcup\mathsf{EQ}_{\mathcal{K}}$ problem parametrized by E_1 and E_2 becomes equivalent to deciding if E_2 has a Hamiltonian cycle, which is NP-hard. □

5.4 Constraint index and relation revealing oracle

With constraint and relation index revealing oracles, we show that if the arity and the alphabet size are constant, any CSP satisfying certain modest requirements becomes NP-hard.

Theorem 5.6. *Let* S *be a* CSP *with constant arity* q *and constant alphabet size* w. H–$\mathrm{S}_{\{\mathcal{C},\mathcal{R}\}}$ *is*

NP-*hard if for every* $\alpha \in [\![w]\!]$*, there is a non-empty relation* $R_\alpha \in \mathcal{R}$ *such that* $(\alpha, \ldots, \alpha) \notin R_\alpha$.

Proof. To show that E–S is NP-hard, it will be reduced to the problem E–3SAT which is essentially the MONSAT problem with clauses of size 3. These are instances of 3SAT where the variables in each clause are either all positive or all negated. Their NP-completeness can be deduced, for example, from Schaefer's characterization [Sch78].

We first extend the relations for S as follows. Let $q' = (w-1)q + 1$ and let $\mathcal{R}' \subseteq [\![w]\!]^{q'}$ be the set of q'-ary relations that can be obtained as an extension of an element of $\mathcal{R} \setminus \{\emptyset\}$ from any q coordinates. Since q and w are constant, the cardinality of $\mathcal{R}'$ is also constant. We claim that $\bigcap_{R \in \mathcal{R}'} R = \emptyset$. Indeed, every $a \in [\![w]\!]^{q'}$ has a sub-sequence $(\alpha, \ldots, \alpha)$ of length q for some $\alpha \in [\![w]\!]$, therefore the extension of R_α from these q coordinates does not contain a. Let $\{R^0, R^1, \ldots, R^h\}$ be a minimal subset of $\mathcal{R}'$ such that $\bigcap_{i=0}^{h} R^i = \emptyset$. Since the empty relation is not in $\mathcal{R}'$, we have $h \geq 1$. Let us set $A^0 = \bigcap_{i \neq 1} R^i$ and $A^1 = \bigcap_{i \neq 0} R^i$. Then $A^0 \bigcap A^1 = \emptyset$, and because of the minimality condition, $A^0 \neq \emptyset$ and $A^1 \neq \emptyset$.

For a boolean variable x, we will use the notation $x^1 = x$ and $x^0 = \bar{x}$. The main idea of the proof is to encode a boolean variable x^1 by the relation A^1 and x^0 by A^0. We think about the elements of A^1 as satisfying x^1, and about the elements of A^0 as satisfying x^0. Then x^1 and x^0 can be both satisfied, but not simultaneously.

To implement the above idea, we extend the relations further, building on the above extension. We suppose without loss of generality that ℓ is a multiple of q', and we set $\ell' = \ell/q'$. Since q' is constant, MONSAT on ℓ' variables is still NP-hard. We take ℓ' pairwise disjoint blocks of size q' of the index set $[\ell]$ and on each block we consider relations $R^0, \ldots, R^h$. We denote by R^i_k the ℓ-ary relation which is obtained by extending R^i from the kth block. Observe that the relations R^i_k are just extensions of elements of $\mathcal{R}$.

After these preparations, we are ready to present the construction. Let $K = \bigwedge_{t=1}^{u} K_t$ be an instance of E–3SAT in ℓ' variables, with each 3-clause of the form $K_t = x^{b_t}_{i_{t,1}} \vee x^{b_t}_{i_{t,2}} \vee x^{b_t}_{i_{t,3}}$, where $i_{t,1}, i_{t,2}, i_{t,3}$ are indices from $[\ell']$ and b_t is either 0 or 1. Then we map K to the instance $\mathcal{C}$ whose constraints are

$$R^{b_t}_{i_{t,1}} \cup R^{b_t}_{i_{t,2}} \cup R^{b_t}_{i_{t,3}},$$

for each $t \in [u]$, and

$$C^j_k = R^j_k,$$

for each $k \in [\ell']$ and $j \in \{2, \ldots, h\}$. This is an instance of E–S since the three relations $R^{b_t}_{i_{t,1}}$, $R^{b_t}_{i_{t,2}}$ and $R^{b_t}_{i_{t,3}}$ are the extensions of the same relation in $\mathcal{R}$. It is quite easy to see that K is satisfiable if and only if $\mathcal{C}$ is satisfiable. Indeed, a satisfying assignment a for the $\mathcal{C}$ can be translated to a satisfying assignment for K by assigning 0 or 1 to x_k according to whether the kth block of a was in A^0_k or A^1_k (taking an arbitrary value if it was in none of the two). Similarly, a satisfying assignment b for K can be translated to a satisfying assignment a for $\mathcal{C}$ by picking any element of $A^{b_k}_k$ for the kth block of a. □

An immediate consequence is that under the same conditions H–$S_{\{\mathcal{C}\}}$ is NP-hard too. For an application of this consequence, let LINEQ stand for the CSP in which that alphabet is identified with a finite field F and the ℓ-ary constraints are linear equations over F.

Claim 5.1. H–$\mathsf{LINEQ}_{\{\mathcal{C}\}}$ *is* NP*-hard.*

Proof. For each $i \in \ell$, we pick two equations: $x_i = 0$ and $x_i = 1$. Observe that $x_i = 0$ is the same as $\{0\}^i$, the ℓ-ary extension of the unary relation $\{0\}$ on the ith position and we have the same if we replace 0 by 1. By the above observation, the H–CSPs built from relations satisfying this condition are are NP-hard. □

5.5 Monotone Graph Properties

In this section, we consider monotone graph properties in the context of constraint index revealing oracle. Recall from the discussion on the monotone graph property framework in the trial and error setting (Chapter 1, Section 2.0.2) that the CSP for a property $\mathcal{P}$, $S_{\mathcal{P}}$, uses only one kind of relation $\mathcal{R} = \{\mathsf{Neg}\}$. This makes it irrelevant if the relation is revealed or not and hence makes the $\mathcal{O}_{\{\mathcal{C}\}}$ and $\mathcal{O}_{\{\mathcal{C},\mathcal{R}\}}$ oracles equivalent for this framework. We then study various monotone graph properties like Spanning Tree, Cycle Cover, etc. (which are polynomial time decidable in the un-hidden setting) and prove that the problems become NP-hard with the constraint index revealing oracle.

Let us examine the CSPs for a property $\mathcal{P}$, $S_{\mathcal{P}}$, in the trial and error setting. In particular, as already noted in Section 5.3, the case with the $\mathcal{O}_{\{\mathcal{C},\mathcal{V}\}}$ oracle is polynomial time equivalent to the "normal" problem. As $|\mathcal{R}| = 1$ in this case, the restricted union of arity extension ($\bigcup$E–$\mathcal{R}$) and the union of arity extension ($\bigcup$X–$\mathcal{R}$) are the same. Now, the arity extension is given by X–$\mathcal{R}$ $= \{\mathsf{Neg}^e \mid e \in \binom{n}{2}\}$, where $\mathsf{Neg}^e(\alpha_1, \ldots, \alpha_{\binom{n}{2}}) = \neg\alpha_e$, and therefore $\bigcup$X–$S_{\mathcal{P}}$= $\{\vee_{e\in E'}\mathsf{Neg}^e \mid E' \subseteq \binom{n}{2}\}$ where $\vee$ denotes the OR operator. That is, the relations in $\bigcup$X–$S_{\mathcal{P}}$ are parametrized by sets E' of possible edges where the relation corresponding to E' is equivalent to one where at least one edge of the proposed graph is not there in E'.

As a consequence, the $\bigcup$X–$S_{\mathcal{P}}$ problem becomes the following: Given a graph $G = (V, E)$, and edge sets on n vertices $E_1, \ldots, E_m \subseteq \binom{[n]}{2}$, does there exist an $A \in W_{\mathcal{P}}$ such that $A \subseteq E$ and A excludes at least one edge from each E_i? While every subset of the edge set $\binom{n}{2}$ is in our disposal, in actual applications, the tricky part is to come up with appropriate subsets $E_1, \ldots, E_m$, which, together with the minimal instances of the graph property in question, yield that the resulting $\bigcup$X–$S_{\mathcal{P}}$ problem is hard. From Theorem 5.3, the complexity of H–$S_{\mathcal{P}\{\mathcal{C}\}}$ can be analyzed by considering the complexity of $\bigcup$X–$S_{\mathcal{P}}$. This will be the main content in our proofs for the various properties (defined in Section 2.0.2) considered in Theorem 5.7.

This framework along with the extensions naturally extends to graphs with one or more designated vertices, to directed graphs, while monotone decreasing properties can be treated by replacing Neg with Id, the identity function.

Theorem 5.7. *The following problems are* NP*-hard:(a)* H–$\mathsf{ST}_{\{\mathcal{C}\}}$*, (b)* H–$\mathsf{DST}_{\{\mathcal{C}\}}$*, (c)* H–$\mathsf{UCC}_{\{\mathcal{C}\}}$*, (d)* H–$\mathsf{DCC}_{\{\mathcal{C}\}}$*, (e)* H–$\mathsf{BPM}_{\{\mathcal{C}\}}$*, (f)* H–$\mathsf{DPATH}_{\{\mathcal{C}\}}$*, (g)* H–$\mathsf{UPATH}_{\{\mathcal{C}\}}$*.*

Proof. We show that the following problems in the hidden model with the constraint index revealing oracle are NP-hard. In each case, we construct an instance of the $\bigcup$X–S$_{\mathcal{P}}$ such that it becomes equivalent to a known NP-hard problem and using Theorem 5.3(b) we can conclude that the hidden version, H–S$_{\mathcal{P}\{\mathcal{C}\}}$, is NP-hard.

(a) Spanning Tree (H–$\mathsf{ST}_{\{\mathcal{C}\}}$)

When $\mathcal{P}$ is *connectedness*, $W_{\mathcal{P}}$ is the set of *Spanning Trees* on n-vertices.
Given $G = (V, E)$, for every vertex $v \in V$, we consider $\binom{n-1}{3}$ edge sets E_{vijk} where

$$E_{vijk} := \{(v, i), (v, j), (v, k)\} \quad 1 \leq i < j < k \leq n.$$

With this choice of E_{vijk}s the $\bigcup$X–ST problem asks if there exists a spanning tree in G which avoids at least one edge from each E_{vijk}. This is equivalent to asking that every vertex v is incident to at most two edges of the spanning tree A. Spanning trees with this property are just Hamiltonian paths in G. In other words, the $\bigcup$X–ST problem is equivalent to asking if G contains a Hamiltonian path i.e. the $\mathsf{HAM-PATH}$ problem in G. Hence, the NP-hard $\mathsf{HAM-PATH}$ in G problem reduces to the H–$\mathsf{ST}_{\emptyset}$ problem in G.

(b) Directed Spanning Tree (H–$\mathsf{DST}_{\{\mathcal{C}\}}$)

Similar to the previous case, $W_{\mathcal{P}}$ is the set of directed spanning trees rooted at vertex 1. Let $G = (V, E)$, be a directed planar graph such that the in-degree and the out-degree for every vertex is at most 2. The $\mathsf{DHAM-PATH}$ problem in G, i.e. determining if there exists directed Hamiltonian path ending at node 1 in G, is NP-hard [GJ79]. Our goal is to reduce the $\mathsf{DHAM-PATH}$ problem in G to the H–$\mathsf{DST}_{\{\mathcal{C}\}}$ problem in G.
For every vertex $v \in V$, of in-degree 2 we consider the edge sets E_v where $E_v := \{(i, v) \mid (i, v) \in E\}$ with $|E_v| \leq 2$ by our choice of G. In addition, for every vertex $v \in V$, of out-degree 2 we consider the edge sets E^v where $E^v := \{(v, i) \mid (v, i) \in E\}$. With these E_vs and E^vs, the $\bigcup$X–DST problem asks if there exists a directed spanning tree rooted at vertex 1 that contains at most one edge coming in and at most one edge originating from every vertex. These constraints restrict the directed spanning tree, A to be a $\mathsf{DHAM-PATH}$ in G, analogously to the undirected case. Hence, the NP-hard $\mathsf{DHAM-PATH}$ problem reduces to the H–$\mathsf{DST}_{\{\mathcal{C}\}}$ problem in G.

(c) Undirected Cycle Cover (H–$\mathsf{UCC}_{\{\mathcal{C}\}}$)

Here, $W_{\mathcal{P}}$ is the set of *undirected cycle covers* on n-vertices. From Hell et al. [HKKK88] we know that the problem of deciding whether a graph has a UCC that does not use the cycles of length, (say) 5 is NP-hard. We construct an equivalent instance of $\bigcup$X–UCC as follows. We choose the edge sets $E_C := \{e \mid e \in C\}$ ranging over every length 5 cycle C in G. Then, a UCC satisfying the above conditions cannot contain any 5-cycles. Hence, an NP-hard problem reduces to the H–$\mathsf{UCC}_{\{\mathcal{C}\}}$ problem in G.

(d) Directed Cycle Cover (H–$\mathsf{DCC}_{\{\mathcal{C}\}}$)

In this case, $W_{\mathcal{P}}$ is the set of *directed cycle covers* on n-vertices. The proof follows similar to the undirected case. The NP-hard problem we are interested in is determining if a graph has a DCC that does not use cycles of length 1 and 2 [GJ79]. This problem can be expressed as $\bigcup$X–DCC by choosing the edge sets $E_C := \{e \mid e \in C\}$ for every length 1 and length 2 cycle C in G.

(e) Bipartite Perfect Matching (H–$\mathsf{BPM}_{\{\mathcal{C}\}}$)

Here, $W_{\mathcal{P}}$ is the set of *perfect matchings* in a complete bipartite graph with n-vertices on each side. There is a one-to-one correspondence between perfect matchings in a bipartite graph $G = (A \cup B, E)$ with n vertices on each side and the directed cycle covers in a graph $G' = (V', E')$ on n vertices. Every edge $(i, j) \in E'$ corresponds to an undirected edge $\{i_A, j_B\} \in E$. With this correspondence the H–$\mathsf{BPM}_{\{\mathcal{C}\}}$ problem in G is equivalent to the H–$\mathsf{DCC}_{\{\mathcal{C}\}}$ problem in G'. Thus, from Theorem 5.7(d) the former becomes NP-hard.

(f) Directed Path (H–$\mathsf{DPATH}_{\{\mathcal{C}\}}$)

We consider $W_{\mathcal{P}}$ as the set of *directed paths* from s to t. It is known that given a layout of a directed graph on a plane possibly containing crossings, the problem of deciding whether there is a crossing-free path from s to t is NP-hard [KLN91]. This condition can be expressed by picking the each edge set E_i as the set of pairs of edges that cross.

(g) Undirected Path (H–$\mathsf{UPATH}_{\{\mathcal{C}\}}$)

In this case, $W_{\mathcal{P}}$ is the set of *undirected paths* from s to t. We can apply the same proof method as the one used for the H–$\mathsf{DPATH}_{\{\mathcal{C}\}}$ problem on an undirected graph. □

5.6 H-CSPs with a promise on instances

In this section we consider an extension of the H–CSP framework where the instances satisfy some property. Formally, let S be a CSP, and let PROM be a subset of all instances. Then S with *promise* PROM is the CSP $\mathrm{S}^{\mathsf{PROM}}$ whose instances are only elements of PROM. One such property is *repetition freeness* where the constraints of an instance are pairwise distinct. We denote by RF the subset of instances satisfying this property. For example $\mathsf{1SAT}^{\mathsf{RF}}$, (as well as H–$\mathsf{1SAT}^{\mathsf{RF}}$) consists of pairwise distinct literals. Such a requirement is quite natural in the context of certain graph problems where the constraints are an inclusion (or non-inclusion) of possible edges. The promise H–CSPs framework could also be suitable for discussing certain graph problems on special classes of graphs (e.g, connected graphs, planar graphs, etc.). For instance, the BPM problem can also be viewed as the problem of asking for a perfect matching on a graph that is *promised to be bipartite*.

We would like to prove an analogue of the transfer theorems for CSPs with promise. For the sake of simplicity, we develop this only for the simplest case i.e. with the $\{\mathcal{C}, \mathcal{V}\}$-revealing oracle. To achieve this, the closure by union CSP extension for a CSP with promise is detailed here. Given a promise PROM for the CSP S of type $\mathcal{R} = \{R_1, \ldots, R_s\}$, the corresponding promise $\bigcup$PROM for $\bigcup$S is defined quite naturally as follows. An instance $\mathcal{C} = \{C_1, \ldots, C_m\}$ of S,

where $C_j = R_{k_j}(x_{j_1}, \ldots, x_{j_q})$, is *included* in an instance $\mathcal{C}' = \{C'_1, \ldots, C'_m\}$ of $\bigcup \mathrm{S}$ if for every $j = 1, \ldots, m$ $C'_j = Q_j(x_{j_1}, \ldots, x_{j_q})$ for $Q_j \in \bigcup \mathcal{R}$ such that $R_{k_j} \subseteq Q_j$. Basically, if for every j, the constraint $C_j \in \mathcal{R}$ is contained in the corresponding $C'_j \in \bigcup \mathcal{R}$, then $\mathcal{C} = \{C_1, \ldots, C_m\}$ is included in $\mathcal{C}' = \{C'_1, \ldots, C'_m\}$. Then, $\bigcup \mathsf{PROM}$ is defined as the set of instances in $\mathcal{C}' \in \bigcup \mathrm{S}$ which include $\mathcal{C} \in \mathsf{PROM}$. Note that it is not necessary for $\mathcal{C}'$ to satisfy the promise PROM. In order for the transfer theorem to work, the notion of a solution in this setting is modified. A *solution under promise* for $\mathcal{C}' \in \bigcup \mathsf{PROM}$ has to satisfy two criteria:(a) it is a satisfying assignment when $\mathcal{C}'$ includes a satisfiable instance $\mathcal{C} \in \mathsf{PROM}$, and (b) it is EXCEPTION when $\mathcal{C}'$ is unsatisfiable. However, in the ambiguous case when all the instances $\mathcal{C} \in \mathsf{PROM}$ included in $\mathcal{C}'$ are unsatisfiable but $\mathcal{C}'$ itself is still satisfiable, the output can either be a satisfying assignment or EXCEPTION. An algorithm *solves* $\bigcup \mathrm{S}^{\bigcup \mathsf{PROM}}$ *under promise* if $\forall \mathcal{C}' \in \bigcup \mathsf{PROM}$, it outputs a solution under promise.

Using the above definition in the transfer theorem's proof allows the algorithm for $\text{H–S}^{\mathsf{PROM}}_{\{\mathcal{C},\mathcal{V}\}}$ to terminate, at any moment of time, with the conclusion NO as soon as it gets enough information about the instance to exclude satisfiability and without making further calls to the revealing oracle. In some ambiguous cases, it can still call the oracle with an assignment which satisfies the $\bigcup \mathrm{S}^{\bigcup \mathsf{PROM}}$-instance. Other cases when the satisfiability of a $\bigcup \mathrm{S}^{\bigcup \mathsf{PROM}}$ instance implies the existence of a satisfiable included $\mathrm{S}^{\mathsf{PROM}}$ instance lack this ambiguity. With these notions the proof of Theorem 5.1 goes through and we obtain the following.

Theorem 5.8. *Let* $\mathrm{S}^{\mathsf{PROM}}$ *be a promise* CSP. *(a) If* $\bigcup \mathrm{S}^{\bigcup \mathsf{PROM}}$ *is solvable under promise in time* T *then* $\text{H–S}^{\mathsf{PROM}}_{\{\mathcal{C},\mathcal{V}\}}$ *is solvable in time* $O((T + s \times \mathrm{comp}(\mathcal{R})) \times m \times \min\{\dim(\bigcup \mathcal{R}), |W_q|\})$. *(b) If* $\text{H–S}^{\mathsf{PROM}}_{\{\mathcal{C},\mathcal{V}\}}$ *is solvable in time* T *then* $\bigcup \mathrm{S}^{\bigcup \mathsf{PROM}}$ *is solvable under promise in time* $O(T \times m \times \mathrm{comp}(\bigcup \mathcal{R}))$.

Theorem 5.8 is applied to various problems under the promise of repetition freeness in Theorem 5.9 before which the kWEIGHT problem which has not been before is explicitly defined here for completeness.

Definition 5.5 (k-Weight problem)**.** *The k-weight problem (kWEIGHT) decides if a Boolean string has Hamming weight at least k i.e. the number of 1's in the string is at least k. Formally, we have $w = 2$, $q = 1$, $\mathcal{R} = \{\{0\}\}$ and W consists of words of length ℓ having Hamming weight at least k. An instance of kWEIGHT is a collection $(C_1, \ldots, C_m)$ of constraints of the form $x_{i_j} = 0$ or formally, $C_j = \{0\}^{i_j}$. The satisfying string for these constraints is $b \in \{0,1\}^\ell$ where $b_t = 0$ if and only if $t \in \{i_1, \ldots, i_m\}$.*

Theorem 5.9. *(a) Repetition free* H–1SAT *with constraint index revealing oracle is easy, that is* $\text{H–1SAT}^{\mathsf{RF}}_{\{\mathcal{C}\}} \in \mathrm{P}$*; (b)* $\text{H–}k\text{WEIGHT}_{\{\mathcal{C}\}}$ *is* NP*-hard for certain k, but* $\text{H–}k\text{WEIGHT}^{\mathsf{RF}}_{\{\mathcal{C}\}} \in \mathrm{P}$ *for every k; (c) Repetition free* H–2SAT*, with constraint index revealing oracle, that is,* $\text{H–2SAT}^{\mathsf{RF}}_{\{\mathcal{C}\}}$ *is* NP*-hard. (d) Repetition free* H–2COL*, that is* $\text{H–2COL}^{\mathsf{RF}}_{\{\mathcal{C}\}}$ *is* NP*-hard.*

Proof. Notice that problems are considered with the $\{\mathcal{C}\}$-revealing oracle but the transfer theorem uses the $\{\mathcal{C}, \mathcal{V}\}$-revealing oracle. However, when $\mathsf{PROM} = \mathsf{RF}$, it is possible to consider the relations that describe the constraints to be extended to their ℓ-ary counterparts in which

case the $\mathcal{O}_{\{\mathcal{C}\}}$ and $\mathcal{O}_{\{\mathcal{C},\mathcal{V}\}}$ oracles become equivalent. Now, the proof proceeds by applying Theorem 5.8 to the appropriate problems.

(a) A repetition free instance of $\bigcup$1SAT is $\mathcal{C} = \{C_1, \ldots, C_m\}$, where each C_j is a disjunction of literals from $\{x_1, \overline{x}_1, \ldots, x_\ell, \overline{x}_\ell\}$ such that there exist m distinct literals $z_1, \ldots, z_m$ with z_j from C_j. A conjunction of literals is satisfiable, if for every $i \in [\ell]$, the literals x_i and $\overline{x}_i$ are not both among them. Hence an algorithm which solves H–1SAT$^{\mathsf{RF}}_{\{\mathcal{C}\}}$ under promise can proceed as follows. Using a maximum matching algorithm it selects pairwise different variables $x_{i_1}, \ldots, x_{i_m}$ such that x_{i_j} or $\overline{x}_{i_j}$ is in C_j. If such a selection is not possible it returns EXCEPTION. Otherwise it can trivially find a satisfying assignment.

(b) An instance of $\bigcup k$WEIGHT is $\mathcal{C}' = \{C'_1, \ldots, C'_m\}$, where there exist subsets $S_1, \ldots, S_m$ of $[\ell]$ such that the relation for C'_j is the set of assignments $\{\mathbf{a} \in \{0,1\}^\ell : a_i = 0 \text{ for some } i \in S_j\}$. Finding a satisfying instance of $\bigcup\mathcal{R}$ is therefore equivalent to finding a hitting set (a transversal) of size (at most) $\ell - k$ for the hypergraph $\{S_1, \ldots, S_m\}$. This problem is NP-hard for, say, $0.01\ell < k < 0.99\ell$ [NH81]. Hence, the H–kWEIGHT$_{\{\mathcal{C}\}}$ problem for this range of ks is NP-hard.

A kWEIGHT$^{\mathsf{RF}}$-instance included in an instance of $\bigcup k$WEIGHT$^{\bigcup\mathsf{RF}}$ corresponding to subsets $S_1, \ldots, S_m$ consists of constraints $x_{i_j} \neq 0$ for m different indices $i_1, \ldots, i_m$ with $i_j \in S_j$. Obviously, such a set of constraints is satisfiable by an element of W if and only if $m \leq \ell - k$. These observations immediately give the following efficient solution under promise for $\bigcup k$WEIGHT$^{\bigcup\mathsf{RF}}$. If $m > \ell - k$ we return EXCEPTION. Otherwise, using a maximum matching algorithm we find m different places $i_1, \ldots, i_m$ with $i_j \in S_j$ (which must exist by the promise) and return an assignment from W which can be found in an obvious way.

(c) For this case, we reduce 3SAT to $\bigcup$2SAT$^{\bigcup\mathsf{RF}}$. Let $\phi = \bigwedge_{j=1}^m C_j$ be a $3-$CNF where

$$C_j = x_{j1}^{b_1} \vee x_{j2}^{b_2} \vee x_{j3}^{b_3}.$$

(Here $b_i \in \{0,1\}$ and x^1 denotes x, x^0 stands for $\overline{x}$.) For each $j = 1, \ldots, m$ we introduce a new variable y_j. We will have $2m$ new clauses:

$$C'_j = x_{j1}^{b_1} \vee x_{j2}^{b_2} \vee x_{j3}^{b_3} \vee y_j^0 \quad \text{and} \quad C''_j = x_{j1}^{b_1} \vee x_{j2}^{b_2} \vee x_{j3}^{b_3} \vee y_j^1 \quad \text{for each } j.$$

Define $\phi' = \bigwedge_{j=1}^m (C'_j \wedge C''_j)$. Then ϕ is satisfiable if and only if ϕ' is satisfiable. In fact, there is a 1 to 2^m correspondence between the assignments satisfying ϕ and those satisfying ϕ': only the values assigned to the first ℓ variables matter. Also, the included constraints $(x_{j1}^{b_1} \vee y_j^1)$ and $(x_{j1}^{b_1} \vee y_j^0)$ for all $j = 1, \ldots, m$ form a system of $2m$ different $2-$CNFs. Furthermore, if ϕ' is satisfied by an assignment then we can select a satisfiable system of $2m$ pairwise distinct sub-constraints: for each j we pick $s \in \{1,2,3\}$ such that $x_{j_s}^{b_s}$ is evaluated to 1 and take $(x_{j_s}^{b_s} \vee y_j^1)$ and $(x_{j_s}^{b_s} \vee y_j^0)$ for $j = 1, \ldots, m$.

