Quantum Fine-Grained Complexity

ILLC Dissertation Series DS-2023-01

Institute for Logic, Language and Computation

For further information about ILLC publications, please contact

Institute for Logic, Language and Computation
Universiteit van Amsterdam
Science Park 107
1098 XG Amsterdam
phone: +31-20-525 6051
e-mail: illc@uva.nl
homepage: http://www.illc.uva.nl/

This research was supported by the Robert Bosch Stiftung,
and additionally supported by NWO Gravitation grants NETWORKS
and QSC, and EU grant QuantAlgo, and QuSoft.

Cover design by Subhasree Patro.
Printed and bound by NBD Biblion.

Quantum Fine-Grained Complexity

Academisch Proefschrift

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. P.P.C.C. Verbeek
ten overstaan van een door het College voor Promoties ingestelde commissie,
in het openbaar te verdedigen in de Agnietenkapel
op donderdag 16 februari 2023, te 13.00 uur

door Subhasree Patro
geboren te Berhampur, Orissa

Promotiecommissie

Promotores:	prof. dr. H.M. Buhrman	Universiteit van Amsterdam
	dr. F. Speelman	Universiteit van Amsterdam
Overige leden:	prof. dr. R.M. de Wolf	Universiteit van Amsterdam
	dr. M. Ozols	Universiteit van Amsterdam
	prof. dr. C. Schaffner	Universiteit van Amsterdam
	dr. B. Serra Loff Barreto	University of Porto
	prof. dr. A. Ambainis	University of Latvia

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

List of publications

This thesis is based on the following papers. In each work, all authors contributed equally unless stated otherwise.

[BPS21] **A Framework of Quantum Strong Exponential-Time Hypotheses**. Harry Buhrman, Subhasree Patro, Florian Speelman. In *Proceedings of the 38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021)*, also presented at *TQC 2020*.

Chapter 3 is based on this paper.

[BLP+22a] **Limits of Quantum Speed-Ups for Computational Geometry and Other Problems: Fine-Grained Complexity via Quantum Walks**. Harry Buhrman, Bruno Loff, Subhasree Patro, Florian Speelman. In *Proceedings of the 13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, also presented at *TQC 2021* and *QIP 2022*.

Chapter 5 is based on this paper.

[BLP+22b] **Memory Compression with Quantum Random-Access Gates**. Harry Buhrman, Bruno Loff, Subhasree Patro, Florian Speelman. In *Proceedings of the 17th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2022)*.

Chapters 4 and 5 are based on this paper.

[ABL+22] **Matching Triangles and Triangle Collection: Hardness based on a Weak Quantum Conjecture**. Andris Ambainis, Harry Buhrman, Koen Leijnse, Subhasree Patro, Florian Speelman. Preprint available at *arXiv:2207.11068*.

Chapter 6 is based on this paper.

The author has also co-authored the following papers in the course of her PhD, which are not included in this thesis.

[CMP22] **Improved Quantum Query Upper Bounds Based on Classical Decision Trees**. Arjan Cornelissen, Nikhil S. Mande, Subhasree Patro. To appear in *Proceedings of the 42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022)*, also presented at *TQC 2022*.

[PP20] **An Overview of Quantum Algorithms: from Quantum Supremacy to Shor Factorization**. Álvaro Piedrafita Postigo, Subhasree Patro. In *Proceedings of the 52nd IEEE International Symposium on Circuits and Systems (ISCAS 2020)*. The first author is the main contributor of this work.

[PPV+21] **Impossibility of Cloning of Quantum Coherence**. Dhrumil Patel, Subhasree Patro, Chiranjeevi Vanarasa, Indranil Chakrabarty, Arun Kumar Pati. In *Journal of Physical Review A **103**, 022422 (2021)*. The first two authors are the main contributors. This work was done during her masters at IIIT-Hyderabad.

Contents

Part I
The Main Part

Chapter 1

Introduction

1.1 Fine-grained complexity

Recent advancements in quantum hardware technologies have made it even more exciting to further the theoretical development of quantum algorithms, because of the possible speedups for computational problems as compared to their respective classical counterparts. Interestingly, for some naturally occurring problems, we can prove limits on how much quantum speedup is achievable. For example, it can be shown that it is not possible to get a super-quadratic quantum speedup for the *unordered search* problem [BBB+97]. Most of these results follow from quantum *query* lower bounds that don't always immediately imply optimal time lower bounds, especially for problems that require super-linear time. In general, for a powerful enough computational model, time lower bounds are notoriously hard to obtain. To overcome this barrier, several recent works, some of which constitute this thesis, have been developing the field of *quantum fine-grained complexity*, giving us the power to prove conditional quantum time lower bounds for many computational problems and thereby elucidating the internal structure of the BQP complexity class.

Such results rely heavily on the notion of a reduction where one tries to solve a problem using an algorithm for another (somewhat unrelated) problem. Reductions were used by Turing in the context of computability theory dating back to as early as 1936. Polynomial-time reductions were used to demonstrate NP-completeness of many computational problems, which resulted in deciding whether a problem is in P or NP (of course conditional on $\mathsf{P} \neq \mathsf{NP}$). However, if one is interested in knowing the exact complexity of a problem, be it a problem in P or in NP, even while ignoring the constants and in our case poly-logarithmic factors, then the notion of NP-completeness doesn't help much. In some sense, the reductions are too coarse-grained to provide any more information about the complexity of those problems.

Fine-grained complexity, on the other hand, provides us with a technique to understand the *exact* complexity of a problem. The broad idea is as follows: pick a well-studied problem P, that on n input variables is conjectured to not be solvable in $p(n)^{1-\varepsilon}$ time in a particular model of computation, for any constant $\varepsilon > 0$. Let Q be another computational problem with a $q(n)$ time algorithm solving it in the same model of computation. Suppose, given an input instance of P we are able to generate input instances for Q such that we can solve P using an algorithm for Q, and if for any $\delta > 0$, a $q(n)^{1-\delta}$ time algorithm for solving Q implies a $p(n)^{1-\varepsilon}$ time

algorithm for P for some $\varepsilon > 0$, then we say that we have a reduction from P to Q. Using this reduction we can now conclude a $q(n)^{1-o(1)}$ time lower bound for Q based on the conjectured $p(n)^{1-o(1)}$ time lower bound for P.

The area of classical fine-grained complexity studies such reductions in a classical model of computation. Some of the well-studied problems in this context are SAT, 3SUM and APSP; often referred to as *key* problems. Based on the conjectured hardness of these key problems classical time lower bounds for a lot of computational problems have been derived; we discuss some of these results in Section 1.2. One can also, for example, see the survey articles [Vas15; Vas19] for a summary of many of these results and their corresponding reductions; Figure 1.1 captures some of these reductions too.

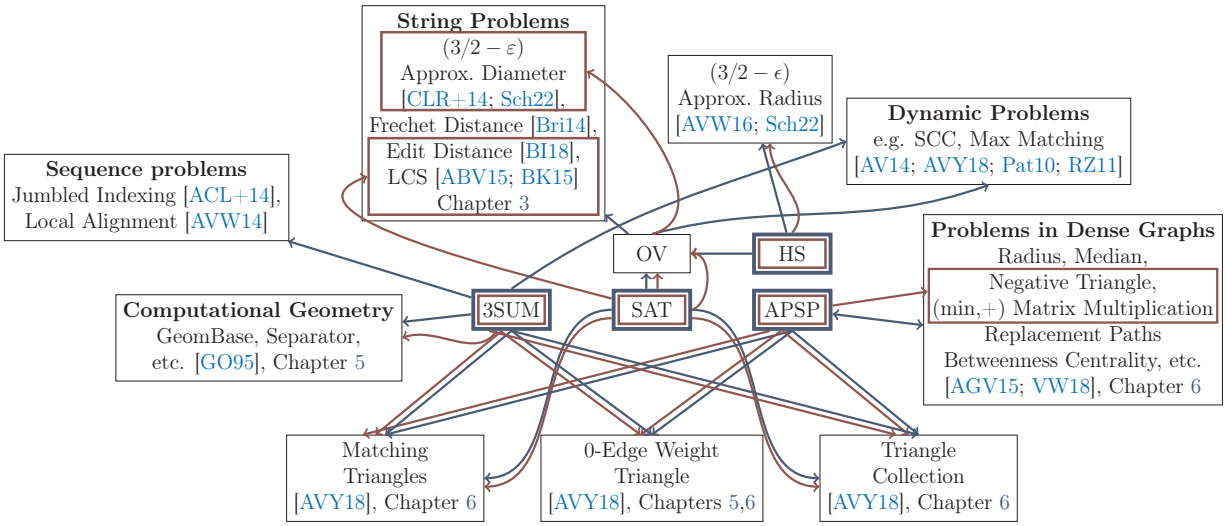

Figure 1.1: Overview of a few fine-grained reductions. The highlighted boxes denote the problems whose hardness is conjectured. The directed edges denote the direction of the reduction. The colour indicates whether the reduction is in the classical or quantum setting, coloured in dark blue and red, respectively.

These key problems are very suitable to use as a basis for such reductions as they are natural to describe and well studied. It is also interesting to see how these key problems computationally relate to other problems from various fields, such as Computer Science, Bio-Informatics, Artificial Intelligence, Physics, etc., providing us with a web of reductions which is of both theoretical and practical interest. More precisely because, if a speed-up is possible for any of the key problems (which happens if there's a speed-up for any problem they are reduced to) we are happy with having a new result about a fundamental problem. On the other hand, if no speed-up is possible, then the web of fine-grained reductions implies interesting lower bounds for a plethora of natural problems, which is an explanation for why better algorithms for these natural problems are not yet found in the considered model of computation.

'*Do these classical reductions hold in the quantum setting as well?*'

This is the primary question we address in this thesis. As we focus on the exact complexities of computational problems it is really important to fix the model

of computation — we fix a quantum model of computation that we formally define in Chapter 2. It doesn't come as a surprise that a computational problem can have different complexities in different models of computation; what is surprising is the unexpected ways this hinders us from directly using the classical reductions in the quantum setting. In this thesis, we discuss these challenges and present results in which we circumvent these challenges.

1.2 Our contributions

We further the field of fine-grained complexity by extending it to the quantum setting. Though our results are proven by adapting some known classical fine-grained reductions from CNF-SAT (and in some cases variants of CNF-SAT), 3SUM and APSP problems to the quantum setting, these adaptations are not trivial, as we will briefly see in the forthcoming sections and in detail in Chapters 3, 5, and 6.

1.2.1 Quantum reductions from CNF-SAT

One of the first problems we study in the context of quantum fine-grained complexity is CNF-SAT, the problem of whether a formula, input in conjunctive normal form, has a satisfying assignment. Classically, CNF-SAT on inputs with n variables is conjectured to require $2^{n(1-o(1))}$ time; also known as the Strong Exponential-Time Hypothesis (SETH). However, SETH fails to hold in the quantum setting because one can solve CNF-SAT in $O^*(2^{n/2})$ time quantumly.[1] The quadratically faster quantum algorithm to solve CNF-SAT is currently the best algorithm, and no further significant improvement to the run-time is known. Consequently, gives rise to the following conjecture.

Conjecture 1.1 (Basic-QSETH in Chapter 3, called QSETH in [ACL+20])**.** There is no bounded error quantum algorithm that solves CNF-SAT on n variables and m clauses in $O(2^{\frac{n}{2}(1-\delta)}m^{O(1)})$ time, for any $\delta > 0$.

The Basic-QSETH conjecture turns out to be useful in proving tight quantum time lower bounds for problems like Orthogonal Vectors, Closest Pair, Bichromatic Closest Pair, etc. [ACL+20]. However, it fails to say anything interesting for string comparison problems like Longest Common Subsequence (LCS) and Edit Distance. The reasons are as follows.

1. Both the best classical and quantum algorithms known for these string problems run in $O(n^2)$ time.

2. A series of classical reductions from CNF-SAT to Edit Distance and LCS, respectively, showed that any subquadratic algorithm for either of these two problems would refute SETH in the classical setting [BI18; BK15].[2]

3. Based on Basic-QSETH (Conjecture 1.1), the quantum complexity of CNF-SAT is at least $2^{n/2(1-o(1))}$ time, which is quadratically smaller than its classical complexity. Which means, even if it is possible to port the classical reductions to

[1] We use $O^*(\cdot)$ to hide polynomial factors of n. All such notational details are mentioned in Chapter 2.

[2] Such SETH-based classical lower bounds are also shown other string comparison problems [Bri14].

the quantum setting, this would only let us prove quantum time lower bounds for Edit Distance and LCS that are at best *linear*. These linear conditional lower bounds are not interesting because we can conclude unconditional lower bounds of the same complexity by encoding hard Boolean properties like Majority or Parity into input instances of these string comparison problems.

Unfortunately, this shows us how Basic-QSETH loses explanatory power in the context of certain string comparison problems; the best known upper bound is quadratic but the obtained lower bound is only linear. Fortunately, we are able to provide a solution for this.

1. Notice that, while it is possible to solve CNF-SAT quadratically faster using a quantum computer using the Grover's search algorithm as a subroutine, we do not know of quantum algorithms that can solve say $\oplus$CNF-SAT (computing the Parity of the number of satisfying assignments) or #CNF-SAT (computing the number of satisfying assignments) significantly faster than best classical algorithms, which we conjecture is because properties like Parity and Count are not amenable to Grover-like speedups.

2. Furthermore, the classical reduction from CNF-SAT to Edit Distance (or LCS) actually encodes a much harder property instead of checking if there is a satisfying assignment to the given input formula; 'harder' in the sense that the quantum query complexity of the property is provably harder than that of the OR problem.

With these observations, the following immediately interesting question surfaces:

'*How much time does it take for a quantum computer to solve a more general property on the set of satisfying assignments to a given* CNF-SAT *input?*'

Surprisingly, this question doesn't have a trivial answer even for properties whose query complexity is known or can be computed. While it is natural to "feel" that query complexity of a property is what translates into the hardness of solving that property on the set of satisfying assignments to a CNF-SAT input, this is actually not always the case. For example, consider the AND property: the query complexity of AND on N input variables is $\mathsf{Q}(\text{AND}) = \Theta(\sqrt{N})$, but, given an n variable CNF formula, one can in $O(\mathsf{polylog}N)$ time (where $N = 2^n$) find whether or not all assignments to this formula are satisfied; check whether each and every clause of the formula contains a variable along with its negation, only then the formula will be satisfiable on every possible assignment. We precisely needed to understand this situation to be able to give a better than linear quantum time lower bound for Edit Distance and LCS.

Keeping in mind these above-mentioned observations and insights, while also addressing technical struggles that came our way, we are able to develop a framework of Quantum Strong Exponential-Time Hypotheses that allows us to conjecture hardness of more general properties on the set of satisfying assignments to a CNF-SAT input (and also circuits of a more complex class). Our conjecture informally stated is as follows:

Conjecture 1.2 (Informal statement of γ-QSETH)**.** Let γ denote a class of representation such as poly-sized depth-2 circuits, poly-sized poly-logarithmic-depth circuits or poly-sized circuits of a more complex class, with n input variables.

- We define the compression oblivious properties corresponding to a class γ as the set of properties P such that the **time** required to compute P on the truth table of the input circuit from γ is lower bounded by the **query** complexity of P on all 2^n-length strings, i.e. $\mathsf{Q}(P)$.

- We then say, for all compression oblivious properties P of class γ given circuit $C \in \gamma$ the time taken to compute P on the truth table of C is at least $\mathsf{Q}(P)$.

As a result, conditional on Conjecture 1.2 we are able to show the following results:

1. We prove that Edit Distance (also LCS) requires $n^{1.5-o(1)}$ time to solve on a quantum computer, conditioned on QSETH. We do this by showing that Edit Distance (also, LCS) can be used to compute a harder property of the set of satisfying assignments with query complexity of $\Omega(N^{3/4})$ when the input is a N-bit Boolean string.

   Following [AHV+16], we are able to show this for a version of QSETH where the input formulas are branching programs instead, giving a stronger result than when assuming the hardness for only CNF inputs.

   As a corollary to the proof of the conditional Edit Distance (also for LCS) lower bound, we can show that the query complexity of the restricted Dyck language is linear for any $k = \omega(\log n)$, partially answering an open question posed by Aaronson, Grier, and Schaeffer [AGS19].[3]

2. Proofs of work is a form of cryptographic proof in which one party (the prover) proves to others (the verifiers) that a certain amount of a specific computational effort has been expended. A key feature of any proof-of-work scheme is that the computational task computed by the prover is moderately *hard* yet feasible while it is fairly easy for the verifiers to *verify* whether the task has been computed correctly.

   Assuming QSETH we are able to show that the *Proofs of Useful Work* scheme of Ball, Rosen, Sabin and Vasudevan [BRS+18] requires $n^{2-o(1)}$ time to solve on a quantum computer, matching the classical complexity of these proofs of work.

3. We also notice that some SETH-based $\Omega(T)$ lower bounds carry over to $\Omega(\sqrt{T})$ QSETH lower bounds, from which we immediately gain structural insight into the complexity class BQP.

We describe the QSETH framework and these results in detail in Chapter 3 of this thesis.

[3] Lower bounds for the restricted Dyck language were recently independently proven by Ambainis, Balodis, Iraids, Khadiev, Klevickis, Prūsis, Shen, Smotrovs and Vihrovs [ABI+20].

1.2.2 Quantum reductions from 3SUM

The kind of quantum fine-grained results we show next includes hardness results based on the quantum hardness of 3SUM, a problem when given a list of n integers, one needs to output whether or not the list contains a triple a, b, c such that $a+b+c = 0$. There is a simple classical algorithm that solves this problem in $\widetilde{O}(n^2)$ time: given input $S = (x_1, \ldots, x_n)$ to the 3SUM problem, sort S in $O(n \log n)$ time. Then, brute force search over all pairs $(a, b) \in S \times S$, and for each pair using binary search check if $-(a+b) \in S$. This entire procedure takes $O(n^2 \log n) = \widetilde{O}(n^2)$ time. Hence, a total of $\widetilde{O}(n^2)$ time.[4] Baran, Demaine and Pătraşcu give a $O(n^2/\max\{\frac{w}{\log^2 w}, \frac{\log^2 n}{(\log\log n)^2}\})$ time algorithm that solves 3SUM in the classical word-RAM model with w-bit words [BDP08].

Unfortunately, even after many years of interest in the problem, the exponent has not been reduced. The conjecture naturally arises that there is no $\delta > 0$ such that 3SUM can be solved in $O(n^{2-\delta})$ classical time, which we refer to as the Classical 3SUM Conjecture. Using this conjecture, one can derive conditional classical lower bounds for a vast collection of computational geometry problems, dynamic problems, sequence problems, etc. [GO95; VW13; Pat10; Vas15].

As expected, the Classical 3SUM Conjecture no longer holds true in the quantum setting, as there is a quadratically faster quantum algorithm for 3SUM: we may use Grover search as a subroutine in the $\widetilde{O}(n^2)$ classical algorithm to solve the problem in $\widetilde{O}(n)$ quantum time. Apart from this quadratic speedup, no further improvement to the quantum-time upper bound is known. Consequently, it was (informally) conjectured [AL20] that the 3SUM problem cannot be solved in sub-linear quantum time.

Conjecture 1.3 (Quantum 3SUM Conjecture [AL20])**.** There is no bounded-error quantum algorithm that solves 3SUM on a list of n integers in $O(n^{1-\delta})$ time, for any constant $\delta > 0$.

It is natural to try to extend the classical 3SUM-based lower bounds to the quantum setting, and one may at first expect this task to be a simple exercise. One soon realises that none of the existing classical reductions can be easily adapted to the quantum regime. Indeed most of the existing classical reductions begin by pre-processing the input in some way, e.g., by sorting it according to some ordering. This is not an issue in the classical setting, as the classical conjectured lower bound for 3SUM is quadratic. Hence, the classical reductions can accommodate any pre-processing of the input that takes sub-quadratic time, such as e.g. sorting (as sorting takes only $n \log n$ time on inputs of length n). However, this pre-processing becomes problematic in the quantum setting, since here we will need a sublinear-time quantum reduction, and even simple sorting (provably) requires linear time on a quantum computer [HNS01].

We present a solution to this problem.

1. The idea of our proof is to adapt Ambainis' quantum walk algorithm for Element Distinctness [Amb07], which was shown to work for the 3SUM problem as well [CE05]. To overcome, for example, the challenge of not being

[4] There is also a slightly less trivial $O(n^2)$ time algorithm as well; we leave this as an exercise to the reader until they reach Chapter 5 where the $O(n^2)$ algorithm is discussed.

3SUM-based quantum lower bounds (our results) ↴		**Classical upper & lower bounds, respectively** (**) ↴	
Problems		**Quantum upper-bound**	
GeomBase	$n^{1-o(1)}$	$\widetilde{O}(n)$ (*)	$O(n^2), n^{2-o(1)}$
3-Points-on-Line	$n^{1-o(1)}$	$O(n^{1+o(1)})$ [AL20]	$O(n^2), n^{2-o(1)}$
Point-on-3-Lines	$n^{1-o(1)}$	$O(n^{1+o(1)})$ [AL20]	$O(n^2), n^{2-o(1)}$
Separator	$n^{1-o(1)}$	$O(n^{1+o(1)})$ [AL20]	$O(n^2), n^{2-o(1)}$
Strips-Cover-Box	$n^{1-o(1)}$	$O(n^{1+o(1)})$ [AL20]	$O(n^2), n^{2-o(1)}$
Triangles-Cover-Triangle	$n^{1-o(1)}$	$O(n^{1+o(1)})$ [AL20]	$O(n^2), n^{2-o(1)}$
Point-Covering	$n^{1-o(1)}$	$O(n^{1+o(1)})$ [AL20]	$O(n^2), n^{2-o(1)}$
Visibility-Between-Segments	$n^{1-o(1)}$	$O(n^{1+o(1)})$ [AL20]	$O(n^2), n^{2-o(1)}$
Hole-In-Union	$n^{1-o(1)}$	$O(n^{1+o(1)})$ (†)	$\widetilde{O}(n^2), n^{2-o(1)}$
Triangle-Measure	$n^{1-o(1)}$	$O(n^2)$ (††)	$O(n^2), n^{2-o(1)}$
Visibility-From-Infinity	$n^{1-o(1)}$	$O(n^2)$ (††)	$O(n^2), n^{2-o(1)}$
Visible-Triangle	$n^{1-o(1)}$	$O(n^{1+o(1)})$ (†)	$O(n^2), n^{2-o(1)}$
Planar-Motion-Planning	$n^{1-o(1)}$	$O(n^2)$ (††)	$O(n^2), n^{2-o(1)}$
3D-Motion-Planning	$n^{1-o(1)}$	$O(n^2)$ (††)	$O(n^2), n^{2-o(1)}$
General-Covering	$n^{1-o(1)}$	$O(n^{1+o(1)})$ [AL20]	$O(n^2), n^{2-o(1)}$
Convolution-3SUM	$n^{1-o(1)}$	$O(n)$ (*)	$O(n^2), n^{2-o(1)}$
0-Edge-Weight-Triangle	$n^{1.5-o(1)}$	$O(n^{1.5})$ (*)	$O(n^3), n^{3-o(1)}$

(*) Using a simple Grover speedup on the classical algorithm.

(†) Implicit in [AL20], by using the classical reduction to Triangles-Cover-Triangle and then using the corresponding quantum algorithm.

(**) All upper bounds are straightforward: For problems like Convolution-3SUM and 0-Edge-Weight-Triangle the best known algorithms use brute force, for the computational-geometry problems, the upper bounds follow from geometry arguments [GO95]. All lower bounds for computational-geometry problems are from [GO95], the lower bound for Convolution-3SUM follows from [Pat10], and, the lower bound for 0-Edge-Weight-Triangle follows from [VW13]; all conditional on the classical hardness of 3SUM.

(††) Unfortunately, the current best quantum upper bound known is the classical upper bound, and there is no matching quantum lower bound known for these problems yet.

Table 1.1: This is a summary of all the Quantum-3SUM-hard problems mentioned in this thesis, with (almost) matching upper bounds for most of them.

able to sort the input in strictly sublinear time; instead of having the reduction sort the entire list we combine a data structure for dynamic sorting together with a quantum walk algorithm. As we will show in detail in Chapter 5, this approach only needs the reduction to sort a small part of the input and thus allows us to show that 3SUM remains hard, even when the entire input is sorted.

2. We will also see that this idea can be extended to allow for any "structuring" of the input (not just sorting) which can be implemented by a dynamic data structure obeying a certain "history-independence" property.

3. Also, interestingly enough, if one were to construct space *inefficient* dynamic data structures that satisfy this "history-independent" property then it is actually easy, but not entirely desirable because of the amount of space it uses. Fortunately, we have a solution for this as well. We present a technique that takes as input a space-inefficient but sparse quantum algorithm and constructs a space-efficient quantum algorithm that is still time efficient. Chapter 4 of this thesis and the content of Section 1.2.3 discusses this compression technique.

In the past, the quantum-walk plus data-structure proof strategy has been used to prove time upper bounds on other problems (e.g., for the closest-pair problem [ACL+20]), and here we use it for the first time as part of a reduction in order to obtain a lower bound. We expect that the same strategy will be applicable to other quantum fine-grained reductions, and our hope is that this will give rise to a landscape of results (in the future), that establishes (conditional) tight lower bounds for quantum algorithms. This will in turn precisely answer the question of how much quantum speed-up is possible for a variety of computational problems.

To summarise, using these strategies we are able to show that various "structured" versions of 3SUM are as hard as the original (unstructured) 3SUM problem, even in the quantum case. Once we have shown that these structured versions of 3SUM are hard, we may then construct direct quantum adaptations of the classical reductions, to show the quantum hardness of *several* computational-geometry problems, of Convolution-3SUM and of the 0-Edge-Weight-Triangle problem. This enables us to prove quantum time lower bounds for these problems, conditioned on the Quantum 3SUM Conjecture. See Table 1.1 for an overview of the quantum time bounds for these problems.

1.2.3 Memory compression with QRAGs

As discussed briefly in the previous section, in the process of proving quantum time lower bounds based on the conjectured hardness of the 3SUM problem, we had to develop dynamic data structures that satisfy the property of being "history-independent". We saw that it is very easy to conceive space *inefficient* data structures that use a polynomial amount of space (polynomial in the input length). While *space efficient* data structures that satisfy these properties exist in the literature, they usually are very complicated to analyse, unlike the space-inefficient ones. As a result, we investigate if there are any sparsification techniques possible in the quantum setting that help us to construct a space-efficient quantum algorithm from a given space inefficient algorithm in a black-box manner. Especially because such techniques are well known in the classical setting.

Classically, if we have an algorithm that uses M memory cells throughout its execution, but sparsely, in the sense that at any point in time, only m out of the M cells will be non-zero, then we may "compress" it into another algorithm which uses only $m \log M$ memory and runs in almost the same time. We may do so by simulating the memory using either a hash table or a self-balancing tree.

In this thesis, we show an analogous result for quantum algorithms equipped with quantum random-access gates. If we have a quantum algorithm that runs in time T and uses M qubits, such that the state of the memory, at any time step, is supported on computational-basis vectors of Hamming weight at most m, then it can be simulated by another algorithm which uses only $O(m \log M)$ quantum memory, and runs in time $\widetilde{O}(T)$. Our main theorem is (informally) stated as,

Theorem 1.4. *Any m-sparse quantum algorithm using time T and M qubits can be simulated with ε additional error by a quantum algorithm running in time $O(T \cdot \log(\frac{T}{\varepsilon}) \cdot \log(M))$, using $O(m \log M)$ qubits.*

We show how this theorem can be used, in a black-box way, to simplify the presentation in the following papers: *Ambainis' walk algorithm for element distinctness* [Amb07], *Quantum algorithms for closest pair and related problems* [ACL+20], and our very own *Fine-grained complexity via quantum walks* [BLP+22a].

Broadly speaking, when there exists a need for a space-efficient history-independent quantum data-structure, it is often possible to construct a space-inefficient, yet sparse, quantum data structure, and then appeal to our main theorem. This results in simpler and shorter arguments.

While this result was conceived and published later than that of the 3SUM related work (Chapter 5 of this thesis), we present this result in Chapter 4 so that the presentation of Chapter 5 in the thesis is simplified.[5]

1.2.4 Quantum complexity of APSP-hard problems

Next, we further extend the field of fine-grained complexity by studying the quantum hardness of the All Pairs Shortest Path (APSP) problem and reductions from it. APSP is defined as follows: given a weighted graph $G = (V, E)$ on n nodes with no negative-weight cycles, for every pair of vertices $(a, b) \in V \times V$, output the shortest distance between vertices a and b if there is a path, else output ∞.

The currently fastest known classical algorithm for APSP runs in $n^3/\exp(\sqrt{\log n})$ time [Wil18], and it has been conjectured that no $O(n^{3-\delta})$ time classical algorithm, for any constant $\delta > 0$, exists. Based on this conjecture, time lower bounds for a lot of problems, for example, Graph Radius, Graph Median, Negative Triangle, and many more, have been shown. However, the fastest known quantum algorithm to solve to solve APSP takes time $\widetilde{O}(n^{2.5})$, but no significant speedups to this algorithm have been found in a long time.[6] Therefore, it is natural to study the consequences of the following conjecture.

Conjecture 1.5 (Quantum APSP Conjecture)**.** There is no bounded-error quantum algorithm that solves APSP on a graph of n nodes in $O(n^{2.5-\delta})$ time, for any constant $\delta > 0$.

[5] Though this result is not directly related to the topic of quantum fine-grained complexity, it still aids in proving some fine-grained results, hence, it is included in this thesis.

[6] Also see Leijnse's master's thesis for a discussion on this [Lei22].

We study the classical reductions from APSP to the problems mentioned in Table 1.2 and observe that almost all those reductions can be trivially adapted to the quantum setting, thus proving tight lower bounds for nearly all of those six problems. The matching upper bounds for most of them can be derived using Grover-like speedups, except for Δ-Matching Triangles and Triangle Collection for which we present quantum algorithms that require careful use of data structures, Ambainis' variable time search and certain other ingredients as subroutines. We elaborate on these results in Chapter 6.

Additionally, in the classical case, hardness results for Δ-Matching Triangles and Triangle Collection can be derived conditioned on hardness of any of the three key problems. More precisely, Abboud, Williams, and Yu have proven that an $O(n^{3-\delta})$ time classical algorithm (for any constant $\delta > 0$ and $\omega(1) \leqslant \Delta(n) \leqslant n^{o(1)}$) for either of these two graph problems would imply faster classical algorithms for k-SAT, 3SUM and APSP, which makes Δ-Matching Triangles and Triangle Collection very interesting to study; this means one can now make fewer hardness assumptions when it comes to understanding hardness of Δ-Matching Triangles and Triangle Collection [AVY18]. Towards that, a weaker conjecture was introduced which states that at least one of the classical 3SUM-conjecture, APSP-conjecture or SETH is true. They called it the extremely popular conjecture.

Analogous to Abboud et al.'s classical result, we are also able to show that an $O(n^{1.5-\delta})$ time quantum algorithm (for any constant $\delta > 0$ and the same ranges of Δ) for either of these two graph problems would imply faster quantum algorithms for k-SAT, 3SUM and APSP. Not only does this allow us to check the consistency of our approaches, but clearly for problems such as Δ-Matching Triangles and Triangle Collection a weaker quantum hardness assumption can be made to show these time lower bounds. Hence, we state the following conjecture analogous to the classical case.

Conjecture 1.6. At least one of Conjectures 1.1, 1.3 or 1.5 is true.

Based on Conjecture 1.6, we prove tight quantum time bounds for Δ-Matching Triangles and Triangle Collection. The upper bounds are non-trivial and will be discussed in detail in Chapter 6. See Table 1.2 for an overview of the APSP-hard problems and their quantum time bounds.

To summarise, we present quantum analogues to some of the classical reductions from key problems like CNF-SAT, 3SUM and APSP. Figure 1.1 captures all the quantum fine-grained reductions presented in this thesis. Only barring a few for most computational problems discussed in the thesis we are able to prove *tight* quantum time bounds. See Tables 1.1 and 1.2 for an overview of these bounds.

1.3 Related work

Recently the field of quantum fine-grained complexity has also expanded to include the following results.

1. Independent from the work presented in Chapter 3 of this thesis, Aaronson, Chia, Lin, Wang, and Zhang [ACL+20] also defined a basic quantum version of the strong exponential-time hypothesis, which assumes that a quadratic speed-up over the classical SETH is optimal. They present conditional quantum lower bounds for Orthogonal Vectors, the Closest Pair problem,

Problem	Bound	Classical	Quantum
$(\min,+)$-Matrix Multiplication	Lower	$n^{3-o(1)}$ [FM71; Mun71]	$n^{2.5-o(1)}$ Lem 6.9
	Upper	$O(n^3)$ $(*)$	$O(n^{2.5})$ $(**)$
All-Pairs Negative Triangle	Lower	$n^{3-o(1)}$ [VW18]	$n^{2.5-o(1)}$ Lem 6.13
	Upper	$O(n^3)$ $(*)$	$O(n^{2.5})$ $(**)$
Negative Triangle	Lower	$n^{3-o(1)}$ [VW18]	$n^{1.5-o(1)}$ Lem 6.15
	Upper	$O(n^3)$ $(*)$	$O(n^{1.5})$ $(**)$
0-Edge-Weight Triangle	Lower	$n^{3-o(1)}$ [VW13]	$n^{1.5-o(1)}$ Lem 6.18
	Upper	$O(n^3)$ $(*)$	$O(n^{1.5})$ $(**)$
Δ-Matching Triangles	Lower	$n^{3-o(1)}$ [AVY18] (†)	$n^{1.5-o(1)}$ Lem 6.20 (†)
	Upper	$O(n^{3-o(1)})$ [AVY18] (†)	$\widetilde{O}(n^{1.5+o(1)})$ Cor 6.33 (†)
Triangle Collection	Lower	$n^{3-o(1)}$ [AVY18]	$n^{1.5-o(1)}$ Lem 6.23
	Upper	$O(n^3)$ $(*)$	$\widetilde{O}(n^{1.5})$ Thm 6.34

Table 1.2: Overview of lower bounds based on a hardness conjecture for APSP, both in the classical and in the quantum setting. Corresponding upper bounds are also provided.
$(*)$: These upper bounds are the most straight forward algorithms, like exhaustive search, and therefore have no particular source.
$(**)$: By applying Grover Search, potentially as subroutine.
(†): Holds only for $\omega(1) \leqslant \Delta \leqslant n^{o(1)}$.

and the Bichromatic Closest Pair problem, by giving fine-grained quantum reductions from k-SAT. All such lower bounds have a quadratic gap with the corresponding classical SETH lower bound. Despite the overlap in the topic, these results turn out to be complementary to the work presented in this thesis: in the current work we focus on defining a more extensive framework for QSETH that generalises in various ways the basic version. Our more general framework can exhibit a quantum-classical gap that is less than quadratic, which allows us to give conditional lower bounds for LCS and Edit Distance of $n^{1.5-o(1)}$ time and an $n^{2-o(1)}$ time lower bound for Useful Proofs of Work (thus a quadratic gap between prover and verifier). For our presented applications, the requirements of the fine-grained reductions are lower, e.g., when presenting a lower bound of $n^{1.5-o(1)}$ time for LCS or Edit Distance it is no problem if the reduction itself takes $\widetilde{O}(n)$ time. Conversely, we do not give the reductions that are given by [ACL+20]; those results are distinct new consequences of QSETH.

2. A quantum analogue of the Hitting-Set Conjecture (HSC), states that the Hitting-Set (HS) problem has at most a quadratic speed-up in the quantum setting – see Schoneveld's Bachelor's Thesis [Sch22]. Using this conjecture, it was possible to prove a linear quantum lower bound for the $(\frac{3}{2}-\varepsilon)$-approximate radius problem. The same lower bound is also proven for the $(\frac{3}{2}-\varepsilon)$-approximate diameter problem, using the quantum version of SETH.

1.4 Organisation of this thesis

In Chapter 2 we present some definitions that are highly relevant to this thesis: the model of computation (on which our results are based), the notion of a quantum

fine-grained reduction, and we also provide some basic preliminaries that will be used throughout the thesis. Chapter 3 constitutes the QSETH framework and some QSETH-based lower bounds as its consequence. Following that, in Chapter 4 we discuss the *memory compression technique*. Chapter 5 contains results on certain 3SUM-hard problems and how we arrive at those reductions. In Chapter 6 we discuss some APSP-hard problems and present non-trivial upper bounds for some of them. We end with concluding remarks in Chapter 7.

Chapter 2

Preliminaries

The primary goal of this chapter is to introduce the notations, the model of computation, and certain techniques that will be used throughout this thesis.

2.1 Notations

Definition 2.1 (Set Notation)**.** We let $[n] = \{1, \ldots, n\}$, and let $\binom{[n]}{\leqslant m}$ be the set of subsets of $[n]$ of size at most m.

Definition 2.2 (Hamming Weight)**.** Let $x \in \{0,1\}^n$, we use $|x|$ to denote the Hamming weight of x, i.e., $|x| = |\{i \mid i \in [n] \text{ and } x_i = 1\}|$.

Definition 2.3 (Asymptotic Notation)**.** Let f and g be non-decreasing functions from $\mathbb{N}$ to $\mathbb{R}$.

- $O(f(n))$ denotes the set of functions $g(n)$ for which there exists a $c \in \mathbb{R}_{\geqslant 0}$ and $n_0 \in \mathbb{N}$ such that $0 \leqslant g(n) \leqslant c \cdot f(n)$ for all $n \geqslant n_0$.
- $o(f(n))$ denotes the set of functions $g(n)$ for which there exists an $n_0 \in \mathbb{N}$ such that $0 \leqslant g(n) < c \cdot f(n)$ for all $c \in \mathbb{R}_{>0}$ and $n \geqslant n_0$.
- $\Omega(f(n))$ denotes the set of functions $g(n)$ for which there exists a $c \in \mathbb{R}_{>0}$ and $n_0 \in \mathbb{N}$ such that $g(n) \geqslant c \cdot f(n)$ for all $n \geqslant n_0$.
- $\omega(f(n))$ denotes the set of functions $g(n)$ for which there exists an $n_0 \in \mathbb{N}$ such that $g(n) > c \cdot f(n)$ for all $c \in \mathbb{R}_{\geqslant 0}$ and $n \geqslant n_0$.
- $\Theta(f(n))$ denotes the set of functions $g(n)$ that are both in $O(f(n))$ and $\Omega(f(n))$.

Generally, O and o are used to indicate upper bounds, Ω and ω to indicate lower bounds and Θ to indicate tight bounds. Throughout the literature, we also use other notations for hiding larger than constant factors: we use $\widetilde{O}(\cdot)$ to hide poly-logarithmic factors, i.e., $\widetilde{O}(g(n)) = O(g(n) \cdot \log^{O(1)}(n))$. This notation can be used on all the asymptotic notations. Besides that, we use $O^*(\cdot)$ to hide polynomial factors, $O^*(g(n)) = O(g(n) \cdot n^{O(1)})$. We also sometimes use $\mathsf{polylog}(n)$, $\mathsf{poly}(n)$ to denote $\log^{O(1)}(n)$ and $n^{O(1)}$ terms, respectively.

2.2 Quantum computing

We briefly sketch the basics and notation of quantum computing and ask the reader to refer to the book by Nielsen and Chuang [NC00] and de Wolf's lecture notes [Wol21] for more details.

Let $\mathcal{H}(N)$ denote the complex Hilbert space of dimension $N = 2^m$. An m-qubit state is an N-dimensional vector in $\mathcal{H}(N)$ with unit ℓ_2-norm; the state can be alternatively viewed as a linear combination of classical m-bit states written as

$$|\psi\rangle = \sum_{i\in\{0,1\}^m} \alpha_i |i\rangle,$$

where $|i\rangle$ denotes the basis state i, amplitudes $\alpha_i \in \mathbb{C}$, and $\sum_i |\alpha_i|^2 = 1$. A measurement of state $|\psi\rangle$ will output i with probability $|\alpha_i|^2$, and the state will then collapse to the observed state $|i\rangle$. We use $\||\psi\rangle\|$ to denote the ℓ_2-norm of $|\psi\rangle$.

Let $\mathcal{U}(N)$ denote the space of unitary linear operators from $\mathcal{H}(N)$ to itself (i.e. the unitary group). This is the only set of non-measuring quantum operations we consider for our algorithms and reductions presented in this thesis. To evolve a quantum state we apply a unitary (= linear and norm-preserving) transformation U to the vector of amplitudes. More precisely for any unitary $U \in \mathcal{U}(N)$ and quantum state $|\psi\rangle \in \mathcal{H}(N)$ we have

1. $U^\dagger U = I$ where $U^\dagger$ denotes the conjugate transpose of U and I denotes the identity matrix of dimension $N \times N$.
2. Let $|\psi'\rangle = U|\psi\rangle$ then (it follows from Item 1 that) $\||\psi'\rangle\| = \|U|\psi\rangle\| = 1$.
3. U is linear, i.e., $U|\psi\rangle = \sum_{i\in\{0,1\}^m} \alpha_i U|i\rangle$.

Let $N' = 2^{m'}$ for an $m' \in \mathbb{N}$. If $|\psi\rangle \in \mathcal{H}(N)$ and $|\phi\rangle \in \mathcal{H}(N')$ are quantum states on m and m' qubits, respectively, then the two-register state $|\psi\rangle \otimes |\phi\rangle = |\psi\rangle|\phi\rangle$ is a quantum state of $m + m'$ qubits which corresponds to the $N \cdot N'$-dimensional vector in $\mathcal{H}(N) \otimes \mathcal{H}(N')$ that is the tensor product of $|\psi\rangle$ and $|\phi\rangle$.

2.3 Model of computation

Ideally a good quantum model of computation would be the *standard quantum circuit* model which is a quantum analogue of the standard classical circuit model - replace AND, OR and NOT gates operating on bits by elementary quantum gates to operate on qubits. However, we instead consider a slightly different model, namely the *quantum query model* of computation. This model differs from the *standard quantum circuit* model because the input to this model is given as a "black-box" (also sometimes referred to as an "oracle") which is realisable by a unitary operation. More precisely, if the input x is an n bit Boolean string, then the oracle corresponding to x, usually denoted as $\mathcal{O}_x$ refers to the following unitary map,

$$\mathcal{O}_x|i\rangle|b\rangle \to |i\rangle|b \oplus x_i\rangle,$$

for all $i \in [n]$ and $b \in \{0, 1\}$. Many famous breakthrough results have been conceived in this model; for e.g., Grover's algorithm [Gro96], Shor's factoring algorithm [Sho97], etc. Moreover this model is very well studied. We can formalise the model in the following way.

2.3.1 Quantum query model

Recall that, we let $\mathcal{H}(N)$ to denote the complex Hilbert space of dimension N, and $\mathcal{U}(N)$ to denote the space of unitary linear operators from $\mathcal{H}(N)$ to itself. We let $\mathcal{B}$ denote a set of universal quantum gates, which we will fix to contain all 1-qubit and 2-qubit gates, i.e., all unitary operators in set $\mathcal{U}(2) \cup \mathcal{U}(4)$, but which could have been chosen from among any of the standard possibilities.

Of particular importance to us will be the set $\mathcal{Q} = \mathcal{B} \cup \{\mathsf{RAG}_n \mid n \text{ a power of } 2\}$ which contains our universal set together with the *random-access gates*, so that $\mathsf{RAG}_n \in \mathcal{U}(n2^n)$ is defined on the computational basis by

$$\mathsf{RAG}_n|i, b, x_1, \ldots, x_{n-1}\rangle = \begin{cases} |i, b, x_1, \ldots, x_{n-1}\rangle, & \text{if } i = 0, \\ |i, x_i, x_1, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_{n-1}\rangle, & \text{if } i \in [n-1], \end{cases}$$

$\forall b, x_1, \ldots, x_{n-1} \in \{0,1\}$. The RAGs are necessary to our setup because otherwise classical random-access operations that require $O(\log n)$ time would take a linear amount of time in the quantum setting. Also, note that RAG_n on states with $i = 0$ behaves like identity; this construction allows us to easily implement a controlled version of RAG_n in a black-box manner as discussed in Section 2.3.5.

We now give a formal definition of what it means to solve a Boolean relation $f \subseteq \{0,1\}^n \times \{0,1\}^m$ using a quantum circuit. This includes the special case when f is a function.

A *quantum circuit* over a gate set $\mathcal{G}$ ($= \mathcal{B}$ or $\mathcal{Q}$) is a tuple $\mathcal{C} = (n, T, S, C_1, \ldots, C_T)$, where $T \geqslant 0$, $n, S \geqslant 1$ are natural numbers, and the C_t give us a sequence of instructions. The number n is the input length, the number T is the *time complexity*, and S is the *space complexity*, also called the *number of wires* or the *number of ancillary qubits* of the circuit. Each instruction C_t comes from a set $\mathcal{I}_{\mathcal{G}}(S)$ of possible instructions, defined below. Given an input $x \in \{0,1\}^n$, at each step $t \in \{0, \ldots, T\}$ of computation the circuit produces an S-qubit state $|\psi_T(x)\rangle \in \mathcal{H}(2^S)$, starting with $|\psi_0(x)\rangle = |0\rangle^{\otimes S}$, and then applying each instruction C_t, as we will now describe.

For each possible q-qubit gate $G \in \mathcal{G} \cap \mathcal{U}(2^q)$, and each possible ordered choice $I = (i_1, \ldots, i_q) \in [S]^q$ of distinct q among S qubits, we have an instruction $\mathsf{APPLY}_{G,I} \in \mathcal{I}_{\mathcal{G}}(S)$ which applies gate G to the qubits indexed by I, in the prescribed order. The effect of executing the instruction $\mathsf{APPLY}_{G,I}$ on $|\psi\rangle \in \mathcal{H}(2^S)$ is to apply G on the qubits indexed by I, tensored with identity on the remaining $S - q$ qubits. I.e., $\mathsf{APPLY}_{G,I} \in \mathcal{U}(2^S)$ corresponds to the unitary transformation defined on each basis state by:

$$\mathsf{APPLY}_{G,I} \cdot |y_I\rangle \otimes |y_J\rangle = (G|y_I\rangle) \otimes |y_J\rangle,$$

where $J = [S] \backslash I$.

Furthermore, for each possible ordered choice $I = (i_1, \ldots, i_{\lceil \log n \rceil}) \in [S]^{\lceil \log n \rceil}$ of distinct $\lceil \log n \rceil$ among S qubits, and each $i \in [S] \backslash I$, we have an instruction $\mathsf{READ}_{I,i} \in \mathcal{I}_{\mathcal{G}}(S)$, which applies the query oracle on the qubits indexed by I and i. I.e., given an input $x \in \{0,1\}^n$, the instruction $\mathsf{READ}_{I,i} \in \mathcal{U}(2^S)$ applies the unitary transformation defined on each basis state by:

$$\mathsf{READ}_{I,i} \cdot |y_I\rangle \otimes |y_i\rangle \otimes |y_J\rangle = |y_I\rangle \otimes |y_i \oplus x_{y_I}\rangle \otimes |y_J\rangle,$$

where $J = [S] \backslash (I \cup \{i\})$.

Hence if we have a sequence $C_1, \ldots, C_T$ of instructions and an input x, we may obtain the state of the memory at time step t, on input x, by $|\psi_0(x)\rangle = |0\rangle^{\otimes S}$ and $|\psi_{t+1}(x)\rangle = C_{t+1}|\psi_t(x)\rangle$.