(d) Here the alphabet is $[\![2]\!] = \{0,1\}$, $q = 2$, $\mathcal{R}$ has one element "$\neq$", that is, the strings $\{10, 01\}$. An instance of 2COL$^{\mathsf{RF}}$ consists of a set of constraints of the form $x_u \neq x_v$ for m

pairwise distinct unordered pairs (u, v) from $\{1, \ldots, \ell\}$ (corresponding to the edges of a graph). An instance of $\bigcup$2COL is a collection $\{C_1, \ldots, C_m\}$, where each C_j is a disjunction of constraints of the form $x_u \neq x_v$. In an equivalent view, an instance of $\bigcup$2COL can be described by the collection of edge sets (graphs) $E_1, \ldots, E_m$ on vertex set $[n]$ and a satisfying assignment can be described by a colouring $c : \{1, \ldots, n\} \to \{0, 1\}$ such that for every j there exists an edge $e_j \in E_j$ with endpoints having different colours. It is clear that if the edge sets $E_1, \ldots, E_m$ are disjoint then the instance is repetition free and the solutions under promise coincide with the solutions in the normal sense.

Let $E_1, \ldots, E_m$ be edge sets describing an instance of $\bigcup$2COL. Put $s_j = |E_j|$. For each j we introduce $2s_j$ new vertices: $uvj1, uvj2$ for each $(u, v) \in E_j$, $2s_j$ new one-element edge sets $E_{uvj1} = \{(u, uvj1)\}$ and $E_{uvj2} = \{(v, uvj2)\}$; while E_j is replaced with an edge set E'_j consisting of s_j edges: $(uvj1, uvj2)$ for each $(u, v) \in E_j$. It turns out that the $\bigcup$2COL problem on the $n + 2\sum_{j=1}^{m} s_j$ vertices with the new $m + 2\sum_{j=1}^{m} s_j$ edge sets is equivalent to the original one and solutions of the two problems can be easily (and efficiently) mapped to each other. The new edge sets are pairwise disjoint and hence the repetition free version of the new $\bigcup$2COL problem is the same as the non-promise version.

Theorem 5.6 shows that non-promise $\bigcup$2COL is NP-hard. By the reduction above, so is its repetition free version. □

On group isomorphism. Isomorphism of a hidden multiplication table with a given group, a problem discussed in [BCZ13], can also be cast in the framework of promise H–CSPs. We consider the following problem GROUPEQ (equality with a group from a class). Let $\mathcal{G}$ be a family of groups on the set $[k]$, that is, a set of multiplication tables on $[k]$ such that each multiplication table defines a group. The task is to decide whether a hidden group structure $b(\,,\,)$ is equal to some $a(\,,\,)$ from $\mathcal{G}$ and if so, to find such an $a(\,,\,)$. (Note that a solution of the latter task will give the whole table for $b(\,,\,)$.)

GROUPEQ($\mathcal{G}$) can be defined as a promise CSP. First we consider the CSP ENTRIES($\mathcal{G}$) with the following parameters and type. We have $w = k$, $W = \mathcal{G}$, $\mathcal{R} = \{\{w\} : w \in [k]\}$, $\ell = k^2$. It will be convenient to consider assignments as $k \times k$ tables with entries from $[k]$, that is, functions $[k]^2 \to [k]$. (Implicitly, we use a bijection between the index set $\{1, \ldots, \ell\}$ and $[k]^2$.) The number of constraints is $m = k^2$ and an instance is a collection $x_{(u_h, v_h)} = b_h$, where $h = 1, \ldots, m$, and $u_h, v_h, b_h \in [k]$. Thus the assignment satisfying ENTRIES($\mathcal{G}$) are $k \times k$ multiplication tables from $\mathcal{G}$ which have prescribed values at k^2 (not necessarily distinct) places.

We say that an instance for ENTRIES($\mathcal{G}$) belongs to the promise GROUP if two conditions are satisfied. Firstly, there is one constraint for the value taken by each place. Formally, the map $\tau : h \mapsto (u_h, v_h)$ is a bijection between $\{1, \ldots, m\}$ and $[k]^2$. As a consequence, by setting $b(u, v) := b_{\tau^{-1}(u,v)}$, we have a constraint $x_{u,v} = b(u, v)$ for pair $(u, v) \in [k]^2$. The second – essential – condition is that the multiplication given by $b(\,,\,)$ defines a group structure on $[k]$. The promise problem GROUPEQ($\mathcal{G}$) is the problem ENTRIES($\mathcal{G}$)$^{\text{GROUP}}$.

We consider the promise problem H–$\mathsf{ENTRIES}(\mathcal{G})^{\mathsf{GROUP}}_{\{V\}}$ (which we denote by H–$\mathsf{GROUPEQ}(\mathcal{G})$ for short) and the corresponding problem $\bigcup\mathsf{ENTRIES}(\mathcal{G})^{\bigcup\mathsf{GROUP}}_{\{V\}}$ (short notation: $\bigcup\mathsf{GROUPEQ}(\mathcal{G})$). In this H–CSP, if $a(\,,\,)$ is different from $b(\,,\,)$, the oracle reveals a pair (u,v) such that $a(u,v) \neq b(u,v)$.

We note here that the case of H–$\mathsf{GROUPEQ}(\mathcal{G})$ where $\mathcal{G}$ consists of all isomorphic copies of a group G in fact covers the problem of finding an isomorphism with G discussed in [BCZ13]. For that problem, the input to the verification oracle is a bijection $\phi : [k] \to G$. Recall that $b(\,,\,)$ encodes the hidden group structure, and we assume G is specified by the binary relation $g(\,,\,)$. Then, in the case when ϕ is not an isomorphism, the oracle has to reveal $u, v \in [k]$ such that, the product $g(\phi(u), \phi(v))$ does not equal $\phi(b(u,v))$ in G. Indeed, given ϕ we can define (and even compute) the multiplication $a_\phi(\,,\,)$ on $[k]$ – by taking $a_\phi(x,y) = \phi^{-1}(g(\phi(x),\phi(y))$ – such that ϕ becomes an isomorphism from the group given by $a_\phi(\,,\,)$ to G. Then ϕ is an isomorphism from the group given by $b(\,,\,)$ if and only if $a_\phi(\,,\,) = b(\,,\,)$. Furthermore, if it is not the case then the oracle given in [BCZ13] reveals a pair (u,v) such that $a_\phi(u,v) = b(u,v)$, exactly what is expected from a revealing oracle for H–$\mathsf{GROUPEQ}(\mathcal{G})$.

An instance of $\bigcup\mathsf{ENTRIES}(\mathcal{G})$ consists of k^2 constraints expressing that $a(u_h, v_h) \in S_h$ where $S_h \in 2^{[k]} \setminus \emptyset$ for $h \in [m] = [k^2]$. An instance of the promise version $\bigcup\mathsf{GROUPEQ}(\mathcal{G})$ (which is equal to $\bigcup\mathsf{ENTRIES}(\mathcal{G})^{\bigcup\mathsf{GROUP}}_{\{V\}}$) should satisfy that $\{(u_h, v_h) : h = 1, \ldots, m\} = [k]^2$, that is, our constraints are actually $x_{(u,v)} \in S(u,v)$ for a map $S(\,,\,) : [k]^2 \to 2^{[k]}$. Furthermore, there is a map $b(\,,\,) : [k]^2 \to [k]$ with $b(u,v) \in S(u,v)$ for every $(u,v) \in [k]^2$ such that $b(\,,\,)$ gives a group structure.

Now we are ready to reprove Theorem 11 in [BCZ13]. Note that our proof is considerably shorter than the original proof.

Theorem 5.10. *The problem* H–$\mathsf{GROUPEQ}(\mathcal{G})$ *is* NP*-hard.*

Proof. Let p be a prime. We show that finding Hamiltonian cycles in Hamiltonian digraphs of size p is reducible in polynomial time to H–$\mathsf{GROUPEQ}(\mathcal{G})$. The former problem is NP-hard, since an algorithm that in polynomial time finds a Hamiltonian cycle in a Hamiltonian digraph obviously can decide if an arbitrary digraph G has a Hamiltonian cycle: it just runs on G and then tests if the outcome is indeed a Hamiltonian cycle.

Choose $\mathcal{G}$ as the set of all group structures on $[p]$. As every group having p elements is isomorphic to Z_p, $\mathcal{G}$ coincides with the group structures on $[p]$ isomorphic to Z_p. We essentially translate the arguments given in [BCZ13] to the setting of $\bigcup\mathsf{GROUPEQ}$ as follows. This suffices to prove the statement due to the transfer theorem stated in Theorem 5.8.

Let $([p], E)$ be a Hamiltonian directed graph (without loops) on $[p]$. Fix $z \in [p]$. For $u \in [p]$, let $S(u,z) = \{v : (u,v) \in E\}$, and $S(u,v) = [p]$ for $v \neq z$. Let $\phi : [p] \to \{0, \ldots, p-1\} = Z_p$ be a bijection that defines a Hamiltonian cycle in $([p], E)$. Then $b(x,y) = a_\phi(x,y) := \phi^{-1}(\phi(x) + \phi(y))$ gives a group structure on $[p]$ (isomorphic to Z_p via ϕ) consistent with the constraints given by

$S(\ ,\ )$. Conversely, if $b(\ ,\ )$ gives a group structure (necessarily isomorphic to Z_p) consistent with $S(\ ,\ )$ then the pairs $(u, b(u, z))$ $(u \in [p])$ form a Hamiltonian cycle in $([p], E)$. Thus finding Hamiltonian cycles in Hamiltonian digraphs on p points can be reduced to $\bigcup$GROUPEQ on p elements. □

As an example, suppose we have $p = 3$, and the edges are $2 \to 1, 1 \to 3, 3 \to 2$. Then set $z = 2$, so $\phi(2) = 1$, $\phi(1) = 2$, $\phi(3) = 0$. (That is, i is the $\phi(i)$th vertex to be visited in this Hamiltonian cycle, where $\phi(i)$ should be understood as modulo p.) It can be verified that $b(x, 2) \in S(x, 2) = \{y : (x, y) \in E\}$. On the other hand, if we set $b(x, y)$ to be isomorphic to Z_p by the correspondence just given by ϕ, then the path $(u, b(u, 2))$ forms a Hamiltonian cycle.

Chapter 6

Probabilistic trials for hidden satisfiability problems

Question: *What is the minimum amount of information needed to solve* 2SAT *efficiently?*

The previous chapter discussed the consequence of using a wide range of oracles in the trial and error model for any classical CSP whose input is unknown. The trials proposed however, consisted of a single trial and algorithms proceeded in a deterministic fashion searching the solution space for the next possible assignment. Given the wide range of oracles discussed, the complexity of a single problem, say 2SAT, fluctuated widely between being intractable i.e. NP-hard or trivial i.e. the hidden instance can be learned completely. However, the question remains if there is any meaningful modification to the model that would allow the solving of simple CSPs without learning every clause in the instance. Since the idea of an unknown input roughly corresponds to the limit on the information one has about the instance, it is natural to phrase the question as the minimum amount of information needed to solve a tractable CSP in polynomial time. Considering SAT's position as a flagship CSP, the question this chapter tries to answer is in the context of 1SAT and 2SAT which can be solved efficiently when there is full information about the instance. On another note, as it is also possible to find the ground states for Q1SAT and Q2SAT efficiently, a quantization of the trial and error approach is also of independent interest. This chapter aims to answer both these questions.

As far as modifications go, a first attempt is to randomize the trials made to any oracle. However, as the complexity of the problem lies in the relational structure imposed by the transfer theorems, randomizing the trials does not significantly decrease the complexity. Then, randomizing the other end of the model i.e. the oracle would make for a second attempt.

The Random Oracle

Randomizing the oracle amounts to the case where the information revealed is a random violated clause. This can be thought of as a relaxation of the clause-index revealing oracle $\mathcal{C}$ as repeating the each trial many times over could reveal the indices of a all violated clauses and not just a single violated clause index. Of course, in this case, the random oracle – denoted by $\mathcal{O}_{\text{RAND}}$ – is not adversarial in nature. So, given a H–SAT instance, accessed via the $\mathcal{O}_{\text{RAND}}$ oracle, a single trial, repeatedly proposed, reveals the indices of all the clauses violated by that trial. This model corresponds more closely to real-life applications where one would expect to see observe a

randomly violated constraint and not an adversarially chosen one. However, this seems to be a powerful assumption with respect to the question of interest as it is possible to learn some clause of a H–kSAT instance in $O(n^k)$ time which would automatically translate to polynomial time algorithms for H–1SAT and H–2SAT. For ease of notation, the hidden SAT problem with the random oracle will be indicated as H–$\mathsf{SAT}_{\mathrm{RAND}}$

Theorem 6.1. *There exists an algorithm in the random oracle trial and error model which either finds a satisfying assignment to a* H–$\mathsf{kSAT}_{\mathrm{RAND}}$ *instance on n variables containing m clauses or learns the instance in time $O(mn^k)$ where the $O(\cdot)$ notation hides a constant dependent on k.*

Proof. Let the hidden instance be defined on variables $\mathbf{x} = \{x_1, \ldots, x_n\}$ with clauses $C_1, \ldots, C_m$. Without loss of generality, assume that all assignments tried fail to satisfy the instance as otherwise, a satisfying assignment has been found aborting the algorithm's run. Recall that the tag of a clause is the set of literals contained in the clause. It will be shown that for each clause tag T involving k literals, it is possible to learn which clauses are of this tag in $O(m)$ times. Since there could be at most $O(n^k)$ clause tags with arity k, the $O(mn^k)$ time follows.

Assume the clause tag to learn is $(x_1 \vee x_2 \vee \ldots \vee x_k)$; other clause tags will have an analogous process. The first trial is the Boolean string $\mathbf{a}_1 = 0^n$ which returns a set S of violated clauses followed by the trial $\mathbf{a}_2 = 0^k 1^{(n-k)}$ which returns S' as the set of violated clauses. Now, consider the intersection $S \cap S'$ – which can be found in $O(m)$ time.

The only clauses C_j in $S \cap S'$ are those violated both by $\mathbf{a}_1$ and $\mathbf{a}_2$. Suppose C_j contained a literal in $\{x_{k+1}, \ldots, x_n\}$, then $\mathbf{a}_2$ would have satisfied it and if C_j contained a literal in $\{\bar{x}_{k+1}, \ldots, \bar{x}_n\}$, then $\mathbf{a}_1$ would have satisfied it. If on the other hand C_j contained a literal in $\{\bar{x}_1, \ldots, \bar{x}_k\}$, then both $\mathbf{a}_1$ and $\mathbf{a}_2$ would have satisfied it leading to the conclusion that C_j only contains literals from the set $\{x_1, \ldots, x_k\}$. In fact, $S \cap S'$ only contains clauses all of whose literals are in $\{x_1, \ldots, x_j\}$. Note that this does not guarantee that each $C_j \in S \cap S'$ has tag $T = \{x_1, \ldots, x_k\}$, only that the tag $T \subseteq \{x_1, \ldots, x_k\}$.

To ensure that each C_j of interest has tag $T = \{x_1, \ldots, x_k\}$, one has to simply weed out the clauses with tag $T \subsetneq \{x_1, \ldots, x_k\}$ i.e. clauses with less than k literals. The way to do this would be to run the same test as above for every $L \subset \{x_1, \ldots, x_k\}$ such that $|L| = k - 1$ and collecting the sets S_L of violated clause indices that have tag $T \subseteq L$. Now, $\cup_L S_L$ for all $k-1$ sized Ls is the list of clauses with at most $k-1$ literals and $(S \cap S') \setminus (\cup_L S_L)$ is the set of clauses with the tag $T = \{x_1, \ldots, x_k\}$. There are k such sets L and the test for each consumes $O(m)$ time leading to a clause tag of size k being learned in $O(km)$ time. This proves the claim. □

Corollary 6.1. *In the random oracle model,* H–$\mathsf{2SAT}_{\mathrm{RAND}}$ *can be solved in polynomial time.*

The corollary follows directly from Theorem 6.1 and shows that H–2SAT in this model can be learned simply, and hence, solved in polynomial time. A more restrictive model is needed to better satisfy the minimal information condition. Specifically, one that does not permit that

learning of the underlying instance completely but still provides polynomial time algorithms for H–1SAT and H–2SAT. For this, the modification is made both over the kind of trials – a probability distribution over assignments – and the oracle's answers – the clause index which is most likely to be violated by the distribution.

6.1 Probabilistic Trials, Quantum Trials

The idea for this model is to propose a probability distribution over trials $\mathbf{D}$ which automatically imposes a probability of violation for each clause in the instance. This can be treated as ordering of the likelihood of a clause being violated by the current trial and the oracle can return the index of a clause with the maximum likelihood of being violated. Any ties can be broken arbitrarily expecting an adversarial behaviour for the worst case scenario. This type of oracle will be indicated by $\mathcal{O}_{\text{MAX}}$ and any problem will be denoted with H–CSP_{MAX}. The complexity of the algorithms discussed in this chapter will be in terms of the total running time where one query to the oracle takes unit time.

6.1.1 Probabilistic trials for H-SAT

A product distribution over the variables suffices for the application of interest. A probabilistic assignment for a set of n variables is a function $\mathbf{a} : [n] \to [0, 1]$ such that $Pr[x_i = 1] = \mathbf{a}(i)$ and $Pr[x_i = 0] = Pr[\overline{x}_i = 1] = 1 - \mathbf{a}(i)$. For the sake of concise notation, these are usually written as $x_i = \mathbf{a}(i)$ and $\overline{x}_i = 1 - \mathbf{a}(i)$. This naturally translates to the notion of the probability of a clause C_j being violated which is defined as $Pr[C_j = 0] := \prod_{\ell \in T(C_j)} Pr[\ell = 0] = \prod_{\ell \in T(C_j)} (1 - \ell)$ which allows the oracle to calculate the probability for each clause being violated. In a slight overload of notation, here ℓ refers both to the identity of a literal as well as to the probability that literal ℓ is set to true. Now, the problem H–SAT_{MAX} (resp. H–kSAT_{MAX}) consists of finding a satisfying assignment for a H–SAT (resp. H–kSAT) instance by proposing probabilistic assignments to the $\mathcal{O}_{\text{MAX}}$ oracle. One way to do this is by *learning* an *equivalent formula* to the hidden instance and solve it to find a satisfying assignment. Learning implies the process of using the information from a series of violations to determine what a clause in the hidden instance could be.

Note that it is possible for an instance to contain clauses which will never be returned by the oracle. For instance, given clauses C_i and C_j such that their tags are related as $T(C_i) \subset T(C_j)$, then clause C_i will always be at least as violated as C_j. Then, C_j will never be returned by the oracle and C_i will be said to *obscure* C_j.

6.1.2 Quantum Trials for H-QSAT

The unknown input version of QSAT, H–QSAT, is defined analogously to the classical case. Given the continuous nature of the energy spectrum for a QSAT instance $H' = \sum_j \Pi'^{(j)}$, the

task is to decide between the following cases

YES: the ground energy is 0 with the instance being frustration free;
NO: the ground energy of H' is at least $m \cdot 2\epsilon^2$.

Here, $\epsilon > 0$ is some threshold parameter that can be assumed to be inverse polynomially small in n. The trials would clearly correspond to quantum states that could be ground states for the hidden instance. However, generalizing the notion of probabilistic trials to the quantum regime, the trials are in fact density matrices of mixed states i.e. an ensemble of pure states. The oracle $\mathcal{O}_{\text{MAX}}$ is then required to return the index of j for which $\text{Tr}(\rho\Pi^{(j)})$ is maximized. Here the energy violation of the density matrix with respect to a clause indicates the probability of the clause being violated. If the total energy of the proposed state is $\leq m \cdot \epsilon^2$ then the oracle will indicate that a satisfying assignment has been found.

6.2 SAT with probabilistic trials

Learning WIDESAT

To start with, it is useful to compare the $\mathcal{O}_{\text{MAX}}$ oracle with the clause-index revealing oracle $\mathcal{O}_{\{\mathcal{C}\}}$ introduced by Bei, Chen and Zhang [BCZ13]. They showed that given a H–SAT instance, it is possible to find a satisfying assignment for it in $O(mn)$ time when also provided with access to a SAT oracle. The latter takes as assignment some SAT formula and provides a satisfying assignment for it if one exists or declares it unsatisfiable. However, they also showed that there exists instances where trying to learn an equivalent instance to the underlying formula could require exponential trials. The set of instances used to show this are clauses containing all n distinct variables and are termed WIDESAT clauses. The reason can be gleaned from the fact that a WIDESAT clause contains just one assignment out of a possible 2^n that violates it and uniquely identifies it. Then, using even a randomized algorithm, if the oracle always answers YES, each trial only eliminates one from an exponential set and identifying the clause requires $\Omega(2^n)$ trials. It is shown in [BCZ13] that exponential trials are required even when the hidden formula consists of only a single WIDESAT clause. In contrast, using the $\mathcal{O}_{\text{MAX}}$ oracle in concert with probabilistic trials allows one to identify a formula with $m = O(1)$ number of WIDESAT clauses in polynomial time. The following lemma is useful in proving the final claim.

Lemma 6.1. *Given a set S of m distinct binary strings of length n. Then there exists a subset of $m-1$ indices such that each string restricted to those indices is unique.*

It can be seen that $m-1$ indices are necessary by consider S as any m-sized subset of $\{e_i \mid 1 \leq i \leq n\}$ where e_i is the with a 1 in position i and 0 everywhere else. Clearly, any set of $m-2$ indices is insufficient to correctly distinguish all the strings.

Proof. The proof proceeds by induction. For the base case, consider 2 distinct strings of length 2. Knowing that they differ at least at one index, restricting to that index gives a set of

$m-1=2-1=1$ index that satisfies the proposition. For the inductive step, let this hold for sets of strings of length $n-1$. Now consider S as the set of m distinct Boolean strings of length n and any two strings in S. Since all strings in the set are distinct, they differ in at least one position (without loss of generality, let it be in the first bit). Divide the set into two groups such that all strings in the first group start with 0 and all strings in the second start with 1. Notice that any two strings in differing groups are distinguished by the first index, but any two strings in the same group must still all be distinct when restricted to the last $n-1$ bits. If the first group is of size k and second is of size $m-k$, then by the induction hypothesis, it is possible distinguish to the strings within the first group with at most $(k-1)$ bits and the strings in the second group require at most $(m-k-1)$ indices. Adding the first index used for the initial comparison, every string in S can be distinguished using at most $(k-1)+(m-k-1)+1=m-1$ indices. □

Proposition 6.1. *Given a hidden* WIDESAT *instance on n variables and m distinct clauses where $m \leq n$, it is possible learn an equivalent instance in $O(\binom{n}{m-1}2^m + mn)$ time using the $\mathcal{O}_{\text{MAX}}$ oracle and probabilistic trials.*

Proof. First, consider the case when the instance has just a single WIDESAT clause. If the assignment $\mathbf{a}(1)=1, \mathbf{a}(2)=\ldots=\mathbf{a}(n)=1/2$ satisfies the hidden formula, then the clause contains x_1 and contains $\overline{x}_1$ otherwise. Repeating this for each variable identifies the clause in $O(n)$ time.

Next is the case when there are $m>1$ clauses. Each WIDESAT clause can be identified by the unique n length string that violates the clause making the instance similar to a list of m Boolean strings of length n. For each $m-1$-sized subset of the variables, try all possible $0-1$ assignments on them, setting the rest to $1/2$. From Lemma 6.1, eventually, the set of $m-1$ indices that identifies the m strings will be chosen and over the set of assignments, each of the m clause indices will be returned as being violated. This allows one to map exactly how these $m-1$ variables are present in each clause. Pick any one clause C_k setting the $m-1$ variables to the violating assignment which will satisfy all other clauses. Now repeat the process given above to learn how the remaining variables occur in C_k. Further repeating this across clauses will enable the learning of the entire WIDESAT instance. The number of such subsets is $\binom{n}{m-1}$ with 2^m trials each is supplemented with an $O(n)$ procedure for each clause giving the time bound. □

6.2.1 Hidden 1SAT

While the previous result seems to imply that it is possible to learn H–1SAT_{MAX} instances the much tougher hidden WIDESAT can be efficiently identified, there is a subtle argument to be made for this not being true. Consider a H–1SAT_{MAX} formula Φ which could contain repeated clauses. Then, there exists a formula $\Phi' \neq \Phi$ such that the answers from the $\mathcal{O}_{\text{MAX}}$ oracle is identical in both cases. In other words,

Proposition 6.2. *There is no algorithm using probabilistic trials and the $\mathcal{O}_{\text{MAX}}$ oracle which, given an unsatisfiable instance Φ, learns all the literals present in Φ – even if granted the use of an arbitrary numbers of queries.*

Proof. Consider the following two H–1SAT instances:

$$\Phi_1 : C_1 = x_1, C_2 = \overline{x}_1, C_3 = x_1, C_4 = \overline{x}_1$$
$$\Phi_2 : C_1 = x_1, C_2 = \overline{x}_1, C_3 = x_2, C_4 = x_2$$

Both of these instances are unsatisfiable. However, note that for any oracle query, it is possible for the oracle to give the same answer (i.e., clause index) for each query. This can be seen by outlining the strategy that the oracle could pursue to obscure the actual instance. If x_1 is more violated than $\overline{x}_1$ or x_2, $\mathcal{O}_{\text{MAX}}$ returns C_1. If $\overline{x}_1$ is more violated than x_1 or x_2, then the oracle returns clause C_2. If x_2 is more violated than x_1 or $\overline{x}_1$, then $\mathcal{O}_{\text{MAX}}$ returns C_3 if x_1 is more violated than $\overline{x}_1$, otherwise it returns C_4. One can easily check that these oracle answers are consistent with both instance. Hence these instances are indistinguishable to adversarial oracle answers, and no algorithm can distinguish between Φ_1 and Φ_2 in this model. □

Here the difficulty in learning an unsatisfiable instance does not lie in the repetition of clauses, but rather in determining for which i do both x_i and $\overline{x}_i$ appear in Φ. As an aside, this does not rule out the possibility of only learning satisfiable H–1SAT$_{\text{MAX}}$ formulas. However, any algorithm to solve H–1SAT$_{\text{MAX}}$ must do so despite the fact that it may not be able to deduce the underlying instance. Surprisingly, it turns out that it is possible to solve H–1SAT$_{\text{MAX}}$ in polynomial time without deducing the underlying instance . The algorithm aims to construct partial assignments and extend them to a satisfying assignment. In this context, any partial assignment that extends to a satisfying assignment and is consistent with the oracle answers is called *good*. Other partial assignments that are not good, are correspondingly termed as *bad*.

Theorem 6.2. *Given a* H–1SAT$_{\text{MAX}}$ *instance Φ on n variables and m clauses, it is possible to determine if Φ is satisfiable in time $O(mn^2)$.*

Proof. Start with an ordering of the variables as $x_1, \ldots, x_n$. The idea is to inductively create a series of n lists $L_1, L_2, \ldots, L_n$. Each list will contain a partial assignment to the variables $x_1, \ldots, x_i$ and will be of size at most m except for L_n which can have size at most $2m$. Note that every partial assignment in the case of $SAT1$ is either good or bad. The guarantee is that when Φ is satisfiable, it is possible to construct each list to contain at least one "good" partial assignment. By extension, L_n contains at least one satisfying assignment and trying all the (at most) $2m$ assignments in L_n should solve the problem.