We say that a quantum circuit $\mathcal{C} = (n, T, S, C_1, \ldots, C_T)$ *computes* or *solves a relation* $f \subseteq \{0,1\}^n \times \{0,1\}^m$ *with error* ε if $\mathcal{C}$ is such that, for every input $x \in \{0,1\}^n$, if we measure the first m qubits of $|\psi_T(x)\rangle$ in the computational basis, we obtain, with probability $\geqslant 1 - \varepsilon$, a string $z \in \{0,1\}^m$ such that $(x, z) \in f$.

Also note that our description of what constitutes a quantum query algorithm subsumes the description that is usually presented in literature, which is a sequence of unitaries $U_0, \mathcal{O}_x, U_1, \mathcal{O}_x, \cdots, U_T$, applied on an initial state and followed by a measurement in the end. Each unitary U_i can be viewed as a series of APPLY instructions and the unitary $\mathcal{O}_x$ representing the "query" operation is analogous to the READ operation we define. We take effort in redefining the quantum query model because it allows us to easily quantify the amount of resources (such as time, queries or space) that a query algorithm uses. For example, given a circuit $\mathcal{C} = (n, T, S, C_1, \ldots, C_T)$, it is easy to see that the time complexity of this circuit is T, the query complexity of $\mathcal{C}$ is the number of READ instructions in $\mathcal{C}$, and space complexity is S.

2.3.2 Complexity of a Boolean relation f

In the earlier section, we could quantify the query, time, and space complexity of an algorithm computing a relation $f \subseteq \{0,1\}^n \times \{0,1\}^m$. We will now define the same quantities for f itself.

Quantum query complexity of f. The *ε-error quantum query complexity* of f, denoted by $\mathsf{Q}_\varepsilon(f)$, is the least number of queries required for a quantum query algorithm to compute f with error ε. When the subscript ε is dropped we assume $\varepsilon = 1/3$; the *bounded-error query complexity* of f is $\mathsf{Q}(f)$. Note that in between queries the algorithm can perform arbitrary unitary operations, which may depend on previously queried input bits, but the cost of these operations do not count towards the query cost.

Quantum time complexity of f. The *ε-error quantum time complexity* of f, denoted by $\mathsf{QTime}_\varepsilon(f)$, is the least amount of time required for a quantum query algorithm to compute f with error ε. Here as well, when the subscript ε is dropped we assume $\varepsilon = 1/3$; the *bounded-error time complexity* of f is $\mathsf{QTime}(f)$.

2.3.3 Quantum random-access machine (QRAM) model

Generally speaking, a quantum circuit is allowed to apply any of the basic operations to any of its qubits. In the definition given above, a quantum random-access gate can specify any permuted subset of the qubits to serve as its inputs. This allows for unusual circuit architectures, which are undesirable.

One may then define a more restricted class of circuits, as follows. We think of the qubits as divided into two parts: *work* qubits and *memory* qubits. We have M memory qubits and $W = O(\log M)$ work qubits, for a total space complexity $S = W + M$. We restrict the circuit so that any unitary gate $G \in \mathcal{B}$, or read instruction, must be applied to work qubits only. And finally, any random-access

gate must be applied in such a way that the addressing qubits (i) and the swap qubit (b) are always the first $\log M + 1$ work qubits, and the addressed qubits $(x_1, \ldots, x_M)$ are exactly the memory qubits, and are always addressed in the same, fixed order, so one can speak of *the first memory qubit, the second memory qubit, etc.* We may then think of a computation as alternating between doing some computation on the work registers, then swapping some qubits between work and memory registers, then doing some more computation on the work registers, and so forth. The final computational-basis measurement is also restricted to measuring a subset of the work qubits.

Under these restrictions, a circuit of time complexity T may be encoded using $O(T \log S)$ bits, whereas in general one might need $\Omega(TS)$ qubits in order to specify how the wires of the circuit connect to the random-access gates.

We will then use the term a *quantum random-access machine algorithm*, or *QRAM algorithm*, for a family of circuits that operate under these restrictions.[1] For example see Figure 2.1.

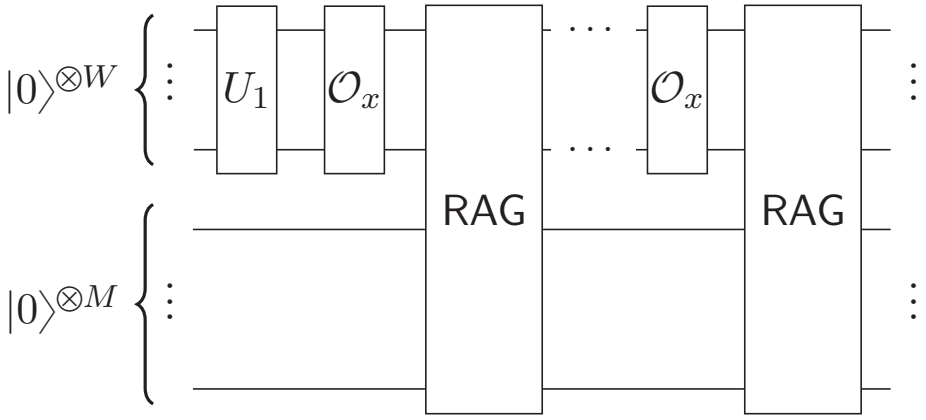

Figure 2.1: An example of a *Quantum Random-Access Machine (QRAM)* algorithm. There are W designated work qubits and M memory qubits. The *random-access gates* can only index into these M memory qubits; this is analogous to accessing any memory cell in "constant" time in the classical RAM model. The constant here depends on the precise architecture as discussed in Section 2.3.4.

2.3.4 Time complexity of simple operations (the constant ζ)

Throughout the thesis we will often describe algorithms that use certain simple operations over a logarithmic number of bits. These may include comparison, addition, bitwise XOR, swapping, and others. In a classical random-access machine, all of these operations can be done in $O(1)$ time, as in such machines it is usually considered that every memory position is a register that can hold $O(\log n)$ bits, and such simple operations are taken to be machine instructions.

[1] Such a computational model has been referred to by several names in the past. For instance, the term *QRAQM* appears in several publications, starting with [Kup05], and *QAQM* has also been used [NS20].

We do not necessarily wish to make such an assumption for quantum algorithms, since we do not really know what a quantum computer will look like just yet. So we will broadly postulate the existence of a quantity ζ, which is an upper-bound on the time complexity of doing such simple operations. We then express our time upper bounds with ζ as a parameter. Depending on the precise architecture of the quantum computer, one may think of ζ as being $O(1)$, or $O(\log n)$.

2.3.5 Controlled unitaries

Sometimes we will explain how to implement a certain unitary, and we wish to have a version of the same unitary that can be activated or deactivated depending on the state of an additional control bit.

The set of all 1-qubit operations together with the 2-qubit CNOT gate is universal, meaning that any other unitary transformation can be *exactly* built from these gates [NC00; Wol21]. Moreover, any 2-qubit unitary can be implemented using a constant number of CNOTs and 1-qubit unitaries [VW04]. As our gate set contains all 1-qubit and 2-qubit operations, the controlled versions of all the 1-qubit gates are already in our gate set, and we can naively construct a controlled version of any 2-qubit gate by first decomposing the 2-qubit gate into a constant number of CNOT and 1-qubit gates, then using the controlled version of each of these gates; the controlled version of CNOT is a Toffoli : $(x; y; z) \to (x; y; z \oplus xy)$ which in turn can be constructed using a constant number of CNOT and 1-qubit gates as shown in Chapter 4 of [NC00].

We can construct the controlled version of RAG_n using the procedure shown in Figure 2.2: all the quantum states $|i, b, x_1, \ldots, x_{n-1}\rangle$ with $i = 0$ are 1-eigenstates of RAG_n, therefore to be able to implement this controlled version, our algorithms will have to maintain an all-zero quantum state of $O(\log n)$ qubits. The controlled-SWAP gates need to only operate on the qubits associated with index i. Therefore, with only $O(\log n)$ extra qubits and $O(\log n)$ additional controlled-SWAP gates, each of which can be implemented with three Toffoli gates, we can implement the desired controlled version of RAG_n.

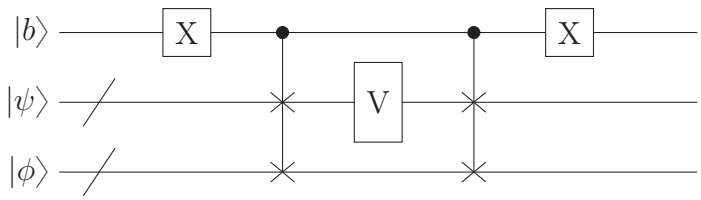

Figure 2.2: Constructing controlled-version of a gate V with only black-box access to V and access to the state $|\phi\rangle$ which is a 1-eigenstate of V, i.e., $V|\phi\rangle = |\phi\rangle$. When the control bit $b = 1$ the output state is $|1\rangle\otimes V|\psi\rangle\otimes|\phi\rangle$, as intended. In the other case, when the control bit $b = 0$ then the state $|\psi\rangle$ is first swapped with state $|\phi\rangle$ and then V operates on $|\phi\rangle$ which is a 1-eigenstate of V; with the controlled SWAP and X gate operating again we get $|0\rangle \otimes |\psi\rangle \otimes |\phi\rangle$ as the output.

If the use of extra memory in the above-mentioned construction is undesirable,

then it is not unreasonable to include a controlled version of RAG_n in our gate set $\mathcal{Q}$ as we already include RAG_n in $\mathcal{Q}$ — it is not hard to see that we can construct a controlled version of RAG_n in (almost) the same cost that is required to actually implement a RAG_n. With this, we can make free use of the following lemma.

Lemma 2.4. *If a unitary U can be implemented using T gates from $\mathcal{Q}$, then the unitary*

$$|b\rangle|x\rangle \mapsto \begin{cases} |b\rangle(U|x\rangle) & \text{if } b = 1 \\ |b\rangle|x\rangle & \text{if } b = 0 \end{cases}$$

can be implemented (without error) using $\widetilde{O}(T)$ gates from $\mathcal{Q}$.

2.3.6 Reversible classical computation

More often than not, we use classical procedures as subroutines in our quantum algorithms and reductions, and during those instances we implicitly claim that the time taken to realise any classical procedure using a quantum circuit is roughly similar to the time taken by a classical circuit realising the procedure. However, we cannot *trivially* embed such classical circuits in our quantum algorithms because in general classical circuits are not reversible. Fortunately, there are certain techniques that can be used to give a workaround to this problem.

To substantiate our claim, we first use the fact that any classical procedure (interpretable as a Boolean function) can be realised with a classical circuit consisting of $\mathsf{AND}, \mathsf{OR}, \mathsf{NOT}$ and classical random-access gates (cRAGs). Let us denote the gate set by $\mathcal{G}_c = \{\mathsf{AND}, \mathsf{OR}, \mathsf{NOT}, \mathsf{cRAG}\}$ and we define cRAG to implement the following map

$$(i, b, x_1, \ldots, x_{n-1}) \to \begin{cases} (i, b, x_1, \ldots, x_{n-1}), & \text{if } i = 0, \\ (i, x_i, x_1, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_{n-1}), & \text{if } i \in [n-1], \end{cases}$$

$\forall b, x_1, \ldots, x_{n-1} \in \{0,1\}$.[2] Secondly, we use the fact that any classical circuit implemented using T gates from $\mathcal{G}_c$ can be simulated by a reversible classical circuit of almost the same size consisting of only $\mathsf{Toffoli}$ and cRAG gates.

It is enough to only prove these results for single-bit-output Boolean functions of the kind $f : \{0,1\}^n \to \{0,1\}$, because any multi-bit-output Boolean function $g : \{0,1\}^n \to \{0,1\}^m$ can be viewed as a series of m different single-bit-output Boolean functions. Having said that, suppose we have a function $f : \{0,1\}^n \to \{0,1\}$, clearly an irreversible function, we first have to define an analogous reversible function corresponding to f, let us denote that by $\tilde{f}$, before attempting to construct a reversible circuit to compute f. Let $\tilde{f} : \{0,1\}^{n+1} \to \{0,1\}^{n+1}$ corresponding to f be the reversible function such that $\tilde{f}(x;0) = (x; f(x))$. We can now do the following.

First, note that every irreversible gate in $\mathcal{G}_c$ can be simulated by a $\mathsf{Toffoli}$ gate using one or two additional bits and by fixing inputs and ignoring outputs; it is also clear from the definition that cRAG is reversible. Let $\mathcal{C}$ be the irreversible circuit computing f. Replace every *irreversible* gate G in $\mathcal{C}$ with a corresponding $\mathsf{Toffoli}$ gate that implements G. The resultant circuit, let us denote this circuit by $\mathcal{C}''$, will now compute f but would collect a lot of "garbage" on its ancillary bits. Hence, copy out the final output on a fresh ancillary bit and then run $\mathcal{C}''$ in reverse.

[2]Note that cRAG is a classical analogue of RAG mentioned in Section 2.3.1.

With this procedure, the reversible circuit runs only about twice as long as the irreversible circuit $\mathcal{C}$ that it simulates, and all garbage generated in the simulation is also disposed. However, the number of additional ancillary bits used in this process is $O(T(n))$, which is undesirable.

However, turns out that it is possible to use space (i.e., ancillary bits) far more efficiently with only a minor blowup to the circuit size by using a *reversible pebble game* idea originally given by Bennett in 1989 [Ben89], and very well explained in Chapter 6 of Preskill's lecture notes [Pre98]. The (broad) idea is to first divide the computation in $\mathcal{C}''$ into smaller chunks of roughly the same size, and then run these steps backwards whenever possible during the course of the computation, so that the ancillary bits that are freed can be reused again and again. More precisely,

Lemma 2.5 ([Ben89; Pre98])**.** *Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function, and let $\mathcal{C}$ be a $T(n)$-size classical circuit, for some monotonically increasing function T, computing f over gate set $\mathcal{G}_c = \{\mathsf{AND}, \mathsf{OR}, \mathsf{NOT}, \mathsf{cRAG}\}$. Then, from the description of $\mathcal{C}$ one can construct a reversible classical circuit $\mathcal{C}'$ that computes $\tilde{f} : \{0,1\}^{n+1} \to \{0,1\}^{n+1}$ such that $\tilde{f}(x;0) = (x; f(x))$ where $\mathcal{C}'$ uses at most $(T(n))^{1+o(1)}$ cRAGs and $\mathsf{Toffoli}$ gates, and at most $\widetilde{O}(\log(T(n))$ additional ancillary bits.*[3]

The resultant circuit $\mathcal{C}'$ from Lemma 2.5 only contains Toffoli gates and cRAGs, therefore $\mathcal{C}'$ can be easily converted to a quantum circuit by replacing all its wires and gates with their respective quantum analogues; we can replace cRAG by RAG defined in Section 2.3.1 and implement a Toffoli gate using a constant number of 1-qubit and CNOT gates [NC00].

[3]Note that we get these numbers by plugging in values (suitable to our requirements) in the analysis given in Chapter 6 of Preskill's lecture notes [Pre98]. However, this is not in any sense "optimal" as there is a trade-off between the number of ancillary bits used and the size of the constructed reversible circuit, thus it really depends on what we want to optimise. If T_{irr}, T_{rev} denote the size of our irreversible and (to be constructed) reversible circuits, respectively, then we have $T_{irr} = \ell^m$, $T_{rev} = (2\ell-1)^m \leqslant T_{irr} \cdot 2^m$ and $M = m(\ell-1)+1$ with M denoting the number of additional ancillary bits used in the simulation. In our case we have $T_{irr} = T(n)$, and by plugging in $m = \log(T(n))/\log(\log n)$ we get the result mentioned in Lemma 2.5.

2.4 Quantum subroutines

Additionally, in the reductions and algorithms presented in this thesis, we make use of some well-known quantum algorithms as subroutines, mostly in a "black-box" manner. Most of these subroutines are bounded error with at most $\frac{1}{3}$ probability of error, which means one cannot directly use them because the errors in each step will quickly add up to something uncontrollably large. However, if the errors were small, say for example $\frac{1}{100\cdot\mathsf{poly}(n)}$, then an algorithm taking $\mathsf{poly}(n)$ steps will be able to tolerate this error, using a folkloric *compute-copy-uncompute* trick to make the imperfect subroutine behave sufficiently "close" to the perfect one. Moreover, we can also cheaply reduce the $\frac{1}{3}$ error probability to any $\epsilon > 0$, by repeating the subroutine for $O(\log \frac{1}{\epsilon})$ times and then outputting the majority of the outcomes. In this section, we formally present some of these useful techniques and mention some of the well-known quantum subroutines relevant to this thesis.

Theorem 2.6 (Perfect versus imperfect subroutines)**.** *Let $\mathcal{A}_f$ be an exact quantum algorithm computing a function $f : \{0,1\}^n \to \{0,1\}$ such that*

$$\mathcal{A}_f : |i\rangle|b\rangle|z\rangle \to |i\rangle|b \oplus f(i)\rangle|z\rangle,$$

$\forall i \in \{0,1\}^n, b \in \{0,1\}, z \in \{0,1\}^*$. Let $\widetilde{\mathcal{A}}_f$ be the bounded-error subroutine equivalent of $\mathcal{A}_f$ such that

$$\widetilde{\mathcal{A}}_f : |i\rangle|b\rangle|z\rangle \to \sqrt{1-p}|i\rangle|b \oplus f(i)\rangle|\psi_{i,b,z}\rangle + \sqrt{p}|i\rangle|b \oplus \overline{f(i)}\rangle|\psi^{\dagger}_{i,b,z}\rangle$$

i.e., the algorithm $\widetilde{\mathcal{A}}_f$ outputs the correct answer with probability $(1-p)$ and answers incorrectly with probability p for all $i \in \{0,1\}^n, b \in \{0,1\}$ and work qubits $z \in \{0,1\}^*$. Then, we can construct a unitary $\mathcal{U}_f$ using $\widetilde{\mathcal{A}}_f$, $\widetilde{\mathcal{A}}_f^{-1}$ and CNOTs to approximately implement $\mathcal{A}_f$ such that for any state $|\phi\rangle = \sum_{i,b,z} \alpha_{i,b,z}|i\rangle|b\rangle|z\rangle$ with $\sum_{i,b,z} |\alpha_{i,b,z}|^2 = 1$ and $\alpha_{i,b,z} \in \mathbb{C}$, we have

$$\|\mathcal{U}_f|\phi\rangle - \mathcal{A}_f|\phi\rangle\| \leqslant \sqrt{2p}.$$

Proof. We will construct a unitary $\mathcal{U}_f$ using $\widetilde{\mathcal{A}}_f, \widetilde{\mathcal{A}}_f^{-1}$ and CNOT gates to simulate $\mathcal{A}_f$ approximately. Note that a SWAP gate can be implemented using three CNOTs. Additionally, we will also use an ancillary qubit in our construction. For simplifying the presentation of the proof, we consider an implicit numbering of the quantum registers $|\phi\rangle$, $|0\rangle$ as $|\phi\rangle_{1,2,3}$ and $|0\rangle_4$, respectively. The broad idea of the construction is to apply $\widetilde{\mathcal{A}}_f$, then copy the output to a safe place, and use $\widetilde{\mathcal{A}}_f^{-1}$ to reverse the computation. More precisely, unitary $\mathcal{U}_f$ constitute the following maps.

$$
\begin{aligned}
|\phi\rangle|0\rangle = &\sum_{i,b,z} \alpha_{i,b,z}|i\rangle|b\rangle|z\rangle|0\rangle \\
\xrightarrow{\text{swap 2,4}} &\sum_{i,b,z} \alpha_{i,b,z}|i\rangle|0\rangle|z\rangle|b\rangle \\
\xrightarrow{\tilde{\mathcal{A}}_f \otimes \mathbb{I}} &\sum_{i,b,z} \alpha_{i,b,z}\left(\sqrt{1-p}|i\rangle|f(i)\rangle|\psi_{i,0,z}\rangle|b\rangle + \sqrt{p}|i\rangle|\overline{f(i)}\rangle|\psi^{\dagger}_{i,0,z}\rangle|b\rangle\right) \\
\xrightarrow{\text{copy 2 to 4}} &\sum_{i,b,z} \alpha_{i,b,z}\left(\sqrt{1-p}|i\rangle|f(i)\rangle|\psi_{i,0,z}\rangle|b\oplus f(i)\rangle + \sqrt{p}|i\rangle|\overline{f(i)}\rangle|\psi^{\dagger}_{i,0,z}\rangle|b\oplus \overline{f(i)}\rangle\right) \\
= &\sum_{i,b,z} \alpha_{i,b,z}\Big(\sqrt{1-p}|i\rangle|f(i)\rangle|\psi_{i,0,z}\rangle|b\oplus f(i)\rangle + \sqrt{p}|i\rangle|\overline{f(i)}\rangle|\psi^{\dagger}_{i,0,z}\rangle|b\oplus \overline{f(i)}\rangle \\
&+\sqrt{p}|i\rangle|\overline{f(i)}\rangle|\psi^{\dagger}_{i,0,z}\rangle|b\oplus f(i)\rangle - \sqrt{p}|i\rangle|\overline{f(i)}\rangle|\psi^{\dagger}_{i,0,z}\rangle|b\oplus f(i)\rangle\Big) \\
\xrightarrow{\text{rearrange}} &\sum_{i,b,z} \alpha_{i,b,z}\Big(\sqrt{1-p}|i\rangle|f(i)\rangle|\psi_{i,0,z}\rangle|b\oplus f(i)\rangle + \sqrt{p}|i\rangle|\overline{f(i)}\rangle|\psi^{\dagger}_{i,0,z}\rangle|b\oplus f(i)\rangle \\
&+\sqrt{p}|i\rangle|\overline{f(i)}\rangle|\psi^{\dagger}_{i,0,z}\rangle|b\oplus \overline{f(i)}\rangle - \sqrt{p}|i\rangle|\overline{f(i)}\rangle|\psi^{\dagger}_{i,0,z}\rangle|b\oplus f(i)\rangle\Big) \\
= &\sum_{i,b,z} \alpha_{i,b,z}\left(\widetilde{\mathcal{A}}_f(|i\rangle|0\rangle|z\rangle)|b\oplus f(i)\rangle + \sqrt{p}|i\rangle|\overline{f(i)}\rangle|\psi^{\dagger}_{i,0,z}\rangle(|b\oplus \overline{f(i)}\rangle - |b\oplus f(i)\rangle)\right) \\
\xrightarrow{\tilde{\mathcal{A}}_f^{-1} \otimes \mathbb{I}} &\sum_{i,b,z} \alpha_{i,b,z}\left(|i\rangle|0\rangle|z\rangle|b\oplus f(i)\rangle + \sqrt{p}\underbrace{\widetilde{\mathcal{A}}_f^{-1}(|i\rangle|\overline{f(i)}\rangle|\psi^{\dagger}_{i,0,z}\rangle)}_{|\eta_{i,b}\rangle \text{ with } \||\eta_{i,b}\rangle\|=1}\underbrace{(|b\oplus \overline{f(i)}\rangle - |b\oplus f(i)\rangle)}_{\|\cdot\|\leqslant\sqrt{2}}\right) \\
\xrightarrow{\text{swap 4,2}} &\sum_{i,b,z} \alpha_{i,b,z}|i\rangle|b\oplus f(i)\rangle|z\rangle|0\rangle + \sqrt{p}\underbrace{\sum_{i,b,z} \alpha_{i,b,z}|\eta'_{i,b}\rangle)}_{\|\cdot\|\leqslant\sqrt{2}} \\
= &\ \mathcal{A}_f|\phi\rangle|0\rangle + \sqrt{p}\underbrace{\sum_{i,b,z} \alpha_{i,b,z}|\eta'_{i,b}\rangle)}_{\|\cdot\|\leqslant\sqrt{2}}.
\end{aligned}
$$

Therefore, $\|\mathcal{U}_f|\phi\rangle - \mathcal{A}_f|\phi\rangle\| \leqslant \sqrt{2p}$. □

Next we describe a procedure to cheaply reduce the error to any $\epsilon > 0$.

Theorem 2.7 (Error Reduction Procedure, see de Wolf's Lecture notes [Wol21])**.** *If an algorithm $\mathcal{A}$, classical or quantum, outputs $0, 1$ with error probability at most $\frac{1}{3}$, then with $O(\log \frac{1}{\epsilon})$ repetitions of $\mathcal{A}$ one can reduce this error to any $\epsilon > 0$.*

Proof. Choose an odd integer N such that $2e^{-2\alpha^2 N} \leqslant \epsilon$ for $\alpha = 1/6$; it suffices to take $N = O(\log \frac{1}{\epsilon})$. Run $\mathcal{A}$ for about N times and output the majority of those N output bits. The probability that this majority is wrong (i.e., that the number of correct output bits is more than αN below its expectation), is at most ϵ by the Chernoff bound. Hence, with probability at least $1 - \epsilon$ we output the correct answer. □

Although the above-mentioned error reduction technique cannot be used directly for all functions in general, for algorithms with non-Boolean outputs that can be

modified to behave as one-sided error algorithms a similar technique (with a much simpler analysis) can be used. More precisely, for algorithms of the following kind: let $\mathcal{A}$ be 1/3-bounded error algorithm computing a Boolean relation $f \subseteq \{0,1\}^n \times \{0,1\}^m$ such that, on an input $x \in \{0,1\}^n$ the algorithm outputs a $z \in \{0,1\}^m$. Moreover, when $z \neq z_0$ for a specific $z_0 \in \{0,1\}^m$ then one can *efficiently*, i.e., in at most poly-logarithmic time, check if $(x,z) \in f$, however, when a $z = z_0$ it may take really long to verify whether or not $(x,z_0) \in f$. In such scenario, we use a slightly modified version of the error reduction procedure mentioned in the proof of Theorem 2.7. The adapted procedure is as follows: choose an integer N such that $\frac{2}{3}^N \leqslant \epsilon$, it then suffices to fix $N = O(\log \frac{1}{\epsilon})$. Plan to run algorithm $\mathcal{A}$ on input x for at most N times. If algorithm $\mathcal{A}$ outputs z_0 then re-run $\mathcal{A}$. However, if $\mathcal{A}$ outputs a $z \neq z_0$ then quickly check whether or not $(x,z) \in f$, if indeed $(x,z) \in f$ then output z and exit, else if $(x,z) \notin f$ then re-run $\mathcal{A}$. If $\mathcal{A}$ has already run N times then output z_0. The probability that $\mathcal{A}$ outputs z_0 incorrectly after $N = c \log \frac{1}{\epsilon}$ rounds is at most $\frac{1}{3}^N \leqslant \epsilon$ for $c \geqslant \frac{1}{\log 3}$. An example of such a Boolean relation which is of much relevance to us is the following problem.

Definition 2.8 (Unordered search problem)**.** Given an input string $x \in \{0,1\}^n$, output an $i \in [n]$ if there exists an i such that $x_i = 1$, else output 0.

The *unordered search* problem qualifies to be a true Boolean relation, because inputs x with Hamming weight $|x| \geqslant 2$ could have several different indices i, j such that $x_i = x_j = 1$ but $i \neq j$. Moreover, if an algorithm outputs an index $i \in [n]$ then we can with a single query to the input, i.e., in constant time, check if the algorithm outputted correctly. Therefore, in this scenario, we use the modified version of the error reduction algorithm, whenever needed. Fortunately, there is an interesting 1/3-bounded error quantum algorithm for this problem.

Theorem 2.9 (Grover's Search Algorithm [Gro96])**.** *Let $x \in \{0,1\}^n$ be an input to the* Unordered Search Problem *(Definition 2.8). Then, there exists a bounded error quantum algorithm that with probability at least 2/3 outputs a $j \in [n]$ if there exists a j such that $x_j = 1$, else outputs 0. This algorithm uses $O(\sqrt{n})$ queries and runs in $O(\sqrt{n})$ time.*

Additionally, Grover's search algorithm can be used in a slightly more generalised setting as well.

Theorem 2.10 (Implicit in [HMW03])**.** *Let $f : [n] \to \{0,1\}$ be a Boolean function computable by an exact or a 1/3-bounded error quantum circuit of size $T(n)$, for some monotonically increasing function T, then there exists a bounded error quantum algorithm, that in $O(T(n)\sqrt{n})$ time with probability at least 2/3, correctly outputs an index $i \in [n]$ such that $f(i) = 1$, else outputs 0.*

Proof Sketch. Let $\mathcal{A}_f$, $\widetilde{\mathcal{A}}_f$ denote the exact and the probabilistic circuits of size $T(n)$, respectively.

In the exact case, we have

$$\mathcal{A}_f|i\rangle|b\rangle|z\rangle \to |i\rangle|b \oplus f(i)\rangle|z\rangle$$

for all $i \in [n], b \in \{0,1\}$ and $z \in \{0,1\}^*$. Using $\mathcal{A}_f$, a Hadamard gate and an additional ancillary qubit, we can implement its corresponding phase oracle $\mathcal{A}_{f,\pm}|i\rangle =$

$(-1)^{f(i)}|i\rangle$. One can now find an $i \in [n]$ such that $f(i) = 1$ (if such an i exists) using Grover's algorithm with $\mathcal{A}_{f,\pm}$ instead of the regular $\mathcal{O}_{x,\pm}$ [Gro96]. The entire circuit will be of size $O(T(n)\sqrt{n})$.

In the case when f is computed with error, we have

$$\widetilde{\mathcal{A}}_f|i\rangle|b\rangle|z\rangle \to \sqrt{\frac{2}{3}}|i\rangle|b \oplus f(i)\rangle|\psi_{i,b,z}\rangle + \sqrt{\frac{1}{3}}|i\rangle|b \oplus \overline{f(i)}\rangle|\psi^{\dagger}_{i,b,z}\rangle.$$

We will present a sub-optimal, $\widetilde{O}(T(n)\sqrt{n})$ time, procedure here and refer to [HMW03] for a more efficient procedure that uses recursive interleaving of amplitude amplification and error-reduction techniques to find an index $i \in [n]$, if any, such that $f(i) = 1$, in $O(T(n)\sqrt{n})$ time [HMW03]. The sub-optimal procedure is as follows: use the error reduction technique as mentioned in Theorem 2.7 to reduce the error to an $\epsilon = \frac{1}{\mathsf{poly}(n)}$, incurring an $O(\log n)$ factor overhead. Following that, use the *compute-copy-uncompute* technique mentioned in the proof of Theorem 2.6 to simulate $\mathcal{A}_f$ approximately, and use the Grover's subroutine on top of that. This entire procedure will then take $\widetilde{O}(T(n)\sqrt{n})$ time. □

Another variation of Grover's algorithm can be used to solve the maximum or minimum finding problems in $O(\sqrt{n})$ time [DH96].

Theorem 2.11 (Maximum or Minimum Finding Algorithm [DH96])**.** *Let $f : [n] \to \mathbb{R}$ be a function. There exists a quantum algorithm, that with probability at least $(1 - 1/2^c)$, outputs the index $i \in [n]$ for which $f(i)$ is minimised (or maximised) in $O(c\sqrt{n})$ time.*

Finally, we mention the *Variable Time Grover Search* subroutine which we use for obtaining non-trivial quantum time upper bounds for certain graph problems.

Theorem 2.12 (Variable Time Grover Search [Amb10])**.** *Let $f : [n] \to \{0, 1\}$ be a Boolean function, and suppose we can compute $f(i)$ in time t_i, then there exists a bounded error quantum algorithm that with probability at least $2/3$, outputs an index $i \in [n]$, if there exists an i such that $f(i) = 1$, else outputs 0 in $\widetilde{O}(\sqrt{\sum_{i=1}^{n} t_i^2})$ time.*

Also note that the results of Theorem 2.10 in the exact case directly follow from Theorem 2.12.

2.5 Quantum basic adversary method

Time lower bounds, whether classical or quantum, are hard to obtain, however there are known techniques to prove *query* bounds in the quantum model of computation considered in this thesis. Additionally, these query lower bounds immediately translate to time lower bounds in this model, but often they are not tight. Even though we care about tight time bounds in this thesis, we sometimes use the query lower bounds to conclude time lower bounds for variants of CNF-SAT problems, as we will see in Chapter 3. We use Ambainis's *quantum adversary method* for proving those query lower bounds.

Theorem 2.13 (Basic quantum adversary method [Amb02])**.** *Let $f : \{0, 1\}^n \to \{0, 1\}$ be a decision problem, with $X \subseteq f^{-1}(0)$ and $Y \subseteq f^{-1}(1)$. Let $R \subseteq X \times Y$ be a relation such that*

1. *For every $x \in X$, there exist at least m different $y \in Y$ such that $(x, y) \in R$.*
2. *For every $y \in Y$, there exist at least m' different $x \in X$ such that $(x, y) \in R$.*
3. *For every $x \in X$ and $i \in [n]$, there are at most ℓ different $y \in Y$ such that $(x, y) \in R$ and $x_i \neq y_i$.*
4. *For every $y \in Y$ and $i \in [n]$, there are at most ℓ' different $x \in X$ such that $(x, y) \in R$ and $x_i \neq y_i$.*

Then any quantum algorithm computing f uses $\Omega(\sqrt{\frac{mm'}{\ell\ell'}})$ queries.

Note that, there are several other, comparable and incomparable, techniques known to prove quantum query lower bounds, for example, the *polynomial method* [BBC+01], the *Kolmogorov complexity method* [LM08], the *spectral adversary method* [BSS03], etc. Additionally, it has been shown that variants of the quantum adversary method such as the spectral adversary [BSS03], strong weighted adversary [Zha04] and the Kolmogorov complexity adversary [LM08] are equivalent [ŠS06]. Moreover, the *negative weight adversary method* [HLŠ07] tightly (up to constant factors) characterises query complexity of functions [LMR+11; Rei11]. However, these generalised methods are hard to use in practice.[4] On the other hand, the *basic adversary method* is simple to use but has some limitations: it is shown that, if f is a total Boolean function, the lower bounds achieved using this method or even the Weighted Adversary method [Amb06] are at most $\sqrt{C_0 C_1}$ where C_i denotes the i-certificate complexity of f [Zha04]; a limitation that leads to an open question in Chapter 3.

2.6 Quantum fine-grained reductions

The need to define quantum fine-grained reductions stems from the fact that quantum algorithms, thus quantum reductions, obey different rules than their classical counterparts. We use a *slightly* modified version of the definition of quantum fine-grained reduction, described by Aaronson *et al.* in [ACL+20]. Note that this definition in turn is inspired by the definition of classical fine-grained reduction given by Williams in [Vas15], appropriately modified to account for quantum operations in a reduction.

First, recall that in the quantum query model, we do not have to write down the elements of an instance explicitly, instead we have "on-the-fly" access to these elements.

Definition 2.14 (Quantum Oracle)**.** Let $X := (x_1, \ldots, x_n)$ be the instance of some problem P and let $\mathcal{O}_X$ be the corresponding oracle that can be accessed in superposition. To realise $\mathcal{O}_X$, we do not need to write down the entire X. Instead, we can design a quantum circuit to realise the mapping

$$\mathcal{O}_X : |i\rangle|b\rangle \to |i\rangle|b \oplus x_i\rangle.$$

Secondly, in our quantum reductions we should be able to access quantum subroutines of the following kind.

[4] These methods are not difficult to use if the person who is using them is A. Belovs.

Definition 2.15 (γ-Oracle for problem Q)**.** Let Q be a computational problem with access to an input instance $X = (x_1, \ldots, x_n)$ given by a quantum oracle $\mathcal{O}_X$ (Definition 2.14). Let $\mathcal{A}(\mathcal{O}_X)$ denote an algorithm or an oracle $\mathcal{A}$ with access to oracle $\mathcal{O}_X$. Now we say $\mathcal{A}_\gamma$ is a γ-oracle for problem Q, if for every instance $\mathcal{O}_X$ of Q it holds that

$$\Pr[\mathcal{A}_\gamma(\mathcal{O}_X) = Q(X)] \geqslant 1 - \gamma,$$

where $Q(X)$ is the solution of instance X for problem Q.

Finally, following is the definition of what constitutes a *quantum fine-grained reduction.*

Definition 2.16 (Quantum Fine-Grained Reduction)**.** Let $p(n)$, $q(n)$ and $r(n)$ be non-decreasing functions of n. Let P and Q be two problems in the quantum query model and $\mathcal{A}_\gamma$ be a γ-oracle for Q with error probability $\gamma \leqslant 1/3$. The problem P is quantum (p, q)-reducible to Q, denoted as $(P, p) \leqslant_{QFG} (Q, q)$, if for every $\epsilon > 0$, there exists a $\delta > 0$, and a quantum algorithm F with access to $\mathcal{A}_\gamma$, constants c, d, and for every $n \geqslant 1$ an integer $\ell(n)$, such that for every n, the algorithm F takes any input of P of size n and satisfies the following:

1. F can solve P with success probability at least $2/3$, in time at most $c \cdot (p(n))^{1-\delta} \cdot \log^{O(1)}(p(n))$.

2. F performs at most $\ell(n)$ quantum queries to $\mathcal{A}_\gamma$. Specifically, in the i-th query, let $X_i := \{X_{1,i}, X_{2,i}, \ldots\}$ be a set of instances of Q. Then F realises the oracles $\{\mathcal{O}_{X_{1,i}}, \mathcal{O}_{X_{2,i}}, \ldots\}$ in superposition and applies $\mathcal{A}_\gamma$ to solve the instances.

3. The following inequality holds

$$\sum_{i=1}^{\ell(n)} t(X_i) \cdot (q(n_i))^{1-\epsilon} \leqslant d \cdot (p(n))^{1-\delta},$$

   where $t(X_i)$ is the time required for F to realise the oracles $\{\mathcal{O}_{X_{1,i}}, \mathcal{O}_{X_{2,i}}, \ldots\}$ in superposition and $n_i := \max_j |X_{j,i}|$.

In Definition 2.16, the input of $\mathcal{A}_\gamma$ is given as a quantum oracle such that $\mathcal{A}_\gamma$ can be a quantum query algorithm with running time strictly less than the input size. Moreover, the quantum reduction F can realise quantum oracles $\{\mathcal{O}_{X_{1,i}}, \mathcal{O}_{X_{2,i}}, \ldots\}$ in superposition, and thus the time required is $\max_i t(X_{i,j})$ (where $t(X_{i,j})$ is the time required to realise $\mathcal{O}_{X_{i,j}}$) instead of $\sum_i t(X_{i,j})$. This also allows F to use fast quantum algorithms to process the information of $\mathcal{A}_\gamma$'s output (for example, amplitude amplification). See Figure 2.4 for reference.

Also note that our definition of quantum fine-grained reduction, with a (slight) modification to Definition 25 in [ACL+20], firstly allows us to account for any pre-processing time that algorithm F might spend on input X, and secondly lets us account for the time taken to reduce the error probability γ to a value suitable for a large number of calls to $\mathcal{A}_\gamma$.

It might not be immediately clear how this definition of a quantum fine-grained reduction helps in proving conditional quantum time lower bounds for any problem. To understand that, let us re-consider the problems P and Q, and let $p(n)$ and $q(n)$ be two non-decreasing functions of n. From Definition 2.16 we see that

if $(P,p) \leqslant_{QFG} (Q,q)$, i.e., P is quantum (p,q)-reducible to Q, then there exists a $\delta > 0$ (corresponding to an $\epsilon > 0$), such that one can solve P in at most $c \cdot \log^{O(1)}(p(n)) \cdot (p(n))^{1-\delta}$ time using a $q(n)^{1-\epsilon}$ time quantum algorithm for Q as subroutines. Moreover, if we have sufficient reason to believe that P cannot be solved in $(p(n))^{1-\alpha}$ time (for any $\alpha > 0$) while $(P,p) \leqslant_{QFG} (Q,q)$ holds, then it shows that Q also cannot have a $q(n)^{1-\epsilon}$ time quantum algorithm for any $\epsilon > 0$. Thus, showing how a conjectured time lower bound of $p(n)^{1-o(1)}$ for P can be used towards proving a time lower bound of $q(n)^{1-o(1)}$ for Q, when $(P,p) \leqslant_{QFG} (Q,q)$.

Consequently, what immediately follows from Definition 2.16 is the next definition.

Definition 2.17 (Quantum Fine-Grained Equivalent)**.** Let P and Q be two problems in the quantum query model and let $p(n)$ and $q(n)$ be two non-decreasing functions of n. We say P and Q are (p,q)-equivalent, denoted by $(P,p) =_{QFG} (Q,q)$, if we have $(P,p) \leqslant_{QFG} (Q,q)$, and $(Q,q) \leqslant_{QFG} (P,p)$.

Additionally, Definitions 2.16 and 2.17 also satisfy the transitivity condition. See Theorem 3.3 and 4.4 of Schoneveld's Bachelor Thesis [Sch22], where this question of *transitivity* of fine-grained reductions was also addressed, but formally proven only in the classical setting.

Theorem 2.18 (Transitivity of Quantum Fine-Grained Reductions)**.** *Let P, Q and R be problems in the quantum query model, and let $p(n), q(n)$ and $r(n)$ be non-decreasing functions of n. If $(P,p) \leqslant_{QFG} (Q,q)$ and $(Q,q) \leqslant_{QFG} (R,r)$, then $(P,p) \leqslant_{QFG} (R,r)$.*

Proof. Let $\mathcal{A}^Q_\gamma$, $\mathcal{A}^R_\gamma$ denote a γ-oracle for Q, R, respectively, with error probability $\gamma \leqslant 1/3$. When the subscript γ is dropped we assume $\gamma = 1/3$. We will first construct a reduction algorithm F_{PR} using the reduction algorithms F_{PQ} and F_{QR} that we will soon define.

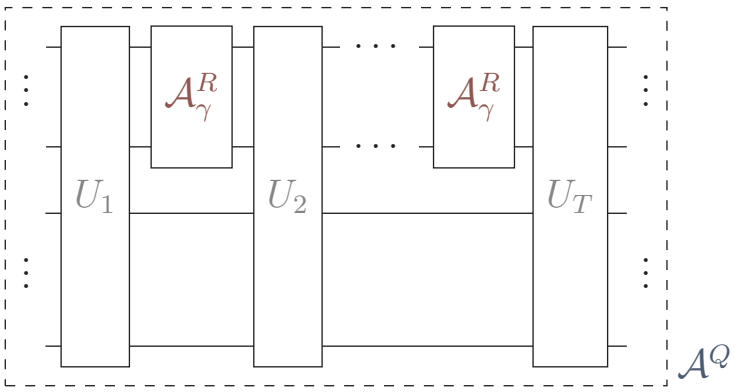

Figure 2.3: Algorithm F_{QR} with access to $\mathcal{A}^R_\gamma$ which is a γ-oracle for problem R, with $T \leqslant m(n)$. One way to implement $\mathcal{A}^Q_{1/3}$ is to run the algorithm F_{QR} on $\mathcal{A}^R_\gamma$ with $\gamma = \frac{1}{3m(n)}$. To implement $\mathcal{A}^Q_\epsilon$ for any $\epsilon > 0$ repeat $\mathcal{A}^Q_{1/3}$ for $O(\log(\frac{1}{\epsilon}))$ times and take the majority of the outcomes.

Construction. Consider an $\epsilon > 0$. Given that $(Q, q) \leqslant_{QFG} (R, r)$, it means for the ϵ we have considered, there exists $\delta > 0$ and a quantum algorithm, let us denote by F_{QR}, that makes quantum queries (in superposition) to $\mathcal{A}^R$ to solve Q, as shown in Figure 2.3. On an input of length n, Algorithm F_{QR} makes at most $m(n)$ queries to $\mathcal{A}^R$, possibly in superposition, and runs for at most $c \cdot (q(n))^{1-\delta} \cdot \log^{O(1)}(q(n))$ time. The time required by F_{QR} to implement i^{th} query is $t(X_i) \cdot (r(n_i))^{1-\epsilon}$, with $t(X_i)$ and n_i as mentioned in Definition 2.16. Additionally, we are also promised that

$$\sum_{i=1}^{m(n)} t(X_i) \cdot (r(n_i))^{1-\epsilon} \leqslant d \cdot (q(n))^{1-\delta}. \tag{2.1}$$

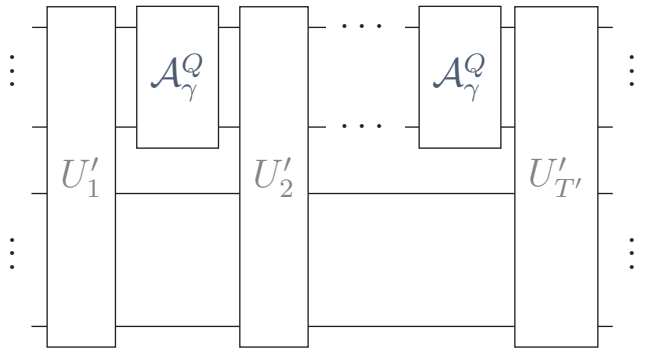

Figure 2.4: Algorithm F_{PQ} with access to $\mathcal{A}^Q_\gamma$ with $T' \leqslant \ell(n)$.

Given that we also have $(P, p) \leqslant_{QFG} (Q, q)$, now consider an algorithm F_{PQ} corresponding to the δ we get from the previous paragraph, and let $\beta > 0$ correspond to this δ. Algorithm F_{PQ} makes at most $\ell(n)$ quantum queries to $\mathcal{A}^Q$ and computes P in at most $c' \cdot (p(n))^{1-\beta} \cdot \log^{O(1)}(p(n))$ time. We will now construct a new algorithm F_{PR} from algorithm F_{PQ} by replacing the $\mathcal{A}^Q$ subroutines with the circuit corresponding to F_{QR}, as shown in Figure 2.5. Once again we are promised that

$$\sum_{i=1}^{\ell(n)} t(X_i) \cdot (q(n_i))^{1-\delta} \leqslant d' \cdot (p(n))^{1-\beta}. \tag{2.2}$$

Analysis.

- Consider the i^{th} query made to $\mathcal{A}^Q$ by F_{PQ}. F_{PQ} takes $t(X_i)$ time to realise the oracles $\{\mathcal{O}_{X_{1,i}}, \mathcal{O}_{X_{2,i}}, \dots\}$ in superposition and uses $\mathcal{A}^Q$ on inputs of length at most $n_i := \max_j |X_{j,i}|$.

  Let us now analyse how much time it must require to run $\mathcal{A}^Q(\mathcal{O}_{X_{i,j}})$. Let $X_{i,j} = X'$. For this we will now use the algorithm F_{QR}. Let us consider the k^{th} query made by F_{QR} to $\mathcal{A}^R$. Algorithm F_{QR} uses $t(X'_k)$ to realise the $\{\mathcal{O}_{X'_{1,k}}, \mathcal{O}_{X'_{2,k}}, \dots\}$ in superposition and uses $\mathcal{A}^R$ on inputs of length at most $n'_k := \max_j |X'_{j,k}|$. Therefore, algorithm F_{PR} can now in at most $t(X_i) \cdot t(X'_k)$ time realise the input oracle to $\mathcal{A}^R$.

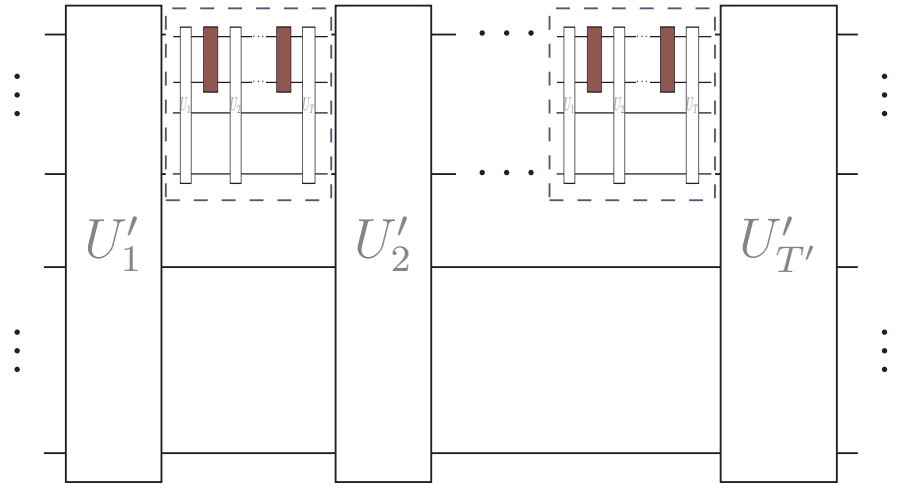

Figure 2.5: Algorithm F_{PR} with access to $\mathcal{A}^R_\gamma$; the darkred shaded areas in the figure corresponding to queries to $\mathcal{A}^R_\gamma$.

In order to prove the theorem, we will first check if the third condition of Definition 2.16 is satisfied. Towards that, let us analyse the following quantity with $i \in [\ell(n)]$ and $k \in [m(n_i)]$,

$$\sum_{i,k} t(X_i) \cdot t(X'_k) \cdot (r(n'_k))^{1-\epsilon} \leqslant \sum_{i}^{\ell(n)} t(X_i) \underbrace{\sum_{k}^{m(n_i)} t(X'_k) \cdot (r(n'_k))^{1-\epsilon}}_{\leqslant d\cdot(q(n_i))^{1-\delta} \text{ from Equation 2.1}} \tag{2.3}$$

$$\leqslant d \cdot \underbrace{\sum_{i}^{\ell(n)} t(X_i)(q(n_i))^{1-\delta}}_{\leqslant d'\cdot(p(n))^{1-\beta} \text{ from Equation 2.2}} \tag{2.4}$$

$$\leqslant d \cdot d' \cdot (p(n))^{1-\beta}. \tag{2.5}$$

Thus, there exists a constant $d'' = d \cdot d'$ such that condition 3 in Definition 2.16 is satisfied.

- The total time taken by F_{PR} is now at most $c'' \cdot (p(n))^{1-\beta} \cdot \log^{O(1)}(p(n))$ with $c'' = \max\{d'', c'\}$.

Therefore, we have shown that for any $\epsilon > 0$, there exists a $\beta > 0$, constants c'', d'', an algorithm F_{PR} that with access to $\mathcal{A}^R_\gamma$ of run time $(r(n))^{1-\epsilon}$ solves P in at most $c'' \cdot \log^{O(1)}(p(n)) \cdot (p(n))^{1-\beta}$ time. Hence, $(P, p) \leqslant_{QFG} (R, r)$. □

As a corollary to Theorem 2.18 we get the following result.

Corollary 2.19. *Let P, Q and R be problems in the quantum query model, and let $p(n)$, $q(n)$ and $r(n)$ be non-decreasing functions of n. If $(P, p) =_{QFG} (Q, q)$ and $(Q, q) =_{QFG} (R, r)$, then $(P, p) =_{QFG} (R, r)$.*

Chapter 3

A Framework of Quantum Strong Exponential-Time Hypotheses

Chapter summary The strong exponential-time hypothesis (SETH) is a commonly used conjecture in the field of complexity theory. It essentially states that determining whether a CNF formula is satisfiable cannot be done faster than exhaustive search over all possible assignments. This hypothesis and its variants gave rise to a fruitful field of research, fine-grained complexity, obtaining (mostly tight) lower bounds for many problems in P whose unconditional lower bounds are very likely beyond current techniques. In this chapter, we introduce an extensive framework of Quantum Strong Exponential-Time Hypotheses, as quantum analogues to what SETH is for classical computation.

Using the QSETH framework, we are able to translate quantum query lower bounds on black-box problems to conditional quantum time lower bounds for many problems in P. As an example, we provide a conditional quantum time lower bound of $n^{1.5-o(1)}$ for the Longest Common Subsequence and Edit Distance problems. We also show that the $n^{2-o(1)}$ SETH-based lower bound for a recent scheme for Proofs of Useful Work carries over to the quantum setting using our framework, maintaining a quadratic gap between the verifier and prover.

Lastly, we show that the assumptions in our framework cannot be simplified further with relativizing proof techniques, as they are false in relativized worlds.

This chapter is based on the following paper:

[BPS21] Harry Buhrman, Subhasree Patro, Florian Speelman. A Framework of Quantum Strong Exponential-Time Hypotheses. In *Proceedings of the 38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021)*.

3.1 Introduction

There is a rich diversity of computational problems that are solvable in polynomial time; some have surprisingly fast algorithms, such as the computation of Fourier transforms or solving linear programs, and some for which the worst-case run time has not improved much for many decades. Of the latter category Edit Distance is a good example: this is a problem with high practical relevance, and an $O(n^2)$ algorithm using dynamic programming has been known for many decades. Even after considerable effort, no algorithm has been found that can solve this problem essentially faster than n^2. The best-known algorithms run in $O(n^2/\log^2 n)$ time [MP80], still a nearly quadratic run time.

Traditionally, the field of (structural) complexity theory has studied the time complexity of problems in a relatively coarse manner — the class P, of problems solvable in polynomial time, is one of the central objects of study in complexity theory.

Consider CNF-SAT, the problem of whether a formula, input in conjunctive normal form, has a satisfying assignment. What can complexity theory tell us about how hard it is to solve this problem? For CNF-SAT, the notion of NP-completeness gives a convincing reason why it is hard to find a polynomial-time algorithm for this problem: if such an algorithm is found, all problems in the complexity class NP are also solvable in polynomial time, showing $\mathsf{P} = \mathsf{NP}$.

Not only is no polynomial-time algorithm known, but (if the clause length is arbitrarily large) no significant speed-up over the brute-force method of trying all 2^n assignments are known. Impagliazzo, Paturi, and, Zane [IP01; IPZ01] studied two ways in which this can be conjectured to be optimal. The first of which is called the *Exponential-Time Hypothesis* (ETH).

Conjecture 3.1 (Exponential-Time Hypothesis)**.** There exists a constant $\alpha > 0$ such that CNF-SAT on n variables and m clauses cannot be solved in time $O(m2^{\alpha n})$ by a (classical) Turing machine.

This conjecture can be directly used to give lower bounds for many natural NP-complete problems, showing that if ETH holds then these problems also require exponential time to solve. The second conjecture, most importantly for the current chapter, is the *Strong Exponential-Time Hypothesis* (SETH).

Conjecture 3.2 (Strong Exponential-Time Hypothesis)**.** There does not exist $\delta > 0$ such that CNF-SAT on n variables and m clauses can be solved in $O(m2^{n(1-\delta)})$ time by a (classical) Turing machine.

The strong exponential-time hypothesis also directly implies many interesting exponential lower bounds within NP, giving structure to problems within the complexity class. A wide range of problems (even outside of just NP-complete problems) can be shown to require strong exponential time assuming SETH: for instance, recent work shows that, conditioned on SETH, classical computers require exponential time for *strong simulation* of several models of quantum computation [HNS18; MT19].

Surprisingly, SETH is not only a very productive tool for studying the hardness of problems that likely require exponential time, but can also be used to study the difficulty of solving problems within P, forming a foundation for the field of *fine-grained complexity*. The first of such a SETH-based lower bound was given in

[Wil05], via a reduction from CNF-SAT to the Orthogonal Vectors problem, showing that a truly subquadratic algorithm that can find a pair of orthogonal vectors among two lists would render SETH false.

The Orthogonal Vectors problem became one of the central starting points for proving SETH-based lower bounds, and conditional lower bounds for problems such as computing the Frechet distance between two curves [Bri14], sequence comparison problems such as the string alignment problem [AVW14] and Dynamic Time Warping [ABV15], can all obtained via a reduction from Orthogonal Vectors. Both the Longest Common Subsequence (LCS) and the Edit Distance problems [BI18] can also be shown to require quadratic time conditional on SETH, implying that any super-logarithmic improvements over the classic simple dynamic programming algorithm would also imply better algorithms for satisfiability — a barrier which helps explain why it has been hard to find any new algorithms for these problems.

All these results give evidence for the hardness of problems relative to classical computation, but interestingly SETH does not hold relative to *quantum* computation. Using Grover's algorithm [Gro96; BV97], quantum computers are able to solve CNF-SAT (and more general circuit satisfiability problems) in time $2^{n/2}$, a quadratic speedup relative to the limit that SETH conjectures for classical computation.

Even though this is in violation of the SETH bound, it is not in contradiction to the concept behind the strong exponential-time hypothesis: the input formula is still being treated as a black box, and the quantum speedup comes 'merely' from the general quadratic improvement in unstructured search.[1]

It could therefore be natural to formulate the quantum exponential time hypothesis as identical to its classical equivalent, but with an included quadratic speedup, as a 'basic QSETH'. For some problems, such as Orthogonal Vectors, this conjecture would already give tight results, since these problems are themselves amenable to a speedup using Grover's algorithm. See for instance the Masters thesis [Ren19] for an overview of some of the SETH-based lower bounds that are violated in the quantum setting.

On the other hand, since the conditional lower bound for all problems are a quadratic factor lower than before, such a 'basic QSETH' lower bound for LCS or Edit Distance would be merely linear. The best currently-known quantum algorithm that computes edit distance takes quadratic time, so we would lose some of the explanatory usefulness of SETH in this translation to the quantum case.

In this chapter, we present a way around this limit. Realise that while finding a single marked element is quadratically faster for a quantum algorithm, there is no quantum speedup for many other similar problems. For instance, computing whether the number of marked elements is odd or even cannot be done faster when allowing quantum queries to the input, relative to allowing only classical queries [BBC+01; FGG+98].

Taking the LCS problem again as an illustrative example, after careful inspection of the reductions from CNF-SAT to LCS [ABV15], we show that the result of such a reduction encodes more than merely the existence of an a satisfying assignment. Instead, the result of these reductions also encodes whether *many* satisfying

[1] For unstructured search this bound is tight [BBB+97; BBH+98]. Bennett, Bernstein, Brassard, and Vazirani additionally show that with probability 1 relative to a random oracle all of NP cannot be solved by a bounded-error quantum algorithm in time $o(2^{n/2})$.

assignments exist (in a certain pattern), a problem that could be harder for quantum computers than unstructured search. The 'basic QSETH' is not able to account for this distinction, and therefore does not directly help with explaining why a linear-time quantum algorithm for LCS has not been found.

We present a framework of conjectures, that together form an analogue of the strong exponential-time hypothesis: QSETH. In this framework, we account for the complexity of computing various properties on the set of satisfying assignments, giving conjectured quantum time lower bounds for variants of the satisfiability problem that range from $2^{n/2}$ up to 2^n.

Summary of results presented in this chapter

- We define the QSETH framework, connecting quantum query complexity to the proving of fine-grained (conditional) lower bounds of quantum algorithms. The framework encompasses both different properties of the set of satisfying assignments, and is also able to handle different input circuit classes – giving a hierarchy of assumptions that encode satisfiability on CNF formulas, general formulas, branching programs, and so on.
  - To be able to handle more-complicated properties of the satisfying assignments, we require such a property to be *compression oblivious* – a notion we define to capture the cases where query complexity is a lower bound for the time complexity, even for inputs that are 'compressible' as a truth table of a small formula.[2] We give various results to initiate the study of the set of compression-oblivious languages.
- Some SETH-based $\Omega(T)$ lower bounds carry over to $\Omega(\sqrt{T})$ QSETH lower bounds, from which we immediately gain structural insight to the complexity class BQP.
- We show that, assuming QSETH, the *Proofs of Useful Work* of Ball, Rosen, Sabin and Vasudevan [BRS+18] require time $n^{2-o(1)}$ to solve on a quantum computer, matching the classical complexity of these proofs of work.
- We prove that the Longest Common Subsequence (and the Edit Distance) problem requires $n^{1.5-o(1)}$ time to solve on a quantum computer, conditioned on QSETH. We do this by showing that LCS (similarly, edit distance) can be used to compute a harder property of the set of satisfying assignments than merely deciding whether one satisfying assignment exists.

  Following [AHV+16], we are able to show this for a version of QSETH where the input formulas are *branching programs* instead, giving a stronger result than assuming the hardness for only CNF inputs.
- As a corollary to the proof of the conditional LCS lower bound, we can show that the query complexity of the restricted Dyck language is linear for any $k = \omega(\log n)$, partially answering an open question posed by Aaronson, Grier, and Schaeffer [AGS19].[3]

[2] This notion is conceptually related to the Black-Box Hypothesis introduced by [BGI+12] and studied by [IKK+17].

[3] Lower bounds for the restricted Dyck language were recently independently proven by Ambainis, Balodis, Iraids, Khadiev, Klevickis, Prūsis, Shen, Smotrovs and Vihrovs [ABI+20].

Related work Independently from the work presented in this chapter, Aaronson, Chia, Lin, Wang, and Zhang [ACL+20] recently also defined a basic quantum version of the strong exponential-time hypothesis, which assumes that a quadratic speed-up over the classical SETH is optimal. They present conditional quantum lower bounds for the orthogonal vectors problem, the closest pair problem, and the bichromatic closest pair problem, by giving fine-grained quantum reductions from CNF-SAT. All such lower bounds have a quadratic gap with the corresponding classical SETH lower bound.

Despite the overlap in topic, these results turn out to be complementary to the current work: in the current chapter we focus on defining a more extensive framework for QSETH that generalises in various ways the basic version. Our more general framework can exhibit a quantum-classical gap that is less than quadratic, which allows us to give conditional lower bounds for LCS and Edit Distance ($n^{1.5-o(1)}$) and useful proofs of work (a quadratic gap between prover and verifier). For our presented applications, the requirements of the fine-grained reductions are lower, e.g., when presenting a lower bound of $n^{1.5-o(1)}$ for LCS or Edit Distance it is no problem if the reduction itself takes time $\widetilde{O}(n)$. Conversely, we do not give the reductions that are given by [ACL+20]; those results are distinct new consequences of QSETH (both of the QSETH that is presented in that work, and of our more extensive QSETH framework).

Structure of this chapter In Section 3.2 we motivate and state the QSETH framework. Following that, in Section 3.3 we present the direct consequences of QSETH, including the maintaining of some current bounds (with a quadratic loss), and the Useful Proof of Work lower bound. In Section 3.4 we present conditional lower bounds for LCS and the Edit Distance problem. The lower bound to the restricted Dyck language we get as a corollary to the proof in Section 3.5.1. Finally, we conclude and present several open questions in Section 3.6.

3.2 The quantum strong exponential-time hypotheses

As we briefly discussed in Chapter 1, almost all known lower bounds for quantum algorithms are defined in terms of *query* complexity, which measures the number of times any quantum algorithm must access the input to solve an instance of a given problem.

Despite the success of quantum query complexity and the fact that we know tight query lower bounds for many problems, the query model does not take into account the computational efforts required after querying the input. In particular, it is not possible to use query complexity to prove any lower bound greater than linear, since any problem is solvable in the query-complexity model after all bits are queried. In general we expect the time needed to solve most problems to be much larger than the number of queries required for the computation, but it still seems rather difficult to formalise methods to provide unconditional quantum time lower bounds for explicit problems. We overcome these difficulties by providing a framework of conjectures that can assist in obtaining *conditional* quantum time lower bounds for many problems in BQP. We refer to this framework as the QSETH framework.

Variants of the classical SETH The Strong Exponential-Time Hypothesis (SETH) was first studied in [IP01; IPZ01], who showed that the lack of a $O(2^{n(1-\delta)})$ for a $\delta > 0$ algorithm to solve CNF-SAT is deeply connected to other open problems in complexity theory. Despite it being one the most extensively studied problems in the field of (classical) complexity theory, the best known classical algorithms for solving k-SAT run in $2^{n-n/O(k)}m^{O(1)}$ time [PPS+05], while the best algorithm for the more-general CNF-SAT is $2^{n-n/O(\log \Delta)}m^{O(1)}$ [CIP06], where m denotes the number of clauses and $\Delta = m/n$ denotes the clause to variable ratio.

Even though no refutation of SETH has been found yet, it is plausible that the CNF structure of the input formulas does allow for a speed-up. Therefore, if possible, it is preferable to base lower bounds on the hardness of more general kinds of (satisfiability) problems, where the input consists of wider classes of circuits. For example, lower bounds based on NC-SETH, satisfiability with NC-circuits as input, have been proven for LCS, Edit Distance and other problems [AHV+16], in particular all the problems that fit the framework presented in [BK15].[4]

Additionally, a different direction in which the exponential-time hypothesis can be weakened, and thereby made more plausible, is requiring the computation of different properties of a formula than whether at least one satisfying assignment exists. For example, hardness of *counting* the number of satisfying assignments is captured by #ETH [DHM+14]. Computing existence is equivalent to computing the OR of the set of satisfying assignments, but it could also conceivably be harder to output, e.g., whether the number of satisfying assignments is odd or even, or whether the number of satisfying assignments is larger than some threshold. In the quantum case, generalising the properties to be computed is not only a way to make the hypothesis more plausible: for many of such tasks it is likely that the quadratic quantum speedup, as given by Grover's algorithm, no longer exists.

3.2.1 The basic QSETH

To build towards our framework, first consider what would be a natural generalisation of the classical SETH.

Conjecture (Basic QSETH)**.** There is no bounded error quantum algorithm that solves CNF-SAT on n variables, m clauses in $O(2^{\frac{n}{2}(1-\delta)}m^{O(1)})$ time, for any $\delta > 0$.

This conjecture is already a possible useful tool in proving conditional quantum lower bounds, as we present an example of this in Section 3.3.1.[5]

We first extend this conjecture with the option to consider wider classes of circuits. Let γ denote a class of representations of computational models. Such a representation can for example be polynomial-size CNF formulas, polylog-depth circuits NC, polynomial-size branching programs BP, or the set of all polynomial-size circuits. The complexity of the latter problem is also often studied in the classical case, capturing the hardness of CircuitSAT.

Conjecture (Basic γ-QSETH)**.** A quantum algorithm cannot, given an input C from the set γ, decide in time $O(2^{\frac{n}{2}(1-\delta)})$ whether there exists an input $x \in \{0,1\}^n$ such that $C(x) = 1$ for any $\delta > 0$.

[4] NC circuits are of polynomial size and poly-logarithmic depth consisting of fan-in 2 gates.

[5] Additional examples of implications from such a version of QSETH can be found in the recent independent work of [ACL+20].

We also define AC^0_2 to be the set of all depth-2 circuits consisting of unbounded fan-in, consisting only of AND and OR gates. This definition is later convenient when considering wider classes of properties, and it can be easily seen that 'basic AC^0_2-QSETH' is precisely the 'basic QSETH' as defined above.

Since both these basic QSETH variants already contain a quadratic speedup relative to the classical SETH, conditional quantum lower bounds obtained via these assumptions will usually also be quadratically worse than any corresponding classical lower bounds for the same problems. For some problems, lower bounds obtained using the basic QSETH, or using γ-QSETH for a wider class of computation, will be tight. However, for other problems no quadratic quantum speedup is known.

3.2.2 Extending QSETH to general properties

We now extend the 'basic γ-QSETH' as defined in the previous section, to also include computing different properties of the set of satisfying assignments. By extending QSETH in this way, we can potentially circumvent the quadratic gap between quantum and classical lower bounds for some problems.

Consider a problem in which one is given some circuit representation of a Boolean function $f : \{0,1\}^n \to \{0,1\}$ and asked whether a property $\mathrm{P} : \{0,1\}^{2^n} \to \{0,1\}$ on the truth table of this function evaluates to 1, that is, given a circuit C the problem is to decide if $\mathrm{P}(\mathrm{tt}(\mathrm{C})) = 1$, where tt(C) denotes the truth table of the Boolean function computed by the circuit C. If one can only access C as a black box then it is clear that the amount of time taken to compute P(tt(C)) is lower bounded by the number of queries made to the string tt(C). However, if provided with the description of C, which we denote by desc(C), then one can analyse C to compute P(tt(C)) possibly much faster.

For example, take the representation to be polynomial-sized CNF formulas and the property to be OR. Then for polynomial-sized CNF formulas this is precisely the CNF-SAT problem. Conjecturing quantum hardness of this property would make us retrieve the 'basic QSETH' of the previous section. Do note that we cannot simply conjecture that any property is hard to compute on CNF formulas: Even though the query complexity of AND on a string of length 2^n is $\Omega(2^n)$ classically and $\Omega(2^{n/2})$ in the quantum case, this property can be easily computed in polynomial time both classically and quantumly when provided with the description of the $n^{O(1)}$ sized CNF formula.

To get around this problem, we can increase the complexity of the input representation: If we consider inputs from AC^0_2, the set of all depth-2 circuits consisting of unbounded fan-in AND and OR gates, we now have a class that is closed under complementation. For this class, it is a reasonable conjecture that both AND, the question whether the input is a tautology and all assignments are satisfying, and OR, the normal SAT problem, are hard to compute.

After this step we can look at further properties than AND and OR. For instance, consider the problem of computing whether there exists an even or an odd number of satisfying assignments. This task is equivalent to computing the PARITY of the truth table of the input formula. How much time do we expect a quantum algorithm to need for such a task?

The quadratic speedup for computing satisfiability, i.e., the OR of the truth table of the input formula, is already captured by the model where the quantum computation only tries possible assignments and then performs Grover's algorithm in

a black box manner. If PARITY is also computed in such a way, then we know from query complexity [BBC+01] that there is no speedup possible, and the algorithm will have to use $\Omega(2^n)$ steps. Our QSETH framework will be able to consider more complicated properties, like PARITY.

Finally, observe that such a correspondence, i.e., between the query complexity of a property and the time complexity of computing this property on the set of satisfying assignments, cannot hold for *all* properties, even when we consider more complicated input classes besides CNF formulas. For instance, consider a property which is 0 on exactly the strings that are truth tables of polynomial-sized circuits, and is PARITY of its input on the other strings. Such a property has high quantum query complexity, but is trivial to compute when given a polynomial-sized circuit as input. We introduce the notion of *compression oblivious* below to handle this problem.

White box and black box computation of a property We formalise the above intuitions in the following way. Let the variable γ denote a class of representation at least as complex as the set AC^0_2, where AC^0_2 denotes the set of poly sized depth-2 circuits consisting of only OR, AND gates of unbounded fan-in and NOT gates. For every n, let $\mathrm{P} : \{0,1\}^{2^n} \to \{0,1\}$ be some function family which defines a property. We define a meta-language L_{P} such that $L_{\mathrm{P}} = \{\mathrm{desc(C)} \mid \mathrm{C}$ is an element from the set γ and $\mathrm{P(tt(C))} = 1\}$. We now define the following terms:

Definition 3.3 (White-box algorithms)**.** An algorithm A decides the property P in **white-box** if A decides the corresponding meta-language L_{P}. That is, given an input string desc(C), A accepts if and only if $\mathrm{P(tt(C))} = 1$. We use $\mathrm{qTimeWB}_\epsilon(\mathrm{P})$ to denote the time taken by a quantum computer to decide the language L_{P} with error probability ϵ.

Definition 3.4 (Black-box algorithms)**.** An algorithm A decides the property P in **black-box** if the algorithm $\mathrm{A}^f(1^n, 1^m)$ accepts if and only if $\mathrm{P(tt}(f)) = 1$. Here, f is the Boolean function computed by the circuit C and m is the upper bound on $|\mathrm{desc(C)}|$ which is the size of the representation[6] that describes f, and A^f denotes that the algorithm A has oracle access to the Boolean function f. We use $\mathrm{qTimeBB}_\epsilon(\mathrm{P})$ to denote the time taken by a quantum computer to compute the property P in the black-box setting with error probability ϵ.

Compression oblivious properties We define the set of *compression oblivious* properties corresponding to γ as the set of properties where the time taken to compute this property in the black-box setting is lower bounded by the quantum query complexity of this property on all strings. Formally,

$$\mathcal{CO}(\gamma) = \{\text{properties P} \mid \mathrm{qTimeBB}_\epsilon(\mathrm{P}|_{\mathrm{S}_\gamma}) \geqslant \Omega(\mathsf{Q}_\epsilon(\mathrm{P})^{1-\alpha}) \text{ for all constants } \alpha > 0\},$$

where $\mathsf{Q}_\epsilon(\mathrm{P})$ denotes the quantum query complexity of the property P in a ϵ-bounded error query model and $\mathrm{S}_\gamma = \{\mathrm{tt(C)} \mid \mathrm{C}$ is an element of the set $\gamma\}$.

[6]For instance a CNF/DNF formula, an NC circuit, or a general circuit.

Defining QSETH For each class of representation γ we now define the corresponding γ-QSETH*, which states that computing any compression-oblivious property P in the *white-box* setting is at least as hard as computing P in the *black-box* setting. More formally, for every class of representation γ, such as the class of depth-2 circuits AC^0_2 or poly-sized circuits of a more complex class, we hypothesise the following:

Conjecture 3.5 (γ-QSETH*)**.** For all properties $\mathrm{P} \in \mathcal{CO}(\gamma)$ and for all constants $\alpha > 0$, we have

$$\mathrm{qTimeWB}_\epsilon(\mathrm{P}\,|_\gamma) \geqslant \Omega(\mathsf{Q}_\epsilon(\mathrm{P})^{1-\alpha}).$$

3.2.3 Observations on Compression Oblivious properties

As the class γ gets more complex, the corresponding γ-QSETH* becomes more credible. The set of compression oblivious properties is an interesting object of study by itself.

First consider some representative examples of whether various natural properties are compression oblivious. Note here that the example property that is not compression oblivious has to be carefully constructed for this to be the case – it is natural to conjecture that for most natural properties the knowledge that the input can be written as the truth table of a small circuit does not help in speeding up the computation.[7]

Example 3.6. The properties AND and OR are in $\mathcal{CO}(\mathsf{AC}^0_2)$: the adversarial set that gives the tight query bound for the property AND (OR) are truth tables of functions that can be represented by $n^{O(1)}$ sized DNF (CNF) formulas. Namely, these are given by the formulas that reject (accept) a single possible input, which can be constructed by using n clauses that each contain a single variable or its negation. Because $\mathsf{Q}_\epsilon(\mathrm{AND}|_{\mathrm{S}_{\mathsf{AC}^0_2}}) = \mathsf{Q}_\epsilon(\mathrm{AND})$ and $\mathrm{qTimeBB}_\epsilon(\mathrm{AND}|_{\mathrm{S}_{\mathsf{AC}^0_2}}) \geqslant \mathsf{Q}_\epsilon(\mathrm{AND}|_{\mathrm{S}_{\mathsf{AC}^0_2}})$, we have $\mathrm{AND} \in \mathcal{CO}(\mathsf{AC}^0_2)$. The same holds for the property OR as well.

Example 3.7. Consider the following property, defined on some string $z \in \{0,1\}^{2^n}$, which we view as the truth table of a formula or circuit:

$$\mathrm{P}_{\text{large-c}}(z) = \mathrm{PARITY}_{2^n}(z) \ \wedge\ [\text{there exists no circuit } C \text{ of size} < 2^{n/100} \text{ s.t. } z = \mathrm{tt}(C).]$$

Because most strings are not a truth table of a small circuit, the query complexity of this property is close to the query complexity of PARITY, i.e., $\mathsf{Q}_\epsilon(\mathrm{P}_{\text{large-c}}) = \Omega(N)$. Nevertheless, the property is always 0 when restricted to truth tables of small circuits, and therefore trivial to compute. Therefore $\mathrm{P}_{\text{large-c}}$ is not compression oblivious for polynomial-sized circuits (or any smaller class of representations).

Example 3.8. Whether PARITY is compression oblivious is unknown: the quantum query complexity of PARITY is $\Omega(N)$. Restricted to inputs which are truth tables of small formulas/circuits, the query complexity is $O(\sqrt{N})$, this is the maximum query complexity for any property when restricted to truth tables of a small circuit class [AIK+04; Kot14]. Conjecturing that PARITY is compression oblivious is natural, and incomparable to (but not necessarily less likely than) the main QSETH statement.

[7] In classical complexity theory, a closely related notion is the Black-Box Hypothesis introduced by [BGI+12] and studied by [IKK+17].

Secondly, we show the following fact about how sets of compression-oblivious properties relate, relative to different computational models.

Fact 3.9. *Given two classes of representations ζ and λ, if $\zeta \subseteq \lambda$ then for every property* P*, we have* $\mathrm{P} \in \mathcal{CO}(\lambda)$ *whenever* $\mathrm{P} \in \mathcal{CO}(\zeta)$.

Proof. If $\zeta \subseteq \lambda$ then also for the corresponding sets of truth tables it holds that $\mathrm{S}_\zeta \subseteq \mathrm{S}_\lambda$. If a property $\mathrm{P} \in \mathcal{CO}(\zeta)$, then $\mathrm{qTimeBB}_\epsilon(\mathrm{P}\,|_{S_\zeta}) \geqslant \Omega(\mathsf{Q}_\epsilon(\mathrm{P})^{1-o(1)})$ also implies $\mathrm{qTimeBB}_\epsilon(\mathrm{P}\,|_{S_\lambda}) \geqslant \mathrm{qTimeBB}_\epsilon(\mathrm{P}\,|_{S_\zeta})$ as S_λ is a superset of S_ζ. Therefore, $\mathrm{P} \in \mathcal{CO}(\lambda)$. □

Given an explicit property P and a class of input representations γ, it would be desirable to unconditionally prove that the property P is *γ-compression oblivious*.[8] This is possible for some simple properties that have query complexity $\Theta(\sqrt{N})$ like OR, corresponding to ordinary satisfiability, and AND. Unfortunately, for more complicated properties, like computing the parity of the number of satisfying assignments, it turns out to be hard to find an unconditional proof that such a property is compression oblivious. The following theorem shows a barrier to finding such an unconditional proof: proving that such a property is compression oblivious implies separating P from PSPACE.

Theorem 3.10. *If there exists a property* P *such that* $\mathsf{Q}_\epsilon(\mathrm{P}) = \widetilde{\omega}(\sqrt{N})$ *and* P *is γ-compression oblivious, and* $\mathrm{P} \in \mathsf{polyL}(N)$*, then* $\mathsf{P} \neq \mathsf{PSPACE}$*. Here* $N = 2^n$ *and γ represents the set of poly-sized circuits on n input variables.*

Here $\mathsf{polyL}(N)$ is same as $\mathsf{SPACE}(\mathsf{polylog}N)$, i.e., class of properties computable in $\mathsf{polylog}N$ amount of space. Note that SETH is already a much stronger assumption than $\mathsf{P} \neq \mathsf{PSPACE}$, therefore this observation leaves open the interesting possibility of proving that properties are compression oblivious assuming that the (Q)SETH holds for simpler properties. (For instance, these simpler properties could include OR and AND, for which it is possible to unconditionally prove that they are compression oblivious.)

Proof. By way of contradiction, assume $\mathsf{P} = \mathsf{PSPACE}$. We are given a promise that the circuit C to which we have black-box access[9] to is in the set γ, where γ is the set of poly-sized circuits on n input variables. Note that if we would have direct access to the input, instead of black-box access, we can easily solve the problem in polynomial time using the assumption $\mathsf{P} = \mathsf{PSPACE}$.

Using a simplified version of the algorithm for the oracle identification problem [AIK+04; Kot14] we can extract a compressed form of the entire input, effectively going from black-box access back to white-box access, from the set γ using only $\widetilde{O}(\sqrt{N})$ queries. The initial query-efficient algorithm is as follows:

1. Define an $N = 2^n$ bit majority string $m = m_1 m_2 ... m_N$ where $m_i = 1$ if the majority of circuits in γ have 1 in their i^{th} bit of their truth table, else $m_i = 0$.

2. Check whether there exists an index j such that the truth table of circuit C disagrees with m at j. Using Grover's algorithm on the implied string $\mathrm{tt}(C) \oplus m$ this can be achieved using $O(\sqrt{N})$ quantum queries to $\mathrm{tt}(C)$.

[8] We call a property P a γ-compression oblivious property if $\mathrm{P} \in \mathcal{CO}(\gamma)$.

[9] By black-box access we mean that for any input $x \in \{0,1\}^n$ we can compute $C(x)$.

If there is no disagreement, then the string m is the truth table of circuit C and without having to further query C, one can go through all the circuits in γ and compute their respective truth tables to identify C. Using the $\mathsf{P} = \mathsf{PSPACE}$ assumption, this can be done in $\mathsf{poly}(n)$ (classical) time.

3. In the case of a disagreement, remove from γ all the circuits that disagreed with $\mathrm{tt}(C)$ at index j, which, by definition of m, means at least half of the elements from γ are removed.

Repeat these steps until there is no disagreement or until $|\gamma| = 1$. Given that γ initially contained all the poly-sized circuits on n input variables. This whole algorithm requires $O(\sqrt{N}\log|\gamma|) = \widetilde{O}(\sqrt{N})$ quantum queries. Using the $\mathsf{P} = \mathsf{PSPACE}$ assumption, we can implement the same algorithm in $\widetilde{O}(\sqrt{N})$ quantum time as follows.

At any point of the algorithm we have to be able to query the index $i \in [N]$ of $\mathrm{tt}(C)$ and the i^{th} bit of the majority string m at that stage, where the majority string keeps changing every time we update the set γ. Querying any index of $\mathrm{tt}(C)$ is straight forward. On the other hand, the string m is too long to efficiently write down, but will have to be defined implicitly. To enable query access to m, the algorithm will maintain a list of tuples recording previous found positions where the truth table of C differed from the most common values: $\{(i, a_i) \mid i \in [N]$ is the index where there was a disagreement and a_i is the value of the i^{th} bit of $\mathrm{tt}(C)\}$. Now, given such a list, it takes $\mathsf{poly}(n)$ space to compute the current value m_i of the majority string at point i: simply iterate over all elements in the original circuit class up to $\mathsf{poly}(n)$ size, check whether the current circuit D is consistent with the list of previous queries, and then keep tally of $D(i)$. Now we can use the $\mathsf{P} = \mathsf{PSPACE}$ assumption to translate this to a hypothetical algorithm which takes $\mathsf{poly}(n)$ time.

Since $O(\sqrt{N})$ queries suffice to find a single disagreement between $\mathrm{tt}(C)$ and the majority string m at any stage, that means a disagreement (if any) can be found in $\widetilde{O}(\sqrt{N})$ quantum time. Given that there are only $\mathsf{poly}(n)$ such stages, that means we have found the compressed form of circuit C from the set of poly-sized circuits in $\widetilde{O}(\sqrt{N})$ time.

We now have the access to the compressed input of length $n^{O(1)}$. As the property $\mathrm{P} \in \mathsf{polyL}(N)$, we can directly compute P in $O((\log N)^{O(1)}) = O(n^{O(1)})$ amount of space, which again translates to $O((\log N)^{O(1)})$ time under the $\mathsf{P} = \mathsf{PSPACE}$ assumption. Therefore, the total number of (quantum) steps taken is $\widetilde{O}(\sqrt{N}) + O((\log N)^{O(1)})$, which is in contradiction to the assumption that P is γ-compression oblivious. □

Unfortunately, merely making such an assumption alone will likely not be enough to enable an easy proof that simple properties with high query complexity are compression oblivious: we show that there exists an oracle such that, if all computations and input models[10] have access to this oracle, QSETH is true but PARITY (for example) is not compression oblivious. This does give a relativization barrier to this question, showing that a non-relativizing proof will be necessary to prove that properties are compression oblivious.

[10]For example, consider circuit SAT for circuits that have access to an oracle.

Theorem 3.11. *There exists an oracle relative to which the basic QSETH holds, but any property $P \in \mathsf{polyL}(N)$ for which $\mathsf{Q}_\epsilon(\mathrm{P}) = \widetilde{\omega}(\sqrt{N})$ is not γ-compression oblivious. Here γ consists of all polynomial-sized circuits (with oracle access).*

Proof. We construct the oracle in two steps. We first start with the Quantified Boolean Formula (QBF) problem as oracle, call this oracle A. Since QBF is complete for PSPACE, and since a call to A can itself be simulated in polynomial space, note that $\mathsf{P}^A = \mathsf{BQP}^A = \mathsf{PSPACE}^A$.

Recall the classic oracle from Baker, Gill, and Solovay [BGS75], relative to which $\mathsf{P} \neq \mathsf{NP}$. This construction occasionally hides a single string of a certain length in the oracle, for a very sparse set of lengths, and shows that it is hard for a Turing machine to find the string in time less than 2^n.

This same construction also works for quantum computation: we will construct the oracle B in steps. Take the i-th oracle quantum Turing machine, with access to oracle A, and consider that it makes at most $o(2^{n_i/2})$ queries when given input 1^{n_i}, where $n_i = 2^{n_{i-1}}$. We aim to construct B such that the language

$$L_B = \{1^n \mid \text{The oracle } B \text{ contains a string of length } n\}$$

cannot be decided by such a machine. Through lower bounds for unstructured search [BBB+97; BBH+98; Amb02; BBC+01], there has to exist a single-string setting of the oracle at B that makes the i-th machine fail. I.e., either B has a single string of length n_i, or the oracle is empty at n_i. Via the query lower bound of unstructured search, this language requires $2^{n/2}$ quantum time.

The final oracle C is just the direct sum of the oracle A and B:

$$C = \{(i, x) \mid (i = 0 \wedge x \in A) \vee (i = 1 \wedge x \in B)\}.$$

Relative to C, both SETH, as in Conjecture 3.2, and the basic QSETH are true (where we consider a relativized 'basic QSETH' that takes as input circuits which can make oracle queries to C). In particular, satisfiability of the circuit which queries its input to C and outputs the result takes time $2^{n/2}$ to compute for a quantum Turing machine which has oracle access to C (since any hypothetical machine which solves this language faster, would be able to decide the hard language L_B).

Now consider the hardness of computing some property P of a string, for which we only get black box access to this string, and such that it's known that the string is a truth table of a polynomial-sized circuit which has access to oracle C. A quantum computer can first search the part of C that corresponds with B for the hidden string, using Grover's algorithm for unstructured search, taking time $2^{n/2}$. Now, after finding the hidden string, part B of the oracle is no longer relevant since any call to it can be efficiently simulated by a short computation, and therefore the oracle is effectively only a QBF oracle, meaning that after finding the string we effectively have $\mathsf{P} = \mathsf{PSPACE}$ relative to the oracle. The quantum algorithm can next use the A part, using the construction in Theorem 3.10, to compute the property P in total time $O^*(2^{n/2}) = \widetilde{O}(\sqrt{N})$. Since we assumed that P has query complexity at least $\widetilde{\omega}(\sqrt{N})$, it follows that P is not compression oblivious relative to the oracle. □

3.3 QSETH lower bounds for OV and uPoW

Recall that AC^0_2 denotes the set of polynomial-sized depth-2 circuits consisting of only OR and AND gates of unbounded fan-in. Because of the simple input structure,

the AC^0_2-QSETH* conjecture is therefore closest to the classical SETH, and implies the 'basic QSETH' as introduced in Section 3.2.1:

Corollary 3.12. *If* AC^0_2-QSETH* *is true then there is no bounded error quantum algorithm that solves* CNF-SAT *on* n *variables,* m *clauses in* $O(2^{(1-\delta)n/2}m^{O(1)})$ *time, for any constant* $\delta > 0$.

Proof. Consider the property OR: $\{0,1\}^{2^n} \to \{0,1\}$. Using the fact that $\mathrm{OR} \in \mathcal{CO}(\mathsf{AC}^0_2)$ we get $\mathrm{qTimeWB}_\epsilon(\mathrm{OR}|_{\mathsf{AC}^0_2}) \geqslant \Omega(\mathsf{Q}_\epsilon(\mathrm{OR})^{1-o(1)}) = \Omega(2^{\frac{n}{2}(1-o(1))})$. Due to the structure of the DNF formulas one can compute the property OR on DNF formulas on n variables, m clauses in $n^{O(1)}m^{O(1)}$ time. This implies that the hard cases in the set AC^0_2 for the OR property are the CNF formulas. Therefore, $\mathrm{qTimeWB}_\epsilon(\mathrm{OR}|_{\mathrm{CNF}}) \geqslant \Omega(2^{\frac{n}{2}(1-o(1))})$ where the set CNF denotes all the polynomial-sized CNF formulas. □

In this section we present several immediate consequences of the AC^0_2-QSETH* conjecture:

1. For some problems, classical SETH-based $\Omega(T)$ time lower bounds carry over to the quantum case, with AC^0_2-QSETH*-based $\Omega(\sqrt{T})$ quantum time lower bounds using (almost) the same reduction.

2. The *Proofs of Useful Work* of Ball, Rosen, Sabin and Vasudevan [BRS+18] require time $n^{2-o(1)}$ to solve on a quantum computer, equal to their classical complexity, under AC^0_2-QSETH*.

3.3.1 Quantum time lower bounds based on AC^0_2-QSETH*

The statement of AC^0_2-QSETH* along with Corollary 3.12 can give quantum time lower bounds for some problems for which we know classical lower bounds under SETH (Conjecture 3.2).

Corollary 3.13. *Let* P *be a problem with an* $\Omega(T)$ *time lower bound modulo* SETH. *Then,* P *has an* $\widetilde{\Omega}(\sqrt{T})$ *quantum time lower bound conditioned under* AC^0_2-QSETH* *if there exists a classical reduction from* CNF-SAT *to the problem* P *taking* $O(2^{\frac{n}{2}(1-\alpha)})$ *(for* $\alpha > 0$*) time or if there exists an efficient reduction that can access a single bit of the reduction output.*[11]

We will now explain how we can preserve the following two classical SETH lower bounds, with a quadratic gap:

Example 3.14. The Orthogonal Vectors (OV) problem is defined as follows. Given two sets U and V of N vectors, each over $\{0,1\}^d$ where $d = \omega(\log N)$, determine whether there exists a $u \in U$ and a $v \in V$ such that $\Sigma_{l\in[d]}u_l v_l = 0$. In [Wil05], Williams showed that SETH implies the non-existence of a sub-quadratic classical algorithm for the OV problem. In the quantum case the best-known query lower bound is $\Omega(n^{2/3})$, which can be achieved by reducing the 2-to-1 Collision problem

[11] Note that we use a version of QSETH that relates to CNF-SAT as opposed to bounded clause-size k-SAT problems. One could also define a quantum hardness conjecture for k-CNF or k-DNF, for an arbitrary constant k, in the same way as the original SETH. This variant is required for reductions that use the fact that k is constant, which can occur through usage of the sparsification lemma [IP01]. For examples where this is necessary within fine-grained complexity, see the *Matching Triangles* problem mentioned in [AVY18] or reductions like in [CDL+16].

to the Orthogonal Vectors problem; however, the known quantum time upper bound is $\widetilde{O}(n)$ [Ren19]. First note that we cannot use Williams' classical reduction directly, since a hypothetical quantum algorithm for OV expects quantum access to the input, and writing down the entire reduction already takes time $2^{n/2}$. Instead, observe that the reduction produces a separate vector for each partial assignment: let $t(n)$ be the time needed to compute a single element of the output of the reduction, then $t(n) = \mathsf{poly}(n)$, which is logarithmic in the size of the total reduction. Let $N = O^*(2^{n/2})$ be the size of the output of the reduction of [Wil05], for some CNF formula with n variables. Any quantum algorithm that solves OV in time N^α, can solve CNF-SAT in time $t(n)O^*(2^{\alpha n/2}) = O^*(2^{\alpha n/2})$. Assuming AC_2^0-QSETH*, this implies that a quantum algorithm requires $N^{1-o(1)}$ time to solve OV for instances of size N.[12]

Example 3.15. The Longest Common Subsequence (LCS) problem is defined as follows. Given two strings a and b over an alphabet set Σ, the $\mathrm{LCS}(a, b)$ is the length of the longest subsequence common to both strings a and b. A reduction by [ABV15] shows that if LCS of two strings of length $O(N)$ can be computed in time $O(N^{2-\delta})$ for some constant $\delta > 0$, then satisfiability on CNF formulas with n variables and m clauses can be computed in $O(m^{O(1)} \cdot 2^{(1-\frac{\delta}{2})n})$ which would imply that SETH (Conjecture 3.2) is false.

Just like in the OV case, we observe that the classical reduction from CNF-SAT to LCS is local, in the sense that accessing a single bit of the exponentially-long reduction output can be done in polynomial time: every segment of the strings that are an output of the reduction, depends only on a single partial satisfying assignment, out of the $2^{n/2}$ possible partial assignments.

This observation directly lets us use the reduction of [ABV15] to give a quantum time lower bound of $N^{1-o(1)}$ for the LCS problem, where N here is the length of the inputs to LCS, conditioned on AC_2^0-QSETH*. However, an unconditional quantum query lower bound of $\Omega(N)$ can also be easily achieved by embedding of a problem with high query complexity, such as the majority problem, in an LCS instance.

We witness that with the AC_2^0-QSETH* conjecture, the SETH-based fine-grained lower bounds at best transfer to a square root lower complexity in the quantum case. This is definitely interesting on its own, but we are aiming for larger quantum lower bounds, in situations where the gap between the classical and quantum complexities is less than quadratic, which is why we focus on our more general framework.

3.3.2 Quantum Proofs of Useful Work

Other applications of AC_2^0-QSETH* include providing problems for which *Proofs of Useful Work (uPoW)* can be presented in the quantum setting. Ball et al. [BRS+18] propose uPoW protocols that are based on delegating the evaluation of low-degree polynomials to the prover. They present a classical uPoW protocol for OV whose security proof is based on the assumption that OV needs $n^{2-o(1)}$ classical time in the worst case setting, implying that the evaluation of a polynomial that encodes the instance of OV has average-case hardness. At the end of this protocol, the verifier is able to compute the number of orthogonal vectors in a given instance.

[12] See the results by Aaronson, Chia, Lin, Wang, and Zhang [ACL+20] for more examples of reductions from (a variant of) QSETH, that also hold for the basic QSETH of our framework.

Therefore, the same protocol also works to verify the solutions to ⊕OV, where ⊕OV denotes the parity version of OV, i.e., given two sets U, V of n vectors from $\{0,1\}^d$ each, output the parity of number of pairs (u, v) such that $u \in U$, $v \in V$ and $\Sigma_{l\in[d]} u_l v_l = 0$, where d is taken to be $\omega(\log n)$. Assuming AC^0_2-QSETH* and assuming PARITY $\in \mathcal{CO}(\mathsf{AC}^0_2)$ we get that ⊕CNF-SAT takes $\Omega(2^{n(1-o(1))})$ quantum time. Due to the classical reduction[13] given by [Wil05], this protocol then implies a conditional quantum time lower bound of $n^{2-o(1)}$ for the ⊕OV problem. Therefore, the uPoW protocol by [BRS+18] also requires quantum provers to take time $n^{2-o(1)}$.