The lists can now be constructed by induction. The base case of L_1 is trivial - just add both $x_1 = 0$ and $x_1 = 1$ to the list. To construct L_{i+1} given a list L_i, first consider the extension $\tilde{L}_{i+1}$ containing all possible extensions of the candidates in L_i to variable x_{i+1}. In other words, for every partial assignment $\mathbf{p}$ in L_i, add $\mathbf{p}0$ and $\mathbf{p}1$ to $\tilde{L}_{i+1}$. If L_i contained a good assignment,

then it is definitely present in $\tilde{L}_{i+1}$ too except that the size of $\tilde{L}_{i+1}$ has doubled to $2|L_i|$. When $i+1 < n$, this is problematic and the new list will have to be modified to have a size at most m. For this purpose, propose the following trials are to the $\mathcal{O}_{\text{MAX}}$ oracle

$$\mathbf{a}(k) = \begin{cases} \mathbf{p}(k) & \text{if } 1 \leq k \leq i+1 \text{ and } \mathbf{p} \in \tilde{L}_{i+1} \\ 1/2 & \textit{otherwise} \end{cases}$$

Collect the violations C_j returned by the oracle and partition the elements of $\tilde{L}_{i+1}$ accordingly dividing the list into at most m equivalence classes. Pick a representative from each equivalence class to construct L_{i+1}.

Clearly L_{i+1} has size at most m by construction. However, the guarantee that L_{i+1} still contains at least one good assignment has to be shown. Starting with a good element $\mathbf{p} \in L_i$, it is also present as the extension $\mathbf{p}^* \in \tilde{L}_{i+1}$. $\mathbf{p}^*$ being good, satisfies all clauses involving variables $x_1, \ldots, x_{i+1}$. When there are no clauses involving $x_{i+2}, \ldots, x_n$, $\mathbf{p}^*$ is also a satisfying assignment as will be indicated by the oracle and the algorithm can be aborted. If not, there exists a clause involving $x_{i+2}, \ldots, x_n$, $\mathbf{p}^*$. Moreover, the clause C_k returned when the trial with $\mathbf{p}^*$ is proposed, is a clause involving $x_{i+2}, \ldots, x_n$ and has a violation of $1/2$ and this equivalence class contains a good assignment. Note that a trial corresponding to a bad assignment $\mathbf{p}'$ would violate some clause involving $\{x_1 \ldots x_{i+1}\}$ by 1 while C_k will only be violated by $1/2$. So, for any bad assignment, C_k will never be returned as the maximal violation. Then, all assignments in this class are good and picking just one of them suffices to maintain the inductive guarantee. The time to construct each list is $O(mn)$, and the algorithm constructs n lists. Hence, the algorithm runs in time $O(mn^2)$. □

6.2.2 Hidden 2SAT

Ideally, the aim would be to generalize the previous method to encompass H–2SAT instances as well. One of the major roadblocks lies in the fact that it is no longer clear how to separate the good and bad assignments just based on the violations returned. The reason for this can be traced back to there being multiple ways to satisfy a 2SAT clause. The following example would make the scenario clearer.

Example 6.1. *Suppose that the list L_i has been constructed as outlined in Theorem 6.2 and it has also been extended to $\tilde{L}_{i+1}$ which has been partitioned into m equivalence classes based on the indices returned by the oracle. Also, let the* H–2SAT *instance contain the following clauses: $C_1 = (x_1 \vee x_2), C_2 = (x_4 \vee x_5), C_3 = (x_1 \vee x_n), C_4 = (\overline{x}_4 \vee x_{i+2}), C_5 = (\overline{x}_{i+2}), \ldots, C_m$ for some i. Consider the C_3 partition of $\tilde{L}_{i+1}$ and two assignments $\mathbf{a}$ and $\mathbf{b}$ in it. Note that these assignments which have a maximum violation $1/2$ with $\mathbf{a}(1) = \mathbf{b}(1) = 0$ and $\mathbf{a}(2) = \mathbf{b}(2) = 1$. Moreover, let the assignments satisfy C_2 using different assignments i.e. $\mathbf{a}(4) = 0, \mathbf{a}(5) = 1$ but $\mathbf{b}(4) = 1, \mathbf{b}(5) = 1$. This is still valid behaviour for assignments C_3 partition of $\tilde{L}_{i+1}$. The problem arises by noticing that a good assignment, like $\mathbf{b}$ can be extended to satisfy C_4 and C_5 but this clearly cannot be done by $\mathbf{a}$. This demonstrates that any partition can contain a mix of*

good and bad assignments such that picking one candidate from each partition does not suffice anymore.

However, not all is lost and in the case that the H–2SAT formula Φ is promised not to contain any repeated clauses, there is the chance to determine some 2SAT formula Φ' from the pattern of violations such that Φ' and Φ have the same set of satisfying assignments i.e. $\mathsf{sat}(\Phi') = \mathsf{sat}(\Phi)$. Note that Proposition 6.2 shows that one cannot always hope to learn Φ directly. The first step would be to understand how to generate a satisfying assignment for a satisfiable Φ efficiently.

Theorem 6.3. *Suppose Φ is a satisfiable* H–$2\mathsf{SAT}_{\mathrm{MAX}}$ *instance on n variables with m clauses. Then it is possible to generate a satisfying assignment for Φ in time $O(n^2)$.*

Proof. The ides is to learn if a certain clause tag is present in the instance or not. Since the oracle will never return the index of an obscured clause, the clause tag is assumed to be unobscured without loss of generality. Then repeating this for each 1SAT and 2SAT clause tag without running into a satisfying assignment would result in determining the set of unobscured clauses in Φ. It is clear that the conjunction of these clauses forms a formula Φ' such that $\mathsf{sat}(\Phi') = \mathsf{sat}(\Phi)$. Therefore, using any 2SAT algorithm which runs in time $O(n+m)$ on Φ' will also find a satisfying assignment for Φ. The following series of trials are performed to determine if there is a clause with the tag $\{x_i, x_j\}$ and at no point is a satisfying assignment found (otherwise the algorithm can be successfully):

1. On querying $\mathcal{O}_{\mathrm{MAX}}$ with the assignment $\mathbf{a} = 0^n$ a violation implies that there exists a clause with the tag: (a) $\{x_i, x_j\}$; (b) $\{x_i, x_k\}$ for some $k \neq i, j$; (c) $\{x_j, x_k\}$ for some $k \neq i, j$; or (d) $\{x_{k_1}, x_{k_2}\}$ for $k_1, k_2 \neq i, j$.
2. Next query the oracle with the assignment $\mathbf{a}(i) = 0$, $\mathbf{a}(j) = 0$ and $\mathbf{a}(k) = 1$ for all $k \neq i, j$. A violation indicates the presence of a clause with the tag: (a) $\{x_i, x_j\}$; (b) $\{x_i, \overline{x}_k\}$ for some $k \neq i, j$; (c) $\{x_j, \overline{x}_k\}$ for some $k \neq i, j$; or (d) $\{\overline{x}_{k_1}, \overline{x}_{k_2}\}$ for $k_1, k_2 \neq i, j$.
3. Next is an explicit test for the presence of the clause $(x_i \vee x_j)$ using two probabilistic assignments. If $(x_i \vee x_j)$ is present, then the same clause returned for both trials. If not, then the returned clauses are guaranteed to be different. The two assignments are

$$\mathbf{a}_1(\ell) = \begin{cases} 0 & \text{if } \ell = i, j \\ \frac{1}{4} & \text{otherwise} \end{cases} \qquad \mathbf{a}_2(\ell) = \begin{cases} 0 & \text{if } \ell = i, j \\ \frac{3}{4} & \text{otherwise} \end{cases}$$

Table 6.1 shows the violations corresponding to each assignment and each possible clause tag consistent with the previous two trials. Clearly, if $(x_i \vee x_j)$ were indeed present in the

	$(x_i \vee x_j)$	$(x_i \vee x_k)$	$(x_j \vee x_k)$	$(x_{k_1} \vee x_{k_2})$	$(x_{k_1} \vee \bar{x}_{k_2})$	$(x_i \vee \bar{x}_k)$	$(x_j \vee \bar{x}_k)$	$(\bar{x}_{k_1} \vee \bar{x}_{k_2})$
$\mathbf{a}_1$	1	3/4	3/4	9/16	3/16	1/4	1/4	1/16
$\mathbf{a}_2$	1	1/4	1/4	1/16	3/16	3/4	3/4	9/16

Table 6.1: Violation of the clauses based on the fractional assignments of 1/4 and 3/4.

formula, then both assignments return it as the maximum violation. If not, the assignments would return some clause with a 3/4 violation in each case. From the table, it is clear that the set of these clauses for each assignment is disjoint forcing a different clause to be returned with each trial. □

The following theorem also settles the question of finding an equivalent formula even when some trial in the procedure ended up satisfying the instance and aborting the process of finding all the unobscured clauses in Φ'. So, in some sense for repetition free H–$\mathsf{2SAT}_{\mathrm{MAX}}$ one is able to do a little more than just finding a satisfying assignment for some hidden instance.

Theorem 6.4. *Suppose* Φ *is a repetition-free* H–$\mathsf{2SAT}_{\mathrm{MAX}}$ *instance on* n *variables with* m *clauses. Then, there is an algorithm to generate a* 2SAT *formula* Φ' *such that* $\mathsf{sat}(\Phi') = \mathsf{sat}(\Phi)$ *in* $O(\mathrm{poly}(n))$.

Proof. Start by running the procedure outlined in Theorem 6.3. Either all unobscured clauses are determined successfully, or a satisfying assignment is found. Assume the latter case here and without loss of generality, let this assignment be 1^n resulting in each clause having at least one positive literal.

Now, the procedure aims to find all variables that must necessarily be set to 1 to satisfy the instance, adding the 1SAT clause that corresponds to these variables to Φ'. TO check if some x_i satisfies this condition, repeat the process in Theorem 6.3 by replacing $\mathbf{a}(i) = 0$ for each trial. If the formula remains unsatisfiable, then x_i must be set to 1. If, however, all the trials still satisfy the formula, then the conclusion would be that all the clauses that x_i occurs in either as a positive or negative literal are clauses on 2 variables. Moreover, since each clause contains a positive literal, these clauses are of the form either $(x_i \vee x_j)$ or $(\overline{x}_i \vee x_j)$. Repeat this process for all variables to reach a point where the unobscured clauses are all 2 variable clauses. Recalling the DP-procedure for 2SAT, this corresponds to being at a "branch point" of the algorithm.

To simplify the exposition, suppose that after the above procedure, the 2 variable clauses still involve all n variables and one is still querying Φ. The next step is to learn clauses which are a mix of positive and negative literals i.e. a clause with the tag $(x_i \vee \overline{x}_j)$. Set $\mathbf{a}(i) = 0, \mathbf{a}(j) = 1$ and repeat the process in Theorem 6.3 to determine if the induced formula is satisfiable. If so, then clearly Φ does not contain $(x_i \vee \overline{x}_j)$ and if not, $(x_i \vee \overline{x}_j)$ is definitely present in Φ as the setting of $\mathbf{a}(i) = 0$ and for all $k \neq i$, $\mathbf{a}(k) = 1$ will satisfy all other 2 variable clauses in the formula.

Finally, the procedure determines all clauses of the form $(x_i \vee x_j)$. The trial proposed is of the form $\mathbf{a}(i) = \mathbf{a}(j) = 0$ and for all $k \neq i, j$, $\mathbf{a}(k) = 1$. The catch is when the formula is not satisfied as that indicates the possibility that the clause could be of the form (a) $(x_i \vee x_j)$; (b) $(x_i \vee \bar{x}_k)$; or (c) $(x_j \vee \bar{x}_k)$; as all such clauses are maximally violated (with probability 1). To narrow down the options, the next trial is replaces the assignment for all $k \neq i, j$ as $\mathbf{a}(k) = 0.5$. If $(x_i \vee x_j)$ were absent from the formula, then the clause tag of the violation would be a mix of positive and negative literals. However, all such clauses have already been determined previously. So, if one

of those clauses is returned as a violation, it is fitting that $(x_i \vee x_j)$ is absent in the formula but if the clause returned has not been previously determined, it points to $(x_i \vee x_j)$ being present in the formula. In this manner, all unobscured clauses can be determined. □

While the above procedures may seem elementary, they accomplish two things. First, they show that H–2SAT$^{\mathsf{RF}}$ with the $\mathcal{O}_{\mathrm{MAX}}$ oracle is in P, whereas Theorem 5.9(c) from Chapter 5 demonstrated the NP-hardness of H–2SAT$^{\mathsf{RF}}$ with the $\mathcal{O}_{\{\mathcal{C}\}}$ oracle. Second, they act as stepping stones to tackle the harder problem of learning an unknown input instance of Quantum 2SAT in this model. Dealing with the general H–2SAT instance with repeated clauses using the $\mathcal{O}_{\mathrm{MAX}}$ oracle is left for future work.

6.3 Quantum SAT in the trial and error model

One of the main motivations to consider the probabilistic trials model was also to use that as an intermediate step to quantize the trial and error setting and consider the behaviour of Quantum SAT – specifically, Q1SAT and Q2SAT. At this point it is worth comparing the notions of *learning* and *solving* hidden instances both in the classical and quantum settings. The classical case is more straightforward where learning an instance means learning all the literals present in each clause, whereas solving means finding a satisfying assignment. For H–1SAT and H–2SAT, learning the instance in polynomial time automatically triggers solving it in polynomial time as well. However, in the quantum setting this simple relation between learning and solving breaks down. The continuous nature of QSAT means we can only learn a projector or find a satisfying assignment up to a specified precision ϵ. However, in the case of hidden Q2SAT after learning the instance up to precision ϵ, it is not clear if a satisfying assignment up to precision $\mathrm{poly}(n, \epsilon)$ can be found in polynomial time.

6.3.1 Hidden Quantum 1SAT

The algorithm used to solve H–1SAT can be extended to solve the H–Q1SAT problem as well. A 1-local projector defined on $\mathbb{C}^2$ is satisfiable if it is of rank at most 1 and can be viewed as setting the direction of the qubit on the Bloch sphere. Unlike the classical case, where the 1SAT clauses can be viewed as either the $|0\rangle\langle 0|$ or $|1\rangle\langle 1|$ projectors, here the projectors can point in any direction in the Bloch sphere. To make handling the continuous nature of the Bloch Sphere easier, it is discretized by using an ϵ-net that covers the whole sphere. This allows for the lists of $0-1$ strings used in H–1SAT to be generalized into lists of n-qubit product states where each qubit is assigned an element of the ϵ-net.

Given a 1-local projector $|\psi\rangle\langle\psi|$, its ground space is spanned by $|\psi^{\perp}\rangle$. This essentially divides the Bloch sphere into two hemispheres, a *good* hemisphere containing states $|\phi\rangle$ having $|\langle\psi|\phi\rangle| \leq \frac{1}{2}$ and a *bad* hemisphere with states having $|\langle\psi|\phi\rangle| > \frac{1}{2}$. An n-qubit state $a = |a_1\rangle|a_2\rangle \ldots |a_n\rangle$ is called *good* if for each qubit i, the state $|a_i\rangle$ is in the good hemisphere and *bad* otherwise. In

other words, when $|\psi_i\rangle$ is the forbidden state for qubit i, a is good if $\forall i,\ |\langle\psi_i|a_i\rangle| \leq \frac{1}{2}$ and *bad* if $\exists i, |\langle\psi_i|a_i\rangle| > \frac{1}{2}$. For the n-qubit state $a = |a_1\rangle|a_2\rangle \ldots |a_n\rangle$, let $a' := |a_1^{\perp}\rangle|a_2^{\perp}\rangle \ldots |a_n^{\perp}\rangle$.

The first step of to construct the H–Q1SAT algorithm is to adapt the process described in Theorem 6.2 to work for for an arbitrary n-qubit state a and provide a list of n-qubit states, $L_{a/a'}$, where at least one state is *good*. This is formally stated in Lemma 6.2.

Lemma 6.2. *Let $a = |a_1\rangle \otimes \ldots \otimes |a_n\rangle$ be an n qubit state where $\{|a_i\rangle, |a_i^{\perp}\rangle\}$ is a basis for qubit i, for $i = 1, \ldots, n$. Then one can produce a list, $L_{a/a'} \subset \bigotimes_{i=1}^{n}\{|a_i\rangle, |a_i^{\perp}\rangle\}$ of at most $2mn$ states such that, if the instance is satisfiable, there is at least one* good *n-qubit state in the list. The time taken to produce this list is $O(n^2m)$.*

Proof. The list is built inductively as in Theorem 6.2 where, at stage k, $L_{k,a/a^{\perp}}$ contains at most m strings at least one of which is good for qubits $1, \ldots, k$ if the instance is satisfiable. This is done by querying the following trials. At stage k, replace $\{0, 1\}$ from Theorem 6.2 with $\{|a_k\rangle, a_k^{\perp}\}$ for qubit k and replace $\frac{1}{2}$ with $\frac{\mathbb{I}}{2}$, the completely mixed state, for qubits $k+1, \ldots, n$ so that

$$L_{k,a/a^{\perp}} \subseteq \bigotimes_{i=1}^{k}\{|a_i\rangle, |a_i^{\perp}\rangle\} \otimes \left(\frac{\mathbb{I}}{2}\right)^{\otimes(n-k)}.$$

This almost finishes the process except for one caveat while constructing $L_{k+1,a/a^{\perp}}$ from $L_{k,a/a^{\perp}}$a – the case when there is no projector on qubits $k+2, \ldots, n$. This situation also occurs at the last step while constructing L_n. In both cases, while proposing a good state, all violations are $\leq \frac{1}{2}$ and any clause index returned by the oracle involves a qubit in $1, \ldots, k+1$. This same clause could also be violated with probability $> \frac{1}{2}$ when a bad string is proposed which will incorrectly be put in the same equivalence class as the good one. Then, picking just one representative from C_j is insufficient and the size of the lists cannot be compressed. To fix this, following additional checks are made:

1. If $k + 1 = n$, just double the number of states on the list, assuming that there is a clause involving n, i.e. the last qubit that is assigned values. This is okay to do at the last step as the list only doubles once.
2. Repeat the procedure n times by placing a different qubit at the last position each time. Let $L_{a/a^{\perp}}$ be the union of all the lists found in this manner and it is of size $2m \cdot n$ which is still polynomial in the size of the problem. For a non-empty instance, at least one of these repetitions will have a clause on the qubit in the last position and so, will contain a good state.

Hence, $L_{a/a^{\perp}}$ with $2mn$ n-qubit states contains at least one good state and by repeating the classical process n times, an $O(mn^2)$ time procedure is obtained for this task. □

However, this only provides an assignment that violates each projector by $\leq \frac{1}{4}$ while the requirement is for assignments that violate each projector by $\leq \epsilon^2$. The key observation involves constructing two lists $L_{a/a'}$ and $L_{b/b'}$ where $b \neq a, a'$ and picking a state from each list. Consider

the case when both states are good. Let the states on qubit i from each list be $|a_i\rangle$ and $|b_i\rangle$ respectively. Each state defines a hemisphere R_{i,a_i} and R_{i,b_i} respectively containing all the states that are bad with respect to the forbidden state for qubit i, $|\psi_i\rangle$. Then, $|\psi_i\rangle$, should be contained in $R_{i,a_ib_i} := R_{i,a_i} \cap R_{i,b_i}$. The optimal choice for b_i, given a_i, would be one where $|R_{i,a_ib_i}| \leq \frac{|R_{i,a_i}|}{2}$. Then, similar to performing a binary search on the Bloch Sphere, repeating this process $\log_2\left(\frac{1}{\epsilon}\right)$ times, will give a region consisting of good approximations to the forbidden state (See Figures 6.1 (a) and (b) for illustrations).

Theorem 6.5. *Let $\epsilon > 0$. Given a* H–$\mathsf{Q1SAT}_{\mathrm{MAX}}$ *instance on n qubits containing m projectors, there exists an $O((2mn)^{2\log\frac{1}{\epsilon}} \cdot mn^2)$ time algorithm, with the property that*

(a) on a frustration free instance, it outputs an assignment where for each projector, the forbidden state is violated with probability $\leq \epsilon^2$;

(b) otherwise, the algorithm outputs UNSAT.

Proof. Initially, with no information, for each qubit i, R_i = Bloch sphere. Now the algorithm executes the following steps:

- Start by picking an arbitrary state, say $\bar{a} = |0\rangle^{\otimes n}$, and construct $L_{|0\rangle^{\otimes n}/|1\rangle^{\otimes n}}$ as per the procedure in Lemma 6.2. For each $a \in L_{|0\rangle^{\otimes n}/|1\rangle^{\otimes n}}$ repeat:
  - a defines the region R_{i,a_i} in this branch of the iteration.
  - For $i = 1, \ldots, n$ pick a basis $\{|b_i\rangle, |b_i^{\perp}\rangle\}$ such that their equator $\frac{|b_i\rangle + |b_i^{\perp}\rangle}{\sqrt{2}}$ bisects R_{i,a_i}.
  - Set $\bar{b} = b_1 \ldots b_n$, construct $L_{\bar{b}/\bar{b}'}$ and for each $b \in L_{\bar{b}/\bar{b}'}$:
    - The tuples (a, b) define the region R_{i,a_ib_i} in this branch.
    - Repeat the process to find $\bar{c}$ to bisect each R_{i,a_ib_i}.
    - Find a new region $R_{i,a_ib_ic_i}$ for each $c \in L_{\bar{c}/\bar{c}'}$.
    - Continue the recursion up to $\log_2\left(\frac{1}{\epsilon}\right)$ depth and let the last list be $L_{z/z^{\perp}}$.
    - Propose $|\phi^{\perp}\rangle = \bigotimes_{i=0}^{n} |\varphi_i^{\perp}\rangle$ where $\forall i, |\varphi_i\rangle \in R_{i,a_ib_i\ldots z_i}$ to the oracle. Output $|\phi^{\perp}\rangle$ if the oracle returns YES, otherwise continue.
- Output UNSAT if none of the trials satisfy the instance.

This algorithm essentially creates a recursion tree with each new string created where the width of the recursion at each point is $2mn$ and the depth is $\log_2\left(\frac{1}{\epsilon}\right)$. This leads to $(2mn)^{\log_2\frac{1}{\epsilon}}$ trials to be proposed at the end and the number of lists created is also $(2mn)^{\log_2\frac{1}{\epsilon}}$, each at a cost of $O(mn^2)$. Hence, the total running time of the algorithm is $O((2mn)^{\log\frac{1}{\epsilon}} \cdot mn^2)$. Figure (6.1) shows exactly how the algorithm given above proceeds for a particular qubit i. Consider a qubit i and the states $a, b, c, \ldots, z$ that are picked in one branch of the recursion tree of the algorithm. To argue the correctness of this algorithm, analyze the region $R_{i,a_ib_i\ldots z_i}$ obtained at the leaf of the recursion tree. Let the forbidden state for qubit i be $|\psi_i\rangle$. At the beginning, let $\forall\ i,\ |R_i| = 1$ (the complete Bloch sphere) and the only guarantee for each list is that there is at least one good string in it. Tracing the path in the recursion tree to the leaf, assume that each step of the recursion picked a good state i.e. $a, b, \ldots, z$ are all good states. For a and $\forall i$, the forbidden state $|\psi_i\rangle$ is in the opposite hemisphere to $|a_i\rangle$ which reduces the size of the region to $|R_{i,a_i}| = 1/2$ as shown in Figure 6.1(a). Taking (a, b) at the next iteration, the region for each

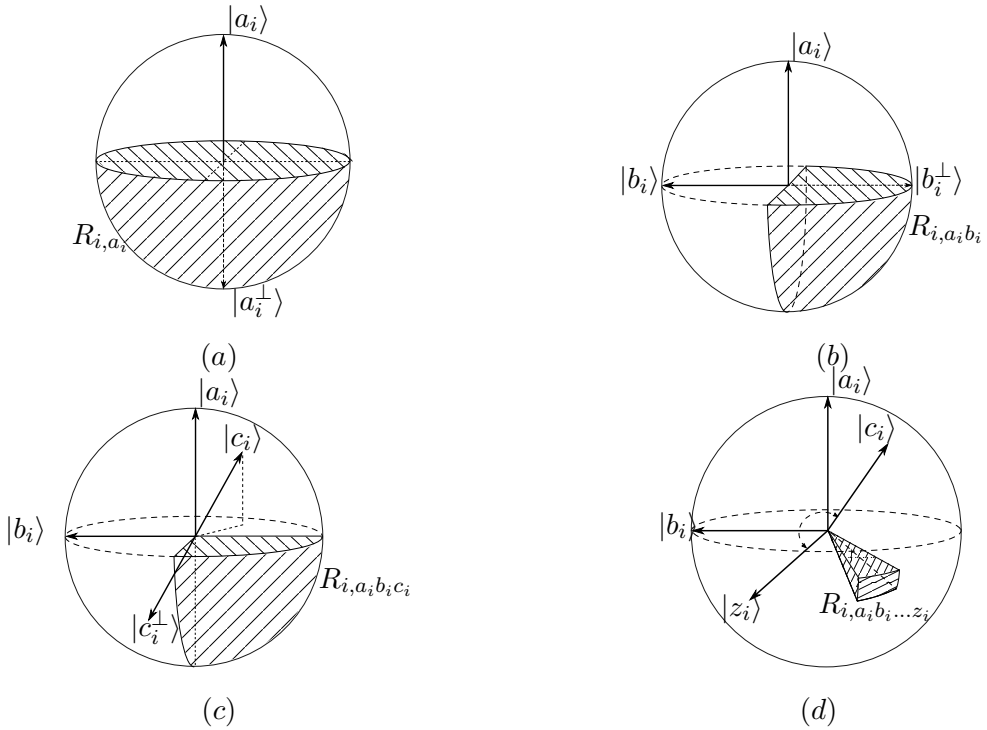


Figure 6.1: Shown here are four stages of the algorithm for a qubit i, with forbidden state $|\psi_i\rangle$, starting with picking string a, b and c followed by the last string z. (a) After picking $|a_i\rangle$ for qubit i, the hemisphere orthogonal to it is R_{i,a_i}; (b) On choosing $|b_i\rangle$, the interesting region is the quadrant R_{i,a_ib_i}; (c) After $|c_i\rangle$ is determined, $|\psi_i\rangle$ should be in the hatched $1/8^{th}$ region of the sphere; (d) Continuing the process of picking strings for $\log_2 \frac{1}{\epsilon}$ steps and picking the final state $|z_i\rangle$ shows that $|\psi_i\rangle$ should be present in $R_{i,a_ib_i\ldots z_i}$.

qubit is the overlap of two hemispheres $R_{i,a_i} \cap R_{i,b_i}$ and by construction, since b_i bisects R_{i,a_i}, the overlaps of the hemispheres also bisect R_{i,a_i} setting $|R_{i,a_ib_i}| = 1/4$ (Figure 6.1(b)). As this pattern continues, each step of the iteration halves the region for qubit i and leaving regions of size at most ϵ at the end of the branch as shown in Figures 6.1(c) and 6.1(d). If the instance is satisfiable, the state proposed will satisfy each projector up to ϵ resulting in the oracle to return YES. Of course, when one of the states chosen is bad, the proposal $|\phi_j^\perp\rangle$ for some qubit j will end up having a large inner product with the forbidden state $|\psi_j\rangle$ and will result in the oracle returning the ID of the projector involving j. This concludes the proof. □

6.3.2 Hidden Quantum 2SAT

The approach to deal with H–$\mathsf{Q2SAT}_{\mathrm{MAX}}$ is similar to the approach used for H–$\mathsf{2SAT}_{\mathrm{MAX}}$ which makes the first step an algorithm to approximately learnt the projectors in the instance. In other words, learn the underlying local Hamiltonian to precision ϵ by finding 2-local projectors $\Pi'^{(j)}$ such that $\|\Pi^{(j)} - \Pi'^{(e)}\| \leq \epsilon$ for every projector Π_e. This yields an approximate local Hamiltonian $H' = \sum_e \Pi'_e$ whose ground energy is at most $m\epsilon^2$ away from the ground energy of the original Hamiltonian $H = \sum_e \Pi_e$. If ϵ is set such that $m\epsilon$ is much smaller than the promise gap of the initial Hamiltonian H (which merely requires $\epsilon < 1/\operatorname{poly}$), then the Hamiltonian H'

will have a promise gap as well.

Theorem 6.6. *Given a* H–$\mathsf{Q2SAT}_{\text{MAX}}$ *problem* $H = \sum_{(u,v)} \Pi_{uv}$ *on* n *qubits, and precision* ϵ*. If the interaction graph for* H *is not Star-like, then there is an* $O(n^4 + n^2 \log\left(\frac{1}{\epsilon}\right))$ *algorithm that can find an approximation* $H' = \sum_{(u,v)} \Pi'_{uv}$ *where* $\forall\ (u,v), \|\Pi'_{uv} - \Pi_{uv}\| \leq \epsilon$

The algorithm proceeds by:

(1) Identifying two pairs of qubits $(i,j) \neq (k,\ell)$ on which two projectors are defined, Π_{ij} and $\Pi_{k\ell}$, and finding a constant approximation for these projectors;
(2) Improving the constant approximation of the two projectors iteratively so that the approximation improves by a factor of 2 in each iteration; and
(3) Using the ϵ-approximation of a projector Π_{ij} to identify the rest of the independent projectors i.e. Π_{uv} such that $(u,v) \neq (i,j)$ and approximating each of them to ϵ-precision.

For the sake of clarity, the discussion initially assume that all the projectors in H' are of rank 1. Generalize to cases where the rank of projectors is > 1 requires only a slight modification that does not affect the running time significantly and will be sketched after describing the algorithm in full. The following series of lemmas will build up to the proof of Theorem 6.6.

Before delving into the details of the implementation for Step 1, consider the procedure **Test-I**, which checks if there is a projector between qubits i,j at a constant distance from $|\psi\rangle\langle\psi|$. The test returns YES if and only if there is a projector at i,j which is close to $|\psi\rangle\langle\psi|$ along with the ID of the projector and NO otherwise. The test is defined for two fixed constants $0 < \nu < 1$ and $0 < \epsilon < 1$, and a ν-net $\mathcal{T}_\nu$ in the space of rank-1 projectors on two qubits.

Test-I$(\mathbf{i},\mathbf{j},|\psi\rangle\langle\psi|)$

- For all possible pairs (k,ℓ) which are different from (i,j) repeat:
  - For all $|\alpha\rangle\langle\alpha| \in \mathcal{T}_\nu$, construct the trial state $\rho_\alpha^{k\ell}$, propose it to the oracle and receive a projector ID as violation where

$$\rho_\alpha^{k\ell} := |\psi\rangle\langle\psi|_{ij} \otimes \left[(1-\epsilon)|\alpha\rangle\langle\alpha|_{k\ell} + \epsilon(\mathbb{I}_{k\ell} - |\alpha\rangle\langle\alpha|_{k\ell})\right] \otimes \left(\frac{\mathbb{I}}{2}\right)^{\otimes(n-4)} \tag{6.1}$$

- If for all trials the oracle answers the *same* ID, output YES together with the ID of the projector, otherwise output NO.

Claim 6.1. *Assume that* $\epsilon + \frac{1}{2}\nu^2 < 1/4$*, and further assume that there is at least one projector in the system that is defined on qubits* $k \neq \ell$*, which are* different *from* i,j*. Then, if Test-I outputs* YES*, there exists a projector* $|\psi'\rangle\langle\psi'|$ *on the* (i,j) *qubits, whose* ID *is the one that was output and*

$$\||\psi'\rangle\langle\psi'| - |\psi\rangle\langle\psi|\| \leq \sqrt{2\epsilon + \nu^2}\,. \tag{6.2}$$

Conversely, if there exists a projector $|\psi'\rangle\langle\psi'|$ *on* (i,j) *such that*

$$\||\psi'\rangle\langle\psi'| - |\psi\rangle\langle\psi|\| < \sqrt{2\epsilon}\,, \tag{6.3}$$

then the test will report it.

Proof. Assume that there is a projector $|\phi'\rangle\langle\phi'|$ on qubits k, ℓ, which are different from i, j. Let $|\alpha\rangle\langle\alpha|$ be the closest member of $\mathcal{T}_\nu$ to $|\phi'\rangle\langle\phi'|$, and calculate the violation energy for this particular assignment. Note that $\mathrm{Tr}[(\mathbb{I} - |\alpha\rangle\langle\alpha|) \cdot |\phi'\rangle\langle\phi'|] = 1 - \mathrm{Tr}(|\alpha\rangle\langle\alpha| \cdot |\phi'\rangle\langle\phi'|)$ and so the violation due to this assignment is

$$\begin{aligned}&(1-\epsilon)\,\mathrm{Tr}(|\alpha\rangle\langle\alpha| \cdot |\phi'\rangle\langle\phi'|) + \epsilon[1 - \mathrm{Tr}(|\alpha\rangle\langle\alpha| \cdot |\phi'\rangle\langle\phi'|)] \\ &= (1-2\epsilon)\,\mathrm{Tr}(|\alpha\rangle\langle\alpha| \cdot |\phi'\rangle\langle\phi'|) + \epsilon\end{aligned}$$

By assumption, $\||\alpha\rangle\langle\alpha| - |\phi'\rangle\langle\phi'|\| \leq \nu$, and therefore by connecting energies and the Frobenius norm,

$$1 - \frac{1}{2}\nu^2 \leq \mathrm{Tr}(|\alpha\rangle\langle\alpha| \cdot |\phi'\rangle\langle\phi'|) \leq 1\,,$$

which implies that

$$1 - \epsilon - \frac{1}{2}\nu^2 \leq \max\,(k,\ell)\text{ violation energy} \leq 1 - \epsilon\,.$$

Similarly, by looking at the state from the ν-net that is closest to $|\phi'^{\perp}\rangle\langle\phi'^{\perp}|$, one can deduce that

$$\epsilon \leq \min\,(k,\ell)\text{ violation energy} \leq \epsilon + \frac{1}{2}\nu^2\,.$$

If all tests returned the same answer then it cannot be due to one of the completely mixed states since there the violation is always $1/4$, but for the maximal α, the violation is at least $1 - \epsilon - \frac{1}{2}\nu^2 > 1/4$. It also cannot be due to the (k, ℓ) projector as the minimal violation energy there is at most $\epsilon + \frac{1}{2}\nu^2 < 1/4$, i.e., less violated than the mixed projectors. Therefore, it must be the projector at (i, j).

Moreover, the violation of (i, j) must be at least as big as the maximal (k, ℓ) violation:

$$\mathrm{Tr}(|\psi\rangle\langle\psi| \cdot |\psi'\rangle\langle\psi'|) \geq 1 - \epsilon - \frac{1}{2}\nu^2\,.$$

Therefore,

$$\||\psi\rangle\langle\psi| - |\psi'\rangle\langle\psi'|\| = \sqrt{2 - 2\,\mathrm{Tr}(|\alpha\rangle\langle\alpha| \cdot |\psi'\rangle\langle\psi'|)} \leq \sqrt{2\epsilon + \nu^2}\,.$$

For the other direction, note that if $\||\psi\rangle\langle\psi| - |\psi'\rangle\langle\psi'|\| < \sqrt{2\epsilon}$, its violation must satisfy $\mathrm{Tr}(|\psi\rangle\langle\psi| \cdot |\psi'\rangle\langle\psi'|) > 1 - \epsilon$, which is bigger than both the violations of the completely mixed state and the maximal violation of (k, ℓ). □

A 2-qubit state $|\psi\rangle_{ij}$ is called *δ-good* for $\delta = \sqrt{2\epsilon + \nu^2}$ if it is returned during a call to **Test-**

$\mathbf{I}(i,j,|\psi\rangle\langle\psi|)$. The idea is that a *$\delta$-good* state for (i,j) would be a constant approximation for the projector on (i,j). Suppose no δ-good state is found for (i,j) then it can be concludes that there is *no projector* on (i,j).

Step 1

- Set $\nu^2 := \frac{1}{16}$, $\epsilon := \frac{1}{32}$ and $\eta := \frac{1}{4}$
- Repeat for all pairs $i \neq j$ until a *δ-good* approximation for some Π_{ij} is found.
  - For all $|\psi\rangle \in \mathcal{T}_\eta$, perform **Test-I**$(i,j,|\psi\rangle\langle\psi|)$. If the test is positive, set $|\psi\rangle\langle\psi|$ as the *δ-good* approximation for Π_{ij}.
- Repeat the above process for all qubit pairs (k,ℓ) that are disjoint from (i,j) till a *δ-good* approximation for $\Pi_{k\ell}$ is found.

Lemma 6.3. *If there exist two projectors in H' acting on disjoint pairs of qubits then Step* 1 *will always succeed in finding independent pairs $(i,j),(k,\ell)$ and projectors $|\psi^{(0)}\rangle\langle\psi^{(0)}|_{ij}$, $|\phi^{(0)}\rangle\langle\phi^{(0)}|_{k\ell}$ such that $\||\psi^{(0)}\rangle\langle\psi^{(0)}|_{ij} - |\psi'\rangle\langle\psi'|_{ij}\| \leq \frac{1}{\sqrt{8}}$ and $\||\phi^{(0)}\rangle\langle\phi^{(0)}|_{k\ell} - |\phi'\rangle\langle\phi'|_{k\ell}\| \leq \frac{1}{\sqrt{8}}$. Moreover, Step* 1 *can be executed in $O(n^4)$ time.*

Proof. Set the parameters according to Step 1. Then from Claim 6.1, **Test-I** succeeds in finding states that are $\sqrt{2\epsilon+\nu^2}$-good $= \frac{1}{\sqrt{8}}$-good approximations. For each **Test-I**$(i,j,|\psi\rangle)$, iterating over all possible disjoint pairs (c,d) gives $\binom{n-2}{2}$ pairs to be checked. The space of 2-qubit states is 4-dimensional space and for any $\gamma > 0$, a γ-net over $\mathbb{C}^2 \otimes \mathbb{C}^2$ will contain $O\left(\frac{1}{\gamma^4}\right)$ points. So, for $\nu, \eta \in O(1)$, one checks only $O(1)$ states in $\mathcal{T}_\nu$ for each qubit pair (c,d) and runs **Test-I** for $O(1)$ states in $\mathcal{T}_\eta$ giving a total of $O(n^2)$ trials proposed. To find the second projector on (k,ℓ) repeat the above process for at most $O(n^2)$ pairs. Hence, Step 1 can be executed in $O(n^4)$ time and the result follows directly. □

At the end of Step 1 there are two projectors $|\psi^{(0)}\rangle\langle\psi^{(0)}|_{ij}$ and $|\phi^{(0)}\rangle\langle\phi^{(0)}|_{k\ell}$ which are a constant approximation of their hidden counterparts Π'_{ij} and $\Pi'_{k\ell}$ i.e. at a distance $\delta \leq \sqrt{2\epsilon+\nu^2}$. For this step, the procedure used is **Test-II** to improve their value to $|\psi^{(1)}\rangle\langle\psi^{(1)}|_{ij}$ (resp. $|\phi^{(1)}\rangle\langle\phi^{(1)}|_{k\ell}$) such that it is at a distance $\leq \frac{\delta}{2}$ from Π'_{ij} (resp $\Pi'_{k\ell}$). Then, Step 2 basically repeats this test c times to improve the value to $\Pi^{(c)}_{ij}$ to a distance $\leq \frac{\delta}{2^c}$ from Π'_{ij} and when $c = O(\log n)$, this will generate a polynomially close approximation to Π'_{ij}.

From Step 1, it is know that Π'_{ij} lies somewhere in a radius of δ around $|\psi^{(0)}\rangle\langle\psi^{(0)}|_{ij}$ and similarly for qubits (k,ℓ). Let $\mathcal{B}_{ij}$ be the ball of radius δ around $|\psi^{(0)}\rangle\langle\psi^{(0)}|_{ij}$ and correspondingly consider $\mathcal{B}_{k\ell}$. The states will be enumerated over $\mathcal{T}^{\mathcal{B}}_{\nu'}$ which is the ν'-net restricted to some ball $\mathcal{B}$ in the space of rank-1 projectors on 2 qubits.

Set $\nu' = \frac{\nu}{2}$, $\eta' = \frac{\eta}{2}$ and $\epsilon' = \frac{\epsilon}{4}$. The test is defined for values of $\nu', \eta', \epsilon' > 0$ as

Test-II (i,j,k,ℓ)

- For (i,j), repeat over all $|\psi\rangle\langle\psi| \in \mathcal{T}_{\eta'}^{\mathcal{B}_{ij}}$ and perform **Test-I**$(i,j,|\psi\rangle\langle\psi|)$ over $\mathcal{T}_{\nu'}^{\mathcal{B}_{k\ell}}$ with parameter ϵ'. If the test outputs YES, set $|\psi^{(1)}\rangle_{ij} = |\psi\rangle$.
- For (k,ℓ), repeat over all $|\phi\rangle\langle\phi| \in \mathcal{T}_{\eta'}^{\mathcal{B}_{k\ell}}$ and perform **Test-I**$(k,\ell,|\phi\rangle\langle\phi|)$ over $\mathcal{T}_{\nu'}^{\mathcal{B}_{ij}}$ with parameter ϵ'. If the test outputs YES, set $|\phi^{(1)}\rangle_{ij} = |\phi\rangle$

Claim 6.2. *If there exists projectors $|\psi^{(l)}\rangle\langle\psi^{(l)}|_{ij}$, $|\phi^{(l)}\rangle\langle\phi^{(l)}|_{k\ell}$ on qubit pairs (i,j), (k,ℓ) that have been approximated to a distance δ, then **Test-II**(i,j,k,ℓ) will give us projectors $|\psi^{(l+1)}\rangle\langle\psi^{(l+1)}|_{ij}$ and $|\phi^{(l+1)}\rangle\langle\phi^{(l+1)}|_{k\ell}$ such that*

$$\||\psi^{(l+1)}\rangle\langle\psi^{(l+1)}|_{ij} - |\psi'\rangle\langle\psi'|_{ij}\| \leq \frac{\delta}{2} \quad \text{and} \quad \||\phi^{(l+1)}\rangle\langle\phi^{(l+1)}|_{k\ell} - |\phi'\rangle\langle\phi'|_{k\ell}\| \leq \frac{\delta}{2}. \tag{6.4}$$

Test-II(i,j,k,l) *requires* $O\left(\left(\frac{\delta}{\eta'}\right)^4 \times \left(\frac{\delta}{\nu'}\right)^4\right)$ *trials where δ is the radius of $\mathcal{B}_{ij}$ and $\mathcal{B}_{k\ell}$.*

Proof. Clearly, as **Test-I** works on the complete space of 2-qubit rank 1 projectors, it will also work on the restricted ball of size δ and when all other parameters remain poly(n) sized. Since the existence of nontrivial projectors Π'_{ij} and $\Pi'_{k\ell}$ has already been determined from Step 1, on is sure to find another state over the new η'-net that approximates the projectors according to the new values. Then, setting the parameters as mentioned in Step 2, the bound follows directly from Claim 6.1. For the number of trials, since (k,ℓ) is fixed for (i,j) and vice-versa, the trials only iterate over the number of states in the δ-ball of a γ-net over the space of 2-qubit states which contains $O\left(\frac{\delta^4}{\gamma^4}\right)$ states. Substituting for the values of γ gives the required number of trials. □

Step 2 To approximate the projectors on qubit pairs $(i,j),(k,\ell)$ to polynomial accuracy, collect parameters η, ν and ϵ from Step 1. Set the counter $c = 0$.

- Set $\nu' = \frac{\nu}{2}$, $\eta' = \frac{\eta}{2}, \epsilon' = \frac{\epsilon}{4}$ and $\delta = \sqrt{2\epsilon + \nu^2}$.
- Let $\mathcal{B}_{ij}$ be the ball of radius δ around $|\psi^{(c)}\rangle\langle\psi^{(c)}|_{ij}$ and correspondingly $\mathcal{B}_{k\ell}$.
- Run **Test-II**(i,j,k,ℓ) which output states $|\psi^{(c+1)}\rangle\langle\psi^{(c+1)}|_{ij}$ and $|\phi^{(c+1)}\rangle\langle\phi^{(c+1)}|_{k\ell}$
- Update the parameters such that $\nu = \nu'$, $\eta = \eta'$ and $\epsilon = \epsilon'$.
- Increment the counter and repeat the process till $c = O(\log n)$.

A crucial part of the analysis for Step 2 is to show that it can be executed in polynomial time. This is ensured by showing that the number of trials for each iteration of **Test-II** actually remains a constant independent of n.

Lemma 6.4. *Given projectors $|\psi^{(1)}\rangle\langle\psi^{(1)}|_{ij}, |\phi^{(1)}\rangle\langle\phi^{(1)}|_{k\ell}$ on qubit pairs $(i,j),(k,\ell)$ that have been approximated to a distance δ, Step 2 will successfully find projectors $|\bar{\psi}\rangle\langle\bar{\psi}|_{ij}$ and $|\bar{\phi}\rangle\langle\bar{\phi}|_{k\ell}$ such that $\||\bar{\psi}\rangle\langle\bar{\psi}|_{ij} - |\psi'\rangle\langle\psi'|_{ij}\| \leq \frac{1}{\text{poly}(n)}$ and $\||\bar{\phi}\rangle\langle\bar{\phi}|_{k\ell} - |\phi'\rangle\langle\phi'|_{k\ell}\| \leq \frac{1}{\text{poly}(n)}$. Additionally, this step can be executed in $O(\log n)$ time.*

Proof. Step 2 starts the first iteration in Step 2 with the parameters $\delta, \frac{\nu}{2}, \frac{\eta}{2}$ and proceeds in each iteration by halving these parameters. In effect, at the t^{th} iteration, the parameters used are $\frac{\delta}{2^{t-1}}, \frac{\eta}{2^t}$ and $\frac{\nu}{2^t}$. The costliest operation in executing Step 2 involves **Test-II** being performed at

each iteration. From Claim (6.2) the t^{th} iteration of **Test-II** can be executed in time expressed via the parameters δ_t, ν_t, η_t as

$$O\left(\frac{\delta_t^8}{\eta_t^4 \nu_t^4}\right) = O\left(\frac{\delta^8}{2^{8(t-1)}} \frac{2^{4t}}{\eta^4} \frac{2^{4t}}{\nu^4}\right) = O\left(\frac{\delta^8}{\eta_t^4 \nu_t^4} 2^8\right) \in O(1)$$

where the last inclusion holds as δ, η and ν start as constants. With $O(\log n)$ iterations, this leads to Step 2 being executed in $O(\log n)$ time. Considering the accuracy of the states output, it is clear that in iteration t the projectors output are $\frac{\delta}{2^t}$ close to the projectors in the hidden instance. For $t = O(\log n)$ this translates to a distance of $\frac{\delta}{2^{O(\log n)}} \leq \frac{\delta}{O(n^c)}$ for some constant $c > 0$ and this in turn is written as $\frac{1}{\text{poly}(n)}$ for $\delta \in O(1)$ and the result follows. □

Step 3 To approximate the remaining projectors, do the following:

- Pick a pair of qubits (u, v) that is independent from at least one of the projectors approximated so far.
- Perform Step 1 to approximate $\Pi_{uv}^{(0)}$ to constant accuracy (if it exists). Otherwise, pick another pair of qubits.
- Let (x, y) be independent of (u, v) such that Π_{xy} has been approximated to $\frac{1}{\text{poly}(n)}$ accuracy. Use $\Pi_{xy}^{(0)}, \ldots, \Pi_{xy}^{(O(\log n))}$ to approximate Π_{uv} to $\frac{1}{\text{poly}(n)}$ accuracy as per Step 2.
- Repeat for all possible independent qubit pairs.

To learn a projector Π_{ik} when Π_{ij} has already been found, set qubit j to the mixed state and choose a different Π_{mn} to use in Steps 1 and 2 for improving the accuracy of Π_{ik}

Dealing with higher rank projectors. As mentioned earlier, the tests have been clearly designed assuming the presence of only rank 1 projectors. To generalize **Test-I** for projectors of rank > 2, don't stop after finding just one state $|\psi\rangle$ that succeeds the test. By continuing to iterate over all 2-qubit states, it is possible to find a constant number of states that span the forbidden subspace and then use any one of the linear algebra techniques to find a basis for that space whose dimensions would give the rank of the projector. This would also approximate the basis up to constant accuracy at the end of **Test-I** as the states that will be returned from the test can be shown as having a low distance (or high violation energy) with at least one of the non-zero components of the high rank projector. Then, repeating **Test-II** for each basis element will successfully approximate each of them to $\frac{1}{\text{poly}(n)}$ accuracy. Note that each of these changes do not significantly affect the runtime of the algorithm or the number of trials proposed.

Proceeding to wrap up the proof of Theorem 6.6

Proof of Theorem 6.6. As outlined previously, putting together the 3 steps is required algorithm. Consider **Test-I**(i, j) returning the projector $|\psi\rangle$ and all the states of the form $\rho_{k\ell}^{\alpha}$ used for the test. From Claim 6.1, $\| |\alpha\rangle_{kl} - \Pi'_{kl} \| \leq \frac{1}{\sqrt{8}}$ for some (k, l) and some α. Then, any state that is output by **Test-I** should be closer to Π'_{ij} to have a larger overlap with it. In case of (i, j) being disjoint from (k, ℓ), there is no problem to ensure this.

However, when no projectors independent of (i,j) exist, consider another projector (i,k). Now, the states used in **Test-I** would be of the form ρ_k^α. Let the reduced density matrix on i with respect to $|\psi\rangle\langle\psi|$ be ρ_i. Then the error threshold used for any state output by **Test-I** in this case would be related to $\max_\alpha \mathrm{Tr}(\rho_i|\alpha\rangle_k\Pi'_{ik}) \leq \mathrm{Tr}(\rho_i \mathrm{Tr}_k(\Pi'_{ik}))$. It is possible for the latter value to be almost 0 in the case that ρ_i almost satisfies Π'_{ik} (e.g. the product state on i satisfies Π'_{ik}). This would lead to an inaccurate error threshold and affect the accuracy of the states output by **Test-I**. This explains the necessity of H' not having a *Star-like* configuration.

For the running time argument, from Lemmas 6.3 and 6.4, the running time for finding a projector up to constant accuracy is $O(n^2)$ and to improve the accuracy to $\beta << 1$ takes $O(\log \frac{1}{\beta})$ time. Step 3 essentially repeats the combination of (Step 1 + Step 2) for n^2 pairs of qubits and results in an overall running time of $O(n^2(n^2 + \log \frac{1}{\beta}))$. Setting $\beta < \frac{1}{n^c}$ for some constant c, makes the overall runtime $O(n^4 + n^2 \log n) = O(n^4)$. Similarly, the correctness also follows from combining Lemmas 6.3 and 6.4. □

Using the H output by the above algorithm in a procedure which could find a good approximation to the ground energy of H' would completely solve H–Q2SAT. At this time, though, existing Q2SAT algorithms [ASSZ16, Bra11, dBG16] are not robust to such errors and seem to require $\frac{1}{\exp(n)}$ precision. Our algorithm for H–Q2SAT does allow one to learn the projectors to exponential precision, since the dependence on ϵ in Theorem 6.6 is merely logarithmic. However, in this parameter regime our algorithm is somewhat unrealistic, as this would require the oracle to be able to distinguish between values that are exponentially close together. This seems to give the oracle too much power - because having the ability to distinguish exponentially close quantum states would allow one to solve PP-hard problems [AL98]. In contrast all of the problems considered are in NP due to the presence of classical, poly(n) size witnesses. If our oracle were constrained to be implementable in polynomial time by an experimenter, acting on polynomially many copies of the proposed state ρ, then one could only learn the instance up to error $\epsilon = \frac{1}{\mathrm{poly}(n)}$. A natural open question is to determine whether one can still solve Q2SAT when one only knows the individual clauses to inverse polynomial precision – a fundamental question about the nature of Q2SAT itself.

While one caveat here is with respect to the precision, another is the exclusion of pathological cases, namely *Star-like* configurations. Learning even projector in the case of the interaction graph being a Star is not possible with the current method. However, the intermediate case when there is exactly one edge in the graph that is not independent of the other edges, seems to have an intermediate albeit slightly unnatural solution. In fact, to explicitly learn some projectors and then solve the instance requires the oracle to distinguish between exponentially small values. In particular, the following lemma holds.

Claim 6.3. *Given a* H–$\mathsf{Q2SAT}_{\mathrm{MAX}}$ *problem* $H' = \sum_{(u,v)} \Pi'_{uv}$ *on* n *qubits,* $\epsilon \leq \frac{1}{n^\beta}$ *for a constant* β *and a function* $f(n) \in \exp(n)$*. If there is exactly one edge* (i,k) *that does not have any independent projector, then there is an* $O(n^4 + n^2 \log f(n))$ *time algorithm that can find an*

approximation $H = \sum_{(u,v)} \Pi_{uv}$ *where*

$$\forall\ (i,j), \neq (i,k) \|\Pi'_{ij} - \Pi_{ij}\| \leq \frac{1}{f(n)}$$

and we can find a 2 *qubit state* $|\Phi\rangle ik$ *in* $O(\epsilon^4)$ *time, such that* $\langle\Phi|\Pi'_{ik}|\Phi\rangle \leq \epsilon^2$.