3.4 QSETH lower bounds for LCS and Edit Distance

In this section we discuss two consequences of the NC-QSETH* conjecture: quantum time lower bounds for the LCS and Edit Distance problems. For length n input strings, the well-known Wagner–Fischer algorithm (based on dynamic programming) classically computes the edit distance in $O(n^2)$ time. A similar algorithm computes LCS in $O(n^2)$ time. Unfortunately, all the best known classical (and quantum) algorithms to compute these problems are also nearly quadratic. As mentioned above, results by [ABV15; BI18] prove that these near-quadratic time bounds might be tight: a sub-quadratic classical algorithm for computing LCS or edit distance would imply that SETH (Conjecture 3.2) is false.

SETH also implies quadratic lower bounds for many other string comparison problems, like Dynamic Time Warping and Frechet Distance, that also have (close to) optimal algorithms that are based on dynamic programming [BK15]. Bouroujeni et al. [BEG+18] give a sub-quadratic quantum algorithm for approximating edit distance within a constant factor which was followed by a better classical algorithm by Chakraborty et al. [CDG+18]. However, no quantum improvements over the classical algorithms in the exact case are known to the best of our knowledge. Investigating why this is the case is an interesting open problem: is it possible to prove better (conditional) lower bounds, or can a better algorithm be found? We formulate the following questions for the example of LCS and the Edit Distance problem.

1. Is there a bounded error quantum algorithm for LCS or Edit Distance that runs in a sub-quadratic amount of time?

2. Is it possible to obtain a superlinear lower bound for LCS or Edit Distance using the 'basic QSETH'?

3. Can we use a different reduction to raise the linear lower bound for LCS or Edit Distance that we achieve under 'basic-QSETH'?

We don't attempt to find a better algorithm for these string problems in this thesis, and it remains possible that no sub-quadratic quantum algorithm for these problems exists. Considering the second question: using the basic QSETH we lose a quadratic factor relative to the classical reduction, so it is clear that it will not be possible to directly translate a classical reduction to the quantum setting – since the quadratic classical SETH bound is tight. Therefore, to prove a 'basic QSETH'

[13] Note that here one can use the classical reduction from CNF-SAT to Orthogonal Vectors that runs in time $\widetilde{O}(2^{n/2})$.

lower bound for a problem where the gap between the best quantum and classical algorithms is less than quadratic, a fundamentally different (inherently quantum) reduction strategy would have to be found.

While the first two questions still remain open, we address the last question in this section. Using (a promise version of) the NC-QSETH* conjecture we prove conditional quantum time lower bounds of $n^{1.5-o(1)}$ for the LCS and Edit Distance problems.[14] Note that, polynomial-size NC circuit can be expressed as a branching program of length $2^{\mathsf{polylog}n}$ and constant width [Bar89].

Therefore, as a global strategy, we analyse reductions [AHV+16] from branching program (Definition 3.16) satisfiability to string problems, and show that solving the string problems (such as LCS) on the result of these (slightly modified) reductions can be used to compute a more complicated property of the branching program, namely the $\mathrm{PP}_{\mathrm{lcs}}$ property (Definition 3.21). The first step then is to give a reduction from BP-$\mathrm{PP}_{\mathrm{lcs}}$, which can be viewed as showing whether or not $\mathrm{PP}_{\mathrm{lcs}}$ on a branching program is satisfied or not, to LCS. This is given as Theorem 3.28, for which the proof is presented in Section 3.4.3.

Theorem (Informal restatement of Theorem 3.28)**.** *There is a reduction from BP-PP_{lcs} on non-deterministic branching programs of size $2^{\mathsf{polylog}n}$ (length Z, width W) to an instance of the* LCS *problem on two sequences of length $M = 2^{n/2}(cW)^{O(\log Z)}$ for some constant c, and the reduction runs in $O(M)$ time.*

We then prove a quantum query complexity lower bound for this property (Corollary 3.59 in Section 3.5), which, together with the assumption that the property is compression oblivious,[15] implies a time lower bound for the LCS problem of $n^{1.5-o(1)}$. The lower bound strategy for the Edit Distance problem is very similar to that of the LCS problem: the 'gadgets' involved have to be constructed in a different way, but these gadgets can then be combined using a very similar method. Therefore, the reduction can be utilised to compute a property of the set of satisfying assignments that is closely related to BP-$\mathrm{PP}_{\mathrm{lcs}}$.

3.4.1 Branching Programs versus NC-circuits

Barrington's Theorem states that any fan-in 2, depth d circuit can be converted into an equivalent *non-deterministic branching program* (BP) of width 5 and size 4^d, over the same set of inputs [Bar89]. Therefore, any polynomial-size NC circuit can be expressed as a BP of length $2^{\mathsf{polylog}n}$ and constant width.

Definition 3.16 (Non-deterministic Branching Programs)**.** A non-deterministic branching program is a directed acyclic graph with n input variables $x_1, \ldots, x_n$. It is a Z layered directed graph with each layer having a maximum of W nodes and the edges can only exists between nodes of neighbouring layers L_i and L_{i+1}, $\forall i \in [Z-1]$. Every edge is labelled with a constraint of the form $(x_i = b)$ where

[14] Note that, independently from our results, Ambainis et al. [ABI+20] recently presented a quantum query lower bound of $n^{1.5-o(1)}$ for the Edit Distance problem, for algorithms that use the natural dynamic-programming approach of first reducing Edit Distance to connectivity on a 2D grid. However, that doesn't rule out the possibility of other $\tilde{O}(n^{1.5-\alpha})$ quantum algorithms for the Edit Distance problem, for $\alpha > 0$.

[15] As discussed in Section 3.2.3, such an assumption is natural, implicit when considering more-complicated QSETH variants, and hard to prove unconditionally.

x_i is an input variable and $b \in \{0,1\}$. One of the nodes in the first layer is marked as the start node, and one of the nodes in the last layer is marked as the accept node. An evaluation of a branching program on an input $x_1, \ldots, x_n$ is a path that starts at the start node and non-deterministically follows an edge out of the current node. The branching program accepts the input if and only if the path ends up in the accept node. The size of this non-deterministic branching program is the total number of edges i.e. $O(W^2 Z)$.

With a non-deterministic branching program S on n inputs, we associate the Boolean function $f = [S]$ as the function computed by the branching program S. We use $\mathrm{tt}(S)$ to denote the truth table of the function f computed by the branching program S and, a standard encoding of S as a binary string is denoted by $\mathrm{desc}(S)$.

Satisfiability on branching programs (BP-SAT) is defined as follows: given a branching program S on n Boolean inputs, decide if there exists a satisfying assignment to S. After careful inspection of the reduction from BP-SAT to LCS by [AHV+16], and with a slight modification, we are able to present a reduction that encodes more than merely the existence of a satisfying assignment. Instead, the value of LCS on the result encodes the number of satisfying assignments (under certain structured constraints), which we can view as showing whether or not a particular property $\mathrm{PP}_{\mathrm{lcs}}$ (Definition 3.20 and Definition 3.21) on branching program S is satisfied (BP-$\mathrm{PP}_{\mathrm{lcs}}$). Consequently, we give the reduction from BP-$\mathrm{PP}_{\mathrm{lcs}}$ to LCS. By making the assumption that $\mathrm{PP}_{\mathrm{lcs}}$ is NC-compression oblivious,[16] we are able to show that computing $\mathrm{PP}_{\mathrm{lcs}}$ on super-polynomial size BPs in the white-box setting takes $\Omega(2^{0.75n(1-o(1))})$ quantum time under NC-QSETH*. Which then, because of the reduction, gives us a conditional quantum time lower bound of $n^{1.5-o(1)}$ time for the LCS problem.

3.4.2 LCS and the alignment framework

Formally, the (Weighted) Longest Common Subsequence problem is defined as follows:

Definition 3.17 (WLCS and LCS)**.** Given two sequences a and b over an alphabet set Σ and a weight function $w : \Sigma \to [K]$, the WLCS is the maximum weight of a sequence that appears as a subsequence in both a and b. Whereas $\mathrm{LCS}(a, b)$ is the maximum weight when the weight of each symbol is 1, which is the length of longest subsequence common to both a and b.

One way to visualise the (W)LCS problem is by thinking about the problem in terms of alignments, as also done in, e.g., [BK15]. Let a, b be two sequences, of length n and m respectively, and let $n \geqslant m$. An *alignment* is a set $A = \{(i_1, j_1), \ldots, (i_k, j_k)\}$ with $0 \leqslant k \leqslant m$ such that $1 \leqslant i_1 < \cdots < i_k \leqslant n$ and $1 \leqslant j_1 < \cdots < j_k \leqslant m$. The set $\mathcal{A}_{n,m}$ denotes the set of all alignments over the index sets $[n]$ and $[m]$. The equivalence between alignments and the (W)LCS problem is captured by the following statement:

[16] Note that, as suggested by Theorem 3.10 and Theorem 3.11 in Section 3.2.3, it's not easy in general to prove compression-obliviousness for properties with query complexity at least $\omega(\sqrt{N})$.

Fact 3.18. *Let a and b be two sequences defined over some alphabet Σ and let $w : \Sigma \to [K]$ be the weight function associated with the symbols in Σ. The* WLCS *between a and b is*

$$\mathrm{WLCS}(a,b) = \max_{A \in \mathcal{A}_{n,m}} \sum_{(i,j)\in A} \mathrm{WLCS}(a[i], b[j]),$$

where n,m denote the length of the sequences a,b respectively. The i^{th} symbol of sequence a is denoted by $a[i]$, while $b[j]$ is the j^{th} symbol of sequence b and

$$\mathrm{WLCS}(a[i], b[j]) = \begin{cases} w(a[i]), & \text{if } a[i] = b[j], \\ 0, & \text{otherwise.} \end{cases}$$

Proof. From the definition of WLCS it is clear that the subsequence is present in both sequences a and b. Which means that the characters of this subsequence will appear in the same order in both the sequences. Therefore, it suffices to check all the alignments in $\mathcal{A}_{n,m}$ to compute $\mathrm{WLCS}(a,b)$. □

We chose the alignment framework to visualise the (W)LCS problem, because in this framework the (W)LCS between two strings can be related to the sum of (W)LCS between pairs of *some* symbols from these two strings, a recursive behaviour that we will extensively use in the following results.

In order to have our proofs simplified, we use the following reduction from WLCS to LCS. The reduction provides us with a way to translate a lower bound for WLCS to a lower bound for LCS.

Lemma 3.19 (Lemma 2 of [ABV15])**.** *Given an alphabet set Σ and a weight function $w : \Sigma \to [K]$, there exists a mapping $f : \Sigma^* \to \Sigma^*$ such that, for any two sequences a and b defined over Σ, the following statement holds:*

$$\mathrm{WLCS}(a,b) = \mathrm{LCS}(f(a), f(b)),$$

where, for any string $s \in \Sigma^*$, $f(s) = \bigcirc_{i=1}^{|s|} f(s[i])$ and, for any symbol $l \in \Sigma$, $f(l) = [l^{w(l)}] \in \Sigma^{w(l)}$. This also implies, computing WLCS *of two sequences of length n over Σ with weights $w : \Sigma \to [K]$ can be reduced to computing* LCS *of two sequences of length $O(Kn)$ over Σ.*

3.4.3 Reduction from BP-PP$_{\mathrm{lcs}}$ to LCS

We now present a conditional quantum time lower bound for LCS as one of the first consequences of our NC-QSETH* (Conjecture 3.5 with γ set to NC).

Outline of the reduction The first part of our reduction mimics the approach by [AHV+16]. Given a non-deterministic branching program S with n input variables, we do the following: let $X_1 = \{x_1, x_2, \ldots, x_{n/2}\}$ and $X_2 = \{x_{n/2+1}, x_{n/2+2}, \ldots, x_n\}$ be the first and the last half of the input variables to S, respectively. Let $A = (a_1, a_2, \ldots, a_{2^{n/2}})$ and $B = (b_1, b_2, \ldots, b_{2^{n/2}})$ be two sequences containing all the elements from the set $\{0,1\}^{n/2}$ in the lexicographical order such that every pair $(a,b) \in A \times B$ together forms an input to S. For each set A and B, our reduction constructs two long sequences x and y, such that these sequences are composed of

subsequences (also referred to as *gadgets*) that correspond to elements of A and B, respectively. Having constructed the gadgets and the sequences x, y only slightly different from [AHV+16], we observe that computing $\mathrm{LCS}(x, y)$ is equivalent to computing the property $\mathrm{PP}_{\mathrm{lcs}}$ of the truth table of the branching program S. Therefore, we establish a connection between BP-$\mathrm{PP}_{\mathrm{lcs}}$ and LCS, solvable in exponential time and polynomial time, respectively.

Defining $\mathrm{PP}_{\mathrm{lcs}}$ and BP-$\mathrm{PP}_{\mathrm{lcs}}$ Unlike satisfiability, $\mathrm{PP}_{\mathrm{lcs}}$ is not a natural property on branching programs. Instead, $\mathrm{PP}_{\mathrm{lcs}}$ is defined based on the observation that the reduction by [AHV+16] can be used to compute a more complicated property on the truth table of branching programs which (in the black-box setting) is not fully amenable to Grover-like speedup.

We define an intermediate property PP_{δ} instead of directly defining $\mathrm{PP}_{\mathrm{lcs}}$ to avoid redundancy when we define a similar property $\mathrm{PP}_{\mathrm{edit}}$ in Section 3.4.7. The intuition behind the property PP_{δ} is as follows: consider a matrix with 0/1 entries, where one needs to traverse from the first column, starting in a row of choice, to any row in the last column. We call these traversals paths, and the goal is to pick a path which encounters as many 1 entries as possible. Such a path does not have to stay in the same row, but is allowed to make jumps between the rows in a path, at some cost. The value of a path now is given by the sum of the entries of the cells that were encountered, with the jump costs when shifting rows subtracted. The property PP_{δ} on a matrix is determined by whether there exists a path whose value is higher than a given threshold.

Definition 3.20 (The PP_{δ} property)**.** Let M be a Boolean matrix of size $K \times L$ where $\mathrm{M}_{ij} = \{0, 1\}$ denotes the entry in the i^{th} row and the j^{th} column. We define a **path** $\mathrm{R} = ((i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k))$ as a sequence of positions in the matrix M which satisfy the following conditions:

1. The column indices in a path are ordered, i.e., $1 = j_1 \leqslant j_2 \leqslant \cdots \leqslant j_k = L$. This ensures that the path can only start from a cell in the first column and must end in a cell in the last column and the path progresses from left to right in the matrix.

2. For all $p \in [K - 1]$, either $i_{p+1} = i_p$ or $i_{p+1} > i_p$ or $i_{p+1} < i_p$. If $i_{p+1} = i_p$ then $j_{p+1} = j_p + 1$. However, if $i_{p+1} \neq i_p$ then we say there is a jump from (i_p, j_p) to (i_{p+1}, j_{p+1}). The upward jumps are different from the downward jumps in the following way: when jumping to a row above, i.e. when $i_{p+1} < i_p$, then $j_{p+1} = j_p + i_p - i_{p+1}$. Whereas, while jumping to a row below, i.e. when $i_{p+1} > i_p$, then $j_{p+1} = j_p$.

3. Finally, $\forall p$ such that $1 < p < K$, if $i_p \neq i_{p-1}$ then $i_{p+1} = i_p$.

Let $\mathrm{PATHS}_{K,L}$ be a set of all possible *paths* for a matrix of size $K \times L$. The *'value'* associated with a path R for a given matrix M depends on the entries of M and a perturbation vector $\vec{\mu}_R$ and is defined as:

$$\mathbf{V}(\mathrm{M}, \mathrm{R}, \vec{\mu}_R) = \sum_{(i_p, j_p) \in \mathrm{R}} C_{\mathrm{M}_{i_p j_p}} + \sum_{\substack{(i_p, j_p) \in \mathrm{R} \\ i_p \neq i_{p+1}}} \underbrace{(-C_{jump}|i_{p+1} - i_p| - C_{\mathrm{M}_{i_p j_p}} - C_{\mathrm{M}_{i_{p+1} j_{p+1}}})}_{\textit{Jump costs without perturbations}} + \sum_{i \in [|\vec{\mu}_R|]} \vec{\mu}_R[i],$$

where $C_0 < C_1$ are some fixed constants, $C_{jump} = \begin{cases} C_{up} & \text{if } i_{p+1} < i_p, \\ C_{down} & \text{if } i_{p+1} > i_p \end{cases}$, is a fixed jump cost depending whether the jump is to a row above or to a row below, and, $\vec{\mu}_R$ is a list of integer parameters in the range $[Y_1, Y_2]$ whose values will be specified later. The size of the vector $|\vec{\mu}_R|$ is equal to the number of jumps in the path R, but, the values of $\vec{\mu}_R$ will not depend on how far the jump is. We now define:

$$\Delta(\mathrm{M}, \Lambda) = \max_{\mathrm{R} \in \mathrm{PATHS}_{K,L}} \mathbf{V}(\mathrm{M}, \mathrm{R}, \vec{\mu}_R),$$

where Λ denotes the range of integer values that the elements of a perturbation vector $\vec{\mu}_*$ can take, which is $\{Y_1, \ldots, Y_2\}$ unless specified.

Given a fixed threshold value T_r[17], let $H = \{\{i, \ldots, j\} | Y_1 \leqslant i \leqslant j \leqslant Y_2\}$ such that the set H contains all possible ranges, we define the property $\mathrm{P}_\delta : \{0,1\}^{K \times L} \times H \to \{0,1\}$ as follows:

$$\mathrm{P}_\delta(\mathrm{M}, \Lambda) = \begin{cases} 1, & \text{if } (\Delta(\mathrm{M}, \Lambda) \geqslant T_r), \\ 0 & \text{if } (\Delta(\mathrm{M}, \Lambda) < T_r). \end{cases}$$

We now define a promise version of the P_δ property, namely $\mathrm{PP}_\delta : \{0,1\}^{K \times L} \to \{0,1\}$ as follows:

$$\mathrm{PP}_\delta(\mathrm{M}) = \begin{cases} 1, & \text{if } (\mathrm{P}_\delta(\mathrm{M}, \Lambda = \{Y_1\}) = 1), \\ 0, & \text{if } (\mathrm{P}_\delta(\mathrm{M}, \Lambda = \{Y_2\}) = 0). \end{cases}$$

Having formally defined the PP_δ we now adapt the definition to formally define the $\mathrm{PP}_{\mathrm{lcs}}$ property.

Definition 3.21 (The $\mathrm{PP}_{\mathrm{lcs}}$ property)**.** We plug in specific values for the variables $C_0 = Y - 1$, $C_1 = Y$, $C_{up} = 2T$, $C_{down} = T'$, $Y_1 = Y - 1$, $Y_2 = O(Y)$ (where $Y_2 > Y_1$) in Definition 3.20 to define the $\mathrm{PP}_{\mathrm{lcs}}$ property. The exact values of Y, T, T', etc. are all mentioned later in Lemma 3.23, 3.24 and 3.26.

Analogous to BP-SAT which is defined as satisfiability on branching programs, i.e., the OR property on its truth table, we would like to define BP-$\mathrm{PP}_{\mathrm{lcs}}$ as computing the $\mathrm{PP}_{\mathrm{lcs}}$ property on the truth table of the branching programs. Notice that $\mathrm{PP}_{\mathrm{lcs}}$ is a property defined on a Boolean matrix. Therefore we need to encode the truth table of the branching program in a matrix: the correspondence between the entries of the truth table and the entries of the matrix on which the property has to be computed will follow from a careful analysis of the result of the reduction of [AHV+16], which translates a branching program into an instance of LCS.

Definition 3.22 (BP-$\mathrm{PP}_{\mathrm{lcs}}$ problem)**.** Given a non-deterministic branching program S with n input variables, decide if $\mathrm{PP}_{\mathrm{lcs}}(\mathrm{M}^{\mathrm{tt}(S)}) = 1$. Here $\mathrm{tt}(S)$ denotes the truth table of the function computed by the branching program S and $\mathrm{M}^{\mathrm{tt}(S)}$ denotes the Matrix Encoding[18] of $\mathrm{tt}(S)$.

[17] We fix $T_r = \frac{3L}{4}C_0 + \frac{L}{4}C_1$.

[18] Matrix Encoding: let $X = X_1 X_2 \ldots X_{2^n}$ be a binary string of 2^n bits. Then the matrix M^X of size $(2^{n/2+1} - 1) \times 2^{n/2}$ (refer to Figure 3.1) is generated by embedding the entries of X in the following way:

$$\mathrm{M}^X_{ij} = \begin{cases} X_{2^{n/2}(i+j-2^{n/2})+j}, & \text{if } (0 < (i + j - 2^{n/2}) \leqslant 2^{n/2}), \\ 0, & \text{otherwise.} \end{cases}$$

Here M^X_{ij} denotes the entry at the i^{th} row and the j^{th} column of the matrix M^X.

The main reduction We now provide the reduction from the BP-PP_{lcs} promise[19] problem to the LCS problem. The main lemmas and facts pertaining to the reduction are mentioned in this section while the detailed proofs are given in Section 3.4.5. We begin with the construction of [AHV+16] which translates branching programs (with partial assignments) to LCS instances:

Vector gadgets For all $a \in A = (a_1, a_2, \ldots, a_{2^{n/2}})$ and $b \in B = (b_1, b_2, \ldots, b_{2^{n/2}})$ where the sequences A and B contain all the elements from the set $\{0,1\}^{n/2}$ in the lexicographical order, we construct the vector gadgets in the almost following way.

Lemma 3.23 ([AHV+16])**.** *Given a branching program of length Z, width W on n input variables, there exists a construction of vector gadgets $G(a), \overline{G}(b)$ for any given $a, b \in \{0,1\}^{n/2}$ such that:*

$$\text{LCS}(G(a), \overline{G}(b)) \text{ is } \begin{cases} Y, & \text{if } a, b \text{ form a satisfying assignment to the given BP,} \\ \leqslant Y - 1, & \text{otherwise.} \end{cases}$$

Here Y is an integer that depends on the length Z and width W of the BP. Also, the gadgets constructed are over the alphabet set Σ_0, where $|\Sigma_0| = 2$ and the vector gadgets are of size $Z^{O(\log W)}$.[20]

Though it is sufficient for the reduction from BP-SAT to LCS to have vector gadgets of the kind mentioned in Lemma 3.23, for our reduction we need the vector gadgets to behave more predictable. Therefore, we modify the vector gadgets in the following way: for any $a, b \in \{0,1\}^{n/2}$, define

$$\begin{aligned} G'(a) &= 2^{Y-1} \bigcirc G(a), \\ \overline{G}'(b) &= \overline{G}(b) \bigcirc 2^{Y-1}, \end{aligned} \tag{3.1}$$

where the symbol 2 represents a symbol that wasn't originally present in Σ_0. We change the alphabet set to $\Sigma_1 = \Sigma_0 \cup \{2\}$ to include the new symbol. We now make the following claim:

Lemma 3.24. *Given a branching program of length Z, width W on n input variables and given the construction of vector gadgets $G(a), \overline{G}(b)$ from Lemma 3.23, there exists a construction of vector gadgets $G'(a), \overline{G}'(b)$ such that, for any given $a, b \in \{0,1\}^{n/2}$:*

$$\text{LCS}(G'(a), \overline{G}'(b)) = \begin{cases} Y, & \text{if } a, b \text{ form a satisfying assignment to the given BP,} \\ Y - 1, & \text{otherwise.} \end{cases}$$

Here Y is an integer that depends on the length Z and width W of the BP. Also, the gadgets constructed are over the alphabet set Σ_1, where $|\Sigma_1| = 3$ and the size of the vector gadgets are of $(Z^{O(\log W)} + Y - 1)$.

[19] Note that the PP_δ property defined is a promise property, that makes PP_{lcs} also a promise property.

[20] The authors [AHV+16] also give a more efficient construction over an alphabet set of size $|\Sigma_0| = O(W \log Z)$ where the size of the vector gadgets decreases to $Z^{8 \log W}$. However, for the purpose of the current work it suffices to consider the earlier simpler construction.

Proof. Given the alphabet set Σ_1, assume a weight function $w : \Sigma_1 \to \{1, Y-1\}$ associated with it. Where for any $l \in \Sigma_1$,

$$w(l) = \begin{cases} Y-1, & \text{if } l = 2, \\ 1, & \text{otherwise.} \end{cases}$$

Consider the following construction over Σ_1:

$$G''(a) = 2 \bigcirc G(a),$$
$$\overline{G}''(b) = \overline{G}(b) \bigcirc 2.$$

We claim that,

$$\mathrm{WLCS}(G''(a), \overline{G}''(b)) = \begin{cases} Y, & \text{if } a, b \text{ form a satisfying assignment to the given BP,} \\ Y-1, & \text{otherwise.} \end{cases}$$

In an optimal alignment either both the symbols 2 align with each other or they don't. In the case where a, b form a satisfying assignment to the given BP, the symbols 2 will not align and the $\mathrm{WLCS}(G''(a), \overline{G}''(b)) = (\mathrm{W})\mathrm{LCS}(G(a), \overline{G}(b)) = Y$. In the case where a, b don't form a satisfying assignment, then $(\mathrm{W})\mathrm{LCS}(G(a), \overline{G}(b)) \leqslant Y-1$, therefore an optimal alignment would align the 2s, thus $\mathrm{WLCS}(G''(a), \overline{G}''(b)) = Y-1$.

Now using the translation of Lemma 3.19 it directly follows that $\forall a, b \in \{0,1\}^{n/2}$,

$$\mathrm{LCS}(G'(a), \overline{G}'(b)) = \mathrm{WLCS}(G''(a), \overline{G}''(b))$$

where $G'(a)$ and $\overline{G}'(b)$ represent the vector gadgets described in Equation 3.1. □

We have successfully constructed a set of well-behaving vector gadgets of size $O(Z^{O(\log W)})$, each over the alphabet set Σ_1. We now proceed to the next part of our reduction where we construct the two final sequences x and y, and show that by computing $\mathrm{LCS}(x, y)$ we are able to compute non-trivial information about the satisfying assignments to the given BP.

Final sequences x and y Let $A = (a_1, a_2, ..., a_{2^{n/2}})$ and $B = (b_1, b_2, ..., b_{2^{n/2}})$ be two sequences containing all the elements from the set $\{0,1\}^{n/2}$ in lexicographical order. The final sequences are constructed in the following way: for every $a \in A$, we construct $G'(a)$, and, similarly for every $b \in B$, we construct $\overline{G}'(b)$ as mentioned in Lemma 3.24. All the vector gadgets corresponding to the set A are grouped together, with extra separators between them, and also padded with dummy gadgets to form the sequence x. Meanwhile, the vector gadgets corresponding to the set B are grouped together to form the sequence y.

$$\begin{aligned} x := \overbrace{(\textstyle\bigcirc_{i=1}^{|A|-1} 5^T r 6^T 7^{T'})}^{x_1} \overbrace{(\textstyle\bigcirc_{a \in A} 5^T G'(a) 6^T 7^{T'})}^{x_2} \overbrace{(\textstyle\bigcirc_{i=1}^{|A|-1} 5^T r 6^T 7^{T'})}^{x_3} \\ y := \underbrace{7^{|x|}}_{y_1} \underbrace{(\textstyle\bigcirc_{b \in B} 5^T \overline{G}'(b) 6^T)}_{y_2} \underbrace{7^{|x|}}_{y_3} \end{aligned} \tag{3.2}$$

Here $T = (c_1 W)^{O(\log Z)} > T' = (c_2 W)^{O(\log Z)}$, for constants c_1,c_2 and r denotes the dummy vector gadget such that $\mathrm{LCS}(r, \overline{G}'(b)) = Y-1$ for all $b \in B$. We refer to [AHV+16] for constructions of these dummy (also known as normalised-vector) gadgets.

Coarse alignment and LCS We will now present the connection between BP-PP$_{\text{lcs}}$ and LCS. But prior to that, we provide some more lemmas and definitions to support our result. In Section 3.4.2 we saw that LCS of two strings can be associated with maximum *alignment value* when viewed as an alignment of symbols. Unfortunately, we cannot use the alignment framework in the same way at a gadget level. Therefore, instead of using the *alignment* framework, which by definition is a set of pairs such that pairing is between two indices, we define a variant, which we call the *coarse alignment*, which is a sequence of pairs, but the pairing here can be between an index and a sequence of indices (or vice-versa). We observe that LCS(x, y) can be expressed as the *value* of an optimal *coarse alignment* of gadgets in x and y.

Definition 3.25 (Coarse alignment)**.** Let $n', n'' \in [n]$ and $n' < n''$, we say $\mathcal{I}_{n',n''} = \{n', n'+1, \ldots, n''\}$ and let $\mathcal{J}_m = [m]$ be two sets of indices.
A *coarse alignment* A is defined as a sequence $((\mathcal{P}_1, \mathcal{Q}_1), (\mathcal{P}_2, \mathcal{Q}_2), \ldots, (\mathcal{P}_k, \mathcal{Q}_k))$, such that:

1. $\forall i \in [k], \mathcal{P}_i$ are sequences and $\bigcup_{i=1}^{k} \mathcal{P}_i = \mathcal{I}_{n',n''}$. Similarly, $\forall i \in [k], \mathcal{Q}_i$ are also sequences and $\bigcup_{i=1}^{k} \mathcal{Q}_i = \mathcal{J}_m$.
2. $\forall i \in [k], \mathcal{P}_i \neq \varnothing$ and $\forall i \in [k], \mathcal{Q}_i \neq \varnothing$.
3. $\forall i, j \in [k]$, if $i \neq j, \mathcal{P}_i \cap \mathcal{P}_j = \varnothing$. Similarly, $\mathcal{Q}_i \cap \mathcal{Q}_j = \varnothing$, whenever $i \neq j$.
4. $\forall i, j \in [k]$, $\forall u \in \mathcal{P}_i$ and $\forall v \in \mathcal{P}_j$, $u < v$ if $i < j$. Similarly, $\forall u \in \mathcal{Q}_i$ and $\forall v \in \mathcal{Q}_j$, $u < v$ whenever $i < j$.
5. $\forall i \in [k]$, $|\mathcal{P}_i| = 1$ or $|\mathcal{Q}_i| = 1$, or both.

Let the set $\mathcal{A}_{n',n'',m}$ denote the set of coarse alignments given the indices sets $\mathcal{I}_{n',n''}$ and $\mathcal{J}_m$. We define the set $\mathcal{C}_{n,m} = \bigcup_{i,j \in [n], i<j} \mathcal{A}_{i,j,m}$ to denote the set of all possible coarse alignments given n and m.

Examples of coarse-alignments: a few examples of a coarse-alignments from the set $\mathcal{C}_{5,3}$ are $C_0 = ((1,2,3),1),(4,2),(5,3))$, $C_1 = ((2,1),(3,2),(4,3))$, $C_2 = ((2,1),(3,(2,3)))$.

We notice that the structure of the sequences x and y in Equation 3.2, i.e., the padding of 5s and 6s (and $7s$ for x) between the gadgets and the padding of the 7s in the beginning and end of the sequence y, ensures that coarse alignments at the gadget level are very useful in describing LCS(x, y):

Lemma 3.26. *Given the two sequences x and y from Equation 3.2, there exists a coarse alignment* $\mathrm{C} \in \mathcal{C}_{(3\cdot 2^{n/2}-2),2^{n/2}}$*, such that* LCS *of the two sequences x and y is:*

$$\mathrm{LCS}(x,y) = \max_{\mathrm{C} \in \mathcal{C}_{(3\cdot 2^{n/2}-2),2^{n/2}}} \underbrace{(\Sigma_{(\mathcal{K},\mathcal{L})\in \mathrm{C}} \mathrm{LCS}(u_{\mathcal{K}}, v_{\mathcal{L}}) + T'(3\cdot 2^{n/2} - f_{\mathcal{P}}(\mathrm{C}) - 1))}_{\text{LCS-value } \textit{of coarse alignment } \mathrm{C}}$$

where $u_{\mathcal{K}} = \bigcirc_{p\in\mathcal{K}} 5^T g_p 6^T 7^{T'}$ *and* $v_{\mathcal{L}} = \bigcirc_{q\in\mathcal{L}} 5^T \overline{G}'(b_q) 6^T$*. Also* $g_p = G'(a_{p-2^{n/2}})$ *when* $0 < (p - 2^{n/2}) \leqslant 2^{n/2}$ *and* $g_p = r$ *otherwise. Here r denotes the dummy gadget. And, for a coarse alignment* $\mathrm{C} = ((\mathcal{P}_1, \mathcal{Q}_1), ..., (\mathcal{P}_k, \mathcal{Q}_k))$*, we define* $f_{\mathcal{P}}(C) = \max(\mathcal{P}_k) - \min(\mathcal{P}_1) + 1$.

Proof of Lemma 3.26 is in 3.4.5.

Main theorem of this section We now present the reduction from the BP-$\mathrm{PP}_{\mathrm{lcs}}$ *promise*[21] problem to the LCS problem. The main lemmas and facts pertaining to the reduction are stated in this section while their detailed proofs are given in Section 3.4.5.

Definition 3.27 (The sets $\mathcal{S}$ and $\mathcal{V}$)**.** Let the set $\mathcal{S}$ denote the set of non-deterministic branching programs (or circuits, depending on the usage context) with n input variables such that:

$$\mathcal{S} = \{S \mid \mathrm{M}^{\mathrm{tt}(S)} \in {\mathrm{PP}_{\mathrm{lcs}}}^{-1}(0) \cup {\mathrm{PP}_{\mathrm{lcs}}}^{-1}(1)\} \text{ and, } \mathcal{V} = \{\mathrm{M}^{\mathrm{tt}(S)} \mid S \in \mathcal{S}\}.$$

This states that $\mathcal{S}$ contains the compressed form of the inputs on which $\mathrm{PP}_{\mathrm{lcs}}$ is defined, while $\mathcal{V}$ contains the matrices that are Matrix Encoding (Figure 3.1) of truth tables of the elements in $\mathcal{S}$.

Theorem 3.28. *There is a reduction from BP-PP_{lcs} on non-deterministic branching programs of size $2^{\mathsf{polylog}n}$ (length Z, width W) from set $\mathcal{S}$ to an instance of the* LCS *problem on two sequences x, y as given in Equation 3.2 of length $M = 2^{n/2}(cW)^{O(\log Z)}$ for some constant c, and the reduction runs in $O(M)$ (quantum) time.*

Proof. The reduction is as follows: let $S \in \mathcal{S}$ be a branching program with n input variables of size $2^{\mathsf{polylog}n}$. Let $X_1 = \{x_1, x_2, \ldots, x_{n/2}\}$ and $X_2 = \{x_{n/2+1}, x_{n/2+2}, \ldots, x_n\}$ be the first and the last half of the input variables to S, respectively. Let $A = (a_1, a_2, \ldots, a_{2^{n/2}})$ and $B = (b_1, b_2, \ldots, b_{2^{n/2}})$ be two sequences containing all the elements from the set $\{0,1\}^{n/2}$ in the lexicographical order such that every pair $(a, b) \in A \times B$ together forms an input to S. Construct the *vector gadgets* as mentioned in Lemma 3.24, and then construct the final sequences x, y as in Equation 3.2. We now show the correctness of our reduction using the following lemma.

Lemma 3.29. *For every n, there exists a constant $C^* \in \mathbb{Z}$ such that*

$$\mathrm{LCS}(x, y) \geqslant C^*$$

if and only if $\mathrm{PP}_{\mathrm{lcs}}(\mathrm{M}^{\mathrm{tt}(S)}) = 1$.

Proof. Consider the sequences x, y in Equation 3.2, then consider the alignment (let's say A) where the vector gadgets from subsequence x_2 (in x) aligns with the corresponding vector gadgets from subsequence y_2 (in y). Let $N = 2^{n/2}$ and recall that $T > T' > Y$, clearly the

$$\mathrm{LCS}(x, y) \geqslant 2TN + k(Y-1) + (N-k)Y + T'(2N-1), \tag{3.3}$$

where k refers to the number of non-satisfying assignments to the given branching program under the alignment A.

Invoking the results of Lemma 3.26 about $\mathrm{LCS}(x, y)$ being related to LCS of subsequences under a coarse-alignment $\mathrm{C} = ((\mathcal{K}_1, \mathcal{L}_1), ..., (\mathcal{K}_t, \mathcal{L}_t)) \in \mathcal{C}_{3N-2,N}$ we can say that

$$\mathrm{LCS}(x, y) \geqslant 2T|\,\mathrm{C}\,| + k_1(Y-1) + k_2 Y + T'(3N - f_{\mathcal{K}}(\mathrm{C}) - 1) \tag{3.4}$$

[21] The property PP_δ (Definition 3.20) is a promise property and so is $\mathrm{PP}_{\mathrm{lcs}}$ (Definition 3.21).

where $f_{\mathcal{K}}(C) = \max(\mathcal{K}_t) - \min(\mathcal{K}_1) + 1$. While k_1 denotes the number of non-satisfying assignments, k_2 denotes the number of satisfying assignments to the given BP under the coarse alignment C, and, $k_1 + k_2 = |\,\mathrm{C}\,|$.

We cannot yet claim equality of the expressions in Equation 3.4 because we have not accounted for the perturbations of the following kind: consider a single element $(\mathcal{K}, \mathcal{L})$ of a coarse alignment, say of the kind $((1,2,3),(5))$. For any term $(\mathcal{K}, \mathcal{L}) \in \mathrm{C}$ that has either $|\mathcal{K}| > 1$ or $|\mathcal{L}| > 1$, there will be additional perturbations that can range between $(Y-1)$ and $|\overline{G}'(b_*)| = O(Y)$, i.e.,

$$2T + (Y-1) \leqslant \mathrm{LCS}(u_{\mathcal{K}}, v_{\mathcal{L}}) \leqslant 2T + O(Y),$$

which means these perturbations that contribute towards $\mathrm{LCS}(x, y)$ have to be accounted for. Refer to Lemma 3.26 for the notations.

We further notice that this problem can be reduced to computing a specific *promise* property $\mathrm{PP}_{\mathrm{lcs}}$ on the assignments to the given branching program as mentioned in the statement of Lemma 3.29. Refer to Definition 3.21 for the definition of this property. If $\mathrm{PP}_{\mathrm{lcs}}(\mathrm{M}^{\mathrm{tt}(S)}) = 1$ then it implies $\mathrm{P}_{\mathrm{lcs}}(M^{\mathrm{tt}(S)}, Y-1) = 1$ because of the promise that the branching program S belongs to the set $\mathcal{S}$. The expression $\mathrm{P}_{\mathrm{lcs}}(M^{\mathrm{tt}(S)}, Y-1) = 1$ implies the existence of a path $\mathrm{P} \in \mathrm{PATHS}_{2L-1,L}$ (here $L = 2^{n/2}$) such that even with a lowest allowed perturbation parameter $\Lambda = Y - 1$, the value of the path is $\mathbf{V}(\mathrm{M}^{\mathrm{tt}(S)}, \mathrm{P}, \vec{\mu}_{\mathrm{P}}) \geqslant T_r$, for a fixed threshold T_r. Which invariably means that for all values of Λ larger than $Y-1$, $\mathbf{V}(\mathrm{M}^{\mathrm{tt}(S)}, \mathrm{P}, \vec{\mu}_{\mathrm{P}}) \geqslant T_r$. Using Algorithm 1 we can generate a coarse alignment $\mathrm{C} \in \mathcal{C}_{3L-2,L}$ for the given path P. The choice of constants C_0, C_1 as given in Definition 3.21 ensures that $\text{LCS-value}(\mathrm{C}, x, y) \geqslant T_r$. Therefore, we can set our constant mentioned in the statement of Lemma 3.29 to $C^* = T_r$.

We now prove the other direction. If $\mathrm{PP}_{\mathrm{lcs}}(\mathrm{M}^{\mathrm{tt}(S)}) = 0$ then $\mathrm{P}_{\mathrm{lcs}}(M^{\mathrm{tt}(S)}, |\overline{G}'(b_*)|) = 0$ because the branching program $S \in \mathcal{S}$. Which in turn implies that for all values of Λ, for all paths $\mathrm{P} \in \mathrm{PATHS}_{2L-1,L}$, $\mathbf{V}(\mathrm{M}^{\mathrm{tt}(S)}, \mathrm{P}, \vec{\mu}_{\mathrm{P}}) < T_r$. Using Algorithm 1 we can generate coarse alignments for every one of these paths, because of the choice of values for the constants C_0 and C_1 in Definition 3.21 we get that the $\text{LCS-value}(\mathrm{C}, x, y)$ for these coarse alignments is strictly less than T_r. Furthermore, using the result from Lemma 3.54 and 3.55 in 3.4.9 we show that if for all values of $\Lambda, \forall\, \mathrm{P} \in \mathrm{PATHS}_{2L-1,L}$, $\mathbf{V}(\mathrm{M}^{\mathrm{tt}(S)}, \mathrm{P}, \vec{\mu}_{\mathrm{P}}) < T_r$ then $\forall\, \mathrm{C} \in \mathcal{C}_{3L-2,L}$, $\text{LCS-value}(\mathrm{C}, x, y) < T_r$ which implies $\mathrm{LCS}(x, y) < T_r = C^*$. □

The proof of Lemma 3.29 establishes the correctness of our reduction. It is also easy to see that the lengths of the vector gadgets are of $(c_1 W)^{O(\log Z)}$ for some constant c_1, therefore, length of the final sequences x and y are of $2^{n/2}(O(W))^{O(\log Z)}$. This constitutes the proof of the reduction from $\text{BP-PP}_{\mathrm{lcs}}$ on branching programs of size $2^{\mathsf{polylog}n}$ (from the set $\mathcal{S}$) to LCS. □

3.4.4 The quantum time lower bound for LCS

In the previous sub-section we gave a reduction from the $\text{BP-PP}_{\mathrm{lcs}}$ problem on branching programs of size $2^{\mathsf{polylog}n}$ from set $\mathcal{S}$ (Definition 3.27) to the LCS problem. Therefore, if we prove that the time taken to compute the $\mathrm{PP}_{\mathrm{lcs}}$ on these branching programs in the white-box setting in ϵ-bounded error model is $\Omega(2^{0.75n(1-o(1))})$ then, because of the reduction, we prove a conditional quantum time lower bound of

$n^{1.5-o(1)}$ for the LCS problem. Additionally, we also define a set $\mathcal{V}$ as set of matrices that are matrix encodings of 2^n bit strings, as shown in Figure 3.1, on which $\mathrm{PP}_{\mathrm{lcs}}$ is defined.[22]

To achieve the quantum time lower bound for the BP-$\mathrm{PP}_{\mathrm{lcs}}$ problem we use the results of the Theorem 3.30 (query complexity of $\mathrm{PP}_{\mathrm{lcs}}$) and Conjectures 3.31 (promise version of NC-QSETH*) and 3.32 ($\mathrm{PP}_{\mathrm{lcs}} \in \mathcal{CO}(\mathsf{NC} \cap \mathcal{S})$) given below.

Theorem 3.30 (Theorem 3.58, Corollary 3.59 in 3.5)**.** *The bounded-error quantum query complexity for computing the property* $\mathrm{PP}_{\mathrm{lcs}}$ *on matrices of size* $(2^{n/2+1}-1) \times 2^{n/2}$ *from* $\mathcal{V}$ *is* $\Omega(2^{0.75n})$.

Conjecture 3.31 (Promise version of NC-QSETH*)**.** For the class of representations NC, i.e., the set of poly-sized circuits of poly-logarithmic depth consisting of fan-in 2 gates, for all properties $\mathrm{P} \in \mathcal{CO}(\mathsf{NC} \cap \mathcal{S})$, we have $\mathrm{qTimeWB}_{\epsilon}(\mathrm{P}\,|_{\mathsf{NC}\cap\mathcal{S}}) \geqslant \Omega(\mathsf{Q}_{\epsilon}(\mathrm{P}\,|_{\mathcal{V}})^{1-o(1)})$.

Recall that Theorem 3.10 and Theorem 3.11 in Section 3.2.3 show obstacles to prove that properties with query complexity at least $\omega(\sqrt{N})$ are compression-oblivious. On the other hand, the only properties that we know are not compression oblivious are artificial constructions that directly refer to circuit complexity, and it is natural to conjecture that all properties which are simple enough are compression oblivious. The conditional lower bound therefore requires the following assumption.

Conjecture 3.32. The property $\mathrm{PP}_{\mathrm{lcs}}$ is $(\mathsf{NC} \cap \mathcal{S})$-compression oblivious.

We can now prove a conditional quantum time lower bound of $n^{1.5-o(1)}$ time for the LCS problem.

Theorem 3.33. *Assuming Conjecture 3.31, the promise version of* NC-QSETH*, *and assuming* $\mathrm{PP}_{\mathrm{lcs}} \in \mathcal{CO}(\mathsf{NC} \cap \mathcal{S})$*, the bounded-error quantum time complexity for computing the* LCS *problem is atleast* $n^{1.5-o(1)}$.

Proof. Combining the results of Theorem 3.30, Conjectures 3.31 and 3.32 we get $\mathrm{qTimeWB}_{\epsilon}(\mathrm{PP}_{\mathrm{lcs}}\,|_{\mathsf{NC}\cap\mathcal{S}}) = \Omega(2^{0.75n(1-o(1))})$. Which implies that, the bounded error quantum time complexity for computing the property $\mathrm{PP}_{\mathrm{lcs}}$ in the white-box setting is $\mathrm{qTimeWB}_{\epsilon}(\mathrm{PP}_{\mathrm{lcs}}\,|_{\mathsf{NC}\cap\mathcal{S}}) = \Omega(2^{0.75n(1-o(1))})$ under the promise version of NC-QSETH*.

With the use of the reduction from the $\mathrm{PP}_{\mathrm{lcs}}$ problem on branching programs with n input variables of size $2^{\mathsf{polylog}n}$ from set $\mathcal{S}$ to the LCS problem (Theorem 3.28) we obtain the conditional quantum time lower bound of $n^{1.5-o(1)}$ for the LCS problem. □

3.4.5 Proof of Lemma 3.26

We will use the following results in order to prove Lemma 3.26. We take the alphabet set to be $\Sigma_2 = \Sigma_1 \cup \{5,6,7\}$ where $\Sigma_1 \cap \{5,6,7\} = \varnothing$, and, all the vector gadgets (refer to Equation 3.1) are defined over alphabet set Σ_1. The associated weight function[23]

[22] Note that the definition of $\mathcal{V}$ here and in Definition 3.27 are equivalent because any 2^n bit string will be the truth table of some branching program on n input variables.

[23] To make the analysis simpler, we keep oscillating between using LCS and WLCS. That is why we have defined a weight function associated to the alphabet set Σ_2.

$w : \Sigma_2 \to [T]$ is as follows: $w(5) = w(6) = T > w(7) = T' > w(2) = Y - 1$ and the weight of all the other symbols are set to 1.