Proof. For every qubit pair except (i,k), use Step 1 to find a constant approximation to the hidden projector and repeat Step 2 for poly(n) iterations, find exponentially close approximations to the hidden projectors. Using these approximations, find an $n-2$-qubit (resp. $n-1$-qubit) state that almost satisfies all projectors except Π'_{ik} following any $O(n+m)$ algorithm to solve Q2SAT [ASSZ16, dBG16]. The $n-1$ qubit state includes either i or k but not both in the satisfiable case. Then, iterating over all 2-qubit (resp. 1-qubit) states on an ϵ-net, and proposing the complete n qubit state to the oracle, will help in finding a state that has low overlap with Π'_{ik}. □

For the remaining case of the Star graph, at the present time, we do not have an algorithm to learn the projectors to any level of accuracy. This is due to the interference of the center of the star with all the projectors skewing the error thresholds used in this type of algorithm. Of course, the brute force technique to find the ground state by iterating over an ϵ-net of all n-qubit states with at most pairwise entanglement leads to an exponential number of trials to be proposed but the power of the oracle doesn't change. Hence, an obvious trade-off between the power of the oracle and the number of trials proposed exists although both techniques currently lead to unnatural algorithmic techniques for the pathological cases.

Barring the Star case, once the approximate Hamiltonian H' is learned, one would like to find a ground state for H' which could approximate the ground state for the H–2SAT instance H. However, unlike the H–2SAT case where the formula learned maintains the satisfiability of the original instance, there is no guarantee that H' does so. In fact, H' turns out to be an ϵ-satisfiable instance of Q2SAT (See Definition 4.10) and for $\epsilon = \frac{1}{\text{poly}(n)}$, H' is an almost frustration free Q2SAT Hamiltonian. Finding a low energy product state to approximate the ground state for H' amounts to solving the Approx − Q2SAT problem for H'. Theorem 4.5 shows the NP-hardness of the Approx − Q2SAT for a specific choice of parameters i.e. where $b = c'a$. However, finding an approximate ground state for H' and, by extension, H is similar to approximating the optimal assignment for robust instances of MAX2SAT[1]. As discussed in Chapter 4, this setting for the Q2SAT case is presently unresolved and left for future work. Hence, the H–$\text{Q2SAT}_{\text{MAX}}$ which now reduces to this setting of the Approx − Q2SAT problem also remains unresolved beyond learning an approximate Hamiltonian H'.

[1]Recall that a robust MAX2SAT instance on n variables has an optimal assignment that is promised to satisfy at least a $1-\epsilon$ fraction of clauses for some $\epsilon = \frac{1}{\text{poly}(n)}$. The approximation algorithm finds a state that satisfies at least $1-\sqrt{\epsilon}$ fraction of clauses [CMM09] in polynomial time

Chapter 7

Lex-first oracle in the trial error model

Question: *What is the minimum amount of information needed to determine if a graph has a monotone graph property?*

The previous chapter considered probabilistic trials in an attempt to quantify the minimum amount of information needed to solve 2SAT. Ultimately, with probabilistic trials, any repetition-free H–2SAT instance can be solved and an unknown instance of Q2SAT can be approximately learned. In this chapter, the same question is posed in the context of monotone graph properties. It is known from Chapter 5 that when the clause index revealing oracle is used, it is not possible to determine if the unknown graph possesses a cycle cover, is bipartite or even connected unless P = NP. However, revealing the edge involved in a constraint trivializes the problem as the underlying instance can be learned with polynomially many trials. An intermediate approach is to consider an oracle which does not reveal an arbitrary violated clause but only the identity of the lexicographically first clause that is violated – i.e., a *lex-first oracle*. For instance, when a certificate C violates clauses as $4, 5, 8, \ldots$ (arranged in ascending order) one learns that "clause 4 is the first clause violated"by this trial. The lex-first oracle denoted as $\mathcal{O}_{\text{FIRST}}$, for a trial $\mathbf{T}$, therefore reveals a clause index which is $\min_j\{C_j(\mathbf{T}) = \mathbb{F}\}$ and any hidden CSP using this oracle is denoted as H–$\text{CSP}_{\text{FIRST}}$. This oracle aims to determine if a graph has a property without learning the complete graph. In fact, there are limitations to completely learning a H–$\text{CSP}_{\text{FIRST}}$ instance, even with unlimited queries made to the lex-first oracle, as shown below.

Limitations to learning

Proposition 7.1. *There is no algorithm which, given an unsatisfiable instance φ and access to the corresponding lex-first oracle, learns all the unique clauses present in φ (even when provided with unlimited queries to the oracle).*

Proof. To show the behaviour of the lex-first oracle in the general CSP setting, consider two H–1SAT instances:

$$\varphi_1 : C_1 = x_1, C_2 = \overline{x}_1, C_3 = x_2, C_4 = \overline{x}_2;$$
$$\varphi_2 : C_1 = x_1, C_2 = \overline{x}_1, C_3 = x_3, C_4 = \overline{x}_4;$$

While it is clear that both these instances are unsatisfiable and $\varphi_1 \neq \varphi_2$, the oracle answers for every trial will be exactly the same for both cases. Any assignment proposed to the oracle will always violate either C_1 (if $a_1 = 0$) or C_2 (if $a_1 = 1$) irrespective of whether the hidden instance is φ_1 or φ_2. Hence, these instances remain indistinguishable to oracle answers and any algorithm in this setting.

A similar behaviour can be demonstrated in the Graph Properties Framework as well. Let the property of interest be connectivity for which the certificate is a spanning tree in the underlying graph. Consider the two disconnected graphs in Figure 7.1 represented in the graph properties framework as ($\overline{x}_{uv}$ denotes that the edge between vertices u, v is missing)

$$G_1 : C_1 = \overline{x}_{12}, C_2 = \overline{x}_{13}, C_3 = \overline{x}_{14}, C_4 = \overline{x}_{24};$$
$$G_2 : C_1 = \overline{x}_{12}, C_2 = \overline{x}_{13}, C_3 = \overline{x}_{14}, C_4 = \overline{x}_{14};$$

As vertex 1 in both G_1 and G_2 (See Figure 7.1) is disconnected from the rest of the graph, the graphs do not contain a spanning tree. Moreover, any spanning tree on the 4 vertices will connect vertex 1 to either vertex 2 (violating C_1), vertex 3 (violating C_2) or vertex 4 (violating C_3). This gives rise to identical oracle answers in both instances and the graphs G_1 and G_2 would remain indistinguishable to any algorithm. □

Figure 7.1: Two disconnected graphs G_1 and G_2

7.1 Structure of Lex-first algorithms

Before dealing with graph properties in the lex-first model, it is useful to understand the general structure underlying every lex-first algorithm. In their introductory paper on the trial and error model, Bei, Chen and Zhang [BCZ13] presented an algorithm that solves hidden SAT with polynomially many queries to the constraint index revealing oracle (i.e., H–SAT$_{\{\mathcal{C}\}}$) provided there is access to a SAT oracle. Recall from Algorithm 2.2 in Chapter 1 that their algorithm maintains a list for each clause that contains all the literals that could be present in the clause. For a SAT instance, each clause C_j has a corresponding list of literals $\hat{C}_j$ which is initialized to all the literals in the system. This list remains consistent with all the trials and corresponding violations revealed by $\mathcal{O}_{\{\mathcal{C}\}}$. For instance, let the all-0s assignment returns clause j as a violation.

Then, the list $\hat{C}_j$ is updated to contain only positive literals as 00 being the forbidden assignment for C_j implies that which ever literals are in the clause, they are falsified by the 00 assignment. This process continues with each trial until a satisfying assignment is found or the entire instance is learned. The SAT oracle is used to generate the next trial in the sequence. From the transfer theorems of Chapter 5, it is known that this achieves the optimal complexity with the clause index revealing oracle.

The lex-first algorithms are in a similar vein with some key changes. The lex-first algorithms also use the lists $\hat{C}_j$ for every clause C_j. Think of the lists as representing the aggregate information one has about the clause C_j. These lists perform a similar function to the A_j^ts used in the transfer theorems (see Theorem 5.1) except that the latter stored assignments violating C_j and the former stores possible variables that could be involved in C_j. For example, if $\hat{C}_j = \{x_1, x_5, \overline{x}_8\}$ for some H–2SAT instance, then $C_j = (\ell_1 \vee \ell_2)$ where $\ell_1, \ell_2 \in \hat{C}_j$. Assuming $\ell_1 \neq \ell_2$, this results in 6 possibilities for what C_j in the hidden instance could be. Successive trials can then be chosen so that each possibility is either ruled out or the algorithm learns what C_j is. So, for H–$2\mathsf{SAT}_{\text{FIRST}}$, $\hat{C}_j$, at any point, is essentially a list of literals that could be in C_j, consistent with the trials proposed and violations received so far. For the Monotone Graph Properties framework, each clause denotes an edge that is missing from the hidden graph. Correspondingly, the list $\hat{C}_j$ is a set of edges such that C_j refers to one of the edges from this set. For example, if $\hat{C}_1 = \{e_2, e_{10}, e_{15}\}$, then $C_1 = (\neg e)$ where $e \in \{e_2, e_{10}, e_{15}\}$. Successive trials will aim to learn which of the three possibilities for e describes the hidden graph. The similarities with Algorithm 2.2 end here.

The first major change is that there is no need for access to a black-box NP oracle as a procedure to generate the next trial from the preceding one is explicitly presented. This is done by showing that given a current trial $\mathbf{T}$ and violation C_j, it is possible to choose a small subset of variables in the trial and change their assignment to create a new trial $\mathbf{T}'$. For H–2SAT, this would mean that a small subset of variables is chosen and their assignment is flipped (i.e., 0 to 1 and vice-versa). For a graph property, it means that some small set of edges from the current trial are removed, and a different set of edges is added to create a new trial. Of course, the results in this work demonstrate that for the examples considered here, this can be done efficiently, in polynomial time, without access to an NP oracle. Informally, if trial $\mathbf{T}_1$ returned C_{j_1} as the violation, the construction picks some suitable candidates $R \subseteq \mathbf{T}_1 \cap \hat{C}_{j_1}$ and changes or removes their assignments to create a partial assignment $\mathbf{T}'$. $\mathbf{T}'$ is then extended to a new trial $\mathbf{T}_2$ such that $\mathbf{T}_2 \neq \mathbf{T}_1$. One of the contributions here is demonstrating how a suitable R for different problems can be chosen efficiently.

The second differing aspect is that the candidates for a clause are eliminated by observing the pattern of the clauses returned in two successive violations and do not depend solely on the current violation observed. Also, the candidates removed at each step are related to the difference between the two corresponding trials and not just the current trial and its violation. This leads to the need for more book-keeping and intricacies in these algorithms. Irrespective

of the problem at hand, this can be seen with some simple reasoning backed by the lex-first behaviour of the oracle. Let the successive trials $\mathbf{T}_1, \mathbf{T}_2$ generate the violations j_1, j_2 respectively. If $j_1 = j_2$, then both trials satisfy all clauses C_j with $j < j_1$. More importantly, C_{j_1} is violated by variables whose assignment remains unchanged between $\mathbf{T}_1, \mathbf{T}_2$. If $j_1 > j_2$, then $\mathbf{T}_1$ satisfied C_{j_2} but $\mathbf{T}_2$ violates it which implies that the additions/changes made to create $\mathbf{T}_2$ are responsible for violating C_{j_2}. Finally, if $j_1 < j_2$, then $\mathbf{T}_1$ violated C_{j_1} but $\mathbf{T}_2$ satisfies it. This implies that the removals/changes made to $\mathbf{T}_1$ were responsible for violating C_{j_1} as, without them, $\mathbf{T}_2$ satisfies C_{j_1}.

Algorithm 7.1 A Lex-first Algorithm Template

Initialize global variables and pick the first guess $\mathcal{G}$. **Repeat**,

1: Propose a trial T_1 and receive index j_1.
2: Construct the next trial T_2 using T_1 and $\hat{C}_{j_1}$.
3: Propose T_2 and receive index j_2.
4: Use $\mathbf{T}_1, \mathbf{T}_2$ to reduce the candidates for C_j in $\hat{C}_j$ where $j = \min\{j_1, j_2\}$
5: Repeat from Step 4 till $\hat{C}_j$ converges to C_j for some j.
6: Use information from the convergence to update $\mathcal{G}$.
7: Update all global variables.

until the oracle says YES or there is no other trial to be found in $\mathcal{G}$

The lex-first algorithm for the unknown input version of a CSP S, H–S, is given in broad strokes in Algorithm 7.1. It is natural to maintain a *guess* $\mathcal{G}$, which is an instance of S that is consistent with the trials proposed and violations revealed by $\mathcal{O}_{\text{FIRST}}$. One way to view this is that for the guess $\mathcal{G}$ paired with the H–S instance G, at any point, $\mathsf{sat}(G) \subseteq \mathsf{sat}(\mathcal{G}) \subseteq W$ where W is the set of admissible assignments for S and $\mathsf{sat}(G), \mathsf{sat}(\mathcal{G})$ denote the trials that satisfy $G, \mathcal{G}$ respectively. Then, starting from a trivial guess such that $W(\mathcal{G}) = W$, as trials are proposed, this guess is modified so that $W(\mathcal{G})$ approaches closer to $W(G)$[1]. Solving H–S$_{\text{FIRST}}$ for the hidden instance G now amounts to finding a trial in $\mathsf{sat}(G) \cap \mathsf{sat}(\mathcal{G})$. So, the lex-first algorithm naturally progresses by proposing trials from $\mathsf{sat}(\mathcal{G})$, ruling them out when they violate G. Note that, $\mathsf{sat}(G) = \mathsf{sat}(\mathcal{G})$ does not necessarily mean that $G = \mathcal{G}$.

Moving to the clause lists, $\hat{C}_j$ is said to *converge* to C_j in the hidden instance when the updated structure of $\hat{C}_j$ at some point reveals at least one of the variables involved in C_j. For example, while dealing with graph properties, convergence occurs when $\hat{C}_j = \{e\}$ and $C_j = (\neg e)$ for an edge e. For compactness, Step 1 to Step 5 of the algorithm is collectively referred to as a *phase* and Step 6 with Step 7 is an *update*. Within a *phase*, Step 2 is the process of constructing the *next trial* and Step 4 is the process of *candidate elimination*. Both steps are detailed with respect to monotone graph properties and 2SAT in the following sections. Now, one can view the algorithm as a series of *phases* with the purpose of each phase being to identify some new *convergence* with the hidden instance. Each phase consists of a sequence of trials proposed to the oracle and the candidates for clauses that turn up as a violation are brought closer to

[1]Here closer implies that $|W(\mathcal{G}) \setminus W(G)|$ shrinks in size progressively.

convergence.

7.2 Graph Properties with the lex-first oracle

Let $\mathcal{P}$ be a monotone graph property and let $\mathrm{S}_{\mathcal{P}}$ be the CSP associated with it as per the Graph Properties framework described in Chapter 1, Section 2.1.1. This section aims to precisely characterize every monotone graph property that can be solved in polynomial time with a lex-first algorithm as given in Algorithm 7.3. An artefact of the conditions required to successfully construct the next trial in this lex-first algorithm is the $\mathcal{P}^{\cap}$ defined below.

Definition 7.1 ($\mathcal{P}^{\cap}$ problem)**.** *The $\mathcal{P}^{\cap}$ problem corresponding to a monotone graph property $\mathcal{P}$, is defined as: given a graph H and a set of edges $E' \subseteq E(H)$, find a certificate $\mathcal{T}$ for $\mathcal{P}$ such that $\mathcal{T} \cap E' \neq \emptyset$.*

Now, for any monotone graph property, the following is demonstrated.

Theorem 7.1. *Let $\mathcal{P}$ be a monotone graph property and its associated CSP be $\mathrm{S}_{\mathcal{P}}$. Then,* H–$\mathrm{S}_{\mathcal{P}\mathrm{FIRST}}$ *is in* P *if the corresponding $\mathcal{P}^{\cap}$ problem can be solved in polynomial time.*

The proof of the theorem explicitly constructs a lex-first algorithm in line with the high level description in Algorithm 7.1. Before describing the algorithm itself, notations and concepts used in the algorithm are given here. Let the hidden instance of H–$\mathrm{S}_{\mathcal{P}}$ be the graph $G = (V, E)$. For ever trial proposed, the aim is for the trial to avoid edges that are known to be absent from G as well as edges that cannot occur in any certificate in G. For this, a *guess graph* $\mathcal{G}$ is maintained as a super-graph of the possible certificates in the hidden graph (recall that $\mathsf{sat}(\mathcal{G}) \supseteq \mathsf{sat}(G)$). Essentially, at any point of time, $\mathcal{G}$ reflects the current knowledge the algorithm has about the hidden graph G. The algorithm starts with $\mathcal{G} = \mathcal{K}_n$, the complete graph on n vertices. As per the framework of Section 2.1.1, each clause $C_j = (\neg e)$ for some $e \notin E$. The candidates for the missing edge that C_j represents is given in the list $\hat{C}_j$ such that it is consistent with the violations seen in the course of the algorithm. For every j, $\hat{C}_j$ is initialized to the set of all possible edges in an n vertex graph i.e. $\hat{C}_j = \{e_1, \ldots, e_{\binom{n}{2}}\}$. Progress in the algorithm would mean that with every trial proposed, some candidates are eliminated from $\hat{C}_j$ for some j. This is done by comparing the differences between successive trials $\mathbf{T}_1$ and $\mathbf{T}_2$ with their respective violations j_1 and j_2.

A clause list $\hat{C}_j$ is said to *converge* to C_j and end a phase when the algorithm finds that the edge(s) in $\hat{C}_j$ will not occur in any certificate for $\mathcal{P}$ in $\mathcal{G}$ and, by extension, in the hidden graph G. The algorithm ends either when a certificate for $\mathcal{P}$ is found in G or it is determined that G does not satisfy $\mathcal{P}$ and the algorithm outputs UNSAT. Edges that are found either to be missing from G or not occurring in any certificate in G are termed *ignored edges*. In other words, the union of edges from all the converged $\hat{C}_j$ forms the set of ignored edges. They are listed in $\mathcal{I}_E$ and an invariant maintained through the algorithm is that $\mathcal{G} = \mathcal{K}_n \setminus \mathcal{I}_E$. Any clause list $\hat{C}_j$ that consists only of ignored edges is called an *ignored clause* and is listed in $\mathcal{I}$. The edges that

change (i.e., removed or added) between successive trials are crucial to identify edges that should be eliminated. They are denoted by $A := \mathbf{T}_1 \setminus \mathbf{T}_2$ and $B := \mathbf{T}_2 \setminus \mathbf{T}_1$. As $\mathcal{G}$ is updated after each phase, it may happen that certain edges become *critical* for $\mathcal{P}$, i.e. they must be present in $\mathcal{G}$ for a certificate to be present in the hidden graph. This set of critical edges is defined as,

$$\mathbf{Crit} := \{e \in \mathcal{G} | \mathcal{G} \setminus \{e\} \text{ does not have a certificate for } \mathcal{P}\}$$

Since the aim is to reduce the possible candidates for a clause while constructing the next trial, a good potential function to track progress is $\sum_{j \notin \mathcal{I}} |\hat{C}_j|$ where $\mathcal{I}$ is the list of *ignored clauses*. The potential function initially starts at $m\binom{n}{2}$ and when all clauses are ignored, it will to 0. As part of the correctness proof of Algorithm 7.3, it will be shown that the lex-first algorithm ensures that each trial successfully reduces the potential function.

7.2.1 An example: Connectivity and spanning trees

To facilitate the discussion of the lex-first algorithm for all graph properties, an example using *graph connectivity* is discussed where the certificates are spanning trees.

Theorem 7.2. *Given an unknown graph G on n vertices accessed via a lex-first oracle $\mathcal{O}_{\text{FIRST}}$, it is possible to determine if G is connected in* poly(n)*-time.*

The proof constructs a lex-first algorithm, given in Algorithm 7.2, following the template of Algorithm 7.1. Step 5 of Algorithm 7.2 details the way to construct a new spanning tree from an existing one with the aim of achieving some progress. Step 4 shows the candidate elimination while Step 5 lists the updates that need to be done if a clause list $\hat{C}_j$ converges (i.e., it is deduced that $C_j = (\neg e)$ for some edge variable e).

Algorithm 7.2 The lex-first algorithm for connectivity

1: Initialize global variables: $\mathcal{G} \leftarrow \mathcal{K}_n$, $\mathcal{I} \leftarrow \emptyset$, $\mathcal{I}_E \leftarrow \emptyset$, $\mathbf{Crit} \leftarrow \emptyset$ and $\forall j$, $\hat{C}_j \leftarrow \{e_1, \ldots, e_{\binom{n}{2}}\}$.
2: If $\mathcal{G}$ has no spanning tree, abort and output UNSAT. Otherwise, find a spanning tree $\mathbf{T}_1$ in $\mathcal{G}$, get a violation j_1 and set $\hat{C}_{j_1} \leftarrow \hat{C}_{j_1} \cap \mathbf{T}_1$.
3: If $\hat{C}_{j_1} \subseteq \mathbf{Crit}$, return UNSAT and abort. If $\hat{C}_{j_1} = \{e"\}$, then $C_{j_1} = (\neg e")$, $j \leftarrow j_1$ and go to Step 5. Otherwise, to create the next trial:
   - Pick an edge e such that $e \in \hat{C}_{j_1} \cap \mathbf{T}_1 \cap \overline{\mathbf{Crit}}$.
   - Pick $e' \in \mathcal{G} \setminus \mathbf{T}_1$ such that $\mathbf{T}_1 \cup \{e'\} \setminus \{e\}$ forms another spanning tree $\mathbf{T}_2$.
4: Use $\mathbf{T}_2$ as the next trial, get the violation j_2 and compare j_1, j_2:
   - If $j_1 = j_2$, $\hat{C}_{j_1} \leftarrow \hat{C}_{j_1} \setminus \{e, e'\}$. Repeat from Step 5.
   - If $j_1 < j_2$, $\hat{C}_{j_1} \leftarrow \{e\} \Rightarrow C_{j_1} = (\neg e)$ and $j \leftarrow j_1$.
   - If $j_2 < j_1$, $\hat{C}_{j_2} \leftarrow \{e'\} \Rightarrow C_{j_2} = (\neg e')$ and $j \leftarrow j_2$.
5: Perform the updates such that: $\mathcal{I}_E \leftarrow \mathcal{I}_E \cup \hat{C}_j$, $\mathcal{G} \leftarrow \mathcal{K}_n \setminus \mathcal{I}_E$, $\mathcal{I} \leftarrow \{j' | \hat{C}_{j'} \subseteq \mathcal{I}_E\}$ and $\mathbf{Crit} \leftarrow \{e \in \mathcal{G} | \mathcal{G} \setminus \{e\}$ is not connected$\}$.
6: Repeat from Step 9 till the oracle returns YES.

Proof. The goal here is to show that Algorithm 7.2 can find a spanning tree in the hidden graph

G, if one exists, or output UNSAT and that this can be done in poly(n) time when G is a graph on n vertices. The correctness of the algorithm hinges on the following arguments:(a) every candidate elimination process results in decreasing the potential function $\sum_{j \notin \mathcal{I}} |\hat{C}_j|$; (b) the algorithm outputs UNSAT, only when G is not connected; (c) the trials do not repeat in an endless loop within any phase. When all three arguments hold, every trial is unique and either decreases $|\hat{C}_j|$ or results in learning the edge in C_j. Then, assuming G has a spanning tree, with at most $O(m\binom{n}{2})$ trials, either the oracle returns YES or all the edges absent from G can be learned using which the spanning tree in G can be found directly.

For the first argument, let $\mathbf{T}_1$ and $\mathbf{T}_2$ be the trials with corresponding violations j_1 and j_2. Also, $\mathbf{T}_2$ has been created by removing an edge e from $\mathbf{T}_1$ and adding the edge e'. The lex-first nature of the oracle suggests that when $j_2 < j_1$, C_{j_2} was satisfied by $\mathbf{T}_1$ but generated a violation with $\mathbf{T}_2$. The only difference between these two trials is the edge e' implying that $C_{j_2} = (\neg e')$. Similarly, when $j_1 < j_2$, the edge that was removed satisfied C_{j_1} implying that $C_{j_1} = (\neg e)$. In both these scenarios, a clause j is added to the list of ignored clauses $\mathcal{I}$ due to which the potential function decreases by $|\hat{C}_j|$. When $j_1 = j_2$, both trials violate C_{j_1} implying that it is one of the edges common to both trials that causes the violation. This excludes the edges $\{e, e'\}$ from $\hat{C}_{j_1}$. As e was originally picked from $\hat{C}_j$ but is now eliminated, $|\hat{C}_j|$ decreases by 1 and accordingly decreases the potential function.

For the second argument, it has to be shown via induction that, if G has spanning trees, all those spanning tree are also present in $\mathcal{G}$ throughout the run of the algorithm. Initially $\mathcal{G}$ is the complete graph $\mathcal{K}_n$ which contains all the possible spanning trees in an n vertex graph. This satisfies the hypothesis. Now, consider the situation where $\mathcal{G}$ is updated. Denote $\mathcal{G}$ before the update as $\mathcal{G}^{(i)}$ and $\mathcal{G}$ after the update as $\mathcal{G}^{(f)}$. By the inductive hypothesis, suppose that $\mathcal{G}^{(i)}$ contained all the spanning trees in G. Notice that the update happens only when the absent edge e represented by a clause C_j is learned. In addition to this, the only difference between $\mathcal{G}^{(f)}$ and $\mathcal{G}^{(i)}$ is the missing edge e. Obviously, e, as it is absent from G, is not part of any spanning tree in G. Hence, $\mathcal{G}^{(f)}$ also contains all the spanning trees in G satisfying the inductive hypothesis. As a result, if $\mathcal{G}$ has no spanning tree, then G also has no spanning tree. Additionally, if an edge is critical for $\mathcal{G}$, it is also critical for G. So, if here exists some j such that $\hat{C}_j \subseteq \mathbf{Crit}$, it implies that C_j represents the absence of a critical edge from G due to which G has no spanning tree. Hence, if the algorithm outputs UNSAT, then G is not connected.

For the final argument, consider the way a new trial is created in Step 5. Since e is not critical to $\mathcal{G}$, it is part of a cycle C in $\mathcal{G}$ and one edge of C is missing from $\mathbf{T}_1$. This missing edge is the required e' as adding it to $\mathbf{T}_1$ would complete C but removing e after that would ensure that there is no cycle in $\mathbf{T}_2$. As all the vertices in C still remain connected after swapping these edges and no other part of $\mathbf{T}_1$ was changed, $\mathbf{T}_2$ remains a spanning tree too. The candidate elimination in Step 4 loops back to Step 5 if $j_1 = j_2$ and $|\hat{C}_{j_1}| > 1$. Observe that the next trial $\mathbf{T}_2'$ is created after the candidate elimination by removing an edge from the updated $\hat{C}_{j_1} \cap \mathbf{T}_1$. However, at this point e has already been removed from $\hat{C}_{j_1}$ and so, it won't be removed from

$\mathbf{T}_1$ to construct $\mathbf{T}'_2$. Hence, $\mathbf{T}'_2 \neq \mathbf{T}_2$ because $e \in \mathbf{T}'_2 \setminus \mathbf{T}_2$. $|\hat{C}_{j_1}| \leq |\mathbf{T}_1| = n-1$, and its size strictly decreases with each repetition. So, the phase can continue for at most $n-2$ trials before $|\hat{C}_{j_1}| = 1$ or $\hat{C}_{j_1} \subseteq \mathbf{Crit}$ and the algorithm will not get stuck in an endless phase loop.

For the complexity, note that it is possible to find a spanning tree in $\mathcal{G}$ in $O(n^2)$ time. A simple graph traversal algorithm can find the cycle C and the edge e' in $O(n + \binom{n}{2} - m) = O(n^2)$ time since $|V| = n$ and $|E| = \binom{n}{2} - m$ in this framework. The candidate elimination step can be executed in $|\hat{C}_{j_1}| = O(n)$ time. So, a clause can be determined in $O(n^3)$ time in the worst case. The step which updates the variables requires at most $O(mn^2)$ time as it may have to modify each clause list. Assuming this happens for each of the m clauses, the algorithm totally consumes $O(m^2n^2 + mn^3) = O(mn^3)$ time before a certificate for G is found or all the clauses are determined. In the latter case, it is possible to find a spanning tree in G in $O(|E|) = O(n^2)$ time. Hence, it is possible to determine if a hidden graph G is connected in $O(mn^3)$ time. □

7.2.2 The Algorithm

To prove Theorem 7.1, Algorithm 7.2 is generalized to work for any monotone increasing graph property $\mathcal{P}$ as described in Algorithm 7.3. The algorithm is designed such that, at any point, if the oracle return YES, then abort and return the corresponding trial as certificate for $\mathcal{P}$ in the hidden graph G. If not, the process is repeated till all the clauses are learned or it is determined that G does not satisfy $\mathcal{P}$.

Algorithm 7.3 A lex-first algorithm for H–S$_{\mathcal{P}\text{FIRST}}$

1: Initialize global variables, setting $\mathcal{G} \leftarrow \mathcal{K}_n$, $\mathcal{I} \leftarrow \emptyset$, $\mathcal{I}_E \leftarrow \emptyset$, $\mathbf{Crit} \leftarrow \emptyset$ and $\forall j$, $\hat{C}_j \leftarrow \{e_1, \ldots, e_{\binom{n}{2}}\}$.
2: If there are no trials in $\mathcal{G}$, abort and output UNSAT.
3: Let $j' \leftarrow \min\{j \mid j \notin \mathcal{I}\}$. Find a certificate for $\mathcal{P}$ in $\mathcal{G}$, $\mathbf{T}_1 = \mathcal{A}(\mathcal{G}, \hat{C}_{j'})$, get a violation j_1 and set $\hat{C}_{j_1} \leftarrow \hat{C}_{j_1} \cap \mathbf{T}_1$.
4: If $\hat{C}_{j_1} \subseteq \mathbf{Crit}$, return UNSAT and abort. If $\hat{C}_{j_1} = \{e"\}$, then $C_{j_1} = (\bar{e"})$, set $j \leftarrow j_1$ and go to Step 6. Otherwise, to create the next trial:
   - Pick an edge, $e \in \hat{C}_{j_1} \cap \mathbf{T}_1 \cap \overline{\mathbf{Crit}}$
   - Find a certificate $\mathbf{T}_2 = \mathcal{A}(\mathcal{G} \setminus \{e\}, \hat{C}_j \setminus \{e\})$
   - Set $A \leftarrow \mathbf{T}_1 \setminus \mathbf{T}_2$ and $B \leftarrow \mathbf{T}_2 \setminus \mathbf{T}_1$.
   - If no $\mathbf{T}_2$ can be created for any choice of $e \in \hat{C}_{j_1} \cap \mathbf{T}_1 \cap \overline{\mathbf{Crit}}$, $j \leftarrow j_1$ and go to Step 6.
5: Use $\mathbf{T}_2$ as the next trial, get the violation j_2 and compare j_1, j_2:
   - If $j_1 = j_2$, $\hat{C}_{j_1} \leftarrow \hat{C}_{j_1} \setminus A$.
   - If $j_1 < j_2$, $\hat{C}_{j_1} \leftarrow \hat{C}_{j_1} \cap A$.
   - If $j_2 < j_1$, $\hat{C}_{j_2} \leftarrow \hat{C}_{j_2} \cap B$, $j_1 \leftarrow j_2$ and $\mathbf{T}_1 \leftarrow \mathbf{T}_2$.
   - $j \leftarrow \min\{j_1, j_2\}$ and if $|\hat{C}_j| > 1$, repeat from Step 4.
6: Update $\mathcal{I}_E \leftarrow \mathcal{I}_E \cup \hat{C}_j$, $\mathcal{G} \leftarrow \mathcal{K}_n \setminus \mathcal{I}_E$, $\mathcal{I} \leftarrow \{j' | \hat{C}_{j'} \subseteq \mathcal{I}_E\}$, $\mathbf{Crit} \leftarrow \{e \in \mathcal{G} | \mathcal{G} \setminus \{e\}$ does not satisfy $\mathcal{P}\}$.
7: Repeat from Step 2 till the oracle returns YES.

The parallels between the lex-first template in Algorithm 7.1 and Algorithm 7.3 are that Step 4 gives the procedure to create the next trial and Step 5 performs candidate elimination. A phase involves creating a series of trials and repeating these two steps in succession until an update

is performed in Step 6. Note that Algorithm 7.3 uses, as a subroutine, an algorithm $\mathcal{A}$ for $\mathcal{P}^{\cap}$ to create the trials that will be proposed to the lex-first oracle. $\mathcal{A}$ takes as input a graph H, a set of edges E' and outputs a certificate $\mathcal{T}$ in H for $\mathcal{P}$ such that $\mathcal{T} \cap E' \neq \emptyset$. The analysis of Algorithm 7.3 is similar to the proof of Theorem 7.2 in that the following arguments are shown to hold:(a) every candidate elimination process results in decreasing the potential function $\sum_{j \notin \mathcal{I}} |\hat{C}_j|$; (b) the algorithm outputs UNSAT, only when G is not connected; (c) the trials do not repeat in an endless loop within any phase.

Proof of Theorem 7.1. Let $\mathcal{A}$ be an algorithm for $\mathcal{P}^{\cap}$ that requires time T to execute and let the maximum certificate size for $\mathcal{P}$ be s. Let j_1, j_2 be two successive violations with their corresponding trials $\mathbf{T}_1$ and $\mathbf{T}_2$. It will be demonstrated that Algorithm 7.3 can, in poly(T, n)-time, find a certificate for $\mathcal{P}$, in the hidden n-vertex graph G, if one exists, or output UNSAT. The key idea is to show that Step 4 and Step 5 don't execute in an endless loop in a phase. This can be done by showing that trials within a phase do not repeat. Also, after at most $O(m)$ successive trials, the potential function $\sum_{j \notin \mathcal{I}} |\hat{C}_j|$ is shown to strictly decrease. Then, assuming G satisfies $\mathcal{P}$, with at most $O(m^2\binom{n}{2})$ unique trials, a certificate can be found or all the edges absent from G can be learned. If all the clauses of G become known, then using $\mathcal{A}(G, E(G))$, a certificate for $\mathcal{P}$ in G (trivially, intersecting the edge set of G) can be found.

To this aim, first it is demonstrated that the algorithm outputs UNSAT, only when G does not satisfy $\mathcal{P}$. Essentially, via induction, it is shown that if G contains certificates for $\mathcal{P}$, then $\mathcal{G}$ always contains these certificates at every point of the algorithm. $\mathcal{G}$ is initialized to the complete graph on n vertices $\mathcal{K}_n$ and $\mathcal{P}$ is a monotone increasing property. Hence, $\mathcal{G}$ will contain every possible certificate for $\mathcal{P}$ on n vertex graphs. The place where $\mathcal{G}$ changes is during the updates. Let $\mathcal{G}$ before the update be denoted $\mathcal{G}^{(i)}$ and after the update let is be denoted as $\mathcal{G}^{(f)}$. Assume that $\mathcal{G}^{(i)}$ contains all certificates for $\mathcal{P}$ in G.

One cause for an update is when there is no choice of $e \in \hat{C}_{j_1} \cap \mathbf{T}_1 \cap \overline{\mathbf{Crit}}$ to create the trial $\mathbf{T}_2$ in Step 4. In this scenario, $\hat{C}_{j_1} \cap \mathbf{Crit} = \emptyset$ as otherwise, any $e' \in \hat{C}_{j_1} \cap \mathbf{Crit}$ would satisfy the requirement to create the next trial. Since, $e \notin \mathbf{Crit}$, there definitely exists at least one other certificate in $\mathcal{G}^{(i)} \setminus \{e\}$. But, the certificate(s) in $\mathcal{G}^{(i)} \setminus \{e\}$ are such that they don't contain any of the $e' \in \hat{C}_{j_1} \setminus \{e\}$ i.e. $\mathbf{T}_1$ is the only certificate in $\mathcal{G}$ involving the edges in $\hat{C}_{j_1}$. However, the C_{j_1} actually represents one of the edges from $\hat{C}_{j_1}$. Additionally, the only certificate involving this edge and the remaining edges from $\hat{C}_{j_1}$ generates a violation. Hence, no valid certificate for G will include the edges in $\hat{C}_{j_1}$ resulting in their being label as ignored edges. Then by creating $\mathcal{G}^{(f)} = \mathcal{G}^{(i)} \setminus \hat{C}_{j_1}$ does not affect the certificates of G in these graphs. The other cause to update occurs when the absent edge e represented by some clause C_j is determined which is similar to the spanning tree case. Using the same reasoning, the certificates of G which are already in $\mathcal{G}^{(i)}$ will remain in $\mathcal{G}^{(f)}$ as well. This satisfies the inductive hypothesis. Again, the reasoning for critical edges carries over from the proof of Theorem 7.2 and so, if the algorithm outputs UNSAT, then G does not satisfy $\mathcal{P}$.

To argue that the trials in each phase are unique and that the potential function strictly decreases, consider the three cases of Step 5 where candidate elimination is done. For each case, it is shown that at least one edge is eliminated from $\hat{C}_j$ for $j = \min\{j_1, j_2\}$ after at most $O(m)$ successive trials. Additionally, for any pair of trials $\mathbf{T}_1, \mathbf{T}_2$, it is shown that subsequent trials $\mathbf{T}'$ constructed after candidate elimination, in a phase, cannot be $\mathbf{T}_1$ or $\mathbf{T}_2$. For some j, let $\hat{C}_j^{(i)}$ denote $\hat{C}_j$ before the candidate elimination and $\hat{C}_j^{(f)}$ denote it after the elimination. The three cases are discussed below:

$j_2 = j_1$: The violation observed is $j_1 = j_2$ and $\mathbf{T}_2$ has been created for some e picked from $\hat{C}_{j_1}^{(i)} \cap \mathbf{T}_1 \cap \overline{\mathbf{Crit}}$. Also, $\hat{C}_{j_1}^{(f)} = \hat{C}_{j_1}^{(i)} \setminus A$ where $A = \mathbf{T}_1 \setminus \mathbf{T}_2$ as per the first case of Step 5. This means that the edges picked for subsequent trials are not from A. Suppose by way of contradiction that, for the subsequent trial, the edge e" in $\hat{C}_{j_1}^{(f)}$ on being removed creates $\mathbf{T}_2$ again. In other words, $e" \in \mathbf{T}_1 \setminus \mathbf{T}_2$ which leads to a contradiction as e" is not in A. Hence, subsequent trials cannot create $\mathbf{T}_2$ again. By forcing that $e \in A$, $e \notin \hat{C}_{j_1}^{(f)}$ due to which $|\hat{C}_{j_1}|$ strictly decreases in size.

$j_2 > j_1$: From Step 5, one of the edges removed from $\mathbf{T}_1$ satisfied C_{j_1} implying that $\hat{C}_{j_1}^{(f)} = \hat{C}_{j_1}^{(i)} \cap A$. Using $\mathcal{A}$ to find the next trial forces some $e' \in \hat{C}_{j_1}^{(i)} \setminus \{e\}$ to be present in both trials. Hence, $\hat{C}_{j_1}^{(f)} \subset \hat{C}_{j_1}^{(i)}$ and $|\hat{C}_{j_1}|$ strictly decreases. Suppose that the $f \in \hat{C}_{j_1}^{(f)}$ has been chosen to create the next trial $\mathbf{T}_2'$ and that $\mathbf{T}_2 = \mathbf{T}_2'$. However, this would mean that the $f' \in \hat{C}_{j_1}^{(f)} \setminus \{f\}$ which is forced to be in $\mathbf{T}_1, \mathbf{T}_2'$ is, by assumption, also in $\mathbf{T}_2$. As $A = \mathbf{T}_1 \setminus \mathbf{T}_2$, this implies that $f' \notin A \Rightarrow f' \notin \hat{C}_{j_1}^{(f)} = \hat{C}_{j_1} \cap A$ which is a contradiction. Hence, any subsequent trial cannot be $\mathbf{T}_2$.

$j_2 < j_1$: Step 5 indicates that it was one of the edges added to create $\mathbf{T}_2$ that violated C_{j_2} such that $\hat{C}_{j_2}^{(f)} = \hat{C}_{j_2}^{(i)} \cap B$ where $B = \mathbf{T}_2 \setminus \mathbf{T}_1$. The nature of the "lex-first" oracle returning the first violated index results in each trial $\mathbf{T}$ being associated with a specific violation j. In other words, any time $\mathbf{T}$ is proposed as a trial, the oracle always returns the same j. This allows for a certain symmetry between the cases when $j_1 < j_2$ and $j_2 < j_1$ with some change of variables. Specifically, let $\mathbf{T}_1' = \mathbf{T}_2$, $\mathbf{T}_2' = \mathbf{T}_1$ which would set $A' = \mathbf{T}_1' \setminus \mathbf{T}_2'$. Now, this case can be viewed as $\mathbf{T}_2'$ created from $\mathbf{T}_1'$ by removing the edges in A' and adding the edges in $B' = \mathbf{T}_2' \setminus \mathbf{T}_1'$ giving violations j_2 followed by $j_1 > j_2$. The reasoning for subsequent trials not being $\mathbf{T}_2' = \mathbf{T}_1$ then follows from the previous case. Moving to the question of decreasing the potential function, it may happen that $\hat{C}_{j_2}^{(f)} = \hat{C}_{j_2}^{(i)}$ and no candidates are eliminated. However, there are only $O(m)$ times that $j_2 < j_1$ can successively occur before $j' = \min\{j | j \notin \mathcal{I}\}$ is reached. For this clause, candidates will have to be eliminated as any violation returned will be $\geq j'$.

For the complexity argument, each execution of Step 3 takes $O(T)$ time, Step 6 is limited by $O([T + m]n^2)$ to find critical edges and modify each clause set and Step 5 takes $O(|\mathbf{T}|) = O(s)$ time as the size of a clause set is limited by the size of the certificate. Each run of Step 4 requires cycling through at most $|\hat{C}_j|$ choices for e before the next trial is found using the algorithm for $\mathcal{P}^{\cap}$ with $E' = \hat{C}_j \setminus \{e\}$. The total running time would be $O(sT)$ since $|\hat{C}_j| \leq s$. On comparing violations j_1, j_2 in Step 5, if $j_1 = j_2$ or $j_1 > j_2$, at least one candidate in $\hat{C}_{j_1}$

will be eliminated and when $j_1 < j_2$, this happens after at most $O(m)$ trials. Hence, in the worst case, a candidate is eliminated in $O(m[sT + s])$ time and a clause can be determined in $O(ms[sT + s]) = O(ms^2T)$ time with $O(ms)$ trials. The total time for each clause can be upper bounded by $O(T + ms^2T + [T + m]n^2) = O(ms^2T + [T + m]n^2)$ and for m clauses, the algorithm requires $O(m^2s^2T + [T + m]mn^2)$ time before all clauses are determined or a certificate for G is found. The size of the certificate can be at most the number of edges in an n-vertex graph giving an overall complexity of $O(m^2n^4T)$ when the certificate size is large i.e. $O(n^2)$. Clearly, with $m = O(n^2)$, the final running time is poly(n)-time when $T =$ poly(n). Hence, any monotone graph property $\mathcal{P}$ has a polynomial time lex-first algorithm if it has a polynomial time algorithm for the $\mathcal{P}^{\cap}$ problem associated with it. □

7.2.3 Applications

Theorem 7.1 is applied to the following properties: (a) Directed Spanning Tree (DST); (b) Undirected Cycle Cover (UCC); (c) Directed Cycle Cover (DCC); (d) Bipartite Perfect Matching (BPM); (e) Directed Path (DPATH); and (f) Undirected Path (UPATH).

Theorem 7.3. *The following properties in the Graph Properties Framework with the have polynomial time algorithms: (a)* H–$\mathsf{DST}_{\text{FIRST}}$*; (b)* H–$\mathsf{UCC}_{\text{FIRST}}$*; (c)* H–$\mathsf{DCC}_{\text{FIRST}}$*; (d)* H–$\mathsf{BPM}_{\text{FIRST}}$*; (e)* H–$\mathsf{DPATH}_{\text{FIRST}}$*; (f)* H–$\mathsf{UPATH}_{\text{FIRST}}$*.*

Proof. If a polynomial time algorithm as mentioned in Theorem 7.1 exists for a property $\mathcal{P}$, then it is straightforward to also find a certificate for $\mathcal{P}$ in a graph $H(V, E)$ by setting $E' = E(H)$. Hence, it is enough to show that all the above properties satisfy the fact that a certificate containing an edge $E' \in A$ can be found in H in polynomial time.

From previous works in graph theory, all the above properties definitely have polynomial time algorithms to find certificates in a given graph H [Die12, Edm67, Tut54]. Now consider the condition in Theorem 7.1. Let E' be the set of edges intersecting which a certificate has to be found. Clearly, if a certificate can be found intersecting a specified edge (u, v), then with $|E'|$ iterations over $\{(u, v)|(u, v) \in A\}$, an intersecting certificate can be found.

Given a graph H and a specified edge (u, v), subdivide (u, v) as given below:

- If $H = (U \cup V, E)$ is a bipartite graph with $u \in U$ and $v \in V$, add two vertices $\tilde{v}, \tilde{u}$ such that $\tilde{U} = U \cup \{\tilde{u}\}$, $\tilde{V} = \cup\{\tilde{v}\}$ and $\tilde{E} = E \cup \{(u, \tilde{v}), (\tilde{u}, v)\} \setminus \{(u, v)\}$ and $H' = (\tilde{U} \cup \tilde{V}, \tilde{E})$.
- If $H = (V, E)$ is an undirected graph, add a vertex x such that $V' = V \cup \{x\}$ and $\tilde{E} = E \cup \{(u, x), (x, v)\} \setminus \{(u, v)\}$ and $H' = (\tilde{V}, \tilde{E})$.
- If $H = (V, E)$ is a directed graph, add a vertex x such that $\tilde{V} = V \cup \{x\}$, $\tilde{E} = E \cup \{(u, x), (x, v)\} \setminus \{(u, v)\}$ where the direction of the newly added edges is the same as the direction of (u, v).

Now for each property, the condition in Theorem 7.1 becomes equivalent to finding a certificate for the property in H' which can be done in polynomial time as they are polynomial time solvable properties. To illustrate, consider each property of interest:

1. H has a BPM containing (u, v) if and only if H' has a BPM $\mathcal{M}'$. Clearly, $\{(u, v'), (u', v)\} \in \mathcal{M}'$ as the only way to match u' and v'. Now $\mathcal{M}' \setminus \{(u, v'), (u', v)\} \cup \{(u, v)\}$ is the required matching in H.
2. H has a UCC if and only if H' has a UCC where the only way to cover x is a cycle passing through $(u, x), (x, v)$. Now replacing these two edges in this cycle cover with (u, v) gives the required cycle cover in H. The DCC case can be argued similarly.
3. H has a UPATH from s to t passing through (u, v) if and only if there are vertex disjoint UPATHs from s to x and from x to t in H'. Now replacing $(u, x), (x, v)$ with (u, v) in the join of these paths is the required path in H. The DPATH can be solved similarly.
4. The subdivision of edges is not needed for the spanning tree problem. Consider a spanning tree $\mathcal{T}$ in H not containing (u, v). This means that adding (u, v) to $\mathcal{T}$ creates a cycle. Then, adding (u, v) to $\mathcal{T}$ and removing an edge of the created cycle will give a spanning tree containing (u, v). The DST case follows similarly. □

If there exists a polynomial time algorithm for $\mathcal{P}^{\cap}$ for a property $\mathcal{P}$, then it is straightforward to also find a certificate for $\mathcal{P}$ in a graph $H(V, E)$ by setting $E' = E(H)$. This along with the above illustration would make one suspect that every polynomial time solvable monotone graph property satisfies the above condition. However, there exists a counter example that refutes this possibility.

Claim 7.1. *Let $\mathcal{P}$ be the property of finding an odd-length directed cycle with the associated CSP* $\mathrm{S}_{\mathcal{P}}$*. Then, it is* NP*-hard to solve* H–$\mathrm{S}_{\mathcal{P}}$ *using the lex-first oracle* $\mathcal{O}_{\mathrm{FIRST}}$*.*

Proof. There exists a linear time algorithm to find an odd-length cycle in a directed graph using breadth first search[2]. However, finding an odd length cycle containing a specific directed edge (v, u) is equivalent to asking if there exists an even length path from u to v, both explicitly specified. This problem has been proved to be NP-complete by LaPaugh and Papadimitriou [LP84]. □

Interestingly, the undirected version of this problem can still be evaluated in polynomial time [LP84] and so the hidden, undirected version of this problem is still in P. This prompts one to ask if there is any undirected graph property which could also serve as a counterexample. This question is currently left for future work.

7.3 2SAT with the lex-first oracle

The previous section showed that using the lex-first oracle certain polynomial time monotone graph properties are still polynomial time decidable in the hidden setting. This prompts the consideration of how well this oracle performs with CSPs whose clauses are not unary. The immediate choice would be to consider an instance of hidden 2SAT. Recall that a 2SAT

[2]Colour vertices at even and odd distances differently. Then look for monochromatic non-tree edges.

formula ϕ is defined n variables $\mathbf{x} = (x_1, \ldots, x_n)$ and m clauses $\varphi = \wedge_{j=1}^{m} C_j$ where each C_j is a disjunction of two literals. The trials are assignments $\mathbf{a} = a_1 \ldots a_n$ with $\mathbf{a} \in \{0,1\}^n$ and a satisfying assignment satisfies each C_j. If a 2SAT formula contains any unary constraint (ℓ), it will be considered as a binary constraint of the form $(\ell \vee \mathbb{F})$ where $\mathbb{F}$ denotes the literal being `false`. Given a set of literals, L, $\overline{L}$ contains the negation of each literal in L while the complementary set is given by $L^c := \{x_1, \overline{x}_1, \ldots x_n, \overline{x}_n\} \setminus L$. A polynomial time algorithm to solve every H–$\mathsf{2SAT}_{\text{FIRST}}$ instance is described by proving that

Theorem 7.4. *Given a* H–2SAT *instance* $\varphi = \wedge_{j=1}^{m} C_j$ *on* n *variables and* m *clauses accessed using a lex-first oracle* $\mathcal{O}_{\text{FIRST}}$*, there exists a* poly$(n)$ *time algorithm that finds a satisfying assignment for* φ *if one exists or outputs* UNSAT*.*

The proof follows from the analysis of Algorithm 7.4. While many notations carry over from Section 7.2, the differences are explicitly defined here. The algorithm ends either when a satisfying assignment to φ is found or it is determined that φ cannot be satisfied.

Since each 2SAT clause contains at most 2 literals, $\hat{C}_j$ is the ordered tuple $\hat{C}_j = (\hat{L}_j, \hat{R}_j)$ for C_j where $\hat{L}_j, \hat{R}_j \subseteq \{\mathbb{F}, x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n\}$. Each component of the tuple is the list of candidates for the corresponding literal of clause j on proposing trial $\mathbf{a}$. In the beginning, both sets will be $\{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n\}$. Note that if at any time $\hat{R}_j$ becomes empty, then C_j is determined to be a unary clause and $\hat{R}_j \leftarrow \{\mathbb{F}\}$ so that the invariant $\forall j, |\hat{L}_j|, |\hat{R}_j| \geq 1$ is maintained at all times. The order of the literals in C_j does not matter i.e. $(x \vee y)$ and $(y \vee x)$ are considered to be equivalent.

Similar to the *guess* graph maintained for graph properties, for the H–$\mathsf{2SAT}_{\text{FIRST}}$ instance φ, the *guess* is a 2SAT *induced formula* Φ such that for all assignments $\mathbf{a} \in \{0,1\}^n$, $\varphi(\mathbf{a}) = \mathbb{T} \Rightarrow \Phi(\mathbf{a}) = \mathbb{T}$. In other words, the induced formula helps maintain the invariant $\mathsf{sat}(\Phi) \supseteq \mathsf{sat}(\phi)$. As Φ is meant to be consistent with all the violations and clauses already known, it is defined as

$$\Phi := \bigwedge_{\{C_j : |\hat{L}_j| = |\hat{R}_j| = 1\}} (\ell_1 \vee \ell_2) \text{ where } \hat{L}_j = \{\ell_1\} \text{ and } \hat{R}_j = \{\ell_2\}$$

Given a guess Φ, $\mathbf{a}_\Phi$ denotes a partial assignment restricted to the variables that appear in Φ such that $\mathbf{a}_\Phi(\Phi) = \mathbb{T}$. Correspondingly, it is possible to partition the n variables depending on if they occur in Φ or not. This is done by defining $V_\Phi := \{i | x_i \text{ or } \overline{x}_i \text{ occurs in } \Phi\}$ and $V_\Phi^c := \{1, \ldots, n\} \setminus V_\Phi$. The literals can also be similarly partitioned with $L_\Phi := \{x_i, \overline{x}_i | i \in V_\Phi\}$ and its complement L_Φ^c. The potential function used to mark progress is $\sum_{j=1}^{m} |\hat{C}_j|$ where $|\hat{C}_j| = |\hat{L}_j| + |\hat{R}_j|$. It initially starts at $m \times 4n$, where each component starts with $2n$ literals and should drop to $2m$. In the analysis of the algorithm the rate at which each trial reduces the potential function is shown. For ease of notation, $\hat{S}_j$ refers to a component of $\hat{C}_j$ and takes the

values

$$\hat{S}_j = \begin{cases} \hat{L}_j & \text{if } |\hat{L}_j|, |\hat{R}_j| > 1 \text{ i.e. } C_j \text{ is unknown;} \\ \hat{R}_j & \text{if } |\hat{L}_j| = 1, |\hat{R}_j| > 1 \text{ i.e. } C_j \text{ is partially known;} \\ \emptyset & \text{if } |\hat{L}_j| = |\hat{R}_j| = 1 \text{ i.e. } C_j \text{ is known.} \end{cases}$$

When a literal in C_j is learned during the course of the algorithm, $\hat{S}_j$ will be updated to reflect if C_j is known, or partially known. Given two n-length Boolean assignments $\mathbf{a}, \mathbf{a}'$, it is useful to find the variables with the same assignment in $\mathbf{a}$ and $\mathbf{a}'$ and those which flipped. For this, define

$$A(\mathbf{a}, \mathbf{a}') := \{x_k | a_k = a'_k = 0\} \cup \{\overline{x}_k | a_k = a'_k = 1\}$$
$$B(\mathbf{a}, \mathbf{a}') := \{x_k | a_k = 0 \text{ and } a'_k = 1\} \cup \{\overline{x}_k | a_k = 1 \text{ and } a'_k = 0\}$$

It is assumed that set operations consume unit time. As the size of the sets is $O(n)$ they will only add a linear factor to the final run-time which still satisfies the polynomial time requirement.