Fact 3.34 (Fact 5.7 in [BK15])**.** *For any similarity measure δ that admits the alignment framework, and given two sequences x, y defined over some alphabet set Σ. We say, for a given ordered partition of $x = x_1 \bigcirc x_2 \bigcirc x_3$, there exists some ordered partition of y, such that $y = y_1 \bigcirc y_2 \bigcirc y_3$ and,*

$$\delta(x, y) = \delta(x_1, y_1) + \delta(x_2, y_2) + \delta(x_3, y_3).$$

Lemma 3.35. *Given two sequences x and y over the alphabet set Σ_2 and associated weight function $w : \Sigma_2 \to [T]$, such that*

$$x = 5a'_1675a'_2675a'_367...5a'_{t_a}67,$$

$$y = 5b'_165b'_265b'_365b'_46...5b'_{t_b}6,$$

and $\forall i, j, a'_i$ and b'_j represents vector gadgets defined over Σ_1. We claim that there exists a coarse alignment C such that,

$$\mathrm{WLCS}(x, y) = \Sigma_{(\mathcal{K},\mathcal{L})\in \mathrm{C}} \mathrm{WLCS}(u_{\mathcal{K}}, v_{\mathcal{L}}),$$

where $u_{\mathcal{K}} = \bigcirc_{p\in\mathcal{K}} 5a'_p67$ and $v_{\mathcal{L}} = \bigcirc_{q\in\mathcal{L}} 5b'_q6$.

Proof. W.l.o.g. assume $t_a \geqslant t_b$. The first observation we make is that the $7s$ in the sequence x don't contribute to the WLCS as there are no matching $7s$ in the sequence y. Clearly, $\mathrm{WLCS}(x, y) > (w(5) + w(6)) \cdot t_b$ as we can achieve that by aligning the symbols $5s$ (also the $6s$) of the sequences x, y. The weight associated with the symbols 5 and 6 is such that $(w(5) + w(6)) > \max(|a'_*|, |b'_*|)$. Therefore, we can say that there exists at least one subsequence '675' in x that aligns with the subsequence '65' in y in an optimal alignment. The existence of an aligning separator pair ('675' in x and '65' in y) in an optimal alignment lets us make the following statement: $\mathrm{WLCS}(x, y) = \mathrm{WLCS}(x', y') + \mathrm{WLCS}(x'', y'')$, such that x', x'' are both of the form '5...67' and $y', y''$are both of the form '5...6'.

$$x = \overbrace{5a'_167...5a'_{i-1}67}^{x'}\ \overbrace{5a'_i67...5a'_{t_a}67}^{x''},$$
$$y = \underbrace{5b'_16...5b'_{j-1}6}_{y'}\ \underbrace{5b'_j6...5b'_{t_b}6}_{y''},$$

Using this argument recursively we can see that the strings x and y gets partitioned into substrings of the form '5...67' and '5...6' respectively, such that $\mathrm{WLCS}(x, y)$ is the sum of pair wise WLCS of these substrings, where only one of the substrings in each pair contains more than zero separators. Hence, proving the claim of Lemma 3.35. □

Lemma 3.26. *Given the two sequences x and y from Equation 3.2, there exists a coarse alignment $\mathrm{C} \in \mathcal{C}_{(3\cdot 2^{n/2}-2),2^{n/2}}$, such that* LCS *of the two sequences x and y is:*

$$\mathrm{LCS}(x, y) = \max_{\mathrm{C}\in\mathcal{C}_{(3\cdot 2^{n/2}-2),2^{n/2}}} \underbrace{(\Sigma_{(\mathcal{K},\mathcal{L})\in \mathrm{C}} \mathrm{LCS}(u_{\mathcal{K}}, v_{\mathcal{L}}) + T'(3 \cdot 2^{n/2} - f_{\mathcal{P}}(\mathrm{C}) - 1))}_{\text{LCS-value } \textit{of coarse alignment } \mathrm{C}}$$

where $u_{\mathcal{K}} = \bigcirc_{p\in\mathcal{K}} 5^T g_p 6^T 7^{T'}$ *and* $v_{\mathcal{L}} = \bigcirc_{q\in\mathcal{L}} 5^T \overline{G}'(b_q) 6^T$. *Also* $g_p = G'(a_{p-2^{n/2}})$ *when* $0 < (p - 2^{n/2}) \leqslant 2^{n/2}$ *and* $g_p = r$ *otherwise. Here* r *denotes the dummy gadget. And, for a coarse alignment* $\mathrm{C} = ((\mathcal{P}_1, \mathcal{Q}_1), ..., (\mathcal{P}_k, \mathcal{Q}_k))$, *we define* $f_{\mathcal{P}}(C) = \max(\mathcal{P}_k) - \min(\mathcal{P}_1) + 1$.

Proof. Using Lemma 3.19 from Section 3.4.2, we can compute $\mathrm{LCS}(x, y)$ by computing $\mathrm{WLCS}(x', y')$ where,

$$\begin{aligned} x' &:= \overbrace{(\bigcirc_{i=1}^{|A|-1} 5r67)}^{x_1} \overbrace{(\bigcirc_{a\in A} 5G'(a)67)}^{x_2} \overbrace{(\bigcirc_{i=1}^{|A|-1} 5r67)}^{x_3}, \\ y' &:= \underbrace{7^{|x'|}}_{y_1} \underbrace{(\bigcirc_{b\in B} 5\overline{G}'(b)6)}_{y_2} \underbrace{7^{|x'|}}_{y_3} \end{aligned} \tag{3.5}$$

and the associated weight function $w : \Sigma_2 \to [T]$ where $w(5) = w(6) = T > w(7) = T' > w(2) = Y - 1$ and the weight of all the other symbols are set to 1.[24]. Let x' be concatenation of three sequences x_1, x_2, x_3, i.e. $x' = x_1 \bigcirc x_2 \bigcirc x_3$, similarly, let $y' = y_1 \bigcirc y_2 \bigcirc y_3$. Suppose that in an alignment the last 7 from y_1 aligns with the last 7 from x' then no element from y_2 can align with any symbol from x'. The LCS-value of such an alignment would be $(3|A| - 2) \cdot w(7) = (3|A| - 2)T'$. It is easy to see that we can achieve a much better LCS-value by aligning x_2 with y_2 hence, $\mathrm{WLCS}(x', y') > (w(5) + w(6))|A| + w(7)(2|A| - 1) > (3|A| - 2)T'$. Therefore, the earlier mentioned alignment is never optimal. Which means, under this observation and by invoking the results of Fact 3.34 we can safely say that there exists some ordered partition of x' such that $x' = \overline{x}_1 \bigcirc \overline{x}_2 \bigcirc \overline{x}_3$ and $\overline{x}_1$, $\overline{x}_3$ are sequences that start with the symbol 5 and 7 respectively, and end with the symbol 7, where as $\overline{x}_2$ is a sequence that starts with a 5 and ends with a 6, and,

$$\mathrm{WLCS}(x', y') = \mathrm{WLCS}(\overline{x}_1, 7^{|x'|}) + \mathrm{WLCS}(\overline{x}_2, y_2) + \mathrm{WLCS}(\overline{x}_3, 7^{|x'|}). \tag{3.6}$$

Further analysis of $\mathrm{WLCS}(\overline{x}_2, y_2)$ under the claims of Lemma 3.35 suggests that there exists a coarse alignment C, such that $\mathrm{WLCS}(\overline{x}_2, y_2) = \Sigma_{(\mathcal{K},\mathcal{L})\in\mathrm{C}} \mathrm{LCS}(u_{\mathcal{K}}, v_{\mathcal{L}})$, while the $\mathrm{WLCS}(\overline{x}_1, 7^{|x'|})$ and $\mathrm{WLCS}(\overline{x}_3, 7^{|x'|})$ depends on the number of $7s$ left in $\overline{x}_1, \overline{x}_3$, respectively. Therefore,

$$\mathrm{LCS}(x, y) = \mathrm{WLCS}(x', y') = \max_{C\in\mathcal{C}_{(3\cdot 2^{n/2}-2),2^{n/2}}} (\Sigma_{(\mathcal{K},\mathcal{L})\in C} \mathrm{LCS}(u_{\mathcal{K}}, v_{\mathcal{L}}) + (3\cdot 2^{n/2} - 1 - f_{\mathcal{P}}(C))T'),$$

as mentioned in the statement of Lemma 3.26. □

3.4.6 Edit Distance and the alignment framework

The lower bound strategy for the Edit Distance problem is very similar to that of the LCS reduction – the underlying reduction from a single partial assignment to a branching programs to Edit Distance is different, but the overall resulting structure on how these partial assignments are combined is almost identical. Therefore, only the main results and theorem statements are presented in this section while proofs of the theorems are presented in a separate section.

[24]Though the number of $7s$ in the sequences x and x' (similarly, in y and y') don't scale according to the mapping mentioned in Lemma 3.19 in Section 3.4.2, it is easy to see that even then the claim holds.

Definition 3.36 (The EDIT DISTANCE problem)**.** Given two strings a and b over an alphabet set Σ, the edit distance between a and b is the minimum number of operations (insertions, deletions, substitutions) on the symbols required to transform string a to b (or vice versa).

Similar to LCS, the EDIT DISTANCE problem also can be visualised by using the alignment framework as presented in Section 3.4.2 (also mentioned in [BK15]). Let a, b be two strings, of length n and m respectively, and let $n \geqslant m$. An *alignment* is a set $A = \{(i_1, j_1), \ldots, (i_k, j_k)\}$ with $0 \leqslant k \leqslant m$ such that $1 \leqslant i_1 < \cdots < i_k \leqslant n$ and $1 \leqslant j_1 < \cdots < j_k \leqslant m$. The set $\mathcal{A}_{n,m}$ denotes the set of all alignments over the index sets $[n]$ and $[m]$. We now claim the following:

Fact 3.37. *Let a,b be strings defined over alphabet set Σ. The edit distance between a and b is*

$$\text{Edit-Dist}(a,b) = \min_{A \in \mathcal{A}_{n,m}} \Big(\underbrace{\sum_{(i,j)\in A} \text{Edit-Dist}(a[i], b[j]) + n + m - 2|A|}_{\text{EDIT-value } \textit{of alignment } A \textit{ with strings } a \textit{ and } b} \Big),$$

where $n = |a|, m = |b|$, $a[i]$ denotes the i^{th} symbol of string a, while $b[j]$ denotes the j^{th} symbol of string b, and, for all $l, l' \in \Sigma$, $\text{Edit-Dist}(l, l') = \begin{cases} 1, & \textit{if } l \neq l', \\ 0, & \textit{otherwise.} \end{cases}$

Proof. There are many ways to transform the string a into the string b and each alignment $A \in \mathcal{A}_{n,m}$ specifies one such way. Every element in A indicates the positions of the symbols in strings a and b, respectively, that is either matched or substituted. For any alignment $A \in \mathcal{A}_{n,m}$ the EDIT-value(A, a, b) denotes the number of operations (insertions, deletions, substitutions) required to transform a to b under the alignment A. As edit distance is defined to be the *minimum* number of operations required to transform a to b, we minimize the EDIT-value over all the alignments in $\mathcal{A}_{n,m}$ to get Edit-Dist(a, b). □

3.4.7 Reduction from BP-PP$_{\text{edit}}$ to the Edit Distance problem

Another consequence of our NC-QSETH* conjecture is the quantum time lower bound of $n^{1.5-o(1)}$ for the EDIT DISTANCE problem. The proof idea is (almost) identical to that of the result about the LCS problem given in Section 3.4.3 and is (again) as follows: we first define a promise property PP$_{\text{edit}}$ (Definition 3.39). We then give a reduction from the problem of computing the property PP$_{\text{edit}}$ on truth tables of some non-deterministic branching programs to the EDIT DISTANCE problem.

Reduction: let S be a branching program of length Z, width W with n input variables. Let $A = (a_1, a_2, \ldots, a_{2^{n/2}})$ and $B = (b_1, b_2, \ldots, b_{2^{n/2}})$ be two sequences containing all the elements from the set $\{0,1\}^{n/2}$ in the lexicographical order such that every pair $(a, b) \in A \times B$ together forms an input to S. For every $a \in A$, construct vector gadget $G(a)$ and for every $b \in B$, construct vector gadget $\overline{G}(b)$ such that,

$$\text{Edit-Dist}(G(a), \overline{G}(b)) = \begin{cases} Q - \rho, & \text{if } (a,b) \text{ forms a satisfying pair to } S, \\ Q, & \text{otherwise,} \end{cases} \tag{3.7}$$

for some constants Q and ρ.[25] The final sequences x and y are constructed by grouping the vector gadgets corresponding to A and B, respectively, in the following way:

$$\begin{aligned} x &:= (\bigcirc_{i=1}^{|A|-1} 5^T r 6^T)(\bigcirc_{a\in A} 5^T G(a) 6^T)(\bigcirc_{i=1}^{|A|-1} 5^T r 6^T) \\ y &:= 7^{|x|}(\bigcirc_{b\in B} 5^T \overline{G}(b) 6^T) 7^{|x|} \end{aligned} \tag{3.8}$$

Here r is a dummy vector gadget such that Edit-Dist$(r, \overline{G}(b)) = Q, \forall b \in B$ [AHV+16] and 5^T (or 6^T) known as a *separator* represents the symbol 5 (or 6) occurring T times. The choice[26] of T is made in a way that in an *optimal alignment* the separators from x align with the separators in y, hence forcing the vector gadgets from x to align with vector gadgets from y.

As mentioned in Fact 3.37 the edit distance between two strings can be associated with minimum EDIT-value of some alignment. However, similar to the situation of LCS, we cannot use the alignment framework in the same way at a gadget level. Therefore, we resort to the usage of *coarse alignment* in this case as well. We observe that Edit-Dist(x, y) can be expressed as the *edit-cost* of an optimal *coarse alignment* of the vector gadgets as given in Lemma 3.38 below.

Lemma 3.38. *There exists a coarse alignment $C \in \mathcal{C}_{(3\cdot 2^{n/2}-2),2^{n/2}}$, such that the edit distance between the two sequences x and y is:*

$$\delta(x, y) = 2|x| + \min_{C\in\mathcal{C}_{(3\cdot 2^{n/2}-2),2^{n/2}}} \text{edit-cost}(C, x, y),$$

where edit-cost$(C, x, y) = \sum_{(i,j)\in C} \delta(u_i, v_j)$ *such that* $u_i = \bigcirc_{p\in i} 5^T g_p 6^T$ *and* $v_j = \bigcirc_{q\in j} 5^T \overline{G}(b_q) 6^T$. *Also* $g_p = G(a_{p-2^{n/2}})$ *when* $0 < (p - 2^{n/2}) \leqslant 2^{n/2}$ *and* $g_p = r$ *otherwise. Here r denotes the dummy gadget.*

The proof of Lemma 3.38 uses results of Lemma 3.52, 3.53, Fact 3.50 and Corollary 3.51 mentioned in Section 3.4.9.

We now provide some definitions required for our reduction.

Definition 3.39 (The PP_{edit} property)**.** The Edit Distance problem is a minimisation problem when viewed as an optimisation problem, and the properties P_{edit} and PP_{edit} need to be defined accordingly. Therefore, we plug in negative values to the constants $C_0 = -Q$, $C_1 = -(Q-\rho)$, $C_{up} = (2T + S_G) = C_{down}$, $Y_1 = -Q$, $Y_2 = 0$ in Definition 3.20 to define the P_{edit} and PP_{edit} properties. The values of Q, ρ, T, S_G are all mentioned in Theorem 3.42 of Section 3.4.7.

Definition 3.40 (BP-PP_{edit} problem)**.** Given a non-deterministic branching program S with n input variables, decide if $\text{PP}_{\text{edit}}(\text{M}^{\text{tt}(S)}) = 1$. Here tt$(S)$ denotes the truth table of the function computed by the branching program S and $\text{M}^{\text{tt}(S)}$ denotes the Matrix Encoding of tt(S).

[25] We again refer to [AHV+16] for the construction of these vector gadgets. Their vector gadgets are of length $Z^{O(\log W)}$ and are constructed over the alphabet set $\Sigma = \{0, 1\}$. However, their results don't require that the vector gadgets $G(\cdot)$ and $\overline{G}(\cdot)$ are of the same length, but our result does. Therefore, we slightly modify their construction as shown in Lemma 3.49 in 3.4.9. We achieve this for another set of constants and over a bigger alphabet set $\Sigma_0 = \{0, 1, 2\}$.

[26] $T = (cW)^{O(\log Z)}$ for some constant c, and W and Z denotes the width and length, respectively, of the given BP.

Definition 3.41 (The sets $\mathcal{S}$ and $\mathcal{V}$)**.** The set $\mathcal{S}$ denotes the set of non-deterministic branching programs (or circuits, depending on the usage context) with n input variables such that:

$$\mathcal{S} = \{S \mid \mathrm{M}^{\mathrm{tt}(S)} \in {\mathrm{PP}_{\mathrm{edit}}}^{-1}(0) \cup {\mathrm{PP}_{\mathrm{edit}}}^{-1}(1)\} \text{ and, } \mathcal{V} = \{\mathrm{M}^{\mathrm{tt}(S)} \mid S \in \mathcal{S}\}.$$

This states that $\mathcal{S}$ contains the compressed form of the inputs on which $\mathrm{PP}_{\mathrm{edit}}$ is defined, while $\mathcal{V}$ contains the matrices that are Matrix Encoding (Figure 3.1) of truth tables of the elements in $\mathcal{S}$.

Having defined the BP-$\mathrm{PP}_{\mathrm{edit}}$ problem and the set $\mathcal{S}$, we now proceed to prove the main theorem of this section.

Theorem 3.42. *There is a reduction from the BP-PP_{edit} problem on BPs of size $2^{\mathsf{polylog}n}$ (length Z and width W) from set $\mathcal{S}$ to an instance of the* Edit Distance *problem on two sequences of length of order $N = Z^{O(\log W)}2^{n/2}$ over the alphabet set $\Sigma_1 = \{0, 1, 2, 5, 6, 7\}$, and the reduction runs in $O(N)$ time.*

Proof. Let $S \in \mathcal{S}$ be a branching program with n input variables of size $2^{\mathsf{polylog}n}$. Construct the vector gadgets as given in Equation 3.7, and the final sequences x and y as mentioned in Equation 3.8. It is easy to see that the length of the final sequences is $O(Z^{O(\log W)}2^{n/2})$ because each vector gadget is of length $Z^{O(\log W)}$. Clearly, this reduction runs in $O(N)$ time, where $N = Z^{O(\log W)}2^{n/2}$. We now claim the correctness of our reduction using the result from the following lemma.

Lemma 3.43. *For every n, there exists a constant $C^* \in \mathbb{Z}$ such that*

$$\text{Edit-Dist}(x, y) \leqslant C^*$$

if and only if $\mathrm{PP}_{\mathrm{edit}}(\mathrm{M}^{\mathrm{tt}(S)}) = 1$.

Proof. If $\mathrm{PP}_{\mathrm{edit}}(\mathrm{M}^{\mathrm{tt}(S)}) = 1$ then it implies $\mathrm{P}_{\mathrm{edit}}(\mathrm{M}^{\mathrm{tt}(S)}, \Lambda = \{-Q\}) = 1$ because of the promise that the branching program S belongs to the set $\mathcal{S}$. The statement $\mathrm{P}_{\mathrm{edit}}(\mathrm{M}^{\mathrm{tt}(S)}, \Lambda = \{-Q\}) = 1$ implies that there exists a *path* $\mathrm{P} \in \mathrm{PATHS}_{2L-1,L}$ (here $L = 2^{n/2}$) such that even with the lowest allowed perturbation parameter $\Lambda = -Q$ the value of the path is $\mathbf{V}(\mathrm{M}^{\mathrm{tt}(S)}, \mathrm{P}, \vec{\mu}_{\mathrm{P}}) \geqslant T_r$, for a fixed threshold T_r. Which invariably means that for all values of $\Lambda \in [-Q, 0]$, $\mathbf{V}(\mathrm{M}^{\mathrm{tt}(S)}, \mathrm{P}, \vec{\mu}_{\mathrm{P}}) \geqslant T_r$. Using Algorithm 1 that generates a coarse alignment $\mathrm{C} \in \mathcal{C}_{3L-2,L}$ for a given path $\mathrm{P} \in \mathrm{PATHS}_{2L-1,L}$ as an input, we get a coarse alignment C corresponding to path P. Due to the choices of C_0, C_1 made in Definition 3.39 we can claim $\text{edit-cost}(\mathrm{C}) = -\mathbf{V}(\mathrm{M}^{\mathrm{tt}(S)}, \mathrm{P}, \vec{\mu}_{\mathrm{P}})$ under a particular perturbation vector $\vec{\mu}_{\mathrm{P}}$. Therefore, we can say $\text{edit-cost}(\mathrm{C}) \leqslant -T_r$ because $\forall \Lambda, \mathbf{V}(\mathrm{M}^{\mathrm{tt}(S)}, \mathrm{P}, \vec{\mu}_{\mathrm{P}}) \geqslant T_r$. Also, from Lemma 3.38 we have that $\text{Edit-Dist}(x, y) = 2|x| + \min_{\mathrm{C} \in \mathcal{C}_{3L-2,L}} \text{edit-cost}(\mathrm{C})$. This implies $\text{Edit-Dist}(x, y) \leqslant 2|x| - T_r$ and, we set our constant $C^* = 2|x| - T_r$.

We now prove the other direction. If $\mathrm{PP}_{\mathrm{edit}}(\mathrm{M}^{\mathrm{tt}(S)}) = 0$ it implies $\mathrm{P}_{\mathrm{edit}}(\mathrm{M}^{\mathrm{tt}(S)}, \Lambda = \{0\}) = 0$ as the branching program $S \in \mathcal{S}$. Which in turn implies that for all other values of $\Lambda \in [-Q, 0]$, $\forall\, \mathrm{P} \in \mathrm{PATHS}_{2L-1,L}$, $\mathbf{V}(\mathrm{M}^{\mathrm{tt}(S)}, \mathrm{P}, \vec{\mu}_{\mathrm{P}}) < T_r$. Using the result from Lemma 3.54 and Lemma 3.55 in Section 3.4.9 we show that if for all $\Lambda \in [-Q, 0], \forall\, \mathrm{P} \in \mathrm{PATHS}_{2L-1,L}$, $\mathbf{V}(\mathrm{M}^{\mathrm{tt}(S)}, \mathrm{P}, \vec{\mu}_{\mathrm{P}}) < T_r$ then $\forall\, \mathrm{C} \in \mathcal{C}_{3L-2,L}$, $\text{edit-cost}(\mathrm{C}) > -T_r$ which implies $\text{Edit-Dist}(x, y) > 2|x| - T_r$. Thus, implying $\text{Edit-Dist}(x, y) > C^*$. □

This constitutes the proof of the reduction from the BP-$\mathrm{PP}_{\mathrm{edit}}$ problem on branching programs of size $2^{\mathsf{polylog}n}$ from the set $\mathcal{S}$ to the Edit Distance problem. □

3.4.8 The quantum time lower bound for the Edit Distance problem

In the previous subsection we gave a reduction from the BP-PP$_{\text{edit}}$ problem on branching programs of size $2^{\mathsf{polylog}n}$ from set $\mathcal{S}$ to the Edit Distance problem. Therefore, if we prove that the time taken to compute the PP$_{\text{edit}}$ on these branching programs in the white-box setting in ϵ-bounded error model is $\Omega(2^{0.75n})$ then because of the reduction we prove a quantum time lower bound of $\Omega(n^{1.5})$ for the Edit Distance problem. Additionally, we also define a set $\mathcal{V}$ a set of matrices that are matrix encoding of 2^n bit strings as shown in Figure 3.1 on which PP$_{\text{edit}}$ is defined.

To achieve the quantum time lower bound for the BP-PP$_{\text{edit}}$ problem we use the results of the Theorem 3.44 (query complexity of PP$_{\text{edit}}$) and Conjectures 3.45 (promise version of NC-QSETH*) and 3.46 (PP$_{\text{edit}} \in \mathcal{CO}(\mathsf{NC} \cap \mathcal{S})$) mentioned below.

Theorem 3.44 (Theorem 3.58, Corollary 3.59 in Section 3.5)**.** *The bounded-error quantum query complexity for computing the property* PP$_{\text{edit}}$ *on matrices of size* $(2^{n/2+1}-1) \times 2^{n/2}$ *from* $\mathcal{V}$ *is* $\Omega(2^{0.75n})$.

Conjecture 3.45 (Promise version of NC-QSETH*)**.** For the class of representations NC, i.e., the set of poly sized circuits of poly-logarithmic depth consisting of fan-in 2 gates, for all properties $\mathrm{P} \in \mathcal{CO}(\mathsf{NC} \cap \mathcal{S})$, we have $\mathrm{qTimeWB}_\epsilon(\mathrm{P}\,|_{\mathsf{NC}\cap\mathcal{S}}) \geqslant \Omega(\mathsf{Q}_\epsilon(\mathrm{P}\,|_{\mathcal{V}})^{1-o(1)})$.

Conjecture 3.46. The property PP$_{\text{edit}}$ is $(\mathsf{NC} \cap \mathcal{S})$-compression oblivious.

Theorem 3.47. *The bounded error quantum time complexity for computing the property* PP$_{\text{edit}}$ *in the white-box setting is* $\mathrm{qTimeWB}_\epsilon(\mathrm{PP}_{\text{edit}}\,|_{\mathsf{NC}\cap\mathcal{S}}) = \Omega(2^{0.75n(1-o(1))})$ *under a promise version of* NC-QSETH*.

Proof. Combining Theorem 3.44, Conjecture 3.45, and Conjecture 3.46 we get $\mathrm{qTimeWB}_\epsilon(\mathrm{PP}_{\text{edit}}\,|_{\mathsf{NC}\cap\mathcal{S}}) = \Omega(\mathsf{Q}_\epsilon(\mathrm{PP}_{\text{edit}}\,|_{\mathcal{V}})^{1-o(1)}) = \Omega(2^{0.75n(1-o(1))})$. □

Theorem 3.48. *Assuming Conjecture 3.45, the promise version of* NC-QSETH*, *and assuming* PP$_{\text{edit}} \in \mathcal{CO}(\mathsf{NC} \cap \mathcal{S})$*, the bounded-error quantum time complexity for computing the* Edit Distance *problem is* $n^{1.5-o(1)}$.

Proof. Using the reduction from the PP$_{\text{edit}}$ problem on branching programs with n input variables of size $2^{\mathsf{polylog}n}$ from set $\mathcal{S}$ to the Edit Distance problem (Theorem 3.42) and using the results from Theorem 3.47 we obtain the conditional quantum time lower bound of $n^{1.5-o(1)}$ for the Edit Distance problem. □

3.4.9 Proofs of Lemma 3.38 and other relevant theorems

We use this section to present the proof of all the facts, lemmas and theorems used as building blocks to prove our main result.

Lemma 3.49. *Given two strings* $a, b \in \{0,1\}^*$ *with* $|a| > |b|$ *such that, either* $\delta(a,b) = Q'$ *or* $\delta(a,b) = Q' - \rho$ *for some constants* $Q', \rho \in \mathbb{Z}^+$ *and* $\rho < Q'$*, we can create strings* $a_{new}, b_{new} \in \{0,1,2\}^*$ *such that,*

$$a_{new} = 2^{|a|} \bigcirc a,$$
$$b_{new} = 0^{|a|-|b|}2^{|a|} \bigcirc b,$$

and,

$$\delta(a_{new}, b_{new}) = Q - \rho \text{ iff } \delta(a,b) = Q' - \rho,$$
$$\delta(a_{new}, b_{new}) = Q \text{ iff } \delta(a,b) = Q',$$

for another constant $Q \in \mathbb{Z}^+$ *and* $\rho < Q$ *and* $|a_{new}| = |b_{new}|$.

Proof. It follows from the construction of a_{new}, b_{new} that $|a_{new}| = |b_{new}|$. It is also easy to see that $\delta(a_{new}, b_{new}) \leqslant |a| - |b| + \delta(a,b)$. We will now prove that $\delta(a_{new}, b_{new}) = |a| - |b| + \delta(a,b)$ for any such a, b. Consider the last 2 of the subsequence $2^{|a|}$ in the string a_{new}. This symbol 2 could either get deleted, or get matched or get substituted. If this 2 gets substituted with a symbol from the substring b in b_{new} then there is a symbol 2 in b_{new} that has to be inserted, which is not an optimal thing to do. If this symbol 2 from a_{new} gets substituted with a symbol in $0^{|a|-|b|}$ then there is a symbol 2 in b_{new} that has to be substituted with a symbol from the substring a in a_{new} or has to be inserted, either ways not an optimal thing to do. A similar argument holds for the case where this 2 from a_{new} gets deleted. Therefore, the only option left is that it gets matched with a 2 in b_{new}. By repeating this argument for all the 2s in a_{new} we can say that in an optimal alignment all the 2s from a_{new} will align with all the 2s from b_{new}. Therefore, $\delta(a_{new}, b_{new}) = |a| - |b| + \delta(a,b)$ and $Q = Q' + |a| - |b|$. □

Lemma 3.38. *There exists a coarse alignment* $C \in \mathcal{C}_{(3\cdot 2^{n/2}-2),2^{n/2}}$*, such that the edit distance between the two sequences* x *and* y *is:*

$$\delta(x,y) = 2|x| + \min_{C \in \mathcal{C}_{(3\cdot 2^{n/2}-2),2^{n/2}}} \text{edit-cost}(C,x,y),$$

where $\text{edit-cost}(C,x,y) = \sum_{(i,j)\in C} \delta(u_i, v_j)$ *such that* $u_i = \bigcirc_{p\in i} 5^T g_p 6^T$ *and* $v_j = \bigcirc_{q\in j} 5^T \overline{G}(b_q) 6^T$*. Also* $g_p = G(a_{p-2^{n/2}})$ *when* $0 < (p - 2^{n/2}) \leqslant 2^{n/2}$ *and* $g_p = r$ *otherwise. Here* r *denotes the dummy gadget.*

Proof. We are given two sequences x and y, such that

$$x := (\bigcirc_{i=1}^{|A|-1} 5^T r 6^T)(\bigcirc_{a\in A} 5^T G(a) 6^T)(\bigcirc_{i=1}^{|A|-1} 5^T r 6^T),$$
$$y := \underbrace{7^{|x|}}_{y_1} \underbrace{(\bigcirc_{b\in B} 5^T \overline{G}(b) 6^T)}_{y_2} \underbrace{7^{|x|}}_{y_3}. \tag{3.9}$$

Recall that $A = (a_1, a_2, \ldots, a_{2^{n/2}})$ and $B = (b_1, b_2, \ldots, b_{2^{n/2}})$ and both the sequences contain all the elements from the set $\{0,1\}^{n/2}$ in the lexicographical order. The gadgets $r, G(a)$ and $\overline{G}(b) \in \{0,1,2\}^*$ and $T > |G(\cdot)| = |\overline{G}(\cdot)| = |r| = S_G$.

Fact 3.50 (Fact 5.7 in [BK15])**.** *Let* $y_1 = 7^{|x|}$*,* $y_2 = \bigcirc_{b\in B} 5^T \overline{G}(b) 6^T$*,* $y_3 = 7^{|x|}$ *as mentioned in Equation 3.9 above. We claim the following statement: the edit distance between the strings* x *and* y *is,*

$$\delta(x,y) = \min_{x_1,x_2,x_3} (\delta(x_1,y_1) + \delta(x_2,y_2) + \delta(x_3,y_3)),$$

where x_1, x_2, x_3 *ranges over all ordered partitions of* x *and* $x = x_1 \bigcirc x_2 \bigcirc x_3$.

Corollary 3.51 (of Fact 3.50)**.** *The edit distance between the strings x and y is*

$$\delta(x, y) = 2|x| + \min_{x_2} \delta(x_2, y_2),$$

such that x_2 ranges over all ordered partitions of x.

Proof. The strings y_1 and y_3 are strings of length $|x|$ (at least as large as $|x_1|$ and $|x_3|$) and consist of symbols that are not used in the entire string x. Therefore, the $\delta(x_1, y_1) = \delta(x_3, y_3) = |x|$ for any choice of x_1 and x_3. □

Lemma 3.52. *Given the two sequences x and y as mentioned above, there exists a substring x_2 of the form $5^T \dots 6^T$ such that the $\delta(x, y) = 2|x| + \delta(x_2, y_2)$ when $y_2 = \bigcirc_{b \in B} 5^T \overline{G}(b) 6^T$.*

Proof. A proof by contradiction. Note that x_2 can be any substring of x. Let us assume that the (minimum) edit distance is achieved with x_2 that is **not** of the form $5^T \dots 6^T$, we then argue that one could change the format of x_2 to $5^T \dots 6^T$ without increasing the cost.

As we assume that x_2 is not of the form $5^T \dots 6^T$ therefore it could be of any of the following forms:

1. Let's consider a scenario where x_2 is of the form $5^{\theta_w} \dots 6^T$ where $0 < \theta_w < T$, note that y_2 is already of the form $5^T \dots 6^T$.

$$x_2 := \overbrace{5^{\theta_w} a_i 6^T}^{w_1} \overbrace{5^T a_{i+1} 6^T \dots 6^T}^{w_2}$$

$$y_2 := \underbrace{5^T b_1 6^T 5^T b_2 6^T \dots 6^T}_{v_1 \bigcirc v_2}$$

   Recall from Fact 3.50 that by fixing $w_1 = 5^{\theta_w} a_i 6^T$ and $w_2 = 5^T a_{i+1} 6^T \dots 6^T$ we have $\delta(x_2, y_2) = \min_{v_1, v_2} \delta(w_1, v_1) + \delta(w_2, v_2)$.

   (a) Let's assume that the minimum is achieved when $v_1 = 5^T b_1 6^{\theta_v}$, where $0 < \theta_v \leqslant T$. In such a scenario, $\delta(w_1, v_1) = \delta(5^{\theta_w} a_i 6^T, 5^T b_1 6^{\theta_v}) = \delta(a_i 6^{T-\theta_v}, 5^{T-\theta_w} b_1)$. On the other hand we have $\delta(5^T a_i 6^T, v_1) = \delta(a_i 6^{T-\theta_v}, b_1) \leqslant \delta(a_i 6^{T-\theta_v}, 5^{T-\theta_w} b_1) = \delta(w_1, v_1)$.[27] Therefore suggesting that if $v_1 = 5^T b_1 6^{\theta_v}$ then setting $w_1 = 5^T a_i 6^T$ doesn't increase the cost.

   (b) Let's assume that the minimum is achieved when $v_1 = 5^T b_1^{\theta_v}$, where $0 < \theta_v \leqslant S_G$. $\delta(w_1, v_1) = \delta(5^{\theta_w} a_i 6^T, 5^T b_1^{\theta_v}) = \delta(a_i 6^T, 5^{T-\theta_w} b_1^{\theta_v}) \geqslant \delta(a_i 6^T, b_1^{\theta_v}) = \delta(5^T a_i 6^T, v_1)$. Again suggesting that if $v_1 = 5^T b_1^{\theta_v}$ then setting $w_1 = 5^T a_i 6^T$ cannot increase the cost.

   (c) Let's assume that the minimum is achieved when $v_1 = 5^{\theta_v}$, where $0 \leqslant \theta_v \leqslant T$. $\delta(w_1, v_1) = \delta(5^{\theta_w} a_i 6^T, 5^{\theta_v}) = \max(T + S_G, T + S_G + \theta_w - \theta_v) > \delta(\varnothing, 5^{\theta_v})$. There by suggesting that if $v_1 = 5^{\theta_v}$ then setting $w_1 = \varnothing$ will definitely cost less.

[27] Consider three strings $s_1 \in \Sigma^*$, and $s_2, s_3 \in \Gamma^*$, where Σ and Γ are two disjoint alphabet sets, i.e. $\Sigma \cap \Gamma = \varnothing$, then $\delta(s_1 \bigcirc s_2, s_3) \geqslant \delta(s_2, s_3)$ (similarly, $\delta(s_2, s_1 \bigcirc s_3) \geqslant \delta(s_2, s_3)$). As the symbols in the string s_1 are different from symbols in s_2 and s_3 and the operations on symbols from s_1 will only be *delete* or *substitute*. Therefore, one can get rid of the string s_1 by removing the symbols that got deleted (hence, reducing the cost) and by inserting the symbols in s_3 that otherwise would have been substituted by the symbols from s_1 (hence, maintaining the cost).

This proves that no matter what form v_1[28] is of, the minimum cost is achieved when $w_1 = 5^T a_i 6^T$ or when $w_1 = \varnothing$, therefore, supporting the claim of Lemma 3.52. We use the same argument symmetrically to prove that x_2 cannot be of the form $5^T \dots 6^{\theta_w}$, where $0 < \theta_w < T$ or of the form $5^{\theta_{w_1}} \dots 6^{\theta_{w_2}}$ for $0 < \theta_{w_1}, \theta_{w_2} < T$.

2. Consider the case where x_2 is of the form $a_i^{\theta_w} \dots 6^T$ where $0 < \theta_w \leqslant S_G$ and y_2 is of the form $5^T \dots 6^T$.

$$x_2 := \overbrace{a_i^{\theta_w} 6^T}^{w_1} \overbrace{5^T a_{i+1} 6^T \dots 6^T}^{w_2}$$

$$y_2 := \underbrace{5^T b_1 6^T 5^T b_2 6^T \dots 6^T}_{v_1 \bigcirc v_2}$$

   (a) Let's assume that the minimum is achieved when $v_1 = 5^T b_1 6^{\theta_v}$, where $0 < \theta_v \leqslant T$. Then, $\delta(w_1, v_1) = \delta(a_i^{\theta_w} 6^T, 5^T b_1 6^{\theta_v}) = \delta(a_i^{\theta_w} 6^{T-\theta_v}, 5^T b_1)$. The last 5 of the substring $5^T b_1$ could either be substituted for a 6 in $a_i^{\theta_w} 6^{T-\theta_v}$ or could be substituted for a $\{0, 1, 2\}$ in $a_i^{\theta_w}$. Either way the $\delta(w_1, v_1) \geqslant S_G + (T - \theta_v) > Q + (T - \theta_v) \geqslant \delta(5^T a_i 6^T, v_1)$. Hence, suggesting that if $v_1 = 5^T b_1 6^{\theta_v}$ then set $w_1 = 5^T a_i 6^T$.

   (b) Let's assume that the minimum is achieved when $v_1 = 5^T b_1^{\theta_v}$, where $0 < \theta_v \leqslant S_G$. $\delta(w_1, v_1) = \delta(a_i^{\theta_w} 6^T, 5^T b_1^{\theta_v}) = T + \max(\theta_w, \theta_v) \geqslant T + \theta_v = \delta(\varnothing, v_1)$. Hence, suggesting that if $v_1 = 5^T b_1^{\theta_v}$ then set $w_1 = \varnothing$.

   (c) Let's assume that the minimum is achieved when $v_1 = 5^{\theta_v}$, where $0 \leqslant \theta_v \leqslant T$. $\delta(w_1, v_1) = \delta(a_i^{\theta_w} 6^T, 5^{\theta_v}) > \delta(\varnothing, 5^{\theta_v})$. Again suggesting that if $v_1 = 5^{\theta_v}$ then set $w_1 = \varnothing$.

   This again proves that no matter what form v_1 is of, the minimum cost is achieved when $w_1 = 5^T a_i 6^T$ or when $w_1 = \varnothing$, therefore, supporting the claim of Lemma 3.52. We use the same argument symmetrically to prove that x_2 cannot be of the form $5^T \dots a_i^{\theta_w}$, where $0 < \theta_w \leqslant S_G$ or of the form $a_i^{\theta_{w_1}} \dots a_j^{\theta_{w_2}}$ for $0 < \theta_{w_1}, \theta_{w_2} \leqslant S_G$.

3. Let's consider a scenario where x_2 is of the form $6^{\theta_w} \dots 6^T$ where $0 < \theta_w \leqslant T$.

$$x_2 := \overbrace{6^{\theta_w}}^{w_1} \overbrace{5^T a_{i+1} 6^T \dots 6^T}^{w_2}$$

$$y_2 := \underbrace{5^T b_1 6^T 5^T b_2 6^T \dots 6^T}_{v_1 \bigcirc v_2}$$

   (a) Let's assume that the minimum is achieved when $v_1 = 5^T b_1 6^{\theta_v}$, where $0 < \theta_v \leqslant T$. $\delta(w_1, v_1) = \delta(6^{\theta_w}, 5^T b_1 6^{\theta_v}) = \max(T + S_G, T + S_G + \theta_v - \theta_w) > Q + |\theta_w - \theta_v| \geqslant \delta(5^T a_i 6^{\theta_w}, 5^T b_1 6^{\theta_v})$. Hence proving that if $v_1 = 5^T b_1 6^{\theta_v}$ then set $w_1 = 5^T a_i 6^T$.

[28] Note that we have not listed the scenario where v_1 is of the form $5^T b_1 6^T *$. The reason being the following: if v_1 is of the form $5^T b_1 6^T *$, then v_2 will be of the same form as of x_2 that we are arguing against.

(b) Let's assume that the minimum is achieved when $v_1 = 5^T b_1^{\theta_v}$, where $0 < \theta_v \leqslant S_G$. $\delta(w_1, v_1) = \delta(6^{\theta_w}, 5^T b_1^{\theta_v}) = T + \theta_v = \delta(\varnothing, v_1)$. Hence proving that if $v_1 = 5^T b_1^{\theta_v}$ then set $w_1 = \varnothing$.

(c) Let's assume that the minimum is achieved when $v_1 = 5^{\theta_v}$, where $0 \leqslant \theta_v \leqslant T$. $\delta(w_1, v_1) = \delta(6^{\theta_w}, 5^{\theta_v}) \geqslant \delta(\varnothing, 5^{\theta_v})$. Hence again proving that if $v_1 = 5^{\theta_v}$ then set $w_1 = \varnothing$.

This again proves that no matter what v_1 is, the minimum cost is achieved when $w_1 = 5^T a_i 6^T$ or when $w_1 = \varnothing$, therefore, supporting the claim of Lemma 3.52. Again we use the same argument symmetrically to prove that x_2 cannot be of the form $5^T \dots 5^{\theta_w}$, where $0 < \theta_w \leqslant T$ or of the form $6^{\theta_{w_1}} \dots 5^{\theta_{w_2}}$ for $0 < \theta_{w_1}, \theta_{w_2} \leqslant T$.

Therefore, proving that there exists an x_2 of the form $5^T \dots 6^T$ such that $\delta(x, y) = 2|x| + \delta(x_2, y_2)$. □

Using Corollary 3.51 and Lemma 3.52 for a chosen partition of $y = y_1 y_2 y_3$ into three substrings, we have shown that there exists a partition of $x = x_1 x_2 x_3$ such that $\delta(x, y) = 2|x| + \delta(x_2, y_2)$ and x_2 is of the form $5^T \dots 6^T$.

We still have to prove that there exists a coarse alignment $\mathrm{C} \in \mathcal{C}_{(3\cdot 2^{n/2}-2),2^{n/2}}$ such that $\delta(x_2, y_2) = \sum_{(i,j)\in \mathrm{C}} \delta(u_i, v_j)$ where u_i and v_j are substrings of x_2 and y_2 as mentioned above. To prove that we use the result from the Lemma 3.53 below.

Lemma 3.53. *Given two substrings x_2 and y_2, both of the form $5^T \dots 6^T$. There exists a separator $6^T 5^T$ in the string x_2 (assuming that x_2 has one) that completely aligns with a separator $6^T 5^T$ from the other string y_2 (also assuming that y_2 has one). A separator is a substring $6^T 5^T$ that repeatedly occurs in both the strings x_2 and y_2.*

Proof. We have already established that x_2 is of the form $5^T \dots 6^T$, and earlier we chose y_2 to be of the form $5^T \dots 6^T$. Let,

$$x_2 = 5^T a'_1 6^T 5^T a'_2 \overbrace{6^T 5^T}^{\text{sep } w} a'_3 6^T \dots 5^T a'_{t_a} 6^T,$$

$$y_2 = 5^T b'_1 6^T 5^T b'_2 6^T 5^T b'_3 \underbrace{6^T 5^T}_{\text{sep}} b'_4 6^T \dots 5^T b'_{t_b} 6^T,$$

such that $\forall i \in [t_a], \forall j \in [t_b], a'_i \in \{0,1,2\}^*$ and $b'_j \in \{0,1,2\}^*$ and $Q < |a'_i| = |b'_j| = S_G \ll T$. Without loss of generality, let's assume $t_a \leqslant t_b$. Then, $\delta(x_2, y_2) \geqslant 2T(t_b - t_a) + S_G(t_b - t_a)$, because there will definitely be $2T(t_b - t_a) + S_G(t_b - t_a)$ number of symbols inserted to convert the string x_2 to y_2. Also, it is easy to see that $\delta(x_2, y_2) \leqslant 2T(t_b - t_a) + S_G(t_b - t_a) + Q \cdot t_a$ We will now show that there is no such optimal alignment of symbols where there is no separator in x_2 that completely aligns with a separator in y_2.

The proof is by contradiction. Let us assume that no separator in x_2 aligns with any separator in y_2. Let $w = 6^T 5^T$ be a separator from x_2 and let w align with a substring v from y_2 in an optimal alignment. The substring v can be of the following forms:

1. Let $v = b_i^{\theta_1} 6^T 5^T b_{i+1}^{\theta_2}$ with $0 \leqslant \theta_1, \theta_2 \leqslant S_G$. This is a trivial case leading to contradiction.

2. Let $v = b_i^{\theta_1}6^{\theta_2}$, with $0 \leqslant \theta_1 \leqslant S_G, 0 \leqslant \theta_2 \leqslant T$. The $\delta(w, v) = \delta(6^T5^T, b_i^{\theta_1}6^{\theta_2}) > 2T - \theta_1 - \theta_2$. Note that this is a deletion cost because of the mismatch in the number of symbols in w and v. As $\theta_1 \leqslant S_G$ and $\theta_2 \leqslant T$. The deletion cost $\geqslant T - S_G$. When $v = 5^{\theta_1}b_i^{\theta_2}$ with $0 \leqslant \theta_1 \leqslant T, 0 \leqslant \theta_2 \leqslant S_G$ we follow the same argument and get deletion cost $\geqslant T - S_G$.

3. Let $v = 6^{\theta_1}5^{\theta_2}$, with $0 \leqslant \theta_1, \theta_2 < T$. As the separator w aligns with the substring v, that means the $6^{T-\theta_1}$s that are *prefixing* v has to be either inserted or substituted by some symbols $\in \{0, 1, 2\}$, or matched with the 6s of a separator appearing before w in the string x_2. For the 6s that get inserted, it is cheaper to match the 6s with the 6s that get deleted instead in $\delta(6^T5^T, 6^{\theta_1}5^{\theta_2})$ $=\delta(6^{T-\theta_1}5^{T-\theta_2}, \varnothing)$. Same argument holds for the 5s that are *suffixing* v. Suppose these 6s are substituted from the 5s or the $\{0, 1, 2\}$ again it costs the same to just delete these and match freely with the 6s that are getting deleted in the alignment of w and v. And say for any reason some 6 in the prefix gets matched with a 6 from a separator preceding w, then the deletion plus the substitution cost is $> T + S_G$. Therefore, it is cheaper to align the separator w with the separator from y_2 that is surrounding v.

4. Let $v = 6^{\theta_1}5^Tb_i^{\theta_2}$, with $0 \leqslant \theta_1 < T, 0 \leqslant \theta_2 \leqslant S_G$. Similar to the argument in item 3, we analyse the cost to generate the substring $6^{T-\theta_1}$s that is prefixing v. Even in this case the deletion plus substitution cost of not aligning the separators is $> T + S_G$. A similar argument holds when $v = b_i^{\theta_1}6^T5^{\theta_2}$, with $0 \leqslant \theta_1 \leqslant S_G, 0 \leqslant \theta_2 < T$.