7.3.1 Algorithm Details

The H–2SAT lex-first algorithm is discussed below and outlined in Algorithm 7.4. The trials are generally assumed to generate violations as otherwise, the algorithm can be aborted on having found a satisfying assignment.

Algorithm 7.4 A H-2SAT lex-first algorithm outline

1: Initialization: Set $\forall j, \hat{L}_j \leftarrow \hat{R}_j \leftarrow \{x_1, \overline{x}_1, \ldots, x_n, \overline{x}_n\}$, $\hat{S}_j \leftarrow \hat{L}_j$ and $\Phi \leftarrow \mathbb{T}$.
2: Using $O(n)$ trials learn at least one clause. Add the clauses learned to Φ.
- If trial $\mathbf{a}$ is found such that $\mathcal{O}_{\text{FIRST}}(\mathbf{a})$ returns YES, return $\mathbf{a}$.

Update the guess Φ
3: Add the clauses learned to Φ. For all literals ℓ such that $\ell = \mathbb{T} \Rightarrow \Phi$ is UNSAT:
- Replace ℓ with $\mathbb{F}$ in all clause lists and remove $\bar{\ell}$ from them. Add $(\bar{\ell} \vee \mathbb{F})$ to Φ.

4: Find $\mathbf{a}_\Phi$ such that $\Phi(\mathbf{a}_\Phi) = \mathbb{T}$. If Φ is unsatisfiable, return UNSAT.

Learn one of the literals present in some clause C_j*:*
5: Let $j_0 \leftarrow \mathcal{O}_{\text{FIRST}}(\mathbf{a}_\Phi 0^{rest})$ and $j_1 \leftarrow \mathcal{O}_{\text{FIRST}}(\mathbf{a}_\Phi 1^{rest})$.
6: If $j_0 \neq j_1$, identify $\ell \notin L_\Phi$ such that $\ell \in C_j$ for $j \leq \min\{j_0, j_1\}$.
7: Else if $j_0 = j_1$, identify $\ell \in L_\Phi$ such that $\ell \in C_j$ for some $j \leq j_0$.
8: If Step 6 or Step 7 find a trial $\mathbf{a}$ such that $\mathcal{O}_{\text{FIRST}}(\mathbf{a})$ returns YES, return $\mathbf{a}$.
9: If ℓ_1, ℓ_2 have been identified such that $C_j = (\ell_1 \vee \ell_2)$ and $C_j \notin \Phi$, go to Step 3.

Learn the second literal in a clause C_j *where it is known that* $\ell \in C_j$*:*
10: Otherwise, to identify the second literal in C_j, fix $\ell = \mathbb{F}$ and repeat from Step 5 with trials where $\ell = \mathbb{F}$ is maintained.

Repeat from Step 3 till the oracle return YES.

To help visualize the flow of the algorithm, it is presented as a flow chart in Figure 7.2 with the main procedures related to the phases highlighted.

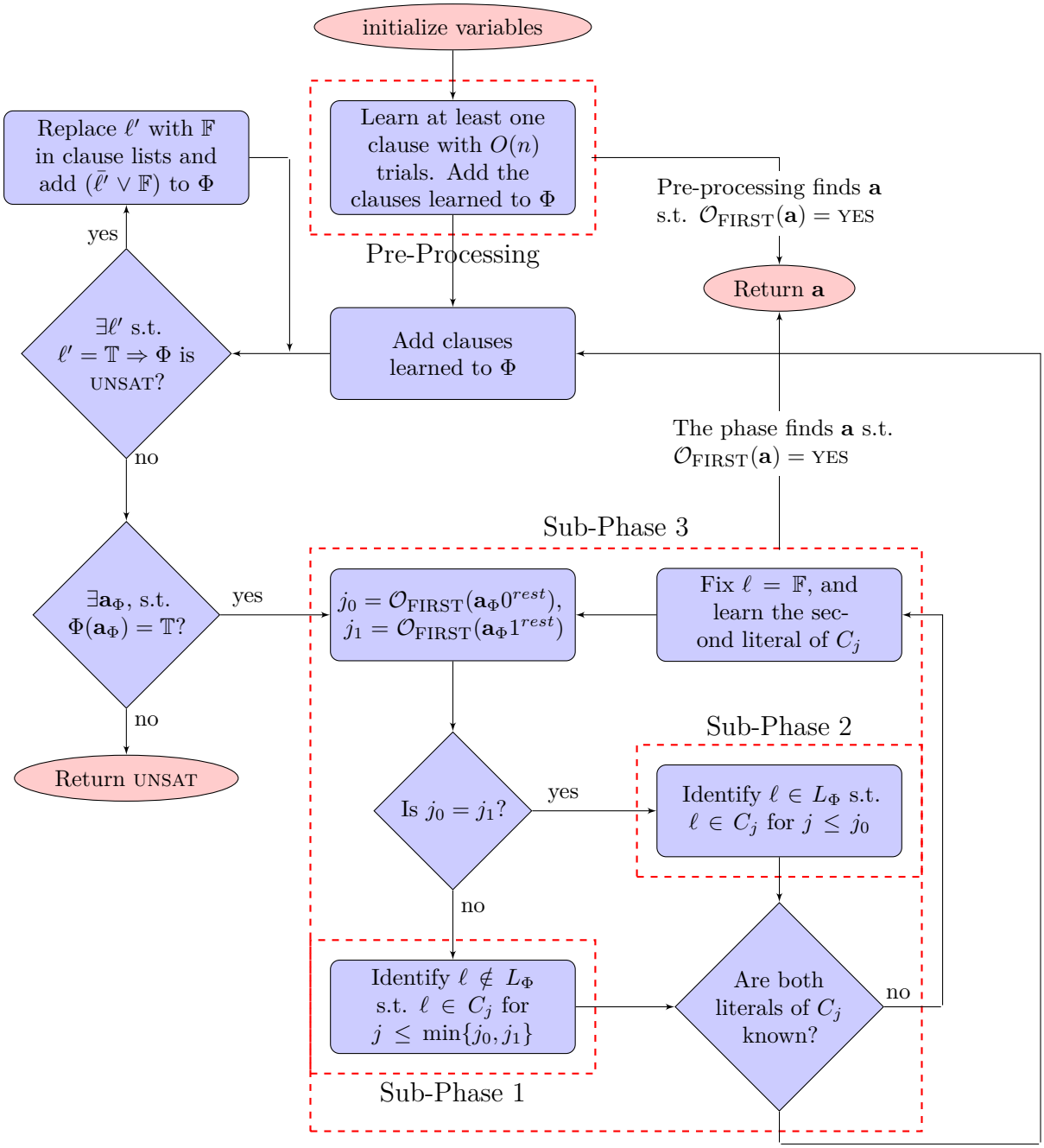


Figure 7.2: Flow chart of Algorithm 7.4

While not appearing so at first glance, the outline in Algorithm 7.4 does line up with the template in Algorithm 7.1. For instance, the execution of Step 6 requires the construction of a series of next trials corresponding to which candidate elimination is performed and Φ is accordingly updated at the end. The same applies to Step 7 and Step 10. Essentially, Step 5 – Step 10 constitutes a single phase which is broken up into *sub-phases*, each with a different notion of convergence: (a) learn a literal not restricted by $\mathbf{a}_\Phi$ i.e., $\ell \notin L_\Phi$ (Step 6); (b) learn a literal that is restricted by $\mathbf{a}_\Phi$ i.e., $\ell \in L_\Phi$ (Step 7); and (c) learn the second literal in a partially known clause (Step 10).

A phase ends when any one of the sub-phases converges by learning a previously unknown literal in a clause. After Φ is accordingly updated, Step 3 updates the clause lists to exclude literals that are forced to be $\mathbb{F}$ by the clauses learned so far in Φ. A new trial is created is by flipping the assignment to some subset of literals while ensuring that these changes do not violate the clauses in Φ at any point.

The proof of Theorem 7.4 is built on the correctness and complexity of Algorithm 7.4. The steps required to execute each sub-phase of the algorithm are described and analyzed in the lemmas below. The goal is to understand how the next trials are created in each sub-phase and how the candidate elimination affects the progress of the algorithm (i.e., $\sum_j |\hat{C}_j|$). Candidate elimination basically infers the presence or absence of a subset of literals from some clause list $\hat{C}_j$. This leads to a series of set operations performed on $\hat{L}_j, \hat{R}_j$ to reflect this information. The relation between the inference drawn and the operations performed is described below. Given a set of literals L, if it is inferred that

- $L \notin C_j \Rightarrow \hat{L}_j \leftarrow \hat{L}_j \setminus L$ and $\hat{R}_j \leftarrow \hat{R}_j \setminus L$.
- $L \in C_j$ as one of the literals in L set C_j to true $\Rightarrow \hat{S}_j \leftarrow \hat{S}_j \cap L$.
- $L \in C_j$ as the literals in L falsified C_j, then both literals of C_j have been set to false. So, $\hat{L}_j \leftarrow \hat{L}_j \cap L$ and $\hat{R}_j \leftarrow \hat{R}_j \cap L$.
- Further, if a literal of C_j is determined as a result of these inferences, $\hat{S}_j$ is updated to point to $\hat{R}_j$ or $\emptyset$ as the case may be.

Note that, the proofs of the lemmas in this section will use the inferences to refer to the execution of the set operations given above. For instance, when it is concluded that a literal $\ell \notin C_j$ during candidate elimination, it implicitly means that the procedure executed $\hat{L}_j \leftarrow \hat{L}_j \setminus \{\ell\}$ and $\hat{R}_j \leftarrow \hat{R}_j \setminus \{\ell\}$. With these operations consuming $O(n)$ time, it will not affect any complexity considerations for the procedure in question. After initializing all the global variables in Algorithm 7.4, Φ is trivially true and every clause is unknown. Step 2 performs some pre-processing before any sub-phase is executed to learn at least one of the clauses in the hidden instance ϕ.

Lemma 7.1 (Step 2 - Pre-processing)**.** *Given a* H–2SAT$_{\text{FIRST}}$ *formula φ, with none of its clauses being known, it is possible to learn the literals of at least one clause of φ using $O(n)$ trials in $O(n)$ time.*

Proof. As per Step 2 of the algorithm, let the two trials $\mathbf{a} = 0^n$, $\mathbf{b} = 1^n$ generate violations j_a and j_b. Note that, $j_a \neq j_b$ as $\mathbf{a}, \mathbf{b}$ indicate that C_{j_a} is the first clause with two positive literals and C_{j_b} is the first clause with two negative literals. Assume without loss of generality that $j_a < j_b$ (the case with $j_b < j_a$ is symmetric). Do the following:

- Using $\mathbf{a}$, for each $i \in \{1, \ldots, n\}$ create the assignment $\mathbf{a}_i$ with $a_i = 1$ and all other variables set to 0. Let the trial $\mathbf{a}_i$ generate a violation j_i.
  - If $j_i = j_a$, then $\{x_i, \overline{x}_i\} \notin C_{j_a}$.
  - If $j_i > j_a$, then $x_i \in C_{j_a}$.

  - If $j_i < j_a$, then $\overline{x}_i \in C_{j_i}$, $\hat{R}_j = \{x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n\}$.
- Next, repeat the same process using $\mathbf{b}$ except that in $\mathbf{b}_i$, $b_i = 0$. Let the trial $\mathbf{b}_i$ generate a violation j_i.
  - If $j_i = j_b$, then $\{x_i, \overline{x}_i\} \notin C_{j_b}$.
  - If $j_i > j_b$, then $\overline{x}_i \in C_{j_b}$.
  - If $j_i < j_b$, then $x_i \in C_{j_i}$, $\hat{R}_j = \{\overline{x}_1, \ldots, \overline{x}_{i-1}, \overline{x}_{i+1}, \ldots, \overline{x}_n\}$.

The complexity is straightforward as the process takes $2n + 2$ trials and modifies one clause list for every trial. Hence, it can be executed in $O(n)$ time.

To argue the correctness of the candidate elimination, first consider the violations (j_a, j_i) generated by the trials $\mathbf{a}, \mathbf{a}_i$ for some $i \in \{1, \ldots, n\}$. When $j_i = j_a$, C_{j_a} is violated by both trials which means C_{j_a} is made of variables whose assignment did not change between the two trials. In other words, the variable x_i whose assignment was flipped between both trials is absent from the clause, i.e., $\{x_i, \overline{x}_i\} \notin C_{j_a}$. When $j_a < j_i$, $\mathbf{a}_i$ satisfied C_{j_a}, i.e., setting $a_i = 1$ (or equivalently, $x_i = \mathbb{T}$) satisfied C_{j_a} leading to the conclusion that $x_i \in C_{j_a}$. When $j_i < j_a$, a similar argument helps in reaching the conclusion that setting $a_i = 1$ (or equivalently, $\overline{x}_i = \mathbb{F}$) causes C_{j_i} to become violated due to which $\overline{x}_i \in C_{j_i}$. Observe that, for all $j < j_a$, C_j is satisfied by both 0^n and 1^n implying that none of these clauses have two positive or two negative literals. In other words, all these clauses contain one positive and one negative literal. Also, C_{j_i} is such that $\mathbf{a}_i$ sets the positive and negative literal in C_{j_i} to $\mathbb{F}$. $a_i = 1$ implies that the positive literal cannot be x_i and with $a_{i'} = 0, \forall i' \neq i$, the negative literal can only be $\overline{x}_i$.

Now, consider the violations (j_b, j_i) generated by the trials $\mathbf{b}, \mathbf{b}_i$ for some i. The correctness follows for the most part from the arguments made above (except that the x_is and $\overline{x}_i$s are swapped), along with the observation that, C_{j_i} has one positive and one negative literal even when $j_a < j_i < j_b$. Suppose that C_{j_i} two positive literals. Then, with only $b_i = 0$ in $\mathbf{b}_i$, C_{j_i} would not have been violated by this assignment, leading to a contradiction.

At the end of the $2n + 2$ trials, for every $i \in \{1, \ldots, n\}$, the first clause containing $\overline{x}_i$ among the clauses C_j with $j \leq j_a$ is known along with the first clause containing x_i among the clauses C_j with $j \leq j_b$. With a simple argument it is shown that the literals in C_1 are guaranteed to be learned at the end of this procedure. Clearly, if $j_a = 1$, this is the case. Suppose $j_a > 1$, then as previously mentioned, with $j_a < j_b$, all clauses C_j with $j \leq j_a$ are of the form $(x_{j_1} \vee \overline{x}_{j_2})$, and this includes the clause C_1. As C_1 will also be the first occurrence for the literals present in it, both literals of C_1 will be learned and this satisfies the claim. □

At the end of the procedure outlined in Lemma 7.1, all the clauses fully determined are added to Φ. The update in Step 5 also tries to ensure that the clause lists only contain literals whose assignment is *unresolved* with respect to Φ. A variable $x_i \in V_\Phi$ is said to be *unresolved* with respect to Φ if there exists $\mathbf{a}, \mathbf{a}' \in \mathsf{sat}(\Phi)$ such that $a_i \neq a'_i$. This definition naturally extends to literals too. Analogously, a variable in $x_i \in V_\Phi$ is said to be *resolved* if $\forall\ \mathbf{a}, \mathbf{a}' \in \mathsf{sat}(\Phi)$, $a_i = a'_i$. Now, the update procedure in $O(nm)$ time removes all the resolved literals from the

clause lists. The rationale for removing resolved literals from clause lists is similar to that for removing ignored edges in the graph properties lex-first algorithm. In this case, it helps minimize the number of trials proposed by ensuring that all subsequent trials do not violate the clauses already learned, i.e., those in Φ. A more detailed argument for the correctness of this procedure is discussed in the proof of Theorem 7.4. A satisfying assignment $\mathbf{a}_\Phi$ can be found in $O(n+m)$ time using one's favourite algorithm for 2SAT. The next step is to expand the number of clauses in Φ by learning more clauses. One way to do this is find a clause involving variables currently not assigned in $\mathbf{a}_\Phi$, i.e., those in V_Φ^c. This is given by the first sub-phase, Step 6, and it is described below.

Lemma 7.2 (Step 6 - First sub-phase)**.** *Let Φ be an induced formula for a* H–2SAT$_{\text{FIRST}}$ *formula φ and let $\mathbf{a}_\Phi$ be a satisfying assignment for Φ. Let the assignments $\mathbf{a}_0 = \mathbf{a}_\Phi 0^{V_\Phi^c}$ and $\mathbf{a}_1 = \mathbf{a}_\Phi 1^{V_\Phi^c}$ produce violations $j_0 \neq j_1$. Then in $O(n)$ time, it is possible to learn a literal ℓ in C_j where $\ell \in L_\Phi^c$ and $j \leq \min\{j_0, j_1\}$.*

Proof. Assume without loss of generality that $j_0 < j_1$ (the $j_1 < j_0$ case is symmetric). Without modifying $\mathbf{a}_\Phi$, do the following:

- Update $\hat{C}_{j_0}$ such that $\{x_k | k \in V_\Phi^c\} \in C_{j_0}$ i.e., $\hat{S}_{j_0} \leftarrow \hat{S}_{j_0} \cap \{x_k | x_k \in L_\Phi^c\}$.
- For each variable $i \in V_\Phi^c$, create $\mathbf{b}_i$ from $\mathbf{a}_0$ by flipping the assignment of b_i from 0 to 1. Let the trial $\mathbf{b}_i$ generate a violation j_i.
  - If $j_i = j_0$, then $\{x_i, \overline{x}_i\} \notin C_{j_0}$ i.e., $\hat{S}_{j_0} \leftarrow \hat{S}_{j_0} \setminus \{x_i, \overline{x}_i\}$.
  - If $|\hat{S}_{j_0}| = 1$, end sub-phase.
  - If $j_i < j_0$, then $\overline{x}_i \in C_{j_i}$. End sub-phase.
  - If $j_i > j_0$, then $x_i \in C_{j_0}$. End sub-phase.

Recall from its definition that $\hat{S}_j$ points to the component of $\hat{C}_j$ with the candidates for the literal of C_j that the algorithm would like to learn. In the case of C_{j_0}, $\hat{S}_{j_0}$ contains the candidates for the literal ℓ. To argue the correctness of this procedure, consider the types of literals present in C_{j_0}. As $j_0 \neq j_1$, and $\mathbf{a}_\Phi$ is common to $\mathbf{a}_0$ and $\mathbf{a}_1$, C_{j_0} is violated by the assignment to the V_Φ^c variables. More specifically, as $0^{V_\Phi^c}$ violates C_{j_0} and $1^{V_\Phi^c}$ satisfies it, C_{j_0} contains at least one positive literal ℓ from L_ϕ^c that is, $\{x_k | k \in V_\Phi^c\}$. This justifies updating $\hat{S}_{j_0} \leftarrow \hat{S}_{j_0} \cap \{x_k | k \in V_\Phi^c\}$. The correctness of the candidate elimination follows from the same arguments as the proof of Lemma 7.1 except that the arguments only involve the literals in L_ϕ^c as only their assignments change.

To show that at least one literal will be learned, notice that if there is some $i \in V_\phi^c$ such that $j_i \neq j_0$, then a literal from C_j for $j = \min\{j_i, j_0\} \leq j_0$ will be learned satisfying the claim. The only case left to consider is when for all $i \in V_\Phi^c$, $j_i = j_0$. Then, after the first step of the process described above, $\hat{S}_{j_0} \subseteq \{x_k | k \in V_\Phi^c\}$ and $|\hat{S}_{j_0}| \leq |V_\Phi^c|$. For each i such that $j_i = j_0$, $\{x_i\} \notin C_{j_0}$ implies that x_i can be removed from $\hat{S}_{j_0}$ when the latter is updated as $\hat{S}_{j_0} \leftarrow \hat{S}_{j_0} \setminus \{x_i\}$. Hence, $|\hat{S}_{j_0}|$ decreases by at most 1 after the update. Then with at most $|V_\phi^c|$ trials such that $j_i = j_0$, the size of $|\hat{S}_{j_0}|$ will decrease until there is only one choice for ℓ. As $\ell \in C_{j_0}$ is determined, this

satisfies the claim.

The complexity is straightforward as the process takes at most $n+2$ trials, modifies one clause list every time and so can be executed in $O(n)$ time. □

Any new clauses learned are added to Φ to update it along with a corresponding $\mathbf{a}_\Phi$. The first sub-phase is not executed if $j_0 = j_1$ and this case is handled by the second sub-phase Step 7. In this case, one would like to learn a literal which is restricted by Φ.

Lemma 7.3 (Step 7 - Second sub-phase)**.** *Let Φ be an induced formula for a* H–2SAT$_{\text{FIRST}}$ *formula φ and let $\mathbf{a}_\Phi$ be a satisfying assignment for Φ. Let the assignments $\mathbf{a}_0 = \mathbf{a}_\Phi 0^{V_\Phi^c}$ and $\mathbf{a}_1 = \mathbf{a}_\Phi 1^{V_\Phi^c}$ produce violations $j_0 = j_1$ respectively. Then it is possible in* poly(n) *time to learn a literal ℓ in C_j for $j \leq j_0$ such that $\ell \in L_\Phi$.*

Proof. Let $\mathbf{a} = \mathbf{a}_1$ and $j = j_1$. For two assignments $\mathbf{b}$, $\mathbf{b}'$, recall that $A(\mathbf{b}, \mathbf{b}')$ is the set of literals that are set to $\mathbb{F}$ in both assignments. In addition, $B(\mathbf{b}, \mathbf{b}')$ is the set of literals that change their values from being $\mathbb{F}$ in $\mathbf{b}$ to being $\mathbb{T}$ in $\mathbf{b}'$ and $\bar{B}(\mathbf{b}, \mathbf{b}')$ is the set of literals that changes from being $\mathbb{T}$ in $\mathbf{b}$ to $\mathbb{F}$ in $\mathbf{b}'$. Also, for C_{j_0}, $\hat{S}_{j_0}$ will contain the candidates for the literal from L_Φ that the algorithm tries to learn. Without modifying the assignment to V_ϕ^c in $\mathbf{a} = \mathbf{a}_\Phi 1^{V_\Phi^c}$, do the following:

- Update $\hat{C}_j$ as $A(\mathbf{a}_0, \mathbf{a}_1) \in C_j$ i.e., $\hat{S}_j \leftarrow \hat{S}_j \cap A(\mathbf{a}_0, \mathbf{a}_1)$. If $|\hat{S}_j| = 1$, end sub-phase.
- Otherwise, to create the next assignment $\mathbf{b}$, for each literal $\ell' \in \hat{S}_j \setminus \{\mathbb{F}\}$ repeat:
  - Set $\ell' = \mathbb{T}$ and find a $\mathbf{b}_\Phi$ with $\ell' = \mathbb{T}$ such that $\Phi(\mathbf{b}_\Phi) = \mathbb{T}$. Set $\mathbf{b} = \mathbf{b}_\Phi 1^{V_\Phi^c}$.
  - Let $\mathcal{B} = B(\mathbf{a}, \mathbf{b})$. If $\hat{S}_j \setminus \mathcal{B} = \emptyset$, repeat with a different literal.
- If $\hat{S}_j \setminus \mathcal{B} = \emptyset$, pick any literal from $\ell \in \hat{S}_j$ and conclude $\ell \in C_j$. End sub-phase.
- Otherwise, let the trial $\mathbf{b}$ generate the violation j'.
  - If $j = j'$, then $\mathcal{B} \notin C_j$ i.e., $\hat{S}_j \leftarrow \hat{S}_j \setminus \mathcal{B}$. Update $\mathbf{a} \leftarrow \mathbf{b}$.
  - If $j < j'$, then $\mathcal{B} \in C_j$ i.e., $\hat{S}_j \leftarrow \hat{S}_j \cap \mathcal{B}$.
  - If $j > j'$, then $\overline{\mathcal{B}} \in C_{j'}$ i.e., $\hat{S}_{j'} \leftarrow \hat{S}_{j'} \cap \overline{\mathcal{B}}$. Update $\mathbf{a} \leftarrow \mathbf{b}$ and $j \leftarrow j'$.
- If $|\hat{S}_j| = 1$, end sub-phase.
- Otherwise repeat by creating a new $\mathbf{b}$.

Note that, as $j_0 = j_1$, and $\mathbf{a}_\Phi$ is common to $\mathbf{a}_0$, $\mathbf{a}_1$, the clause C_{j_0} is made up of two literals from L_Φ such that $\mathbf{a}_\Phi$ sets these literals to $\mathbb{F}$. This is exactly the set of literals in $\mathcal{A}$. Hence, it is justified to set $\hat{S}_{j_0} \leftarrow \hat{S}_{j_0} \cap \mathcal{A}$. For all $j \leq j_0$, C_j is satisfied both by $\mathbf{a}_0$ and $\mathbf{a}_1$ and can fall into one of two categories. Consider splitting an assignment into two parts: one part acting on V_Φ and the other on V_Φ^c. Either the C_j is such that it is satisfied by $\mathbf{a}_\Phi$ in which case it contains at least one literal from L_Φ; or, C_j is satisfied by the assignments to V_Φ^c. In the latter case, as both $0^{V_\Phi^c}$ and $1^{V_\Phi^c}$ satisfied C_j, it contains at least one positive and one negative literal from L_Φ^c. Clearly, as the process described above does not change the assignments on V_Φ^c, these clauses will not show up as violations during the execution of this procedure.

The first part of showing the correctness of this process is to analyze how the next trial $\mathbf{b}$ is

created. Observe that it is always possible to create a suitable $\mathbf{b}_\Phi$ with $\ell' = \mathbb{T}$. Clearly $|\hat{S}_j| > 1$ and so there is definitely some $\ell' \in \hat{S}_j \setminus \{\mathbb{F}\}$. Suppose by way of contradiction that it is not possible to find a $\mathbf{b}_\Phi$ by setting $\ell' = \mathbb{T}$. Then, there is a literal $\ell' \in \hat{S}_j \setminus \{\mathbb{F}\}$ such that $\ell' = \mathbb{T} \Rightarrow \Phi$ is UNSAT. In other words, for all $\mathbf{a}' \in \mathsf{sat}(\Phi)$, $\ell' = \mathbb{F}$. This, by definition means that ℓ' is a literal resolved to be $\mathbb{F}$. However, the clause lists are updated in Step 5 so that $\hat{S}_j$ contains only $\mathbb{F}$ or unresolved literals. Any subsequent updates to $\hat{S}_j$ during the process results only in removing literals and not adding any literals (resolved or otherwise) to it. Hence $\ell' \notin \hat{S}_j \setminus \{\mathbb{F}\}$ which leads to a contradiction.