5. Let $v = 5^{\theta_1}b_i6^{\theta_2}$, with $0 \leqslant \theta_1, \theta_2 \leqslant T$. Consider the last 6 of the string w. Whether this 6 matches with a 6 from 6^{θ_2} in v or substitutes any symbol from the b_i or 5_1^{θ} in v the deletion and the substitution cost is $> T$.

6. Let $v = 5^{\theta_1}b_i6^T5^{\theta_2}$, with $0 \leqslant \theta_1 \leqslant T$, $0 \leqslant \theta_2 < T$. The argument here is similar to that of the argument in item 3, where we analyse the cost of generating the substring $5^{T-\theta_2}$ which succeeds the substring v in y_2. Either we align the separators completely or pay a deletion plus substitution cost $> T + S_G$. Same argument can be used when $v = 6^{\theta_1}5^Tb_i6^{\theta_2}$, with $0 \leqslant \theta_1 < T$, $0 \leqslant \theta_2 \leqslant T$. Just that, here we analyse the cost of generating the substring $6^{T-\theta_1}$ which precedes v.

7. Let $v = 6^{\theta_1}5^Tb_i6^T5^{\theta_2}$, with $0 \leqslant \theta_1, \theta_2 < T$. The deletion and substitution cost to generate the substring $6^{T-\theta_1}$ which is a prefix to the substring v (or $5^{T-\theta_2}$ a suffix to v) is $> T + S_G$.

The deletion and substitution cost induced when a separator 6^T5^T from x_2 doesn't align with a separator from y_2 is $\geqslant S_G$[29]. As there are a total of $(t_a - 1)$ such separators in x_2, if no separator from x_2 aligns with any separator from y_2 the total cost of the transformation then becomes $\geqslant 2T(t_b - t_a) + S_G(t_b - t_a) + S_G \cdot (t_a - 1)$. Therefore, such a transformation is not an optimal transformation because the edit distance $\delta(x_2, y_2) \leqslant 2T(t_b - t_a) + S_G(t_b - t_a) + Q \cdot t_a$ and as long as $t_a > 1$ we have

[29]The deletion and substitution costs induced when a separator 6^T5^T from x_2 doesn't align with a separator from y_2 is either $\geqslant T - S_G$ or $> T$ or $> T + S_G$. As $S_G \ll T$, the deletion and substitution costs are always higher than S_G.

$S_G \cdot (t_a - 1) > Q \cdot t_a$.[30] Thus, we prove that in an optimal transformation there will always exist a separator in x_2 that will freely align with a separator in y_2. □

As proved in Lemma 3.53, the existence of a completely aligning separator pair in an optimal alignment lets us make the following statement: $\delta(x_2, y_2) = \delta(x', y') + \delta(x'', y'')$, such that x', x'', y', y'' are all of the form $5^T \dots 6^T$.

$$x_2 = \overbrace{5^T a'_1 6^T 5^T a'_2 \dots 6^T}^{x'} \overbrace{5^T a'_i 6^T \dots 5^T a'_{t_a} 6^T}^{x''},$$

$$y_2 = \underbrace{5^T b'_1 6^T 5^T b'_2 6^T 5^T b'_3 \dots 6^T}_{y'} \underbrace{5^T b'_j 6^T \dots 5^T b'_{t_b} 6^T}_{y''},$$

Using this argument recursively we can see that the strings x_2 and y_2 get partitioned into substrings of the form $5^T \dots 6^T$ such that $\delta(x_2, y_2)$ is the sum of pair wise edit distance of these substrings, where only one of the substrings in each pair contains more than zero separators. This proves the claim of our Lemma 3.38. □

Below are elaborate proofs of certain lemmas and facts used in proving the correctness of reductions from BP-PP$_{\text{lcs}}$ to LCS, and from BP-PP$_{\text{edit}}$ to Edit Distance problems, respectively.

Algorithm 1: Convert a given *path* P to a coarse alignment C.

```
Result: Given as input P ∈ PATHS_{2L−1,L}, generate C ∈ C_{3L−2,L}.
C=[ ], i=1,j=1, k=|P|;
while (i ⩽ k) do
    (a+1,b)=P[i];
    (c+1,d)=P[i+1];
    if (a ≠ c) then
        if (d=b) then
            C[j]=((a + b, . . . , c + d), d);
            j=j+1;
        else
            C[j]=(c + d, (b, . . . , d));
            j=j+1
        end
        i=i+2;
    else
        C[j]=(a + b, b);
        j=j+1;
        i=i+1;
    end
end
return C;
```

[30] Recall that, $S_G = |G(\cdot)| = |\overline{G}(\cdot)|$ and $Q = \delta(G(a), \overline{G}(b))$ such that (a, b) wasn't a satisfying assignment. Clearly, $S_G > Q$. Also one can choose to make the gadgets in such a way that $S_G(t-1) > Q \cdot t$ for all $t > 1$.

Lemma 3.54. *Algorithm 1 when given a path* $\mathrm{P} = ((i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k)) \in \mathrm{PATHS}_{2L-1,L}$ *as an input, outputs a coarse alignment* $\mathrm{C} \in \mathcal{C}_{3L-2,L}$*, such that* $|\,\mathrm{C}\,| \leqslant |\,\mathrm{P}\,|$.

Proof. We present Algorithm 1 that when given as an input a *path* $\mathrm{P} \in \mathrm{PATHS}_{2L-1,L}$, it generates a sequence $\mathrm{C} = ((p_1, q_1), (p_2, q_2), \ldots, (p_m, q_m))$. It is easy to see from the algorithm that $k = |\,\mathrm{P}\,| \geqslant m = |\,\mathrm{C}\,|$. We now show that the sequence C generated by this algorithm indeed is a *coarse alignment* $\mathrm{C} \in \mathcal{C}_{3L-2,L}$.

For all neighbouring pairs $(i_l, j_l), (i_{l+1}, j_{l+1}) \in \mathrm{P}$ that the algorithm reads as inputs it checks if there is a *jump*[31] between these neighbouring pairs. If there is no jump in the path P at (i_l, j_l) and (i_{l+1}, j_{l+1}), then the algorithm just adds a term (p_*, q_*) to the sequence C such that $|p_*| = |q_*| = 1$ and $p_* = i_l + j_l - 1$ and $q_* = j_l$ and changes the position of the pointer to the next term. But, if there is a jump in the path P at (i_l, j_l) and (i_{l+1}, j_{l+1}) then the algorithm checks whether the jump is to a row above or to a row below. When the jump is to a row below then the algorithm adds a sequence (p_*, q_*) such that $p_* = (i_l + j_l - 1, \ldots, i_{l+1} + j_{l+1} - 1)$ ensuring that $|p_*|$ is the number of rows jumped below and $q_* = j_l = j_{l+1}$ ensuring that $|q_*| = 1$ and changes the pointer to the next but one term. When the jump is to a row above then the algorithm adds a sequence (p_*, q_*) such that $q_* = (j_l, \ldots, j_{l+1})$ ensuring that $|q_*|$ is the number of rows jumped above and $p_* = i_l + j_l + 1 = i_{l+1} + j_{l+1} + 1$[32] ensuring that $|p_*| = 1$ and then changes the pointer to the next but one term.

As the definition of a *path* requires that $1 = j_1 < j_2 < \cdots < j_k = L$. Therefore, it is easy to see that $\forall r \in [m-1], q_r <_e q_{r+1}$[33] and $\forall r, s \in [m]$, $q_r \cap q_s = \varnothing$ and $\cup_{r=1}^{m} q_r = [L]$. Also using the same definition we know that either $i_l = i_{l+1}$ (implying $j_{l+1} = j_l + 1$) or $i_l < i_{l+1}$ (implying $j_{l+1} = j_l$) or $i_l > i_{l+1}$ (implying $j_{l+1} = j_l + i_l - i_{l+1}$) therefore, ensuring that $i_{l+1} + j_{l+1} \geqslant i_l + j_l$ therefore proving that $\forall r \in [m-1], p_r <_e p_{r+1}$ and $\forall r, s \in [m]$, $p_r \cap p_s = \varnothing$ and $\cup_{r=1}^{m} p_r = [(i_1 + j_1 - 1) \ldots (i_k + j_k - 1)]$.

It is given that $\mathrm{P} \in \mathrm{PATHS}_{2L-1,L}$ that implies $1 \leqslant i_1 \leqslant 2L - 1, j_1 = 1$ and $1 \leqslant i_k \leqslant 2L - 1, j_k = L$. Therefore, it is now clear that the sequence C produced by Algorithm 1 is indeed a coarse alignment $\mathrm{C} \in \mathcal{C}_{3L-2,L}$. □

Lemma 3.55. *Algorithm 2 when given a coarse alignment* $\mathrm{C} \in \mathcal{C}_{3L-2,L}$ *(with* $L = 2^{n/2}$*) as an input outputs a sequence* P*. This sequence* P *is* ***either*** *a path* $\mathrm{P} \in \mathrm{PATHS}_{2L-1,L}$ ***or*** *there exists another coarse alignment* $\mathrm{D} \in \mathcal{C}_{3L-2,L}$ *for which a path* $\mathrm{R} \in \mathrm{PATHS}_{2L-1,L}$ *can be generated using Algorithm 2 and the* edit-cost(D) $\leqslant$ edit-cost(C) [34]*.*

Proof. Let $\mathrm{C} = ((p_1, q_1), (p_2, q_2), \ldots, (p_m, q_m))$ be a coarse alignment from the set $\mathcal{C}_{3L-2,L}$. We classify every element $(i, j) \in \mathrm{C}$ into *bad* and *good* terms in the following way:

$$\mathrm{bad}(i,j) = \begin{cases} 1, & \text{if } \exists a \in i \text{ and } \exists b \in j \text{ such that } (a-b) < 0 \text{ or } (a-b) \geqslant 2L-1 \\ 0 & \text{otherwise.} \end{cases}$$

The edit-cost for alignment C will be:

[31] Refer to the Definition 3.20 for the definitions of a *path* and also *jumps* in a path.

[32] According to the definition of a *path*, when a jump is to a row above then $i_l + j_l = i_{l+1} + j_{l+1}$.

[33] Given two sequences a and b, $a <_e b$ implies $\forall u \in a, \forall v \in b, u < v$.

[34] Lemma 3.55 and Fact 3.56 hold in context of LCS with a slight change in the statement, that is LCS-value(D) $\geqslant$ LCS-value(C).

$$\begin{aligned}\text{edit-cost(C)} &= \Sigma_{(i,j)\in C}\delta(u_i, v_j) \\ &= \underbrace{\Sigma_{(i,j)\in \text{C},(\text{bad}(i,j)=1)}\delta(u_i, v_j)}_{\text{bad terms}} + \underbrace{\Sigma_{(i,j)\in \text{C},(\text{bad}(i,j)=0)}\delta(u_i, v_j)}_{\text{good terms}},\end{aligned}$$

where $u_i = \bigcirc_{l'\in i} 5^T g_{l'} 6^T$[35] and $v_j = \bigcirc_{l''\in j} 5^T \overline{G}(b_{l''}) 6^T$.

Fact 3.56. *For every coarse alignment* $\text{C} \in \mathcal{C}_{3L-2,L}$ *that contains bad terms, there exists a coarse alignment* $\text{D} \in \mathcal{C}_{3L-2,L}$ *such that* D *consists of good terms and the* edit-cost(D) $\leqslant$ edit-cost(C).

Proof. A bad term (i, j) in the coarse alignment C implies $\exists a \in i, \exists b \in j$ such that $(a - b) < 0$ or $(a - b) \geqslant 2L - 1$. Let $K = 2L - 1$. This term can be a bad term in three different ways:

1. Category 1: $\forall a \in i, \forall b \in j$, either $(a - b) < 0$ or $0 \leqslant (a - b) < K$.
2. Category 2: $\forall a \in i, \forall b \in j$, either $(a - b) \geqslant K$ or $0 \leqslant (a - b) < K$.
3. Category 3: $\exists a \in i, \exists c \in i$, such that $(a - j) < 0$ and $(c - j) \geqslant K$.

Consider the coarse alignment $\text{N} = ((1, 1), (2, 2), \ldots, (L, L))$. Clearly the edit-cost(N) $\leqslant L{\cdot}Q$ and the corresponding path for N using Algorithm 2 is $\text{P}_N = ((1, 1), (1, 2), \ldots, (1, L))$. Let $\text{C} = ((p_1, q_1), (p_2, q_2), \ldots, (p_m, q_m))$ and let us label each of these terms into bad or good terms. Suppose C contains a bad term (p, q) of category 3, then $|p| \neq 1$ because bad category 3 requires that the following condition is met: $\exists a \in p$ such that $(a - q) < 0$ and $\exists b \in p$ such that $(b - q) \geqslant K$. Combining both these conditions we get $(b - a) \geqslant K$. The edit-cost(C) $\geqslant \delta(u_p, v_q) > K \cdot (2T + S_G) >$ edit-cost(N) which proves this fact. Therefore, we can safely only consider cases where the coarse alignment consists of bad terms of category 1 and 2.

Let $\text{G}_1 = ((p_{k'}, q_{k'}), (p_{k'+1}, q_{k'+1}), \ldots, (p_{k''}, q_{k''}))$ be the first group of bad terms in C. It is easy to see that the bad terms come in groups of 2 or more. This means that the entire preceding group $\text{G}_0 = ((p_1, q_1), (p_2, q_2), \ldots, (p_{k'-1}, q_{k'-1}))$ and the next term $\text{G}_2 = ((p_{k''+1}, q_{k''+1}))$ has to be good. We now claim that there exists a group $\text{G}' = ((p'_1, q'_1), (p'_2, q'_2), \ldots, (p'_{k'''}, q'_{k'''}))$ that covers the set of indices of G_1 and has all good terms and the edit-cost(G′) $\leqslant$ edit-cost(G_1). As the terms $(p_{k'}, q_{k'})$ and $(p_{k''}, q_{k''})$ of G_1 can be of any of the two bad categories we have a total of four cases to consider:

1. Let $(p_{k'}, q_{k'})$ of category 1 and $(p_{k''}, q_{k''})$ of category 2: this scenario suggests that there will be two neighbouring terms (p_l, q_l) and (p_{l+1}, q_{l+1}) such that they are of bad category 1 and 2 respectively. As (p_l, q_l) is of category 1, that implies $\exists a \in p_l$ and $\exists b \in q_l$ such that $(a - b) < 0$. Similarly, as (p_{l+1}, q_{l+1}) is of category 2, that implies $\exists c \in p_{l+1}$ and $\exists d \in q_{l+1}$ such that $(c - d) \geqslant K = 2L - 1$. Combining these two inequalities we get $(c - a) - (d - b) > K$. As (p_l, q_l) and (p_{l+1}, q_{l+1}) are neighbouring terms in a coarse alignment the edit-cost($((p_l, q_l), (p_{l+1}, q_{l+1}))$) $> \delta(u_{p_l} \bigcirc u_{p_{l+1}}, v_{q_l} \bigcirc v_{q_{l+1}}) > (2T + S_G) \cdot K >$ edit-cost(N). Therefore proving this fact.

[35] Note that $g_{l'} = G(a_{l'+1-2^{n/2}})$ when $0 \leqslant (l' - 2^{n/2}) < 2^{n/2}$ and $g_{l'} = r$ otherwise.

2. Let $(p_{k'}, q_{k'})$ of category 2 and $(p_{k''}, q_{k''})$ of category 1: this scenario doesn't exist because of the following reason. Let (p_l, q_l) and (p_{l+1}, q_{l+1}) be two neighbouring terms such that they are of bad category 2 and 1 respectively. This implies $\exists a \in p_l$ and $\exists b \in q_l$ such that $(a - b) \geqslant K = 2L - 1$. Similarly, as (p_{l+1}, q_{l+1}) is of category 1, that implies $\exists c \in p_{l+1}$ and $\exists d \in q_{l+1}$ such that $(c - d) < 0$. Combining both these inequalities we get $(a - c) - (b - d) > K$. As (p_l, q_l) and (p_{l+1}, q_{l+1}) are elements of a coarse alignment both $(a - c)$ and $(b - d)$ will be negative. That implies $|b - d| > K = 2L - 1$ which is not possible because the indices in q_* ranges between $1 \dots L$.

3. Let both $(p_{k'}, q_{k'})$ and $(p_{k''}, q_{k''})$ be of category 1: we first claim that in this scenario all the intermediate bad terms in the group G_1 will also be of category 1 because of the impossibility result from scenario 2.

   Let $(p_l, q_l), (p_m, q_m) \in \mathrm{G}_1$ be two nearest terms of the form $|p_l| = 1, |q_l| \neq 1$ and $|p_m| \neq 1, |q_m| = 1$. The only other intermediate terms in between these terms are of the form (p_*, q_*) such that $|p_*| = |q_*| = 1$ where edit-cost$((p_*, q_*)) = Q$[36]. We now do the following: w.l.o.g. let us assume $|q_l| \geqslant |p_m|$. We remove the $(p_m - 1)$ maximum most elements from the set q_l and $(p_m - 1)$ minimum most elements from the set p_m. For an element that we remove from the set q_l we pair it with an intermediate element of type $|p_*| = 1$. Furthermore, we pair an intermediate element of type $|q_*| = 1$ with the element that we have removed from the set p_m and accordingly make a shift in the earlier pairing of the intermediate terms. Thereby reducing the total cost by a positive quantity[37]. We know that such pairs exists in G_1 because the group G_0 and G_2 only consists of good terms. We keep repeating this process until we get rid of all the pairs like (p_l, q_l) and (p_m, q_m). Also note that this process reduces the edit-cost. Therefore, by following the procedure repeatedly we have converted the group of bad terms G_1 into a new group G' which spans all the indices spanned by G_1 and also has the $\leqslant$ edit-cost(G_1). And because we have got rid of all the pairs $(p_l, q_l), (p_m, q_m)$ therefore, either all the terms in G' are of the form $|p_*| = 1, |q_*| \neq 1$ and $|p_*| = 1, |q_*| = 1$ or are of the form $|p_*| \neq 1, |q_*| = 1$ and $|p_*| = 1, |q_*| = 1$ or all are of the form $|p_*| = |q_*| = 1$.

   We now have to prove that all the terms in G' are good terms. Let $\Delta' = \max(p_{k'-1}) - \max(q_{k'-1})$ and $\Delta'' = \min(p_{k''+1}) - \min(q_{k''+1})$. These $0 \leqslant \Delta' < K$ and $0 \leqslant \Delta'' < K$ as the $(p_{k'-1}, q_{k'-1})$ and $(p_{k''+1}, q_{k''+1})$ were the good terms outside of G_1. The difference between the number of indices spanned by the p_* terms and the number of indices spanned by the q_* term in G_1 is $\Delta'' - \Delta'$. Two cases to consider again:

   (a) Case $\Delta' > \Delta''$: the group G' should consist of a term where (p', q') such that $|p'| = 1$ and $|q'| \neq 1$ because the number of indices spanned by the q'_* terms is higher than the number of indices spanned by the p'_* terms. Let $\mathrm{G}' = ((p'_1, q'_1), (p'_2, q'_2), \dots, (p'_l, q'_l))$ and let $\forall i \in [l], \Delta_i = p'_i - \max(q'_i)$ therefore, $\Delta_1 \leqslant \Delta'$ and $\Delta_l = \Delta''$. As all the other terms will also be either of the form $|p'_*| = 1$ and $|q'_*| \neq 1$ or of the form $|p'_*| = 1$ and $|q'_*| = 1$

[36]The edit-cost$((p_*, q_*)) = \delta(u_{p_*}, v_{q_*}) = \delta(5^T g_{p_*} 6^T, 5^T \overline{G}(b_{q_*}) 6^T) = \delta(5^T r 6^T, 5^T \overline{G}(b_{q_*}) 6^T)$ because as $p_* < L$ we have $g_{p_*} = r$

[37]The minimum gain here is $2 \cdot (2T + S_G) - 3Q$ which is positive because $T \gg S_G > Q$.

(as proved in the previous paragraph) therefore, $\forall i \in [l-1], \Delta_{i+1} \leqslant \Delta_i$. Therefore, we have $\Delta' \geqslant \Delta_1 \geqslant \Delta_2 \geqslant \cdots \geqslant \Delta_l = \Delta''$ implying that $\forall i \in [l],\ \text{bad}(p'_i, q'_i) = 0$.

(b) Case $\Delta'' > \Delta'$: the group G' should consist of a term where (p', q') such that $|p'| \neq 1$ and $|q'| = 1$ because the number of indices spanned by the p_* terms is higher than the number of indices spanned by the q_* terms. All the other terms will also be either of the form $|p_*| \neq 1$ and $|q_*| = 1$ or $|p_*| = 1$ and $|q_*| = 1$. Let $\mathrm{G}' = ((p'_1, q'_1), (p'_2, q'_2), \ldots, (p'_l, q'_l))$ and let $\forall i \in [l], \Delta_i = \min(p'_i) - q'_i$ therefore, $\Delta_1 = \Delta'$ and $\Delta_l \leqslant \Delta''$. As all the other terms will also be either of the form $|p'_*| \neq 1$ and $|q'_*| = 1$ or of the form $|p'_*| = 1$ and $|q'_*| = 1$ (as proved in the previous paragraph) therefore, $\forall i \in [l-1], \Delta_{i+1} \geqslant \Delta_i$. Therefore, we have $\Delta' = \Delta_1 \leqslant \Delta_2 \leqslant \cdots \leqslant \Delta_l \leqslant \Delta''$ implying that $\forall i \in [l],\ \text{bad}(p'_i, q'_i) = 0$.

(c) Case $\Delta'' = \Delta'$: all the terms in the group G' should be of the form $|p'| = 1$ and $|q'| = 1$ because the number of indices spanned by the p_* terms is same as the number of indices spanned by the q_* terms. Let $\mathrm{G}' = ((p'_1, q'_1), (p'_2, q'_2), \ldots, (p'_l, q'_l))$ and let $\forall i \in [l], \Delta_i = p'_i - q'_i$ therefore, $\Delta_1 = \Delta'$ and $\Delta_l = \Delta''$. Also, $\forall i \in [l-1], \Delta_{i+1} = \Delta_i$ because all the terms in the group G' are of the form $|p'| = 1$ and $|q'| = 1$. Therefore, $\Delta' = \Delta_1 = \Delta_2 = \cdots = \Delta_l = \Delta''$ implying that that $\forall i \in [l],\ \text{bad}(p'_i, q'_i) = 0$.

4. Let both $(p_{k'}, q_{k'})$ and $(p_{k''}, q_{k''})$ be of category 2: in this scenario all the intermediate bad terms in the group G_1 will also be of category 2 because of the impossibility result from scenario 2. And the rest of the argument is same as in scenario 3.

Note that all the above-mentioned steps were to analyse the first group of bad terms. We keep repeating this procedure till we arrive at a coarse alignment D that only contains good terms. As we see that the procedure mentioned above never increases the edit-cost we can therefore safely say that edit-cost(D) $\leqslant$ edit-cost(C). □

Fact 3.57. *Algorithm 2 outputs a path* $\mathrm{R} \in \mathrm{PATHS}_{2L-1,L}$ *when the input is a coarse alignment* $\mathrm{D} \in \mathcal{C}_{3L-2,L}$ *containing only good terms.*

Proof. Run Algorithm 2 on the coarse alignment $\mathrm{D} = ((p_1, q_1), (p_2, q_2), \ldots, (p_m, q_m))$ as input and let the output sequence be $\mathrm{R} = ((i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k))$. We will now prove that $\mathrm{R} \in \mathrm{PATHS}_{2L-1,L}$ when D contains only good terms.

Given an input $\mathrm{D} = ((p_1, q_1), (p_2, q_2), \ldots, (p_m, q_m))$, Algorithm 2 checks each term $(p_l, q_l), \forall l \in [m]$ and creates two (or one) new terms $(\min(p_l) - \min(q_l) + 1, \min(q_l))$ and $(\max(p_l) - \max(q_l) + 1, \max(q_l)), \forall l \in [m]$ and creates the sequence R. As D contains all good terms, it is clear that $\forall r \in [k]$, we have $1 \leqslant i_r \leqslant K = 2L - 1$ and $1 \leqslant j_r \leqslant L$. Using the definition of a *coarse alignment* (Definition 3.25) we know that $\forall l \in [m-1], q_l <_e q_{l+1}$[38], and $\cup_{i=1}^{m} q_i = [L]$ therefore, we have $1 = j_1 \leqslant j_2 \leqslant \cdots \leqslant j_k = L$.

Consider the term $(p_l, q_l) \in \mathrm{D}$, the algorithm generates the following terms $(\min(p_l) - \min(q_l) + 1, \min(q_l))$, $(\max(p_l) - \max(q_l) + 1, \max(q_l))$[39] for the sequence

[38] Given two sequences a and b, $a <_e b$ implies $\forall u \in a, \forall v \in b, u < v$.

[39] Note that if $|p_l| = |q_l| = 1$ then both the terms are same and the algorithm just adds one term.

Algorithm 2: Convert a given coarse alignment C to a sequence P.

```
Result: Given a coarse alignment C ∈ 𝒞 generate a sequence P.
P={}, i=0, j=0, m=|C|;
while (i<m) do
    (p,q)=C[i];
    if (|p| ≠ 1 or |q| ≠ 1) then
        if (|p| ≠ 1) then
            P[j]=(min(p) − q + 1, q);
            P[j+1]=(max(p) − q + 1, q);
            j=j+2;
        else
            P[j]=(p − min(q) + 1, min(q));
            P[j+1]=(p − max(q) + 1, max(q));
            j=j+2;
        end
    else
        P[j]=(p − q + 1, q);
        j=j+1;
    end
    i=i+1;
end
return P;
```

R. If $|p_l| \neq 1$ then the algorithm generates $(\min(p_l) - q_l + 1, q_l)$, $(\max(p_l) - q_l + 1, q_l)$, clearly generating two terms $(i_r, j_r), (i_{r+1}, j_{r+1}) \in R$ such that $i_{r+1} > i_r$ while $j_{r+1} = j_r$ thus, satisfying another condition of a *path*. Also, when $|q_l| \neq 1$ then the algorithm generates $(p_l - \min(q_l) + 1, \min(q_l))$, $(p_l - \max(q_l) + 1, \max(q_l))$, clearly generating two terms $(i_s, j_s), (i_{s+1}, j_{s+1}) \in$ R such that $i_{s+1} < i_s$ while $j_{s+1} = j_s + i_s - i_{s+1}$ thus, satisfying another condition.

Consider two neighbouring terms $(p_l, q_l), (p_{l+1}, q_{l+1}) \in$ D. Suppose the last (or the only) term generated for (p_l, q_l) by the algorithm is $(i_r, j_r) = (\max(p_l) - \max(q_l) + 1, \max(q_l))$ then the first (or the only) term generated for (p_{l+1}, q_{l+1}) will be $(i_{r+1}, j_{r+1}) = (\min(p_{l+1}) - \min(q_{l+1}) + 1, \min(q_{l+1}))$. According to the definition of a *coarse alignment*, we have $p_l \cap p_{l+1} = \varnothing$ and $p_l <_e p_{l+1}$ which means that $\min(p_{l+1}) = \max(p_l) + 1$. Also the condition $q_l \cap q_{l+1} = \varnothing$ and $q_l <_e q_{l+1}$ implies $\min(q_{l+1}) = \max(q_l) + 1$ thus making sure that $j_{r+1} = j_r + 1$. Combining these two conditions we get that $i_{r+1} = i_r$. Suppose, $i_{r+1} = i_r$, then $\max(p_l) + 1 - \max(q_l) = \min(p_{l+1}) - \min(q_{l+1}) + 1$ which would imply that $\min(q_{l+1}) = 1 + \max(q_l)$ proving $j_{r+1} = j_r + 1$. Thus, satisfying another two conditions for a *path*.

Therefore, we see that if the input is a coarse alignment $\mathrm{D} \in \mathcal{C}_{3L-2,L}$ containing only good terms, then the Algorithm 2 generates a path $\mathrm{R} \in \mathrm{PATHS}_{2L-1,L}$. □

We thus prove Lemma 3.55 using Fact 3.56 and Fact 3.57. □

0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	1
0	0	0	0	1	1	1	1
0	0	0	0	0	0	1	1
0	0	0	0	0	1	1	1
0	0	1	1	0	0	0	0
1	1	0	1	1	0	1	0
1	0	0	1	0	1	0	0
1	0	1	0	0	0	0	0
0	1	0	0	1	0	0	0
1	0	1	1	0	0	0	0
0	1	1	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0

Figure 3.1

3.5 Quantum query lower bound for property P_δ

Theorem 3.58. *The bounded-error quantum query complexity for computing the property P_δ on a matrix of size $(2^{n/2+1}-1)\times 2^{n/2}$ is $\Omega(2^{0.75n})$, where the 2^n non-fixed entries are given by the encoding as shown in Figure 3.1.*

Proof. The matrices of size $(2^{n/2+1}-1)\times 2^{n/2}$ that have 2^n bit string encoded in them are of the form shown in Figure 3.1. We prove the quantum query lower bound of P_δ on these matrices using the quantum adversary method by [Amb02], but, instead of analysing the matrix M of size $2N-1\times N$ (where $N=2^{n/2}$) we analyse the sub-matrix M′ of size $\frac{N}{2}\times\frac{N}{2}$ as shown in the Figure 3.1.

Let $\mathcal{M}=\{\mathrm{M}\,|\,\mathrm{M}$ are matrices of size $2N-1\times N$ of the form in Figure 3.2$\}$. Computing the property P_δ on matrices $\mathrm{M}\in\mathcal{M}$ for a threshold $T_r=\frac{3N}{4}C_0+\frac{N}{4}C_1$ is equivalent to computing P_δ on the sub-matrices M′ (corresponding to each M) of size $\frac{N}{2}\times\frac{N}{2}$ for a different threshold value $V=\frac{N}{4}C_0+\frac{N}{4}C_1$ where the problem is to decide whether there exists a path whose value is greater than V.[40]

Recall that the property $\mathrm{P}_\delta:\{0,1\}^{\frac{N}{2}\times\frac{N}{2}}\times\{Y_1,\ldots,Y_2\}\to\{0,1\}$ (Definition 3.20) is a function of a matrix M and a range Λ which is only required to calculate the *jump* costs. However, the adversarial sets X and Y that we define to compute the quantum query complexity of the property P_δ don't need any reference to the *jump* costs, therefore, for the sake of simplicity of the proof we just define the property to be $\mathrm{P}_\delta:\{0,1\}^{\frac{N}{2}\times\frac{N}{2}}\to\{0,1\}$.

Building the adversarial sets X and Y We choose a relation $R\subseteq X\times Y\subseteq {\mathrm{P}_\delta}^{-1}(0)\times {\mathrm{P}_\delta}^{-1}(1)$ where $X=\{x|(x,y)\in R\}$ and $Y=\{y|(x,y)\in R\}$. The relation R is chosen such that,

1. For each matrix $x\in X$, each row in the matrix x has exactly $\frac{N}{4}$ number of 1s. Which implies that for all $x\in X$, the cost for each row is equal to

[40] Using a simple geometrical argument one can prove that a square matrix M′ of size $\frac{N}{2}\times\frac{N}{2}$ fits inside the white region of matrix M of size $2N-1\times N$, refer to Figure 3.1 or Figure 3.2.

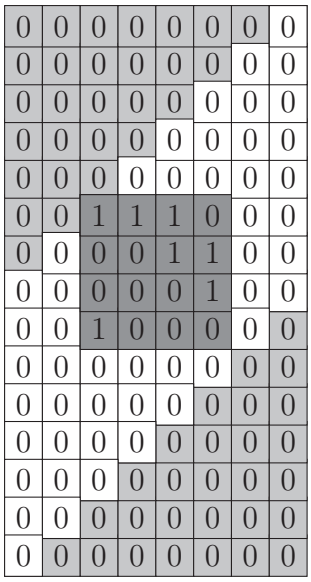

Figure 3.2: Example of a boolean matrices in $\mathcal{M}$ and its corresponding sub-matrix M' (highlighted in dark gray). Notice that all the cells in the white and light gray region are 0s.

$\frac{N}{4}C_0 + \frac{N}{4}C_1 = V$. We now have to ensure that for all matrices $x \in X$, all the paths also have its value lesser than or equal to V, which is addressed in item 3.

2. For each matrix $y \in Y$, there is only one row in the matrix y that has exactly $(\frac{N}{4} + 1)$ number of 1s and rest of the rows in y have exactly $\frac{N}{4}$ number of 1s. Which implies that for all $y \in Y$, there is a row whose value is $(\frac{N}{4} - 1)C_0 + (\frac{N}{4} + 1)C_1 > V$, as $C_0 < C_1$ (see Definition 3.20).

3. We impose additional constraints in building the sets X and Y so that considering only paths without any *jumps* is enough to decide whether or not there exists a path with value greater than V. We introduce a set of symbols that will be useful in understanding our construction.

   For a chosen even number k, we construct some symbols recursively in the following way:

   (a) $0_i \subseteq \{+_{i-1}, -_{i-1}, 0_{i-1}\}^k$ such that, number of $+_{i-1}$-type symbols in 0_i is equal to number of $-_{i-1}$-type symbols in 0_i.

   (b) $+_i \subseteq \{+_{i-1}, -_{i-1}, 0_{i-1}\}^k$ such that, number of $+_{i-1}$-type symbols in $+_i$ is one more than number of $-_{i-1}$-type symbols in $+_i$, but overall there is only one 1 more than the number of 0s.

   (c) $-_i \subseteq \{+_{i-1}, -_{i-1}, 0_{i-1}\}^k$ such that, number of $+_{i-1}$-type symbols in $-_i$ is one less than number of $-_{i-1}$-type symbols in $-_i$, but overall there is only one 0 more than the number of 1s.

   Base case symbols $\{0_0, +_0, -_0, \}$ are defined as follows:

   (a) $0_0 \subseteq \{0, 1\}^k$ such that, number of 1s in 0_0 is equal to number of 0s in 0_0. Therefore, k has to be even.

(b) $+_0 \subseteq \{0,1\}^k$ such that, number of 1s in $+_0$ is two more than number of 0s in $+_0$.

(c) $-_0 \subseteq \{0,1\}^k$ such that, number of 1s in $-_0$ is two less than number of 0s in $-_0$.

For all $x \in X$, each row of the matrix x contains k symbols from the set $\{+_t, -_t, 0_t\}$ such that number of $+_t$ symbols is equal to the number of $-_t$ symbols. While for all $y \in Y$, only one row in the matrix y contains one more of $+_t$ symbol when compared to the number of $-_t$ symbols. The rest of the rows are balanced just like the rows of the matrices belonging to set X. Example of such a matrices can be viewed in Figure 3.3.

The choice of k and t will be such that (1) $k^{t+2} = \frac{N}{2}$, condition that ensures that number of elements in each row of the matrices is $\frac{N}{2}$ and (2) $C_1 \cdot k \cdot (t+2) < C_{jump}$, a condition that ensures that the increase in the value is less than the *jump* cost, because of which we don't need to consider paths with jumps for computing the query complexity of P_δ on matrices in X and Y. These conditions will be reviewed again in the later parts of the proof where we calculate the query lower bound of P_δ in the ϵ-bounded error setting.

+	+	−	−
−	0	0	+
+	−	+	−
0	−	0	+

+	+	−	−
−	0	0	+
+	**0**	+	−
0	−	0	+

Figure 3.3: Example of a boolean matrix $x \in X$ (left) and a boolean matrix $y \in Y$ (right) such that $(x, y) \in R$. The symbol $'+' = +_t$, $'0' = 0_t$ and $'-' = -_t$.

The quantum query complexity of P_δ

1. For each $x \in X$, we have at least $(\frac{k}{2})^{t+2} \cdot \frac{N}{2}$ number of ys, such that $(x, y) \in R$.
2. For each $y \in Y$, we have at least $(\frac{k}{2})^{t+2}$ number of xs, such that $(x, y) \in R$.
3. We can visualise a matrix of size $\frac{N}{2} \times \frac{N}{2}$ as a string of length $\frac{N^2}{4}$. For each $x \in X$ and $i \in [\frac{N^2}{4}]$, there is only one input $y \in Y$ such that $(x, y) \in R$ and $x_i \neq y_i$.
4. For each $y \in Y$ and $i \in [\frac{N^2}{4}]$, there is only one input $x \in X$ such that $(x, y) \in R$ and $x_i \neq y_i$.
5. Therefore, the quantum adversary method by [Amb02] gives a quantum lower bound of $\Omega(\frac{k^{(t+2)}}{2^{(t+2)}} \cdot \sqrt{N})$ for distinguishing the sets X and Y.
6. As mentioned earlier we chose the values k and t such that matrices in both these sets X and Y have their optimal paths without any jumps. The maximum number of ones that can be gained with a jump is $k \cdot (t+1) + \frac{k}{2} < k \cdot (t+2)$. The increase in the value due to these 1s is upper bounded by $C_1 \cdot k \cdot (t+2)$. Therefore, as long as we chose the values k and t such that $C_1 \cdot k \cdot (t+2) < C_{jump}$ there will not be any jump whatsoever.

7. We have to choose the values of k and t such that the lower bound mentioned in item 5 is maximised while satisfying the two following constraints: (1) $k^{t+2} = \frac{N}{2}$ and (2) $C_1 \cdot k \cdot (t+2) < C_{jump}$. From (1) we get $t + 2 = \log_k(N/2)$, hence, $k \cdot (t+2) = \frac{k}{\log k} \log(\frac{N}{2})$. Therefore, the lower bound is $\Omega(\frac{N}{2}^{(1.5-\frac{1}{\log k})})$. By fixing $k = \omega(1)$, we get the lower bound of $\Omega(N^{1.5})$.[41] Therefore a lower bound of $\Omega(2^{0.75n})$ as $N = 2^{n/2}$.

Therefore, we can conclude that the bounded-error quantum query lower bound for computing P_δ on matrices that have 2^n bit strings encoded in them as shown in Figure 3.1 is $\Omega(2^{0.75n})$. □

Corollary 3.59. *Let $\mathcal{V}$ be the set of matrices of size $(2^{n/2+1} - 1) \times 2^{n/2}$ that have 2^n bit strings encoded in them as shown in Figure 3.1, such that $\mathrm{PP}_{\mathrm{lcs}}$ is defined on the matrix. The bounded-error quantum query complexity for computing $\mathrm{PP}_{\mathrm{lcs}}$ on matrices in $\mathcal{V}$ is $\Omega(2^{0.75n})$.*[42]

Proof. The property $\mathrm{PP}_{\mathrm{lcs}}$ defined at 3.21 is a promise version of property $\mathrm{P}_{\mathrm{lcs}}$. The results of Theorem 3.58 also hold for the property $\mathrm{PP}_{\mathrm{lcs}}$ because the adversarial sets that we construct in that theorem don't depend on the perturbation range Λ. Therefore, for a constant ϵ, $\mathrm{Q}_\epsilon(\mathrm{PP}_{\mathrm{lcs}}|_{\mathcal{V}}) = \Omega(2^{0.75n})$. □

3.5.1 Lower bound for the restricted Dyck language

Consider the restricted Dyck language, the language of balanced parentheses of depth bounded by k. The study of the quantum query complexity of this language was initiated in [AGS19] by Aaronson, Grier, and Schaeffer where they provide an $\widetilde{O}(\sqrt{n})$ algorithm to decide the language for a constant k. As a corollary to Theorem 3.58 in Section 3.5 we show that the query complexity of restricted Dyck language is linear for any $k = \omega(\log n)$, partially answering an open question posed by the authors in [AGS19].

Theorem 3.60. *The quantum query complexity of the restricted Dyck language is $\Omega(n^{1-o(1)})$ for any $k = \omega(\log n)$.*

Proof (sketch). In the proof of Theorem 3.58 in Section 3.5 we have constructed sets of length n 0/1 strings such that, for each string in the sets, every prefix is at most some distance d away from balanced, and such that the query complexity of deciding whether these strings are precisely balanced or not is $\Omega(n)$, whenever $d = \omega(\log n)$.

We can directly use the rows of the matrices that form the adversary set in Theorem 3.58 and use this as the adversary set to lower bound the query complexity of the restricted Dyck language with $k = 2d$ in the following way:

1. Interpret 0 as an open bracket '(', and 1 as a closing bracket ')'.

2. For any row r from the matrices of the adversarial set, we define

$$r' = 0^d r 1^d$$

[41] Recall from Definition 3.20 that C_0, C_1 are constants and C_{jump} can be set to $\omega(\log N)$.
[42] A similar argument holds for $\mathrm{PP}_{\mathrm{edit}}$.

If r is balanced, and satisfies the promise of never being d-far from balanced, this is a valid 1-instance for the restricted Dyck language with $k = 2d$. Additionally, if r is not balanced, then this is a valid 0-instance of the restricted Dyck language. Therefore, the adversary bound also is valid for this problem (losing an additive $2d$ in the lower bound, which is negligible in the parameter range that needs to be considered). □

3.6 Summary, future directions and open questions

Summary We presented a quantum version of the strong exponential-time hypothesis, as QSETH, and demonstrated several consequences from QSETH. These included the transfer of previous Orthogonal-Vector based lower bounds to the quantum case, with a quadratically lower time bound than the equivalent classical lower bounds. We also showed two situations where the new QSETH does not lose this quadratic factor: a lower bound showing that computing edit distance or LCS requires time $n^{1.5-o(1)}$ for a quantum algorithm, and an $n^{2-o(1)}$ quantum lower bound for Proofs of Useful Work [BRS+18], both conditioned on QSETH.

Future directions and open questions Possible future applications for the QSETH framework are numerous.

- Most importantly, the QSETH can potentially be a powerful tool to prove conditional lower bounds for additional problems in BQP. The most natural candidates are other string problems, such as Dynamic Time Warping for example, but there are many other problems for which the 'basic QSETH' does not immediately give tight bounds.

- Our proofs for showing $n^{1.5-o(1)}$ lower bounds for Edit Distance and LCS use (almost) identical arguments, except in the gadget constructions where they are different. One would expect that there is a more general framework unifying our results for Edit Distance and LCS, and possibly for the other string problems. Towards that, the paper by [BK15] would be a good starting point.

- Additionally, the notion of *compression oblivious* properties are potentially interesting as an independent object of study. We expect most natural properties to be compression oblivious, but leave as an open question what complexity-theoretic assumptions are needed to show that, e.g., the parity function is compression oblivious.

Future directions also include a careful study of quantum time complexity of the other core problems in fine-grained complexity, such as 3SUM and APSP which we will discuss in the upcoming chapters of this thesis. Just like with satisfiability, the basic versions of these problems are amenable to a Grover-based quadratic speedup. It is possible (or maybe even required) that extensions of those key problems can (should) also be used to prove stronger conditional lower bounds, in a similar way to the reduction that was used for LCS or Edit Distance in the current work.

Chapter 4

Memory Compression with Quantum Random-Access Gates

Chapter summary In the classical Random Access Machine (RAM) model, we have the following useful property. If we have an algorithm that uses M memory cells throughout its execution, and in addition is sparse, in the sense that at any point in time only m out of M cells will be non-zero, then we may "compress" it into another algorithm which uses only $m \log M$ memory and runs in almost the same time. We may do so by simulating the memory using either a hash table, or a self-balancing tree.

We show an analogous result for quantum algorithms equipped with quantum random-access gates. If we have a quantum algorithm that runs in time T and uses M qubits, such that the state of the memory, at any time step, is supported on computational-basis vectors of Hamming weight at most m, then it can be simulated by another algorithm which uses only $O(m \log M)$ memory, and runs in time $\widetilde{O}(T)$.

We show how this theorem can be used, in a black-box way, to simplify the presentation in several papers. Broadly speaking, when there exists a need for a space-efficient history-independent quantum data structure, it is often possible to construct a space-inefficient, yet sparse, quantum data structure, and then appeal to our main theorem. This results in simpler and shorter arguments.

This chapter based on the following paper:

[BLP+22b] Harry Buhrman, Bruno Loff, Subhasree Patro, Florian Speelman. Memory Compression with Quantum Random-Access Gates. In *Proceedings of the 17th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2022)*.

4.1 Introduction

This work arose out of our other work on quantum fine-grained complexity, contents of Chapter 5, where we had to make use of a quantum walk, similar to how Ambainis uses for his algorithm for element distinctness [Amb07]. An essential aspect of these algorithms is the use of a history-independent data structure. In the context of our work, we needed three slightly different data structures of this type, and on each of these occasions we saw a similar scenario. If we were only concerned with the time complexity of our algorithm, and were willing to tolerate a polynomial increase in the space complexity (the number of qubits used by the algorithm), then there was a very simple data structure that would serve our purpose. If, however, we wanted the algorithm to be space-efficient as well, then we needed to resort to more complicated data structures.

And we made the following further observation: we were using the simple yet space-inefficient data structures that were rather *sparse*, in the sense that although M qubits were being used in total, all the amplitude was always concentrated on computational-basis vectors of Hamming weight at most $m \ll M$. The analogous classical scenario is an algorithm that uses M memory registers, but at any time step all but m of these registers are set to 0. In the classical case, we know how to convert any such an *m-sparse* algorithm into an algorithm that uses $O(m \log M)$ memory, by using, e.g., a hash table. We wondered whether the same thing could be said of quantum algorithms. This turned out to be possible, and the main purpose of this chapter is to explain how it can be done. We will take an arbitrary *sparse* quantum algorithm, and *compress* it into a quantum algorithm that uses little space.

Our main theorem is as follows (informally stated):

Theorem 4.1. *Any m-sparse quantum algorithm using time T and M qubits can be simulated with ε additional error by a quantum algorithm running in time $O(T \cdot \log(\frac{T}{\varepsilon}) \cdot \log(M))$, using $O(m \log M)$ qubits.*

We will prove this result using quantum radix trees in Section 4.2. The result can also be proven, with slightly worse parameters, using hash tables, but we will not do so here. We consider sparse algorithms in the QRAM model of computation (described in Section 2.3 of the preliminaries Chapter 2). Recall that, in our defined model of computation we allow the algorithm to use quantum random-access gates a.k.a. RAGs, and the only caveat is that the *compressed* simulation requires such gates, even if the original algorithm does not.

The $\log M$ factor in the time bound can be removed if we assume that certain operations on $O(\log M)$ bits can be done at $O(1)$ cost. This includes only simple operations such as comparison, addition, bitwise XOR, or swapping of two blocks of $O(\log M)$ adjacent qubits.[1] All these operations can be done at $O(1)$ cost in the usual classical Random-Access Machines.

The techniques used to prove our main theorem are not new: quantum radix-trees first appeared in a paper by Bernstein, Jeffery, Lange and Meurer [BJL+13] (see also Jeffery's PhD thesis [Jef14]). One other contribution of our work in this chapter is to present BJLM technique in full, as in currently available presentations

[1] The qubits in each block are adjacent, but the two swapped blocks can be far apart from each other.

of the technique, several crucial aspects of the implementation are missing or buggy.[2]

But our main contribution is to use these techniques at the right level of abstraction. Theorem 4.1 is very general, and can effectively be used as a black box. One would think that Theorem 4.1, being such a basic and fundamental statement about quantum computers, and being provable essentially by known techniques, would already be widely known. But this appears not to be the case, as papers written as recently as a year ago could be significantly simplified by appealing to such a theorem. Indeed, we believe that the use of Theorem 4.1 will save researchers a lot of work in the future, and this is our main motivation for writing this work.

To illustrate this point, we overview three papers [Amb07; ACL+20; BLP+22a], first two in Section 4.3 and the last (our very own work) in Chapter 5. All these results make use of a quantum walk together with a history-independent, space-efficient but complicated data structure. As it turns out, we can replace these complicated data structures with very simple tree-like data structures. These new, simple data structures are memory inefficient but sparse, so we may then appeal to Theorem 4.1 to get similar upper bounds. The proofs become shorter: we estimate each of these papers could be cut in size by 4 to 12 pages. And furthermore, using simpler (memory inefficient but sparse) data-structures allows for a certain *separation of concerns*: when one tries to describe a space-efficient algorithm, there are several bothersome details that one needs to keep track of, and they obscure the presentation of the algorithm. By using simpler data structures, these bothersome details are disappear from the proofs, and are entrusted to Theorem 4.1.

4.2 Compressing sparse QRAM algorithms

In classical algorithms, we may have an algorithm which uses M memory registers, but such that, at any given time, only m out of these M registers are non-zero. In this case we could call such an algorithm *m-sparse*. The following definition is the quantum analogue of this.

Definition 4.2 (Sparse QRAM algorithms)**.** Let $\mathcal{C} = (n, T, W, M, C_1, \dots, C_T)$ be a QRAM algorithm using time T, W work qubits, and M memory qubits. Then, we say that $\mathcal{C}$ is m-sparse, for some $m \leqslant M$, if at every time-step $t \in \{0, \dots, T\}$ of the algorithm, the state of the memory qubits is supported on computational basis vectors of Hamming weight $\leqslant m$. I.e., we always have

$$|\psi_t\rangle \in \mathsf{span}\left(|u\rangle|v\rangle \;\middle|\; u \in \{0,1\}^W, v \in \binom{[M]}{\leqslant m} \right)$$

In other words, if $|\psi_t\rangle$ is written in the computational basis:

$$|\psi_t\rangle = \sum_{u\in\{0,1\}^W} \sum_{v\in\{0,1\}^M} \alpha_{u,v}^{(t)} \cdot \underbrace{|u\rangle}_{\text{Work qubits}} \otimes \underbrace{|v\rangle}_{\text{Memory qubits}},$$

then $\alpha_{u,v}^{(t)} = 0$ whenever $|v| > m$.

[2]For example, some operations are defined which are not unitary. Or, there is no mention of error in the algorithms, but they actually cannot be implemented in an error-free way using a reasonable number of gates from any standard gate set.

Let $\mathcal{C} = (n, T, W, M, C_1, \ldots, C_T)$ be the circuit of an m-sparse QRAM algorithm computing a relation f with error ε and let the state of the algorithm at every time-step t, when written in the computational basis, be

$$|\psi_t\rangle = \sum_{u \in \{0,1\}^W} \sum_{v \in \binom{[M]}{\leqslant m}} \alpha_{u,v}^{(t)} \underbrace{|u\rangle}_{\text{Work qubits}} \otimes \underbrace{|v\rangle}_{\text{Memory qubits}} . \tag{4.1}$$

Using the description of $\mathcal{C}$ and the assumption that this algorithm is m-sparse we will now construct another QRAM algorithm $\mathcal{C}'$ that uses much less space, about $O(m \log M)$ qubits, and computes f with almost same error probability with only $O(\log M \log T)$ factor worsening in the run time.

Main observation As the state of the memory qubits in $|\psi_t\rangle$ for any t is only supported on computational basis vectors of Hamming weight at most m, one immediate way to improve on the space complexity is to succinctly represent the state of the sparsely used memory qubits. The challenge, however, is that every instruction C_i in $\mathcal{C}$ might not have an *easy* analogous implementation in the succinct representation. So we will first present a succinct representation and then show that, for every instruction C_i in the original circuit $\mathcal{C}$, there is an analogous instruction or a series of instructions that evolve the state of the succinct representation in the same way as the original state evolves due to the application of C_i.

A succinct representation Let $v \in \{0,1\}^M$ be a vector with $|v| \leqslant m$ (with $|v\rangle$ being the corresponding quantum state that uses M qubits). Whenever m is significantly smaller than M (i.e., $m \log M < M$) we can instead represent the vector v using the list of indices $\{i\}$ such that $v[i] = 1$. Such a representation will use much fewer (qu)bits. Let S_v denote the set of indices i such that $v[i] = 1$. We will then devise a quantum state $|S_v\rangle$, that represents the set S_v using a quantum data structure. This representation will be unique, meaning that for every sparse computational-basis state $|v\rangle$ there will be a unique corresponding quantum state $|S_v\rangle$, and $|S_v\rangle$ will use much fewer qubits. Then for every time-step t, the quantum state $|\psi_t\rangle$ from Equation 4.1 has a corresponding *succinctly represented* quantum state $|\phi_t\rangle$ such that

$$|\phi_t\rangle = \sum_{u \in \{0,1\}^W} \sum_{v \in \binom{[M]}{\leqslant m}} \alpha_{u,v}^{(t)} |u\rangle \otimes |S_v\rangle. \tag{4.2}$$

By using such a succinct representation, we will be able to simulate the algorithm $\mathcal{C}$ with $O(m \log M)$ qubits, with an $O(\zeta \log \frac{T}{\delta})$ additional factor overhead in time and an additional δ probability of error; recall that we use ζ to denote the time that simple operations like comparison, addition, bitwise XOR, swapping, and others take on $O(\log n)$ (qu)bits (Section 2.3.4).

To obtain the desired succinct representation $|S_v\rangle$, we use the quantum radix trees appearing in an algorithm for the subset-sum problem by Bernstein, Jeffery, Lange, and Meurer [BJL+13] (see also [Jef14]).[3] Several crucial aspects of the implementation were missing or buggy, and required some amount of work to complete

[3] More generally, their idea can be applied to convert any classical *pointer-based* history-independent data structure with reversible updates, into a history-independent quantum data structure.

and fix. The resulting effort revealed, in particular, that the data-structure is unlikely to be implementable efficiently without error (as it relies on a particular gate which cannot be implemented in an error-free way using the usual basic gates). So we include all the required details here in this chapter.

4.2.1 Radix tree

A quantum radix tree is a quantum data structure inspired by the classical radix tree whose definition is as follows.

Definition 4.3. A radix tree is a rooted binary tree, where the edges are labelled by non-empty binary strings, and the concatenation of the labels of the edges along any root-to-leaf path results in a string of the same length ℓ (independent of the chosen root-to-leaf path). The value ℓ is called the *word length* of the tree.

There is a bijective correspondence between radix trees R of word length ℓ and subsets $S \subseteq \{0,1\}^\ell$. Given R, we may obtain S as follows. Each root-to-leaf path of R gives us an element $x \in S$, so that x is the concatenation of all the edge labels along the path.

If R corresponds to S, we say that R *stores*, or *represents* S, and write $R(S)$ for the radix tree representing S, i.e., for the inverse map of what was just described (see below).

An example of a radix tree appears in Figure 4.1.

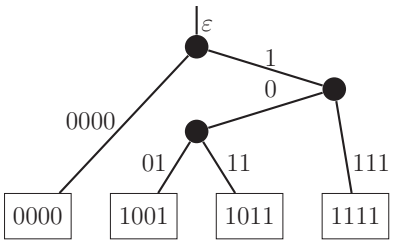

Figure 4.1: A radix tree storing the set $\{0000, 1001, 1011, 1111\}$

Given a set $S \subseteq \{0,1\}^\ell$, we obtain $R(S)$ recursively as follows: the empty set corresponds to the tree having only the root and no other nodes. We first find the longest common prefix $p \in \{0,1\}^{\leqslant \ell}$ of S. If $|p| > 0$, then we have a single child under the root, with a p-labelled edge going into it, which itself serves as the root to $R(S')$, where S' is the set of suffixes (after p) of S. If $|p| = 0$, then the root will have two children. Let $S = S_0 \cup S_1$, where S_0 and S_1 are sets of strings starting with 0 and 1, respectively, in S. The edges to the left and right children will be labelled by p_0 and p_1, respectively, where $p_0 \in \{0,1\}^{\leqslant \ell}$ is the longest common prefix of S_0 and $p_1 \in \{0,1\}^{\leqslant \ell}$ is the longest common prefix of S_1. The left child serves as a root to $R(S_0')$, where S_0' is the set of suffixes (after p_0) of S_0. Analogously, the right child serves as a root to $R(S_1')$, where S_1' is the set of suffixes (after p_1) of S_1.

Basic operations on radix trees The allowed basic operations on a radix tree are insertion and removal of an element. Classically, an attempt at inserting an

element already in S will result in the identity operation. Quantumly, we will instead allow for *toggling* an element in/out of S.

Representing a radix tree in memory We now consider how one might represent a radix tree in memory. For this purpose, suppose we wish to represent a radix tree $R(S)$ for some set $S \subseteq \{0,1\}^\ell$ of size $|S| \leqslant m$. Let us assume without loss of generality that m is a power of 2, and suppose we have at our disposal an array of $2m$ memory blocks.

Each memory block may be used to store a node of the radix tree. If we have a node in the tree, the contents of its corresponding memory block will represent a tuple (z, p_1, p_2, p_3). The value $z \in \{0,1\}^{\leqslant \ell}$ stores the label in the edge from the node's parent, the values $p_1, p_2, p_3 \in \{0, 1, \ldots, 2m\}$ are pointers to the (block storing the) parent, left child, and right child, respectively, or 0 if such an edge is absent.

It follows that each memory block is $O(\ell + \log m)$ bits long. In this way, we will represent $R(S)$ by a binary string of length $O(m(\ell+\log m))$. The root node is stored in the first block, empty blocks will be set to 0, and the only thing that needs to be specified is the *memory layout*, namely, in which block does each node get stored. For this purpose, let $\tau : R(S) \to [2m]$ be an injective function, mapping the nodes of $R(S)$ to the $[2m]$ memory blocks, so that $\tau(\text{root}) = 1$. For any $S \subseteq \{0,1\}^\ell$ of size $|S| \leqslant m$, we then let

$$R_\tau(S) \in \{0,1\}^{O(m(\ell+\log m))}$$

denote the binary string obtained by encoding $R(S)$ as just described.

BJLM's quantum radix tree We see now that although there is a unique radix tree $R(S)$ for each S, there is no obvious way of making sure that the representation of $R(S)$ in memory is also unique. However, this bijective correspondence between S and its memory representation is a requirement for quantum algorithms to use interference. The idea of Bernstein et al. [BJL+13], then, is to represent S using a superposition of *all possible layouts*. I.e., S is to be uniquely represented by the (properly normalised) quantum state:

$$\sum_\tau |R_\tau(S)\rangle.$$

The trick, then, is to ensure that this representation can be efficiently queried and updated. In their discussion of how this might be done, the BJLM paper [BJL+13] presents the broad idea but does not work out the details, whereas Jeffery's thesis [Jef14] glosses over several details and includes numerous bugs and omissions. To make their idea work, we make use of an additional data structure.

4.2.2 Prefix-sum tree

In our implementation of the *quantum prefix tree*, we will need to keep track of which blocks are empty and which are being used by a node. For this purpose, we will use a data-structure that is famously used to (near-optimally) solve the dynamic prefix-sum problem.

Definition 4.4. A *prefix-sum tree* is a complete rooted binary tree. Each leaf node is labelled by a value in $\{0,1\}$, and each internal node is labelled by the number of 1-valued leaf nodes descending from it.

Let $F \subseteq [\ell]$ for ℓ a power of 2. We use $P(F)$ to denote the prefix-sum tree where the i^{th} leaf node of the tree is labelled by 1 iff $i \in F$.

A prefix-sum tree $P(F)$ will be represented in memory by an array of $\ell - 1$ blocks of memory, holding the labels of the inner nodes of $P(F)$, followed by ℓ bits, holding the labels of the leaf nodes. The blocks appear in the same order as a breadth-first traversal of $P(F)$. Consequently, for every $F \in \{0,1\}^\ell$ there is corresponding binary string of length $(\ell - 1)\log \ell + \ell$ that uniquely describes $P(F)$.

We will overload notation, and use $P(F)$ to denote this binary string of length $(\ell - 1)\log \ell + \ell$.

Allocating and deallocating The idea now is to use the prefix tree as a memory allocator. We have $2m$ blocks of memory, and the set F will keep track of which blocks of memory are unused, or "free".

We would then like to have an operation that allocates one of the free blocks. To implement Bernstein et al.'s idea, the choice of which block to allocate is made in superposition over all possible free blocks. I.e., we would like to implement the following map U_{alloc} and also its inverse, U_{free}.

$$U_{\text{alloc}} : |P(F)\rangle|0\rangle|0\rangle \to \frac{1}{\sqrt{|F|}} \sum_{i \in F} |P(F \backslash \{i\})\rangle|i\rangle|0\rangle, \tag{4.3}$$

The second and third registers have $O(\log m)$ bits. We do not care for what the map does when these registers are non-zero, or when $F = \varnothing$. We will guarantee that this is never the case.

Note that each internal node of the prefix tree stores the number of elements of F that are descendants to that node. In particular, the root stores $|F|$. In order to implement U_{alloc}, we then start by constructing the state

$$\frac{1}{\sqrt{|F|}} \sum_{j=1}^{|F|} |j\rangle. \tag{4.4}$$

While this might appear to be simple, it actually requires us to use a gate

$$U_{\text{superpose}} : |k\rangle|0\rangle \mapsto \frac{1}{\sqrt{k}} \sum_{j=1}^{k} |k\rangle|j\rangle. \tag{4.5}$$

This is much like choosing a random number between 1 and a given number k on a classical computer. Classically, such an operation cannot be done exactly if all we have at our disposition are bitwise operations (since all achievable probabilities are then dyadic rationals). Quantumly, it is impossible to implement $U_{\text{superpose}}$ efficiently without error by using only the usual set of basic gates.

So the reader should take note: it is precisely this gate which adds error to BJLM's procedure. This gate can be implemented up to distance ε using $O(\log \frac{m}{\varepsilon})$ basic gates, where $2m$ is the maximum value that k can take. I.e., using so many gates we can implement a unitary U such that the spectral norm $\|U - U_{\text{superpose}}\| \leqslant \varepsilon$.[4]

[4]This is done by using Hadamard gates to get a superposition between 1 and the smallest power of 2 which is greater than $\frac{k}{\varepsilon}$, and then breaking this range into k equal intervals plus a remainder of size $< k$. The *remainder subspace* will have squared amplitude $\leqslant \varepsilon$.

We will need to choose $\varepsilon \approx \frac{1}{T}$, which is the inverse of the number of times such a gate will be used throughout our algorithm.

Once we have prepared state as in Equation 4.4, we may then use binary search, going down through the prefix tree to find out which location i corresponds to the j^{th} non-zero element of F. Using i, as we go up we can remove the corresponding child from $P(F)$, in $O(\zeta \cdot \log m)$ time, while updating the various labels on the corresponding root-to-leaf path. This requires the use of $O(\log m)$ work bits, which are $|0\rangle$ at the start and end of the operation. During this process, the register holding j is also reset to $|0\rangle$, by subtracting the element counts we encounter during the deletion process from this register. The inverse procedure U_{free} is implemented in a similar way.

4.2.3 Quantum radix tree

We may now define the quantum radix tree.

Definition 4.5 (Quantum Radix Tree)**.** Let ℓ and m be powers of 2, $S \subseteq \{0,1\}^\ell$ be a set of size $s = |S| \leqslant m$, and let $R(S)$ be the classical radix tree storing S. Then, the *quantum radix tree* corresponding to S, denoted $|R_Q(S)\rangle$ (or $|R_Q^{\ell,m}(S)\rangle$ when ℓ and m are to be explicit), is the state

$$|R_Q(S)\rangle = \frac{1}{\sqrt{N_S}} \cdot \sum_\tau |R_\tau(S)\rangle|P(F_\tau)\rangle,$$

where τ ranges over all injective functions $\tau : R(S) \to [2m]$ with $\tau(\text{root}) = 1$, of which there are $N_S = \frac{(2m-1)!}{(2m-|R(S)|)!}$ many, and $F_\tau = [2m] \backslash \tau(R(S))$ is the complement of the image of τ.

Basic operations on quantum radix trees The basic allowed operations on a quantum radix trees are look-up and toggle, where the toggle operation is analogous to insertion and deletion in classical radix tree. Additionally, we also define a swap operation which will be used to simulate a RAG gate.

Lemma 4.6. *Let $|R_Q(S)\rangle = |R_Q^{\ell,m}(S)\rangle$ denote a quantum radix tree storing a set $S \subseteq \{0,1\}^\ell$ of size at most m. We then define the following data structure operations.*

1. `Lookup`*. Given an element $e \in \{0,1\}^\ell$, we may check if $e \in S$, so for each $b \in \{0,1\}$, we have the map*

$$|e\rangle|R_Q(S)\rangle|b\rangle \mapsto |e\rangle|R_Q(S)\rangle|b \oplus (e \in S)\rangle.$$

2. `Toggle`*. Given $e \in \{0,1\}^\ell$, we may add e to S if S does not contain e, or otherwise remove e from S. Formally,*

$$|e\rangle|R_Q(S)\rangle \mapsto \begin{cases} |e\rangle|R_Q(S \cup \{e\})\rangle, & \text{if } e \notin S, \\ |e\rangle|R_Q(S \backslash \{e\})\rangle, & \text{if } e \in S. \end{cases}$$

3. `Swap`*. Given an element $e \in \{0,1\}^\ell$, $b \in \{0,1\}$ and a quantum radix tree storing a set S, we would like* `swap` *to be the following map,*

$$|e\rangle|R_Q(S)\rangle|b\rangle \mapsto \begin{cases} |e\rangle|R_Q(S \cup \{e\})\rangle|0\rangle, & \text{if } e \notin S \text{ and } b = 1, \\ |e\rangle|R_Q(S \backslash \{e\})\rangle|1\rangle, & \text{if } e \in S \text{ and } b = 0, \\ |e\rangle|R_Q(S)\rangle|b\rangle, & \text{otherwise.} \end{cases}$$

These operations can be implemented in worst case $O(\zeta \cdot \log m)$ time and will be error-free if we are allowed to use an error-free gate for $U_{superpose}$ (defined in Equation 4.5), along with other gates from set $\mathcal{Q}$.

Proof. Let $|b\rangle, |e\rangle$ denote the quantum states storing the elements $b \in \{0,1\}$ and $e \in \{0,1\}^\ell$, respectively. The data structure operations such as `lookup`, `toggle` and `swap` can be implemented reversibly in $O(\zeta \cdot \log m)$ time in the following way.

`Lookup` We wish to implement the following reversible map U_{lookup},

$$U_{lookup} : |e\rangle|R_Q(S)\rangle|b\rangle \mapsto |e\rangle|R_Q(S)\rangle|b \oplus (e \in S)\rangle. \tag{4.6}$$

We do it as follows. First note that, by Definition 4.5,

$$|R_Q(S)\rangle = \frac{1}{\sqrt{N_S}} \sum_\tau |R_\tau(S)\rangle|P(F_\tau)\rangle.$$

We will traverse $R_\tau(S)$ with the help of some auxiliary variables. Starting at the root node, we find the edge labelled with a prefix of e. If no such label is found then e is not present in $R_\tau(S)$. Otherwise, we traverse to the child reached by following the edge labelled by a prefix of e. Let us denote the label by L. If the child is a leaf node then terminate the process, stating that e is present in $R_\tau(S)$, else, recurse the process on e' and the tree rooted at that child node. Here e' is the binary string after removing L from e. When at some point we have determined whether $e \in S$ or not, we flip the bit b, or not. Eventually, we may conclude that $e \notin S$ before traversing the entire tree, at which point we skip the remaining logic for traversing $R_\tau(S)$ downwards (by using a control qubit). After we have traversed $R_\tau(S)$ downwards and determined whether $e \in S$, we need to undo our traversal, which we do by following the p_1 pointers (to the parent nodes) until the root is again reached, and the auxiliary variables are again set to 0.

Each comparison with the edge labels, at each traversed node, takes $O(\zeta)$ time. Hence, the entire procedure takes $O(\zeta \cdot \log m)$ time.

`Toggle` Let U_{toggle} denote the following map,

$$U_{toggle} : |e\rangle|R_Q(S)\rangle \to \begin{cases} |e\rangle|R_Q(S \backslash \{e\})\rangle, & \text{if } e \in S, \\ |e\rangle|R_Q(S \cup \{e\})\rangle, & \text{if } e \notin S \end{cases} \tag{4.7}$$

The `toggle` operation primarily consists of two main parts: the memory allocation or de-allocation, followed by insertion or deletion, respectively.

We again traverse $R_\tau(S)$ with the help of some auxiliary variables. We start with the root node of $R_\tau(S)$, and traverse the tree downwards until we know, as above, whether $e \in S$ or not. If $e \notin S$, we will know where we need to insert nodes into $R_\tau(S)$, in order to transform it into $R_\tau(S \cup \{e\})$. Below, we will explain in detail how such an insertion must proceed. It turns out that we may need to insert either one node, or two, but never more. We may use the work qubits to compute the contents of the memory blocks that will hold this new node (or new nodes). These contents are obtained by XORing the appropriate bits of e and the appropriate parent/child pointers of the nodes we are currently traversing in the tree.

We may then use the U_{alloc} gate (once or twice) to obtain the indices of the blocks that will hold the new node(s). We then use RAG gates to swap in the contents of these blocks into memory. A fundamental and crucial detail must now be observed: the index of the memory blocks into where we inserted the new nodes is now left as part of the work qubits. This cannot be and must be dealt with, because every work bit must be again set to zero at the end of the procedure. However, a copy of this index now appears as the child pointer (p_2 or p_3) of the parents of the nodes we just created, and these pointers can thus be used to zero out the index. It is then possible to traverse the tree upwards in order to undo the various changes we did to the auxiliary variables.

If $e \in S$, on the other hand, we then do the inverse procedure. We will then know which nodes need to be removed from $R_\tau(S)$ (it will be either one or two nodes). By construction, these nodes will belong to blocks not in F_τ. We begin by setting these blocks to zero by swapping the blocks into the workspace (using the RAG gate), XORing the appropriate bits of e and the appropriate child/parent pointers so the blocks are now zero, and swapping them back. These blocks will then be set to zero, and we are left with a state akin to the right-hand side of (Equation 4.3). We then use the U_{free} gate to *free* the blocks, i.e., add their indices to F_τ once again. At this point we can traverse the tree upwards once more, in order to reset the auxiliary variables to zero, as required.

We now give further detail on how one must update $R_\tau(S)$ in order to insert a new element e into S. We must create a node $N := (z, p_1, p_2, p_3)$ corresponding to the element e stored at the memory location assigned by U_{alloc} procedure. Let us denote the address by k. Start with the root node of $R_\tau(S)$. If e has no common prefix with any of the labels of the root's outgoing edges, which can only happen if the root has one child, then set z to e, p_1 pointing to the root node, and, p_2 and p_3 set to 0. Moreover, set the value of the root's p_2 pointer to k if node N ends up as the left child to the root, else set root's p_3 pointer to k. In the case when e has a common prefix with one of the labels of the root's outgoing edges, let us denote the label by L and the child node by C, then further two scenarios arise: either label L is completely contained in e, which if is the case then we traverse the tree down and run the insertion procedure recursively on e' (which is e after removing the prefix L) with the new root set C. In the case where label L is not completely contained in e, we create an internal node N' with its z variable set to the longest common prefix of e and L (which we denote by L'), p_1 pointing to root, p_2 pointing to C and p_3 pointing to N (or vice versa depending on whether node N gets to be the right or the left child). We run the U_{alloc} procedure again to get a memory location to store N'. Having done that, we now change the z value of node C to be the prefix of L after L', and the p_1 value of node C to be the memory location of N'. Additionally, we also set z of node N to be e', the suffix of e after L', and we let p_1, p_2, p_3 to be, respectively, a pointer to N', 0 and 0.

Each step in the traversal takes time $O(\zeta)$, for a total time of $O(\zeta \cdot \log m)$.

The procedure to update $R_\tau(S)$ in order to delete an element e from S is analogous to the insertion procedure mentioned above, which also can be implemented in $O(\zeta \cdot \log m)$ time.

`Swap` Let U_{swap} denote the following map,

$$U_{swap} : |e\rangle|R_Q(S)\rangle|b\rangle \mapsto \begin{cases} |e\rangle|R_Q(S \cup \{e\})\rangle|0\rangle, & \text{if } e \notin S \text{ and } b = 1, \\ |e\rangle|R_Q(S\backslash\{e\})\rangle|1\rangle, & \text{if } e \in S \text{ and } b = 0, \\ |e\rangle|R_Q(S)\rangle|b\rangle, & \text{otherwise.} \end{cases}$$

To implement U_{swap}, we first run the U_{lookup} on the registers $|e\rangle$, $|R_Q(S)\rangle$ and $|b\rangle$. Conditional on the value of register $|b\rangle$ (i.e., when $b = 1$), we run U_{toggle} on the rest of the registers. We then run U_{lookup} again to attain the desired state. To summarise, the unitary $U_{swap} = U_{lookup} \cdot C_{toggle} \cdot U_{lookup}$, where C_{toggle} is controlled version of U_{toggle} (as per Lemma 2.4). Thus, the `swap` procedure takes a total time of $O(\zeta \cdot \log m)$. □

An error-less, efficient implementation of the unitary $U_{\text{superpose}}$ is impossible by using only the usual sets of basic gates. Furthermore, it is unreasonable to expect to have an error-free $U_{\text{superpose}}$ at our disposal. However, as we explained in page 87, there is a procedure to implement $U_{\text{superpose}}$ using gates from the gate set $\mathcal{B}$ up to spectral distance ε, using only $O(\log \frac{m}{\epsilon})$ gates.

Corollary 4.7. *Let $|R_Q(S)\rangle = |R_Q^{\ell,m}(S)\rangle$ denote a quantum radix tree storing a set $S \subseteq \{0,1\}^\ell$ of size at most m. The data structure operations `look-up`, `toggle` and `swap`, as defined in the statement of Lemma 4.6 can be implemented in $O(\zeta \cdot \log \frac{m}{\epsilon})$ time and ϵ probability of error using gates from the gate set $\mathcal{Q}$. Here ζ is the number of gates required from set $\mathcal{Q}$ to do various basic operations on a logarithmic number of qubits.*

4.2.4 The simulation

Recall from Section 2.3.4 that we take ζ to be the number of gates required to do various basic operations on a logarithmic number of qubits. In our use below, it never exceeds $O(\log M)$.

Theorem 4.8. *Let T, W, $m < M = 2^\ell$ be natural numbers, with M and m both powers of 2, and let $\varepsilon \in [0, 1/2)$. Suppose we are given an m-sparse QRAM algorithm using time T, W work qubits and M memory qubits, that computes a Boolean relation F with error ε.*

Then we can construct a QRAM algorithm which computes F with error $\varepsilon' > \varepsilon$, and runs in time $O(T \cdot \log(\frac{T}{\varepsilon' - \varepsilon}) \cdot \zeta)$, using $W + O(\log M)$ work qubits and $O(m \log M)$ memory qubits.

Proof. Let $\mathcal{C} = (n, T, W, M, C_1, \ldots, C_T)$ be the circuit of the given m-sparse QRAM algorithm computing a relation F with error ε and, let the state of the algorithm at every time-step t, when written in the computational basis be

$$|\psi_t\rangle = \sum_{u \in \{0,1\}^W} \sum_{v \in \binom{[M]}{\leqslant m}} \alpha_{u,v}^{(t)} \cdot \underbrace{|u\rangle}_{\text{W qubits}} \otimes \underbrace{|v\rangle}_{\text{M qubits}} \tag{4.8}$$

where the set $\binom{[M]}{\leqslant m}$ denotes all vectors $v \in \{0,1\}^M$ such that $|v| \leqslant m$. Using the description of $\mathcal{C}$ and the fact that this algorithm is m-sparse we will now construct another QRAM algorithm $\mathcal{C}'$ with the promised bounds. The algorithm C' will

have $w' = W + O(\log M)$ work bits, and $O(m \log M)$ memory bits. The memory is to be interpreted as an instance $|R_Q(S)\rangle$ of the quantum radix tree described above. Then $|v\rangle$ will be represented by the quantum radix tree $|R_Q(S_v)\rangle$, where $S_v = \{i \in [M] \mid v_i = 1\}$ is the set of positions where $v_i = 1$, so that each position $i \in [M]$ is encoded using a binary string of length ℓ.

The simulation is now simple to describe. First, the quantum radix tree is initialised. Then, each non-RAG instruction $C_i \in \mathcal{C}$ operating on the work qubits of $\mathcal{C}$ is applied in the same way in $\mathcal{C}'$ to same qubits among the first W qubits of $\mathcal{C}'$. Each RAG instruction, on the other hand, is replaced with the U_{swap} operation, applied to the the quantum radix tree. The extra work qubits of $\mathcal{C}'$ are used as ancillary qubits for these operations, and we note that they are always returned to zero.

If we assume that the U_{swap} operation can be implemented without error, we then have a linear-space isomorphism between the two algorithms' memory space, which maps the state $|\psi_t\rangle$ of $\mathcal{C}$ at each time step t to the state $|\phi_t\rangle$ of $\mathcal{C}'$ after t simulated steps:

$$|\phi_t\rangle = \sum_{u,v} \alpha_{u,v}^{(t)} \cdot \underbrace{|u\rangle}_{W} \otimes \underbrace{|0\rangle}_{O(\log M)} \otimes \underbrace{|R_Q(S_v)\rangle}_{O(m \log M)}.$$

Thus, if U_{swap} could be implemented without error, we could have simulated $\mathcal{C}$ without additional error. Otherwise, as per Corollary 4.7, we may implement the U_{swap} unitary with an error parameter $\Omega(\frac{\varepsilon'-\varepsilon}{T})$, resulting in a total increase in error of $\varepsilon' - \varepsilon$, and an additional time cost of $O(T \log \frac{T}{\varepsilon'-\varepsilon})$. □

4.3 Simplifications of previous works

It is possible to use our main theorem to simplify the presentation of the following three results: Ambainis' quantum walk algorithm for solving the k-Element Distinctness problem [Amb07], Aaronson et al.'s quantum algorithms for the Closest Pair problem (Closest Pair), and our very own work on Fine-Grained Complexity via Quantum Walks [BLP+22a] (Chapter 5 of this thesis).

All these results use quantum walk together with complicated, space-efficient, history-independent data structures. As we will see, it is possible to replace these complicated data structures with simple variants of the prefix-sum tree (Section 4.2.2), where the memory use is sparse, and then invoke the main theorem of this chapter.

4.3.1 Ambainis' walk algorithm for Element Distinctness

Ambainis' description and analysis of his data structure is complicated, and roughly 6 pages long, whereas a presentation of his results with a simple data structure and an appeal to our theorem requires less than 2 pages, as we will now see. Also, the presentation of the algorithm is considerably muddled by the various difficulties and requirements pertaining to the more complicated data structure. In a presentation of his results that would then appeal to Theorem 4.1, we have a very clear separation of concerns.

Ambainis' algorithm is a $\widetilde{O}(n^{\frac{k}{k+1}})$-time solution to the following problem:

Definition 4.9 (k-Element Distinctness)**.** Given a list L of n integers in Σ are there k elements $x_{i_1}, \ldots, x_{i_k} \in L$ such that $x_{i_1} = \cdots = x_{i_k}$.

Ambainis' algorithm for k-Element Distinctness [Amb07] is quantum walk algorithm on a Johnson graph $J(n, r)$ with $r = n^{k/k+1}$ and runs in $\widetilde{O}(n^{k/k+1})$ time. The crucial ingredient in making the algorithm time efficient is the construction of data-structure which can store a set $S \subseteq [n] \times \Sigma$ of elements of size r, under efficient insertions and removals, so that one may efficiently query at any given time whether there exist k elements $(i_1, x_1), \ldots, (i_k, x_k)$ in S with distinct indices $i_1, \ldots, i_k$ but equal labels $x_1 = \cdots = x_k$. Ambainis makes use of skip-lists and hash tables, ensuring that all operations run in $O(\log^4(n + |\Sigma|))$ time. However, if one does not care about space-efficiency, there is a much simpler data structure that serves the same purpose. The following definition is illustrated in Figure 4.2.

Definition 4.10. Let $S \subseteq [n] \times \Sigma$, with $|S| = r$ and $|\Sigma| = n^{O(1)}$ a power of 2, and such that every $i \in [n]$ appears in at most one pair $(i, x) \in S$. The *k-element-distinctness tree* that represents S, denoted $T_k(S)$, is a complete rooted binary tree with $|\Sigma|$ leaves. Each leaf node $x \in \Sigma$ is labelled by a bit vector $\mathtt{B}_x \in \{0,1\}^n$ and a number $\mathtt{count}_x \in \{0, \ldots, n\}$, so that $\mathtt{B}_x[i] = 1$ iff $(i, x) \in S$, and the $\mathtt{count}_x$ is the Hamming weight of $\mathtt{B}_x$. Each internal node w is labelled by a bit $\mathtt{flag}_w \in \{0,1\}$ which indicates whether there exists a leaf x, descendent of w, with $\mathtt{count}_x \geqslant k$.

Memory Representation A k-element-distinctness tree is represented in the memory by an array of $|\Sigma| - 1$ bits of memory holding the flags of the internal nodes, followed by $|\Sigma|$ blocks of $n + \lceil \log n \rceil$ bits of memory each, holding the labels of the leaf nodes. The blocks appear in the same order as a breadth-first traversal of $T_k(S)$. Consequently, for every $S \subseteq [n] \times \Sigma$ there is a corresponding binary string of length $|\Sigma| - 1 + (n + \lceil \log n \rceil)|\Sigma|$ that uniquely encodes $T_k(S)$. Crucially, if $|S| = r$, then at most $O(r(\log \Sigma + \log n))$ of these bits are 1. So for $|\Sigma| = \mathsf{poly}(n)$, the encoding is $\widetilde{O}(r)$-sparse.

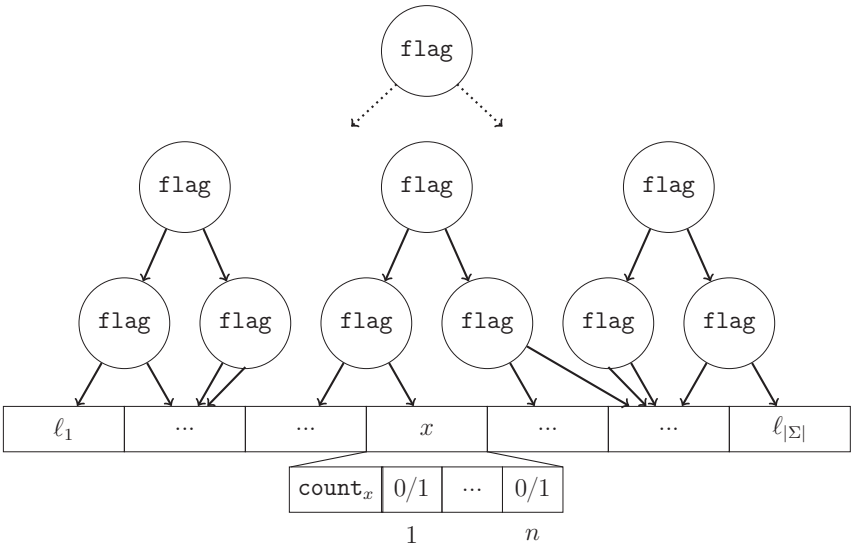

Figure 4.2: Data structure for the k-Element Distinctness problem.

Implementation of data structure operations It is clear from the definition of k-element-distinctness tree and its memory representation that a tree $T_k(S)$ represents a set $S \subseteq [n] \times \Sigma$ in a history-independent way. We will now argue that all the required data structure operations take $O(\log n)$ time in the worst case. Let (i, x) denote an element in $[n] \times \Sigma$.

- **Insertion** To insert (i, x) in the tree, first increase the value of the $\texttt{count}$ variable of the leaf x, and set $\texttt{B}_x[i] = 1$. Then, if $\texttt{count} \geqslant k$, set $\texttt{flag}_w = 1$ for all w on the root-to-x path. This update requires $O(\log n)$ time as $|\Sigma| = \mathsf{poly}(n)$.

- **Deletion** The procedure to delete is similar to the insertion procedure. To delete (i, x) in the tree, first decrease the value of the $\texttt{count}_x$ and set $\texttt{B}[i] = 0$. If $\texttt{count} < k$, then, for all w on the root-to-x path which do not have both children w_0, w_1 with $\texttt{flag}_{w_0} = \texttt{flag}_{w_1} = 1$, set $\texttt{flag}_w = 0$. This requires $O(\log n)$ time.

- **Query** To check if the tree has k distinct indices with the same x, we need only check if $\texttt{flag}_{\text{root}} = 1$, which takes $O(1)$ time.

Runtime, error and memory usage Using the above data-structure, the runtime of Ambainis' algorithm is now $\widetilde{O}(n^{\frac{k}{k+1}})$ time. The total memory used is $O(n|\Sigma|)$ bits. However, note that at any point of time in any branch of computation Ambainis' walk algorithm stores sets of size $r = O(n^{\frac{k}{k+1}})$. Hence their algorithm with this data structure is a $\widetilde{O}(n^{\frac{k}{k+1}})$-sparse algorithm. Thus, invoking Theorem 4.1 we conclude the following.

Corollary 4.11. *There is a bounded-error QRAM algorithm that computes k-Element Distinctness in $\widetilde{O}(n^{k/k+1})$ time using $\widetilde{O}(n^{k/k+1})$ memory qubits.*

4.3.2 Quantum algorithms for Closest-Pair and related problems

The paper of Aaronson et al. [ACL+20] provides quantum algorithms and conditional lower-bounds for several variants of the Closest Pair problem (Closest Pair).

Let $\Delta(a, b) = \|a - b\|$ denote the Euclidean distance. We then describe the Closest Pair problem under Euclidean distance Δ, but we could have chosen any other metric Δ in d-dimensional space which is strongly-equivalent to the Euclidean distance (such as ℓ_p distance, Manhattan distance, ℓ_∞, *etc*).

Definition 4.12 (Closest Pair ($\mathsf{CP}(n, d)$) problem)**.** In the $\mathsf{CP}(n, d)$ problem, we are given a list P of n distinct points in $\mathcal{R}^d$, and wish to output a pair $a, b \in P$ with the smallest $\Delta(a, b)$.

We may also define a threshold version of CP.

Definition 4.13. In the TCP(n, d) problem, we are given a set $P = \{p_1, \ldots, p_n\}$ of n points in $\mathbb{R}^d$ and a threshold $\varepsilon \geqslant 0$, and we wish to find a pair of points $a, b \in P$ such that $\Delta(a, b) \leqslant \varepsilon$, if such a pair exists.

For simplicity, so we may disregard issues of representation of the points, we assume that all points are specified using $O(\log n)$ bits of precision. By translation, we can assume that all the points lie in the integer hypercube $[L]^d$ for some $L = \mathsf{poly}(n)$, and that $\delta \in [L]$, also.

It is then possible to solve CP by running a binary search over the (at most n^2) different values of $\delta \in \{\Delta(p_i, p_j) \mid i, j \in [n]\}$ and running the corresponding algorithm for TCP. This will add an additional $O(\log n)$ factor to the running time.

The TCP(n, d) problem is a query problem with certificate complexity 2. If one is familiar with quantum walks, it should be clear that we may do a quantum walk on the Johnson graph over n vertices, to find a pair with the desired property, by doing $O(n^{2/3})$ queries to the input. Again, if one is familiar with quantum walks, one will realise that, in order to implement this walk efficiently, we must dynamically maintain a set $S \subseteq [n]$, and at each step in the quantum walk, we must be able to add or remove an element i to S, and answer a query of the form: does there exist a pair $i, j \in S$ with $\Delta(p_i, p_j) \leqslant \varepsilon$?

The only difficulty, now, is to implement an efficient data structure that can dynamically maintain S in this way, and answer the desired queries, while being time and space efficient. Aaronson et al. construct a data-structure which can store a set $S \subseteq [n] \times [L]^d$ of points of size r, under efficient insertions and removals, so that one may query at any given time whether there exist two points in S which are ε-close. They do so by first discretizing $[L]^d$ into a hypergrid of width $\varepsilon/\sqrt{d}$, as explained below, and then use a hash table, skip list, and a radix tree to maintain the locations of the points in the hypergrid.

The presentation of the data structure in the paper is roughly 6 pages long, and one must refer to Ambainis' paper for the error analysis, which is absent from the paper. As we will see, a simple, sparse data structure for the same purpose can be described in less than 2 pages, and then an appeal to Theorem 4.1 gives us the same result up to log factors.

Discretization We discretize the cube $[L]^d$ into a hypergrid of width $w = \frac{\varepsilon}{\sqrt{d}}$, and let $\mathrm{id}(p)$ denote the box containing p in this grid. I.e., we define a function $\mathrm{id}(p) : [L]^d \to \{0,1\}^{\lceil d \log(L/\varepsilon) \rceil}$ by

$$\mathrm{id}(p) = (\lfloor p(1)/w \rfloor, \ldots, \lfloor p(d)/w \rfloor) \qquad \text{(represented in binary)}.$$

Let $\Sigma = \{0,1\}^{\lceil d \log(L/\varepsilon) \rceil}$ denote the set of all possible boxes. We say that two boxes $g, g' \in \Sigma$ are neighbours if

$$\sqrt{\sum_{i=1}^{d} \|g(i) - g'(i)\|^2} \leqslant \sqrt{d}.$$

A loose estimate will show there can be at most $(2\sqrt{d}+1)^d$ neighbours for any box. This method of discretization ensures the following crucial property:

Observation 4.14 (Observation 45 [ACL+20])**.** *Let p, q be any two distinct points in $[0, L]^d$, then*

1. *if* $\mathrm{id}(p) = \mathrm{id}(q)$*, then* $\Delta(p, q) \leqslant \varepsilon$*, and*

2. *if* $\Delta(p,q) \leqslant \varepsilon$, *then* $\mathrm{id}(p)$ *and* $\mathrm{id}(q)$ *are neighbours.*

From Observation 4.14, it follows that $i, j \in [n]$ exist with $\Delta(p_i, p_j) \leqslant \varepsilon$, if and only if we have one of the following two cases:

- Either there is such a pair i, j with $\mathrm{id}(p_i) = \mathrm{id}(p_j)$.
- Or there is no such pair, and then there must exist two neighbouring boxes $\mathrm{id}(i)$ and $\mathrm{id}(j)$, each containing a single point, with $\Delta(p_i, p_j) \leqslant \varepsilon$.

We now describe the data structure itself. Let us assume without loss of generality that n is a power of 2.

Definition 4.15 (Data Structure for CP)**.** Let $S \subseteq [n] \times \Sigma$, with $|S| = r$, and such that every $i \in [n]$ appears in at most one pair $(i, x) \in S$. The *closest-pair tree* that represents S, denoted by $T_{CP}(S)$, is a complete rooted binary tree with $|\Sigma|$ leaves. Each leaf node $x \in \Sigma$ is labelled by a number $\mathtt{external}_x \in \{0, \ldots, n\}$, and a prefix-sum tree $P(S_x)$ representing the set $S_x = \{i \in [n] \mid (i, x) \in S\}$. Each internal node w is labelled by a bit $\mathtt{flag}_w \in \{0, 1\}$. These labels obey the following rules:

- If $|S_x| = 1$, then $\mathtt{external}_x$ is the number of boxes $y \neq x$, which are neighbours of x, and which have $|S_y| = 1$ and $\Delta(p_i, p_j) \leqslant \varepsilon$ for the (unique) $j \in S_y$.
- If $|S_x| \geqslant 2$, then $\mathtt{external}_x = 0$.
- The $\mathtt{flag}_w = 1$ if any of the children x descendants to the internal node w have either $|S_x| \geqslant 2$ or $|S_x| = 1$ and $\mathtt{external}_x \geqslant 1$.

It follows from the above discussion that there exist two elements $(i, x), (j, y) \in S$ with $\Delta(p_i, p_j) \leqslant \varepsilon$ if and only if $\mathtt{flag}_{\mathrm{root}} = 1$ in $T_{CP}(S)$. We now show how to efficiently maintain $T_{CP}(S)$ under insertions and removals.

Memory representation A TCP tree is represented in the memory by an array of $|\Sigma|-1$ bits of memory holding the flags of the internal nodes, followed by $|\Sigma|$ blocks of $n \log n + n$ bits of memory each, holding the labels of the leaf nodes. The blocks appear in the same order as a breadth-first traversal of $T_{CP}(S)$. Consequently, for every $S \subseteq [n] \times \Sigma$ there is a corresponding binary string of $|\Sigma| - 1 + (n \log n + n)|\Sigma|$ that uniquely encodes $T_{CP}(S)$. Crucially, if $|S| = r$, then at most $O(r(\log|\Sigma| + \log n))$ of these bits are 1. Since $|\Sigma| = L^{O(d)} = \mathsf{poly}(n)$ (recall $d = O(1)$), the encoding is $\widetilde{O}(r)$-sparse.

Implementation of data structure operations It is clear from the definition of TCP tree and its memory representation that a tree $T_{CP}(S)$ represents a set $S \subseteq [n] \times \Sigma$ in a history-independent way. We will now argue that all the required data structure operations take $O(\log n)$ time in the worst case. For every $(i, x) \in [n] \times [L]^d$ there is a corresponding $(i, z) \in [n] \times \Sigma$, with $z = \mathrm{id}(x)$, stored in the data structure.

- **Insertion** To insert (i, x) in the tree, first go to the memory location corresponding to leaf x. Begin by inserting i in the prefix-sum tree $P(S_x)$. Then three cases arise

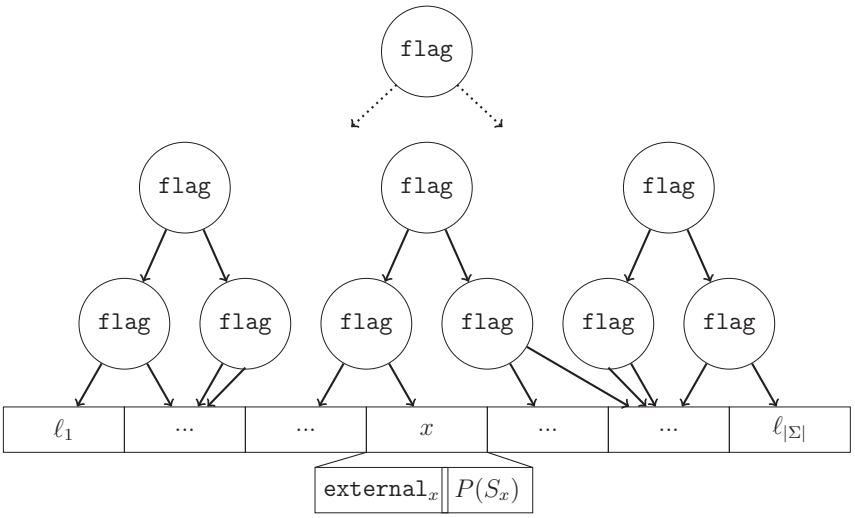

Figure 4.3: Data structure for the CP problem.

- If $|S_x| = 1$ then for every neighbour y of x with $|S_y| = 1$ do the following: using the prefix-sum tree at leaf y obtain the only non-zero leaf index j of $P(S_y)$. This operation takes $\log n$ time. Then check if $\Delta(p_i, p_j) \leqslant \varepsilon$, if yes then increase the values of both $\texttt{external}_x$ and $\texttt{external}_y$ by 1. If this caused $\texttt{external}_y > 0$ then set $\texttt{flag}_w = 1$ for all internal nodes w on the path from leaf y to the root of $T_{CP}(S)$.