If $\hat{S}_j \setminus \mathcal{B} = \emptyset$, then $\hat{S}_j \cap \mathcal{B} = \hat{S}_j$. If this holds for all literals in $\hat{S}_j \setminus \{\mathbb{F}\}$, clearly $\mathbb{F} \notin \hat{S}_j$ and $\hat{S}_j \setminus \{\mathbb{F}\} = \hat{S}_j$ as the fixed literal $\mathbb{F}$ cannot, by definition, be in $\mathcal{B}$. Additionally, it means that all assignments in $\mathsf{sat}(\Phi)$ require that the literals in $\hat{S}_j$ either all be $\mathbb{T}$ or all be $\mathbb{F}$. In other words, the assignment to all these variables has to be the same leading to the conclusion that these literals are logically equivalent in the formula, i.e., for all $\ell, \ell' \in \hat{S}_j, \ell \Leftrightarrow \ell'$. Replacing a literal in a clause with an equivalent literal does not change the set of satisfying assignments of both formulas. This allows for any literal from $\hat{S}_j$ to be picked as the representative present in the clause. The literal in C_j is now learned up to an equivalence and so, the sub-phase can be ended.

The candidate elimination used here is similar to those used in previous procedures with one difference. As the trial changes from $\mathbf{a}$ to $\mathbf{b}$, a set of literals may change their assignment instead of just one literal. However, $\mathcal{B}$ and $\mathcal{B}'$ capture the set of literals that change their assignments. Using these sets, the correctness of candidate elimination will directly follow from the proof of Lemma 7.1. Finally, notice that sub-phase ends only when $|\hat{S}_j| = 1$ which is when it has been determined that for $\hat{S}_j = \{\ell\}$, $\ell \in C_j$. At the beginning of the procedure $j = j_0$ and during candidate elimination, j is always set to the $\min\{j, j'\} \leq j \leq j_0$. Hence, the clause whose literal is revealed is some C_j where $j \leq j_0$. This satisfies the lemma's claim.

To calculate the complexity, observe that the procedure tries to create the next trial by flipping the literals of $\hat{S}_j \setminus \{\mathbb{F}\}$ where $|\hat{S}_j \setminus \{\mathbb{F}\}| \leq n$ as there are at most n variables in V_Φ. Whenever a trial $\mathbf{b}$ is created, it happens by finding a $\mathbf{b}_\Phi$ where some literal ℓ' is set to $\mathbb{T}$. This can be done using Unit Propagation [DLL62]. For example, if $\Phi = (x_1 \vee x_2) \wedge (\overline{x}_2 \vee x_3) \wedge (x_3 \vee x_4)$ and $\mathbf{a}_\Phi = 1001$, then setting $\overline{x}_1 = \mathbb{T}$ would propagate by forcing $x_2 = \mathbb{T}$ to satisfy the first clause which in turn forces $x_3 = \mathbb{T}$ to satisfy the second clause. A slight modification to any linear time 2SAT algorithm to maintain $\ell = \mathbb{T}$ suffices to find $\mathbf{b}_\Phi$ in $O(m + n)$ time if it exists. All other operations are just set operations and consume $O(1)$ time. It may happen that $\hat{S}_j \setminus \mathcal{B} = \emptyset$ for $\mathbf{b}$ and so it may require at most $|\hat{S}_j| = O(n)$ tries to find a suitable $\mathbf{b}$ for candidate elimination. In total, it may take $O(mn + n^2)$ time to find a suitable $\mathbf{b}$.

To bound the total number of trials needed, consider the size of $\mathcal{B}$ when candidate elimination occurs. Clearly, $\hat{S}_j \setminus \mathcal{B} \neq \emptyset \Rightarrow \hat{S}_j \cap \mathcal{B} \subsetneq \hat{S}_j \Rightarrow |\hat{S}_j \cap \mathcal{B}| < |\hat{S}_j| \leq n$. So, with $O(n)$ trials where all the violations are at least j it is possible to learn a literal in C_j. However, it is possible that some violation $j' < j$ is observed in which $|\hat{S}_{j'} \cap \overline{\mathcal{B}}| \leq |\hat{S}_{j'}|$ and if they are equal, no candidate is eliminated. As there are m clauses, after at most $O(m)$ jumps where j decreases, the algorithm

reaches the $j' = \min\{j | C_j \text{ has a literal from } L_\Phi\}$. For this j' all violations will be at least j' and a literal in $C_{j'}$ can be learned. Hence, at most$O(nm)$ trials are needed in the worst case. The complexity of the procedure is just $O(mn \cdot (mn + n^2)) = \text{poly}(n)$. □

While this completes the ways to learn a single literal in a clause, it is usually assumed that it is the first literal in the clause that one is trying to identify. With a little bit of extra processing, the same processes in Lemmas 7.2 and 7.3 can be recycled to also learn the second literal in a clause. This also forms the third and last sub-phase, Step 10. For this step, any modifications to C_j imply modifications to $\hat{S}_j = \hat{R}_j$.

Lemma 7.4 (Step 10 - Third sub-phase)**.** *Let Φ be an induced formula of a* H–2SAT$_{\text{FIRST}}$ *formula φ with $\mathbf{a}_\Phi$ satisfying Φ. If one literal, ℓ, of C_j is known, then it is possible in* poly(n) *time to learn the second literal of C_j or identify a clause $C_{j'}$ with $j' < j$ where at least one literal of $C_{j'}$ is not known.*

Proof. The idea is to force the first literal in C_j to be set to $\mathbb{F}$ so that the only way to satisfy C_j is if the second literal is set to $\mathbb{T}$. This will allow it to be identified using the pattern of violations. The process is different depending on if the first literal is in L_Φ or L_Φ^c. Let ℓ be the first literal of C_j that is already known and assume that ℓ is already set to $\mathbb{F}$ and the violation with the current trial $\mathbf{a}$ is j. Assume that no violation $j' < j$ is returned during this process as otherwise, the process can be aborted and the algorithm tries to learn the literals of $C_{j'}$.

- Mark ℓ so that its assignment will not be changed within this procedure.
- To test if the second literal of C_j is in L_Φ^c, run the procedure from Lemma 7.2 on the literals in L_Φ^c.
- Otherwise, to test if the literal is in L_Φ use the procedure in Lemma 7.3 with the following modification when $\ell \in L_\Phi$: For all $\ell' \in \hat{S}_j \cap L_\Phi$, if $\ell' = \mathbb{T}$ forces $\ell = \mathbb{T}$ to satisfy Φ, replace ℓ' with $\mathbb{F}$ in $\hat{S}_j$.

The complexity is pretty straightforward. The procedure above takes $O(mn)$ time for the first step and poly(n) time for the second and so can run in poly(n) time in total.

To argue the correctness of the process, note that if no $j' < j$ was returned, then all violations were either of the form $j' = j$ or $j' > j$. Using the proof of Lemma 7.2 in the first step, with the former type of violation, some literals are eliminated as possibilities for C_j and the latter violation actually reveals the second literal. For the second step, the argument gets more involved. Observe that a new trial satisfying Φ will be created from $\mathbf{a}$ by setting some literal in $\hat{S}_j \cap L_\Phi$ to $\mathbb{T}$. During this process, using the proof of Lemma 7.3, it is known that the value of more than one literal may change. Consider the scenario where the value of ℓ is forced to change (i.e, from $\mathbb{F}$ to $\mathbb{T}$). Then, there is a literal $\ell' \in \hat{S}_j \cap L_\Phi$ such that setting $\ell' = \mathbb{T}$ forces $\ell = \mathbb{T}$ in every assignment $\mathbf{b}_\Phi$ that satisfies Φ. Suppose that ℓ' was the second literal in the clause C_j. Clearly, for all $\mathbf{b} \in \mathsf{sat}(\Phi)$ if $\mathbf{b}$ sets ℓ' to $\mathbb{T}$, then it also sets ℓ to $\mathbb{T}$. This will satisfy C_j as both literals are $\mathbb{T}$. However, even if ℓ' is $\mathbb{F}$, the only way to satisfy the clause is if ℓ is set to $\mathbb{T}$. Essentially, $\mathsf{sat}(C_j)$ reduces to the satisfiability of the unary clause (ℓ). This is represented as

$(\ell \vee \mathbb{F})$ justifying the replacement of ℓ' with the symbol $\mathbb{F}$. Now, the end of the second step will reveal the second literal of C_j. □

Finally, the proof of Theorem 7.4 is given below combining the correctness and running times of the pre-processing and sub-phases discussed previously.

Proof of Theorem 7.4. Algorithm 7.4 can be viewed, after the pre-processing step, as a series of updates and phases. Further, each phase is broken up into three sub-phases being executed where one of them ends the phase by learning previously unknown literal in some clause. Following this, appropriate updates are applied to the induced formula Φ and the clause lists.

To argue the correctness of the algorithm, it is shown that the algorithm outputs UNSAT only when φ is unsatisfiable. If not, it is possible in poly(n) time, using Lemmas 7.1, 7.2, 7.3, 7.4, to learn the clauses in φ up to some equivalence of literals and use any 2SAT algorithm to find a satisfying assignment for φ. To this aim, it is shown that every assignment $\mathbf{a}$ that satisfies φ will also satisfy Φ. In other words, throughout the run of the algorithm, the following invariant is maintained $\mathsf{sat}(\Phi) \supseteq \mathsf{sat}(\varphi)$ and Φ is indeed an induced formula for φ. This clearly holds after initialization because when $\Phi = (\mathbb{T})$ then $\mathsf{sat}(\Phi) = \{0,1\}^n \supseteq \mathsf{sat}(\varphi)$. Let this hypothesis hold for some Φ before it enters the update where the guess is denoted $\Phi^{(i)}$ and via induction, it is shown that Φ after the update, denoted $\Phi^{(f)}$ will also satisfy the hypothesis. The update can be broken up into two parts: one part where the clauses learned are added to Φ and the second where resolved literals are identified and replaced with $\mathbb{F}$ in the clause lists.

Take the first part where $\hat{L}_j = \{\ell_1\}, \hat{R}_j = \{\ell_2\}$, $\ell_1, \ell_2 \neq \mathbb{F}$ has been identified for some clause j. If the literals were learned by candidate elimination, then the correctness of this procedure from Lemma 7.1 and Lemma 7.3 indicates that C_j in φ is actually $(\ell_1 \vee \ell_2)$. As C_j is sub-formula of φ, $\mathsf{sat}(C_j) \supseteq \mathsf{sat}(\varphi)$.

When candidate elimination is not used, it is possible that $C_j = (\ell_1' \vee \ell_2)$ in φ and $\ell_1' \neq \ell_1 \neq \mathbb{F}$. This could happen in the second sub-phase when one of the literals from $\hat{S}_j$ is picked to represent the literal in C_j. The proof of Lemma 7.3 guarantees that in this case, the literal picked is logically equivalent to the literal in C_j. This means, for all $\mathbf{a} \in \mathsf{sat}(\Phi^{(i)}) \supseteq \mathsf{sat}\,\varphi$, ℓ, ℓ_1 are either both $\mathbb{F}$ or both $\mathbb{T}$. This implies that $\mathsf{sat}((\ell_1 \vee \ell_2)) = \mathsf{sat}(C_j) \supseteq \mathsf{sat}(\varphi)$.

Next, consider the case when $\ell_2 = \mathbb{F}$ but $C_j = (\ell_1 \vee \ell_2')$ in φ where $\ell_2' \neq \mathbb{F}$. This occurs in the third sub-phase while trying to learn a literal from $L_{\Phi^{(i)}}$. From the proof of Lemma 7.4, observe that this happens only if the only way to satisfy C_j is to set ℓ_1 to $\mathbb{T}$. Then $\mathsf{sat}(C_j) = \mathsf{sat}((\ell_1 \vee \mathbb{F}))$ implying that $\mathsf{sat}((\ell_1 \vee \mathbb{F})) \supseteq \mathsf{sat}(\varphi)$. In all these cases, coupled with the fact that $\mathsf{sat}(\Phi^{(i)}) \supseteq \mathsf{sat}(\varphi)$, when $\Phi^{(f)} = \Phi^{(i)} \bigwedge (\ell_1 \vee \ell_2)$, $\mathsf{sat}(\Phi^{(f)}) \supseteq \mathsf{sat}(\varphi)$.

Finally, for the second part, while replacing a resolved literal ℓ, notice that $\Phi^{(i)}$ is such that for all $\mathbf{a} \in \mathsf{sat}(\Phi^{(i)})$, $\bar{\ell}$ is set to $\mathbb{T}$. Adding the clause $(\bar{\ell} \vee \mathbb{F})$ does not change the set of satisfying assignments for it. So, $\mathsf{sat}(\Phi^{(f)}) = \mathsf{sat}(\Phi^{(i)}) \supseteq \mathsf{sat}(\varphi)$.

To argue the complexity, the pre-processing in Step 2 takes $O(mn)$ time from Lemma 7.1 to learn at least one clause. This is followed by Step 3 that takes $O(mn)$ time and $O(m+n)$ time to learn a new $\mathbf{a}_\Phi$ for the updated Φ. A literal is learned in a clause because a phase achieves convergence. For this the first literal in some clause is learned after the execution of the first two sub-phases i.e. Step 6 and Step 7. The second literal in a clause can be learned after the third sub-phase i.e. Step 10. Hence, from Lemmas 7.2, 7.3 and 7.4, a clause can be learned in at most poly(n) time. This upper bounds the total time to learn a clause to poly(n) time. This is repeated for at most m clauses, the total running time is still poly(n) as $m = O(n^2)$. □

This concludes the results on the lex-first trial and error oracle.

References

[ABG+16] Itai Arad, Adam Bouland, Daniel Grier, Miklos Santha, Aarthi Sundaram, and Shengyu Zhang. On the complexity of probabilistic trials for hidden satisfiability problems. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, pages 12:1–12:14, 2016.

[AKLT87] Ian Affleck, Tom Kennedy, Elliott H Lieb, and Hal Tasaki. Rigorous results on valence-bond ground states in antiferromagnets. *Physical Review Letters*, 59(7):799–802, 1987.

[AL98] Daniel S. Abrams and Seth Lloyd. Nonlinear quantum mechanics implies polynomial-time solution for NP-complete and #P problems. *Physical Review Letters*, 81:3992–3995, 1998.

[APT79] Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979. Erratum: Information Processing Letters 14(4): 195 (1982).

[AS08] Noga Alon and Asaf Shapira. Every monotone graph property is testable. *SIAM Journal on Computing*, 38(2):505–522, 2008.

[ASSZ16] Itai Arad, Miklos Santha, Aarthi Sundaram, and Shengyu Zhang. Linear time algorithm for quantum 2SAT. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 15:1–15:14, 2016.

[ASSZ17] Itai Arad, Miklos Santha, Aarthi Sundaram, and Shengyu Zhang. Almost frustration-free Quantum 2SAT. *Paper in preparation.*, 2017.

[BCD+13] Xiaohui Bei, Ning Chen, Liyu Dou, Xiangru Huang, and Ruixin Qiang. Trial and error in influential social networks. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*, pages 1016–1024, 2013.

[BCZ13] Xiaohui Bei, Ning Chen, and Shengyu Zhang. On the complexity of trial and error. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 31–40, 2013.

[BCZ15] Xiaohui Bei, Ning Chen, and Shengyu Zhang. Solving linear programming with constraints unknown. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 129–142, 2015.

[BDW02] Harry Buhrman and Ronald De Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002.

[Bra11] Sergey Bravyi. Efficient algorithm for a quantum analogue of 2-SAT. *Contemporary Mathematics*, 536:33–48, 2011.

[BS16] Anne Broadbent and Christian Schaffner. Quantum cryptography beyond quantum key distribution. *Designs, Codes and Cryptography*, 78(1):351–382, 2016.

[BT97] Béla Bollobás and Andrew Thomason. *Hereditary and Monotone Properties of Graphs*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[BV05] Sergey Bravyi and Mikhail Vyalyi. Commutative version of the local Hamiltonian problem and common eigenspace problem. *Quantum Information & Computation*, 5(3):187–215, 2005.

[Cal11a] Steven Callander. Searching and learning by trial and error. *American Economic Review*, 101(6):2277–2308, 2011.

[Cal11b] Steven Callander. Searching for good policies. *American Political Science Review*, 105(04):643–662, 2011.

[CCD+11] Jianxin Chen, Xie Chen, Runyao Duan, Zhengfeng Ji, and Bei Zeng. No-go theorem for one-way quantum computing on naturally occurring two-level systems. *Physical Review A*, 83(5):050301, 2011.

[Che09] Hubie Chen. A rendezvous of logic, complexity, and algebra. *ACM Computing Surveys*, 42(1):2:1–2:32, 2009.

[CHTW04] Richard Cleve, Peter Høyer, Benjamin Toner, and John Watrous. Consequences and limits of nonlocal strategies. In *19th Annual IEEE Conference on Computational Complexity (CCC 2004), 21-24 June 2004, Amherst, MA, USA*, pages 236–249, 2004.

[CJ15] Fiona R. Cross and Robert R. Jackson. Solving a novel confinement problem by spartaeine salticids that are predisposed to solve problems in the context of predation. *Animal Cognition*, 18(2):509–515, 2015.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[CM16a] Steven Callander and Niko Matouschek. The risk of failure: Trial and error learning and long-run performance. Technical report, Mimeo, 2016.

[CM16b] Toby S. Cubitt and Ashley Montanaro. Complexity classification of local Hamiltonian problems. *SIAM Journal on Computing*, 45(2):268–316, 2016.

[CMM09] Moses Charikar, Konstantin Makarychev, and Yury Makarychev. Near-optimal algorithms for maximum constraint satisfaction problems. *ACM Transactions on Algorithms*, 5(3):32:1–32:14, July 2009.

[Coh93] Henri Cohen. *A Course in Computational Algebraic Number Theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1993.

[Coo71] Stephen A. Cook. The complexity of theorem proving procedures. In *Proceedings of the Third Annual ACM Symposium*, pages 151–158, New York, 1971. ACM.

[dBG16] Neil de Beaudrap and Sevag Gharibian. A linear time algorithm for quantum 2-SAT. In *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, pages 27:1–27:21, 2016.

[Dic17] Merriam-Webster Online Dictionary. "Trial and Error". *https://www.merriam-webster.com/dictionary/trial%20and%20error*, 2017. Retrieved on 2017-04-10.

[Die12] Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

[DLL62] Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, July 1962.

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, July 1960.

[ECP10] Jens Eisert, Marcus Cramer, and Martin B. Plenio. Area laws for the entanglement entropy - a review. *Reviews of Modern Physics*, 82(277), 2010.

[Edm67] Jack Edmonds. Optimum Branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240, 1967.

[EIS76] Shimon Even, Alon Itai, and Adi Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4):691–703, 1976.

[FF56] Lester R. Ford and Delbert R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8(3):399–404, 1956.

[Gha10] Sevag Gharibian. Strong NP-hardness of the quantum separability problem. *Quantum Information & Computation*, 10(3):343–360, March 2010.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[GJS76] Michael R Garey, David S. Johnson, and Larry Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976.

[GKNR09] Oded Goldreich, Michael Krivelevich, Ilan Newman, and Eyal Rozenberg. Hierarchy theorems for property testing. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 504–519. Springer, 2009.

[GN13] David Gosset and Daniel Nagaj. Quantum 3-SAT is QMA1-complete. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 756–765, 2013.

[Gol11] Oded Goldreich. Introduction to testing graph properties. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation - In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman*, pages 470–506. Springer, 2011.

[Got97] Daniel Gottesman. *Stabilizer codes and quantum error correction*. PhD thesis, California Institute of Technology, 1997.

[GS62] David Gale and Lloyd S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69(1):9–15, 1962.

[Har11] Tim Harford. *Adapt: Why Success Always Starts with Failure*. Little, Brown, 2011.

[Her06] Israel N Herstein. *Topics in algebra*. John Wiley & Sons, 2006.

[HG96] Cecilia M. Heyes and Bennett G. Galef, editors. *Social Learning In Animals: The Roots of Culture*. Elsevier Science, 1996.

[HHH96] Michał Horodecki, Paweł Horodecki, and Ryszard Horodecki. Separability of mixed states: necessary and sufficient conditions. *Physics Letters A*, 223(1):1–8, 1996.

[HKKK88] Pavol Hell, David G. Kirkpatrick, Jan Kratochvíl, and Igor Kríz. On restricted two-factors. *SIAM Journal on Discrete Mathematics*, 1(4):472–484, 1988.

[HvdHL16] David Harvey, Joris van der Hoeven, and GrÃľgoire Lecerf. Even faster integer multiplication. *Journal of Complexity*, 36:1–30, 2016.

[IKQ+14] Gábor Ivanyos, Raghav Kulkarni, Youming Qiao, Miklos Santha, and Aarthi Sundaram. On the complexity of trial and error for constraint satisfaction problems. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, Lecture Notes in Computer Science, pages 663–675. Springer, 2014.

[Imm12] Neil Immerman. *Descriptive complexity*. Springer Science & Business Media, 2012.

[JWZ11] Zhengfeng Ji, Zhaohui Wei, and Bei Zeng. Complete characterization of the ground-space structure of two-body frustration-free Hamiltonians for qubits. *Physical Review A*, 84:042338, 2011.

[Kar72] Richard Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

[Kar01] Marek Karpinski. Approximating bounded degree instances of NP-hard problems. In *Fundamentals of Computation Theory, 13th International Symposium, FCT 2001, Riga, Latvia, August 22-24, 2001, Proceedings*, pages 24–34, 2001.

[Kit03] Alexei Kitaev. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303(1):2–30, 2003.

[KKR06] Julia Kempe, Alexei Kitaev, and Oded Regev. The complexity of the local Hamiltonian problem. *SIAM Journal on Computing*, 35(5):1070–1097, May 2006.

[KLN91] Jan Kratochvíl, Anna Lubiw, and Jaroslav Nešetřil. Noncrossing subgraphs in topological layouts. *SIAM Journal on Discrete Mathematics*, 4(2):223–244, March 1991.

[Kot12] Joanne Kotz. Phenotypic screening, take two. *SciBX*, 5(15):380, 2012.

[KR03] Julia Kempe and Oded Regev. 3-local Hamiltonian is QMA-complete. *Quantum Information & Computation*, 3(3):258–264, 2003.

[Kro67] Melven R. Krom. The decision problem for a class of first-order formulas in which all disjunctions are binary. *Mathematical Logic Quarterly*, 13(1-2):15–20, 1967.

[KSV02] Alexei Kitaev, Alexander Shen, and Mikhail N. Vyalyi. *Classical and Quantum Computation*. American Mathematical Society, Boston, MA, USA, 2002.

[Lev73] Leonid A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266, 1973.

[LGS88] László Lovász, Martin Grötschel, and Alexander Schrijver. Geometric algorithms and combinatorial optimization. *Berlin: Springer-Verlag*, 33:34, 1988.

[LMSS10] Christopher R. Laumann, Roderich Moessner, Antonello Scarddichio, and S. L. Sondhi. Random quantum satisfiability. *Quantum Information & Computation*, 10(1):1–15, 2010.

[LP84] Andrea S. LaPaugh and Christos H. Papadimitriou. The even-path problem for graphs and digraphs. *Networks*, 14(4):507–513, 1984.

[LS09] Troy Lee and Adi Shraibman. *Lower Bounds in Communication Complexity*. Foundations and trends in theoretical computer science. Now Publishers, 2009.

[LUM+12] Jonathan A. Lee, Mark T. Uhlik, Christopher M. Moxham, Dirk Tomandl, and Daniel J. Sall. Modern phenotypic drug discovery is a viable, neoclassic pharma strategy. *Journal of medicinal chemistry*, 55(10):4527–4538, 2012.

[Mit82] Tom M. Mitchell. Generalization as search. *Artificial intelligence*, 18(2):203–226, 1982.

[MM14] Frances K. McSweeney and Eric S. Murphy. *The Wiley Blackwell handbook of operant and classical conditioning*. John Wiley & Sons, 2014.

[Mon16] Ashley Montanaro. Quantum algorithms: an overview. *npj Quantum Mechanics*, 2, 2016.

[NC00] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[Nel08] Richard R. Nelson. Bounded rationality, cognitive maps, and trial and error learning. *Journal of Economic Behavior & Organization*, 67(1):78 – 89, 2008.

[NH81] Simeon C. Ntafos and S. Louis Hakimi. On the complexity of some coding problems. *IEEE Transactions on Information Theory*, 27(6):794–796, 1981.

[NM07] Daniel Nagaj and Shay Mozes. New construction for a QMA complete three-local Hamiltonian. *Journal of Mathematical Physics*, 48(7):072104–072104, July 2007.

[Oxl06] James G. Oxley. *Matroid Theory (Oxford Graduate Texts in Mathematics)*. Oxford University Press, Inc., New York, NY, USA, 2006.

[Per96] Asher Peres. Separability criterion for density matrices. *Physical Review Letters*, 77(8):1413–1415, 1996.

[PY12] Bary S.R. Pradelski and H. Peyton Young. Learning efficient nash equilibria in distributed systems. *Games and Economic Behavior*, 75(2):882 – 897, 2012.

[Sac07] Subir Sachdev. *Quantum phase transitions*. Wiley Online Library, 2007.

[SC03] John ER Staddon and Daniel T Cerutti. Operant conditioning. *Annual Review of Psychology*, 54(1):115–144, 2003.

[Sch78] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*, pages 216–226, 1978.

[SCT00] Karyl B. Swartz, Shaofu Chen, and Herbert S. Terrace. Serial learning by rhesus monkeys: Ii. learning four-item lists by trial and error. *Journal of Experimental Psychology: Animal Behavior Processes*, 26(3):274, 2000.

[Sun17] Aarthi Sundaram. Trial and error for graph properties with the lex-first oracle. *Paper in preparation.*, 2017.

[Tho98] Edward Lee Thorndike. Animal intelligence: An experimental study of the associative processes in animals. *Psychological Monographs: General and Applied*, 2(4):i–109, 1898.

[Tho83] William H. Thorpe. The origins and rise of ethology: The science of the natural behaviour of animals. *Journal of the History of the Behavioral Sciences*, 19(1):95–97, 1983.

[Tut54] William T. Tutte. A short proof of the factor theorem for finite graphs. *Canadian Journal of Mathematics*, 6:347–352, 1954.

[Tut59] William T. Tutte. Matroids and graphs. *Transactions of the American Mathematical Society*, 90(3):527–552, 1959.

[Val84] Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[VLRK03] Guifre Vidal, José I. Latorre, Enrique Rico, and Alexei Kitaev. Entanglement in quantum critical phenomena. *Physical Review Letters*, 90:227902, Jun 2003.

[WB03] Pawel Wocjan and Thomas Beth. The 2-local Hamiltonian problem encompasses NP. *eprint arXiv:quant-ph/0301087*, January 2003.

[You09] H. Peyton Young. Learning by trial and error. *Games and Economic Behavior*, 65(2):626 – 643, 2009.

[ZTM13] Wei Zheng, Natasha Thorne, and John C McKew. Phenotypic screens as a renewed approach for drug discovery. *Drug discovery today*, 18(21):1067–1073, 2013.