  After going over all neighbours, check if $\texttt{external}_x \geqslant 1$, if it is then set $\texttt{flag}_w = 1$ for all internal nodes w on the path from leaf x to the root of $T_{CP}(S)$. This process takes at most $(2\sqrt{d}+1)^d \log n$ time as there will be at most $(2\sqrt{d}+1)^d$ neighbours, which is $O(\log n)$ for $d = O(1)$.

- If $|S_x| = 2$ using the prefix-sum tree $P(S_x)$ obtain the only other non-zero leaf index $i' \neq i$ of $P(S_x)$. Then for all neighbours y of x with $|S_y| = 1$ do the following: using the prefix-sum tree $P(S_y)$ obtain the only non-zero index j of $P(S_y)$. Check if $\Delta(p_{i'}, p_j) \leqslant \varepsilon$, and if so decrease the value of $\texttt{external}_y$ by 1. If that results in making $\texttt{external}_y = 0$ then set $\texttt{flag}_w = 0$ for the parent of y, unless the other child y' of the parent of y has $|S_{y'}| \geqslant 2$ or $\texttt{external}_{y'} \geqslant 1$. Likewise, among all the internal nodes w that are on the path from the root to y's parent, update the $\texttt{flag}_w$ accordingly, i.e., set $\texttt{flag}_w = 1$ if any child u of w has $\texttt{flag}_u = 1$, and otherwise set $\texttt{flag}_w = 0$.

  Having done that, set $\texttt{external}_x = 0$ and set $\texttt{flag}_w = 1$ for all internal nodes w from leaf x to the root $T_{CP}(S)$. This process also takes $O(\log n)$ time (when d is a constant).

- If $|S_x| > 2$ then do nothing.

- **Deletion** The procedure to delete is similar to the insertion procedure.

- **Query** To check if the tree has a pair $(i, x), (j, y) \in S$ such that $\Delta(p_i, p_j) \leqslant \varepsilon$, we need only check if $\texttt{flag}_{\text{root}} = 1$, which takes $O(1)$ time.

Runtime, error and memory usage Using the above data-structure, the runtime of this TCP algorithm is now $\widetilde{O}(n^{\frac{2}{3}})$ time. The total memory used is $\widetilde{O}(n|\Sigma|)$ bits. However, note that at any point of time in any branch of computation this algorithm stores sets of size $r = O(n^{\frac{2}{3}})$. Hence their algorithm with this data structure is a $\widetilde{O}(n^{\frac{2}{3}})$-sparse algorithm. Thus, invoking Theorem 4.1 we conclude the following.

Corollary 4.16. *There is a bounded-error QRAM algorithm that computes TCP in* $\widetilde{O}(n^{2/3})$ *time using* $\widetilde{O}(n^{2/3})$ *memory qubits.*

4.3.3 Fine-grained complexity via quantum walks

Results presented in Chapter 5 show how the quantum 3SUM conjecture, which states that there exists no truly sublinear quantum algorithm for 3SUM, can be used to imply several other quantum time lower bounds. The reductions use quantum walks together with complicated space-efficient data structures. We had already realised, when writing the paper [BLP+22a], that simple yet space-inefficient data structures could be used instead, and will include this observation in the next chapter, so we will not repeat it here.

Chapter 5

Fine-Grained Complexity via Quantum Walks

Chapter summary In this chapter, we further extend the theory of fine-grained complexity to the quantum setting. A fundamental conjecture in the classical setting states that the 3SUM problem cannot be solved by (classical) algorithms in time $O(n^{2-\varepsilon})$, for any $\varepsilon > 0$. We formulate an analogous conjecture, the Quantum 3SUM Conjecture, which states that there exist no sublinear $O(n^{1-\varepsilon})$-time quantum algorithms for the 3SUM problem.

Based on the Quantum 3SUM Conjecture, we show new time lower bounds for several computational problems. Most of our lower bounds are optimal, in the sense that they match known upper bounds, and hence they imply tight limits on the quantum speedup that is possible for these problems.

These results are proven by adapting to the quantum setting known classical fine-grained reductions from the 3SUM problem. This adaptation is not trivial, however, since the original classical reductions require pre-processing the input in various ways, e.g. by sorting it according to some order, and this pre-processing (provably) cannot be done in sublinear quantum time.

We overcome this bottleneck by combining a quantum walk with a classical dynamic data-structure having a certain "history-independence" property. This type of construction has been used in the past to prove upper bounds, and here we use it for the first time as part of a reduction. This general proof strategy allows us to prove tight lower bounds on several computational-geometry problems, on Convolution-3SUM and on the 0-Edge-Weight-Triangle problem, conditional on the Quantum 3SUM Conjecture.

We believe that this proof strategy will be useful in proving tight (conditional) lower bounds, and limits on quantum speed-ups, for many other problems.

This chapter is based on the following two papers:

[BLP+22a] Harry Buhrman, Bruno Loff, Subhasree Patro, Florian Speelman. Limits of Quantum Speed-Ups for Computational Geometry and Other Problems: Fine-Grained Complexity via Quantum Walks. In *Proceedings of the 13th Innovations in Theoretical Computer Science Conference (ITCS 2022), QIP 2022.*

[BLP+22b] Harry Buhrman, Bruno Loff, Subhasree Patro, Florian Speelman. Memory Compression with Quantum Random-Access Gates. In *Proceedings of the 17th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2022).*

5.1 Introduction

The world is investing in quantum computing because of so-called *quantum speed-ups*: quantum algorithms can solve many computational problems faster than their classical counterparts. However, as also witnessed in earlier chapters, the amount of speed-up that is possible varies among different computational problems. It is expected that quantum computers will remain an expensive resource for a long time, and the extent to which a quantum speed-up is possible, or not possible, may one day be a key factor in deciding whether or not to invest in the use of quantum computation, for example in an industrial setting.

Such a consideration is not a mere abstraction. It is folk knowledge among researchers in quantum error correction that current error-correction techniques are so costly, meaning the constant-factor overhead they impose is so great, that quadratic quantum speed-ups will offer no advantage when compared to their classical counterparts. Babbush et al. [BMN+21] consider N-qubit quantum algorithms that work by making quantum calls to certain primitives, so that a quantum algorithm will do, e.g., M calls, and the corresponding classical algorithm will do M^2 calls, to the same primitive. Here $M \gg N \approx 100$ (so, e.g., we might using Grover to search for a satisfying assignment to a CNF with roughly 100 variables). They then estimate how large M must be, in order for quantum computers to offer a significant advantage over their classical counterparts, and conclude:

> *[We find that, even when] using state-of-the-art surface code constructions under a variety of assumptions, (...) quadratic speedups will not enable quantum advantage on early generations of fault-tolerant [quantum computers] unless there is a significant improvement in how we would realise quantum error-correction.*

It can further be said that the estimates appearing in [BMN+21] are extremely generous on the quantum side, in many respects. So, even allowing for incremental improvements to current quantum error correction, improvements in qubit technology, and so forth, this *uselessness* of quadratic quantum speedups is likely to assert itself in practice, for decades to come.

It is therefore essential, also from this perspective, to understand how much quantum speed-up is possible for specific computational problems (for example, so as not to overstate the potential of early-generation quantum computers). For this purpose, we would need to have tight upper and lower bounds on both classical and quantum algorithms.

Furthering the study of quantum fine-grained complexity, in this chapter we explore quantum fine-grained reductions to derive quantum time lower bounds for several problems in P, conditioned on a natural, conjectured quantum hardness for the 3SUM problem. These lower bounds will often tightly match upper bounds given by known quantum algorithms, and similar tight upper and lower bounds have also been proven in the classical setting. Together, these tight classical and quantum bounds are finally able to tell us exactly how much quantum speed-up is possible for various problems, which is the main goal of this line of research. For the problems we study, we will conclude that a quadratic speed-up is, in fact, the best possible.

5.1.1 The conjectured hardness of 3SUM

The 3SUM problem is defined as follows: given as input a list S of n integers and we wish to know if there exist a, b, c in S such that $a + b + c = 0$. There is a simple classical algorithm that solves this problem in $\widetilde{O}(n^2)$ time: sort S, then for every pair $(a, b) \in S \times S$, use binary search to check if $-(a + b) \in S$. This entire process takes $O(n^2 \log n)$ time. There is also a slightly less trivial algorithm that can solve 3SUM in $O(n^2)$ time.[1] Unfortunately, even after many years of interest in the problem, the exponent has not been reduced. The conjecture naturally arises that there is no $\epsilon > 0$ such that 3SUM can be solved in $O(n^{2-\epsilon})$ classical time. We refer to this conjecture as the Classical 3SUM Conjecture. Using this conjecture, one can derive conditional classical lower bounds for a vast collection of computational geometry problems, dynamic problems, sequence problems, etc. [GO95; VW13; Pat10; Vas15].

The Classical 3SUM Conjecture, however, is no longer true in the quantum setting, as there is a faster *quantum* algorithm for 3SUM: we may use Grover search as a subroutine in the $\widetilde{O}(n^2)$ classical algorithm to solve the problem in $\widetilde{O}(n)$ quantum time. Apart from this quadratic speedup, no further improvement to the quantum-time upper bound is known. It is worth mentioning that there is a sub-linear $O(n^{3/4})$ quantum *query* algorithm for computing 3SUM [Amb07; CE05] — with a matching lower bound of $\Omega(n^{3/4})$ [BŠ13] — this query algorithm, however, is not time efficient. Consequently, it was conjectured [AL20] that the 3SUM problem cannot be solved in sub-linear quantum time in the QRAM model.[2]

Conjecture 5.1 (Quantum 3SUM Conjecture [AL20])**.** There does not exist a $\delta > 0$ such that 3SUM on a list of n integers can be solved in $O(n^{1-\delta})$ time using a quantum computer.

It is then natural to try to extend the classical 3SUM-based lower bounds to the quantum setting, and one may at first expect this task to be a simple exercise. However, one soon realises that none of the existing classical reductions can be easily adapted to the quantum regime. Indeed, most of the existing classical reductions begin by pre-processing the input in some way, e.g., by sorting it according to some ordering, and this pre-processing turns out to be essential for the reduction to work efficiently. This is not an issue in the classical setting, as the classical conjectured lower bound for 3SUM is quadratic. Hence, the classical reductions can accommodate any pre-processing of the input that takes sub-quadratic time, such as sorting. However, this pre-processing becomes problematic in the quantum setting, since here we will need a sublinear-time quantum reduction, and even simple sorting requires linear time on a quantum computer [HNS01].

We present a workaround for this problem. The idea of the proof is to adapt Ambainis' quantum walk algorithm for element distinctness [Amb07]. For example, to enable reductions that need a sorted input, then instead of having the reduction sort the entire list, we combine a data structure for dynamic sorting together with a quantum walk algorithm. As we will show, this approach only needs the reduction to

[1] The algorithm is as follows: sort S in $O(n \log n)$ time, then for every $a \in S$ check if there exist elements $u, -v \in a + S, S$, respectively, such that $u = -v$. This can be done by simultaneously traversing the lists $a + S$ and S in $O(n)$ time. Repeat this process for all $a \in S$. Hence, the algorithm takes a total of $O(n^2)$ time.

[2] There is no formal definition of a model of computation in [AL20], therefore, we conjecture the same in the QRAM model we define in preliminaries Chapter 2.

sort a small part of the input and allows us to show that 3SUM remains hard, even when the entire input is sorted. As we will see, this idea can be extended to allow for any "structuring" of the input (not just sorting) which can be implemented by a dynamic data structure obeying a certain "history-independence" property. The proof will be sketched in Section 5.1.2.

This *quantum-walk plus data-structure* proof strategy has been used to prove upper bounds on other problems (e.g., for the closest-pair problem [ACL+20]), and here we use it for the first time as part of a reduction in order to obtain a lower bound. We expect that the same strategy will be applicable to other quantum fine-grained reductions, and our hope is that this will give rise to a landscape of results, that establish (conditional) tight lower bounds for quantum algorithms. This, in turn, will precisely answer the question of how much quantum speed-up is possible for a variety of computational problems.

Using this strategy we are able to show that various "structured" versions of 3SUM are as hard as the original (unstructured) 3SUM problem, even in the quantum case. Once we have shown that these structured versions of 3SUM are hard, we may then construct direct quantum adaptations of the classical reductions, to show the quantum hardness of several computational-geometry problems (Sections 5.3 and 5.4), and of the Convolution-3SUM and 0-Edge-Weight-Triangle problems (Section 5.5). This enables us to prove quantum time lower bounds for these problems, conditioned on the Quantum 3SUM Conjecture.

5.1.2 Main idea: reductions via quantum walks

The Classical 3SUM Conjecture states that there is no sub-quadratic classical algorithm to solve the 3SUM problem. However, the statement of this conjecture can be shown to be equivalent to the same statement for a promise version of 3SUM where the input S is sorted. That is because, if there was a sub-quadratic algorithm for 3SUM on sorted inputs, then given any (unsorted) input one can first sort the entire input with additional $O(n \log n)$ pre-processing time, and then use the sub-quadratic algorithm for sorted 3SUM, resulting in a sub-quadratic algorithm for unsorted 3SUM. In fact, one can make a more general statement in this regard. An input to the 3SUM problem is a list $S \in \Sigma^n$ of n integers (possibly with repetitions). One may consider a family $\{q_i\}$ of *queries*, i.e., each $q_i : \Sigma^n \to A_i$ is a function on lists of integers, for some set A_i of possible answers to the query. (For example, $q_i(S)$ could output the i^{th} smallest integer in S.) We may then ask about static data structures that allow us to efficiently answer these queries. (For example, we may consider the sorted version of S to be a data structure that allows us to efficiently obtain the i^{th} smallest element of S.) Then we may generally state that, if it is possible to preprocess an input S in sub-quadratic time to produce a static data structure that allows us to answer any query in $n^{o(1)}$ time, then the "structured" variant of Classical 3SUM Conjecture, where we give the algorithm access to all the queries $q_i(S)$ for free, is equivalent to the original version of Classical 3SUM Conjecture.

Most of the known fine-grained reductions from 3SUM, in the classical setting, can be explained in the following way: one first shows that a certain "structured" variant of 3SUM is just as hard as the original 3SUM problem, and then one reduces the structured variant of 3SUM to another problem. While for some reductions [GO95] require the input list to be sorted in the usual order of the integers,

other reductions require the input to be structured in some other way, for example, reductions in [Pat10; VW13] require that the elements are hashed into buckets and every element in the bucket can be accessed efficiently.

The reduction from "unstructured" to "structured" 3SUM is usually trivial to do in classical sub-quadratic time, but not so in quantum sub-linear time (e.g., a quantum computer cannot sort in sublinear time [HNS01]). This is the main difficulty in translating the classical reductions to the quantum setting.

Our main observation is that, if a certain analogous *dynamic* data-structure problem can be solved efficiently by a dynamic data-structure possessing a certain "history-independence" property, then it is possible to use a quantum walk in order to show that the "structured" variant of Quantum 3SUM Conjecture, where we give the algorithm access to the queries for free, is equivalent to the original unstructured version of the Quantum 3SUM Conjecture.[3] It is this insight that underlies all of our reductions, and which we expect will open up the way to many other fine-grained reductions in the quantum setting.

One might formally state our observation as follows.

Theorem 5.2. *Let $\{q_i\}$ be a collection of queries over* 3SUM *inputs, i.e., each q_i is a function over inputs $S \in \Sigma^n$ for* 3SUM*. Suppose that,*

- *there exists an efficient deterministic classical dynamic data-structure that allows us to answer the queries q_i, under updates to S, where an update consists of replacing an element in the list S by a different element. By efficient we mean that any query or update can be carried out in at most $n^{o(1)}$ time. And,*
- *the dynamic data structure satisfies a certain "history-independence" property,*[4] *which means that the data structure corresponding to each set S has a unique representation in memory, which only depends on the current value of S (so it is independent of the initial value of S, and of the subsequent updates which resulted in the current value of S)*

then, conditioned on the Quantum 3SUM Conjecture, there is no $O(n^{1-\varepsilon})$ time quantum algorithm for 3SUM*, for any $\varepsilon > 0$, even if the queries $q_i(S)$ can be done at unit cost.*

Hereafter, we refer to these versions of 3SUM, where queries $q_i(S)$ have unit cost, as "structured" versions of the 3SUM problem. To be clear, by *being able to do the queries at unit cost*, we mean that the algorithm is given access to an oracle gate, implementing the unitary transformation:

$$|i, b\rangle \mapsto |i, b \oplus q_i(S)\rangle.$$

The distinction between an arbitrary dynamic data structure and a history-independent solution should be understood as follows. Generally speaking, a solution to a dynamic data structure problem could represent data in a way which depends on the specific sequence of updates which were applied to the initial data. For

[3] The *history-independence* is necessary to achieve the appropriate amplitude amplification/cancellation in the quantum walk: if two update histories for the data structure (e.g. insertions/removals) lead to the same data contents (e.g. same list), then there should be a single basis state that represents the result.

[4] Also mentioned and discussed in [Amb07; ACL+20] and in Chapter 4 of this thesis.

example, self-balancing trees are a solution to the dynamic sorting problem, but the specific balancing of the tree which is kept in memory depends on the sequence of updates which were applied, so different sequences of insertions and deletions might lead to the same list, but will nonetheless be represented differently in memory. A history-independent data structure, however, has a fixed a-priori representation for each possible data value. So, for example, in the dynamic sorting problem, a history-independent data structure must represent each possible list in a unique, or canonical way in memory.

Our idea Let $S = (x_1, \ldots, x_n) \in \Sigma^n$ be an unstructured input to 3SUM. We will now discuss quantum query algorithms for solving 3SUM. Such algorithms can access the input only via a unitary $|i, b\rangle \mapsto |i, b \oplus x_i\rangle$. Each application of this unitary is called a *query*. But, in accordance to data-structure nomenclature, we have also called *queries* to the functions q_i. So to distinguish the two, in this section we will use *input queries* to refer to queries to the input, in the sense of query complexity, and let us use *data-structure queries*, to refer to the values $q_i(S)$.

Consider the quantum walk algorithm for Element Distinctness by [Amb07]. It was observed by Childs and Eisenberg [CE05] that this algorithm can be used to solve any problem, such as 3SUM, where we wish to find a constant-size subset that satisfies a given property. Although this algorithm is optimal and sub-linear for 3SUM when we only measure the number of input queries (it uses $\Theta(n^{3/4})$ input queries, and this is required [BŠ13]), the algorithm still requires linear time, essentially because an $\Omega(n^{1/4})$-time operation is performed between each input query.

This optimal query algorithm for 3SUM is a quantum walk on the *Johnson graph*, the graph of $\binom{n}{r}$ vertices with each vertex of the graph labelled by an r-sized subset of $[n]$, and where there is an edge between two vertices if and only if the two corresponding sets differ by exactly two elements. This resulting graph $J(n, r)$ is a good-enough expander, so that a quantum walk will be able to find an r-sized subset of $[n]$ containing indices to three elements of S that sum to zero, in queries sublinear in n.[5] To do so, the quantum-walk algorithm maintains the list of values $(x_{i_1}, \ldots, x_{i_r})$ entangled together with the basis state representing the current r-sized subset $\{i_1, \ldots, i_r\} \subseteq [n]$ that is being traversed. Using this list of values, as a part of the quantum walk algorithm, a subroutine checks (in superposition) if there is a 3SUM solution in $(x_{i_1}, \ldots, x_{i_r})$. While this step requires no additional input queries, so the total number of input queries is $O(n^{3/4})$, the actual implementation of this subroutine requires a significant amount of time (namely time $r = \Omega(n^{1/4})$), which then makes the resulting quantum walk algorithm for 3SUM linear, at best.

It is this subroutine, i.e., the subroutine that checks for a 3SUM solution in the r-sized set of values, that we would like to further speed up. Now suppose that we had a faster-than-linear algorithm for a "structured" version of 3SUM. I.e., the algorithm works in sublinear time, provided it is given certain data-structure queries $q_i(S)$ as part of the input. Now, if we could efficiently answer these data-structure queries at any point during the entire quantum walk, then we could use this faster-than-linear algorithm to speed-up the subroutine. To do so, we need a dynamic data structure that allows us to efficiently answer the data-structure queries, under the kind of updates that are required at each step of the quantum walk. For the

[5] For an excellent introduction to quantum walks, see Chapter 8 of Ronald de Wolf's lecture notes [Wol21].

quantum walk on the Johnson graph, each update corresponds to replacing a single element in the list of values $(x_{i_1}, \ldots, x_{i_r})$.

An important detail remains: in order for the quantum walk to work, it is necessary that there is a unique basis state corresponding to each node in the quantum-walk graph (otherwise we won't have the desired amplitude interference). It is for this reason that the dynamic data-structure structure is required to have a history-independence property.

Proof of Theorem 5.2. In order to prove this theorem, we will first go through the steps of the more general version of Ambainis' quantum walk algorithm for Element Distinctness given by Childs and Eisenberg [CE05].

Let $S \in \Sigma^n$ be an input to the 3SUM problem. Let $r = n^\beta$ for some $\beta \in (0, 1)$ which will be fixed later (so that r is an integer). The graph G used in Ambainis' construction is a Johnson graph $J(n, r)$ with vertices all labelled by r-sized subsets of $[n]$. Let $V, V' \subset [n]$ with $|V| = |V'| = r$. Vertices labelled by V and V' are connected if and only if $|V \cap V'| = r - 1$, i.e., V' can be obtained by replacing a single element of V.

Given a subset $I \subseteq [n]$, we use $S[I]$ to denote all the elements $\{(i, S[i]) : i \in I\}$. Now suppose we have a history-independent classical deterministic dynamic data structure for answering a family of data-structure queries $\{q_i\}$, where each $q_i : \Sigma^r$. For $V \subseteq [n]$ of size $|V| = r$, let $D(S[V])$ denote the (unique) state of the data-structure corresponding to $S[V]$. I.e., given $D(S[V])$, we are able to answer any query $q_i(S[V])$ in time $n^{o(1)}$. And if we change V to V' by replacing a single element of V, we are able to update $D(S[V])$ to $D(S[V'])$, also in time $n^{o(1)}$.

To define a quantum walk on G, define an orthonormal basis of quantum states $|V\rangle$, one basis state for each r-subset V. The key idea is to store values from the list, and the contents of the data-structure, along with the subset V. So the full quantum state has the form $|V, D(S[V]), k\rangle$ where $k \in [n]$. If $|V| = r$ then k denotes an element in $[n] \backslash V$ to be added to V. We say a vertex V is marked if $S[V]$ is a positive 3SUM instance (of smaller size), i.e., if there are $p, q, r \in V$ such that $S[p] + S[q] + S[r] = 0$.

The quantum walk algorithm is analogous to Grover's algorithm, where the aim is to make the amplitude on marked vertices large enough that with probability at least $1 - o(1)$ the final measurement collapses on a marked vertex, i.e., a vertex labelled by an r-subset that contains a solution to 3SUM problem. The algorithm starts with a state (in what is known to be the "setup" phase),

$$|s\rangle = \frac{1}{\sqrt{c}} \sum_{|V|=r} |V, D(S[V])\rangle \sum_{k \notin V} |k\rangle, \tag{5.1}$$

which is a uniform superposition of all the states on subsets of size r and $c = (n - r)\binom{n}{r}$ is the normalisation constant.[6,7]

[6]Refer to the circuit construction in András Pál Gilyén's Master's thesis [Gil14] for creating a uniform superposition over all the vertices of Johnson Graph $J(n, r)$. This construction uses $\widetilde{O}(r)$ elementary quantum gates in total and their results extend for any $r = n^\beta$ with $0 < \beta < 1$.

[7]Note that, the circuit construction that we refer to creates a uniform superposition of vertices in $J(n, r)$ with the vertices represented in $O(r \log n)$ sized array of qubits. This representation of vertices does not allow for time-efficient insertions and deletions. Therefore, we first encode all the vertices V (in superposition) in a data structure that enables the (walk) updates on the vertex

There are two main operations in this algorithm: a walk operation U_{walk} and a phase flip operation $U_{phaseFlip}$ which is

$$U_{phaseFlip}|V, D(S[V])\rangle = \begin{cases} -|V, D(S[V])\rangle & \text{if } V \text{ is marked} \\ |V, D(S[V])\rangle & \text{if } V \text{ is not marked.} \end{cases} \tag{5.2}$$

The full algorithm is $(U_{walk}^{t_1} U_{phaseFlip})^{t_2}$ where $t_1 = O(\sqrt{r})$ and $t_2 = O((n/r)^{1.5})$. The total time taken by the algorithm is

$$T_{setup}(|s\rangle) + t_1 \cdot t_2 \cdot T_{unitary}(U_{walk}) + t_2 \cdot T_{unitary}(U_{phaseFlip}), \tag{5.3}$$

where $T_{setup}(|s\rangle)$ denotes the time taken to setup the initial state $|s\rangle$ that also includes the time taken to query values of the subset of indices of size r. The term $T_{unitary}(U)$ denotes the number of elementary gates required to implement a unitary U.

In the setup phase, for every vertex V we initialise the dynamic data-structure corresponding to $S[V]$. We may think of $S[V]$ as obtained via the $(0, \ldots, 0)$ list by updating each position i with $S[i]$. Hence, the setup time for each vertex, which consists of computing $D(S[V])$ for all V in superposition, is at most $r \cdot n^{o(1)}$.

Now, because the data structure supports efficient updates, the U_{walk} unitary can be implemented in time $n^{o(1)}$: the walk operator is a product of four unitary transformations,

- two Grover's diffusion operators that can be implemented in $O(\log n)$ time [Gro96; Amb07] and some data structure operations (such as insertions, deletions, lookups) on set V, which because of the prefix-sum tree data structure we use to store V also takes at most $O(\log n)$ time, and
- two update operations where an element is inserted and some other element is deleted from set V and $D(S[V])$, hence it is sufficient that the dynamic data structure supports replacement of values in $n^{o(1)}$ time.

The unitary $U_{phaseFlip}$ in Equation 5.2 adds a negative phase to the marked states and none to the unmarked states, which means $U_{phaseFlip}$ implements a subroutine that checks whether or not a vertex V is marked by going through its input-query values $S[V]$ and checking if there is a 3SUM solution present in $S[V]$. Currently, there is no known time-efficient method to implement this subroutine.[8]

Instead, suppose there was a 1/3-bounded error quantum subroutine $\mathcal{A}$ that could solve this structured version of 3SUM on a set of r elements in $O(r^{1-\alpha})$ for some constant $\alpha > 0$, we could then implement the $U_{phaseFlip}$ by calling subroutine $\mathcal{A}$. However, we would have to repeat $\mathcal{A}$ several times to reduce the error to a value

states to occur time efficiently. In this chapter, we use the notation V to denote a vertex already encoded in such a data structure. Such encoding procedures exist and are efficient (reversibly as well), i.e. run in $\widetilde{O}(r)$ time where r is the size of the set that is being encoded, for e.g., the use of skip-list data structure in [Amb07], [ACL+20]. We instead use a prefix-sum tree $P(V)$ over a bit vector of length n where the i^{th} coordinate of the vector is 1 iff $i \in V$, see Definition 4.4 from Chapter 4 for the definition of a prefix-sum tree; this is going to be space inefficient as we will be using n sized vectors to store r sized sets. However, because of the r-sparsity of our reduction algorithm we can get rid of the space inefficiency by invoking Theorem 4.8 of Chapter 4.

[8] One would require a dynamic data structure for efficiently answering 3SUM queries, which is not known to exist.

small enough such that the state of the algorithm is sufficiently close to the state that would have been if the subroutine was exact; see Theorems 2.6 and 2.7 in Chapter 2. This process would incur a $\mathsf{polylog}(n)$ factor overhead in time. Secondly, to implement the input oracle for subroutine $\mathcal{A}$ we instead use a data structure query, which we are promised can be achieved in $n^{o(1)}$ time. With the additional promise that the data structure is deterministic, the only other source of error is the one induced by the walk algorithm itself which is at most $O(1/\mathsf{poly}(n))$. (See for e.g., Equation 56 in [CE05].) Therefore, the time complexity of the walk algorithm then becomes

$$r \cdot n^{o(1)} + t_1 \cdot t_2 \cdot n^{o(1)} + t_2 \cdot n^{o(1)} \cdot r^{1-\alpha}, \tag{5.4}$$

which, after ignoring all the $n^{o(1)}$ factors is

$$r + t_1 t_2 + t_2 r^{1-\alpha}. \tag{5.5}$$

Substituting the values of $t_1 = O(\sqrt{r})$ and $t_2 = O((n/r)^{1.5})$ the total time taken in Equation 5.5 roughly becomes

$$r + \frac{n^{1.5}}{r} + \frac{n^{1.5}}{r^{1.5}} \cdot r^{1-\alpha}. \tag{5.6}$$

Given that $r = n^\beta$ for a $\beta \in (0, 1)$, it is easy to see that for every $0 < \alpha < 1$, there exists a β such that $\max(\frac{1}{2}, \frac{1}{2\alpha+1}) < \beta < 1$, and then the value of (Equation 5.6) becomes strictly sublinear. It then follows that there is no sub-linear quantum time algorithm for solving the structured version of 3SUM, unless Quantum 3SUM Conjecture is false. □

We have deliberately omitted one other crucial possibility in the statement of the above theorem and its proof; a situation where the dynamic data structure is *not* deterministic, i.e., the data structure queries $\{q_i\}$ and the updates to the list S fail with a certain probability? Randomness seems to be required because no dynamic sorting data structure is known that is simultaneously time-efficient, space-efficient, history-independent, and deterministic. However, a solution exists if any of these four requirements are removed. We will (in Section 5.3.1) present a deterministic data structure *highly space inefficient* but one that satisfies all the requirements of Theorem 5.2. In the online version of the paper on which this chapter is based [BLP+22a], we prove similar results using a probabilistic but space efficient namely the skip-list data structure; that result follows a different line of analysis as one has to account for the failure of the data structure both for the updates and the queries. However, we will not be presenting that result in this thesis because we now have a memory compression technique (like the one mentioned in Chapter 4) with which we can simulate a space-efficient algorithm from a space inefficient one (as long as the original algorithm uses space sparsely). What still remains is an interesting open question in classical data structures to provide, or disprove the existence of, a dynamic data structure that *simultaneously* satisfies all four requirements.[9]

[9]The question might arise: *why do we care for the structure to be space efficient?* This is for two reasons. On the one hand, we expect memory to be an expensive resource for quantum computers, so algorithms using a large amount of memory, even in regimes that are practical classically, might never be so quantumly. On the other hand, making our reductions space efficient allows us to weaken the Quantum 3SUM Conjecture to say that no space-efficient quantum algorithm can solve 3SUM in sublinear time. We do not explicitly state this outside of this footnote, but all the lower bounds in this chapter also follow from this weaker conjecture.

5.1.3 Applications

We use our proof strategy to show, conditional on Quantum 3SUM Conjecture, tight lower bounds on several computational-geometry problems, on Convolution-3SUM, and on the 0-Edge-Weight-Triangle problem. Our lower bounds show that the quantum speed-up is at most quadratic for all of these problems.

Our lower bounds on Convolution-3SUM and 0-Edge-Weight-Triangle tightly match the Grover-based speed-up that quantum algorithms can get for these problems.

Our quantum reductions from 3SUM to computational-geometry problems are complementary to a recent paper by Ambainis and Larka [AL20], where they present quantum speed-ups for several such problems. Our results show, under the Quantum 3SUM Conjecture, that all of the speed-ups obtained by Ambainis and Larka are optimal. There are also computational-geometry problems for which the Quantum 3SUM Conjecture gives us a lower bound, but for which no quantum speedup is known.

Table 5.1 (on page 109) summarises our results. It also includes the best-known *classical* upper and lower bounds.

5.1.4 Structure of this chapter

In Section 5.2 we describe various simple variants of the 3SUM problem and show that the Quantum 3SUM Conjecture is equivalent for these versions. (These are not the structured versions we mentioned earlier, here the proof of equivalence is very simple.) In Section 5.3, using the approach we sketched above (in Section 5.1.2), we give a full proof that, under the Quantum 3SUM Conjecture, two "structured" variants of 3SUM also require $n^{1-o(1)}$ time on a quantum computer. As direct implications of these hardness results, in Section 5.4 we present conditional quantum time lower bounds for several computational geometry problems. Lastly, in Section 5.5, we present conditional quantum time lower bound for Convolution-3SUM and 0-Edge-Weight-Triangle problems. This requires us to prove, under the Quantum 3SUM Conjecture, that a third "structured" variant of 3SUM also requires $n^{1-o(1)}$ time on a quantum computer.

3SUM-based quantum lower-bounds (our results) ↴		Classical upper & lower bounds, respectively (**) ↴	
Problems		**Quantum upper-bound**	
GeomBase	$n^{1-o(1)}$	$\widetilde{O}(n)$ (*)	$O(n^2), n^{2-o(1)}$
3-Points-on-Line	$n^{1-o(1)}$	$O(n^{1+o(1)})$ [AL20]	$O(n^2), n^{2-o(1)}$
Point-on-3-Lines	$n^{1-o(1)}$	$O(n^{1+o(1)})$ [AL20]	$O(n^2), n^{2-o(1)}$
Separator	$n^{1-o(1)}$	$O(n^{1+o(1)})$ [AL20]	$O(n^2), n^{2-o(1)}$
Strips-Cover-Box	$n^{1-o(1)}$	$O(n^{1+o(1)})$ [AL20]	$O(n^2), n^{2-o(1)}$
Triangles-Cover-Triangle	$n^{1-o(1)}$	$O(n^{1+o(1)})$ [AL20]	$O(n^2), n^{2-o(1)}$
Point-Covering	$n^{1-o(1)}$	$O(n^{1+o(1)})$ [AL20]	$O(n^2), n^{2-o(1)}$
Visibility-Between-Segments	$n^{1-o(1)}$	$O(n^{1+o(1)})$ [AL20]	$O(n^2), n^{2-o(1)}$
Hole-In-Union	$n^{1-o(1)}$	$O(n^{1+o(1)})$ (†)	$\widetilde{O}(n^2), n^{2-o(1)}$
Triangle-Measure	$n^{1-o(1)}$	Open!	$O(n^2), n^{2-o(1)}$
Visibility-From-Infinity	$n^{1-o(1)}$	Open!	$O(n^2), n^{2-o(1)}$
Visible-Triangle	$n^{1-o(1)}$	$O(n^{1+o(1)})$ (†)	$O(n^2), n^{2-o(1)}$
Planar-Motion-Planning	$n^{1-o(1)}$	Open!	$O(n^2), n^{2-o(1)}$
3D-Motion-Planning	$n^{1-o(1)}$	Open!	$O(n^2), n^{2-o(1)}$
General-Covering	$n^{1-o(1)}$	$O(n^{1+o(1)})$ [AL20]	$O(n^2), n^{2-o(1)}$
Convolution-3SUM	$n^{1-o(1)}$	$O(n)$ (*)	$O(n^2), n^{2-o(1)}$
0-Edge-Weight-Triangle	$n^{1.5-o(1)}$	$O(n^{1.5})$ (*)	$O(n^3), n^{3-o(1)}$

(*) Using a simple Grover speed-up on the classical algorithm.

(†) Implicit in [AL20], by using the classical reduction to Triangles-Cover-Triangle and then using the corresponding quantum algorithm.

(**) All upper bounds are straightforward: For problems like Convolution-3SUM and 0-Edge-Weight-Triangle the best known algorithms use brute force, for the computational-geometry problems, the upper bounds follow from geometry arguments [GO95]. All lower-bounds for computational geometry problems are from [GO95], the lower-bound for Convolution-3SUM follows from [Pat10], and, the lower-bound for 0-Edge-Weight-Triangle follows from [VW13]; all conditional on the classical hardness of 3SUM.

Table 5.1: This is a summary of all the Quantum-3SUM-hard problems mentioned in this chapter, with (almost) matching upper bounds for most of them.

5.2 Simple variants of 3SUM

The standard 3SUM problem is defined as follows: given a list S of n integers, do there exist elements $a, b, c \in S$ such that $a + b + c = 0$? Furthermore, there are several other variants of the 3SUM problem that have been useful as intermediary steps for reductions in the classical case. We will begin by considering the following two simple variants:

1. 3SUM': given a list S of n integers, are there $a, b, c \in S$ such that $a + b = c$?
2. 3SUM-3ListVersion: given three lists A, B, C of n integers each, are there $a \in A$, $b \in B$ and $c \in C$ such that $a + b = c$?

We now show that 3SUM, 3SUM', and 3SUM-3ListVersion can all be quantumly reduced to each other with $O(\sqrt{n})$ pre-computation time, followed by an *on-the-fly* fast reduction for any query. Meaning, the reduction is given input X and outputs Y in the following sense: after some pre-computation on X, obtaining any of the integers in the list or lists in Y can be done in $O(1)$ time ("on-the-fly") by querying X. This establishes, therefore, that the Quantum 3SUM Conjecture can be equivalently stated for any of these simple variants.

The reduction from 3SUM to 3SUM-3ListVersion is simple, and requires no pre-computation. We set $A = S, B = S$ and $C = -S$, and now $a \in A, b \in B, c \in C$ have $a + b = c$ if and only if $a, b, (-c) \in S$ have $a + b + (-c) = 0$. The reduction from 3SUM' to 3SUM-3ListVersion is also simple and on-the-fly with no required pre-computation. Simply set $A = S$, $B = S$ and $C = S$.

The reduction from 3SUM-3ListVersion to 3SUM', is slightly more complicated and is almost identical to the reduction presented in Theorem 3.1 by [GO95], and is as follows: as a pre-computation step, compute the element $m = 2\max(A, B, C)$. This part takes $O(\sqrt{n})$ quantum time. Create a list S of size $3n$: for each $a \in A$ put $a' = a + m$ in S, for each $b \in B$ put $b' = b + 3m$ in S, and, for each $c \in C$ put $c' = c + 4m$ in S. Clearly, if $a + b = c$ then $a' + b' = c'$. Without loss of generality one can assume the elements of the lists A, B, C are strictly positive because one can add a big number k to all the elements in lists A, B and $2k$ to the elements in list C. Additionally, with some elementary calculations one can easily see that whenever there are three elements in S such that $a' + b' = c'$, the corresponding a, b, c come from three different sets A, B, C, respectively.

The reduction from 3SUM-3ListVersion to 3SUM is very similar to the above, and is given in Theorem 3.1 by [GO95]. We will not repeat it here. As above, the reduction contains a pre-computation step where the maximum of all the lists A, B, C is computed, making the quantum reduction take $O(\sqrt{n})$ pre-computation time. Thereafter, it is an on-the-fly fast reduction for any query.

Hence it follows that the Quantum 3SUM Conjecture is equivalent to the same conjecture stated for 3SUM', or for 3SUM-3ListVersion.

5.3 Lower bounds for two structured versions of 3SUM

In 1995, Gajentaan and Overmars [GO95] showed that 3SUM can be reduced to several computational geometry problems, proving that these problems cannot have truly sub-quadratic classical algorithms unless the Classical 3SUM Conjecture is

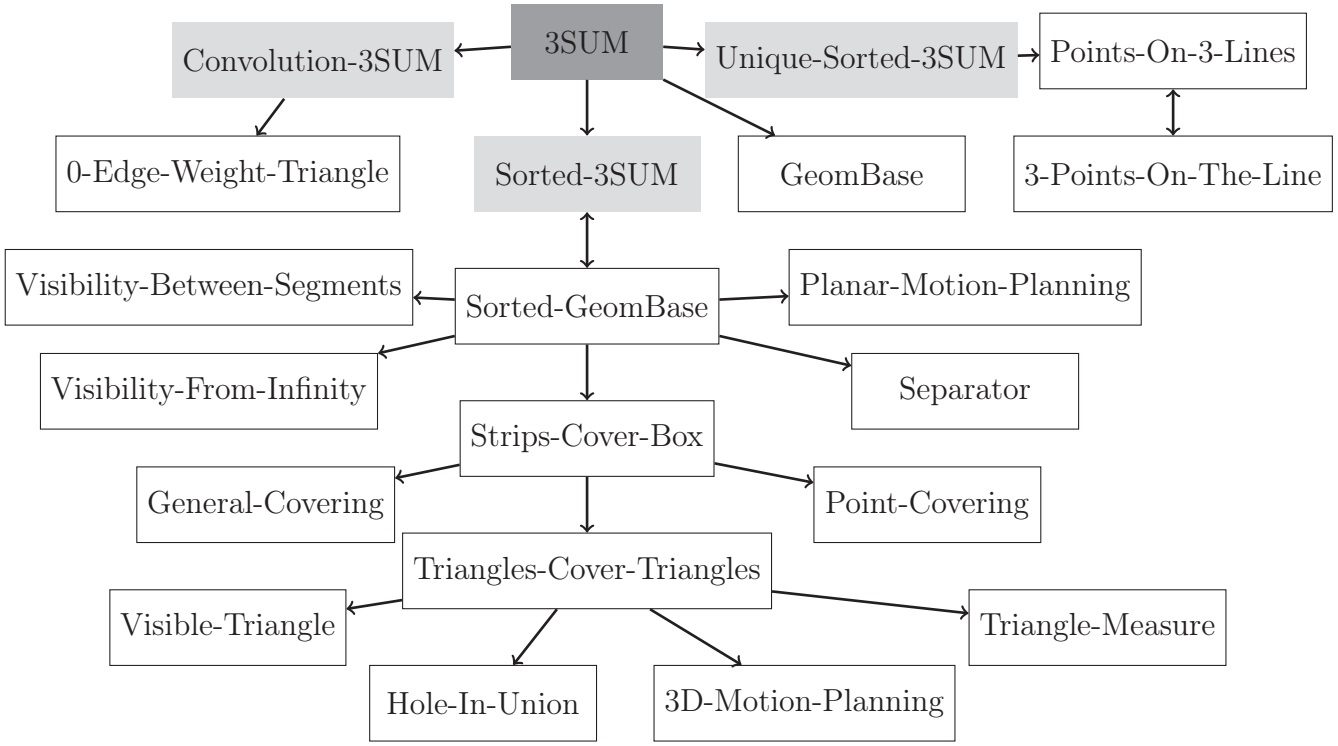

Figure 5.1: Overview of the different reductions between 3SUM, Convolution-3SUM, 0-Edge-Weight-Triangle and some computational geometry problems. The same reductions can be shown both classically and quantumly, but the classical reductions from 3SUM to Convolution-3SUM, Sorted-3SUM, and Unique-Sorted-3SUM cannot be trivially translated to the quantum setting.

false. These results are proven by first exhibiting a reduction from 3SUM to some fundamental computational geometry problems (Sorted-GeomBase and 3-Points-on-Line), and then constructing further reductions among computational geometry problems. See Figure 5.1 for an overview of such reductions.

The reductions among computational geometry problems are all simple to adapt to the quantum setting. We will do so in Section 5.4. However, the fundamental reduction to Sorted-GeomBase requires sorting the 3SUM instance, and the fundamental reduction to 3-Points-on-Line requires both sorting and removing duplicate elements. This means that we cannot trivially adapt these reductions to the quantum setting. We overcome this obstacle by employing a dynamic data structure, that allows efficient updates and queries, and then invoke Theorem 5.2.

5.3.1 Hardness of Sorted-3SUM and Unique-Sorted-3SUM

We will now define a dynamic data structure that is deterministic, history-independent, efficient, and moreover supports two families of queries $\{q_i\}, \{q'_i\}$, where $q_i(S)$ is the i^{th} smallest integer in a list S and $q'_i(S)$ is the i^{th} smallest integer in the set version of list S, respectively. Using this data structure with the quantum walk algorithm in Theorem 5.2 we are able to show that conditioned on Quantum 3SUM Conjecture, both Sorted-3SUM and Unique-Sorted-3SUM require $n^{1-o(1)}$ time in the

quantum setting.

Definition 5.3 (Sorted-3SUM)**.** Given a sorted list $S \in \Sigma^n$ of n integers, i.e., $S[i] \leqslant S[j]$ whenever $i < j$ for all $i, j \in [n]$, are there $a, b, c \in S$ such that $a+b+c = 0$?

Definition 5.4 (Unique-Sorted-3SUM)**.** Given a sorted list $S \in \Sigma^n$ of n unique integers, i.e., $S[i] < S[j]$ whenever $i < j$ for all $i, j \in [n]$, are there $a, b, c \in S$ such that $a + b + c = 0$?

Definition 5.5 (Data Structure for Sorted-3SUM and Unique-Sorted-3SUM)**.** Let $S \subseteq [n] \times \Sigma$, with $|S| = r$, and such that every $i \in [n]$ appears in at most one pair $(i, x) \in S$. The *3SUM tree* that represents S, denoted by $T_{3\text{SUM}}(S)$, is a complete rooted binary tree with $|\Sigma|$ leaves. Each leaf node $x \in \Sigma$ is labelled by a bit $\mathtt{b}_x \in \{0, 1\}$, number $\mathtt{count}_x \in \{0, \ldots, r\}$, and a bit vector $\mathtt{B}_x \in \{0, 1\}^n$ so that set $\mathtt{B}_x[i] = 1$ iff $(i, x) \in S$, the $\mathtt{count}_x$ is the hamming weight of $\mathtt{B}_x$, and $\mathtt{b}_x = 1$ iff $\mathtt{count}_x > 0$. Each internal node w is labelled by two numbers $\mathtt{count\_children}_w \in \{0, \ldots, r\}$ and $\mathtt{count\_unique\_children}_w \in \{0, \ldots, r\}$. These labels obey the following rules:

- The variable $\mathtt{count\_children}_w = \sum_{x \text{ a descendent of } w} \mathtt{count}_x$.
- The variable $\mathtt{count\_unique\_children}_w = \sum_{x \text{ a descendent of } w} \mathtt{b}_x$, i.e., the variable $\mathtt{count\_unique\_children}_w$ stores the total number of leaves x that are descendent of w such that $\mathtt{count}_x > 0$.

See Figure 5.2 for an example of the data structure.

Memory representation A *3SUM tree* is represented in the memory by an array of $|\Sigma|-1$ blocks of $O(\log n)$ bits of memory each, holding the contents of the internal nodes, followed by $|\Sigma|$ blocks of $O(n)$ bits of memory each, holding the labels of the leaf nodes. The blocks appear in the same order as a breadth-first traversal of $T_{3\text{SUM}}(S)$. Consequently, for every $S \subseteq [n] \times \Sigma$ there is a corresponding binary string of $\widetilde{O}(n|\Sigma|)$ bits that uniquely encodes $T_{3\text{SUM}}(S)$. Crucially, when $|S| = r$ we have at most $\widetilde{O}(r \log |\Sigma|)$ of these bits are 1. Since $|\Sigma| = \mathsf{poly}(n)$, the encoding is $\widetilde{O}(r)$-sparse.

Queries and updates The following operations can be implemented efficiently using this data structure $T_{3\text{SUM}}(S)$.

1. **Find** To find if there exists an element $(i, x) \in T_{3\text{SUM}}(S)$ check the value corresponding to variable $\mathtt{B}_x[i]$; if $\mathtt{B}_x[i]$ then $(i, x) \in T_{3\text{SUM}}(S)$. Using a RAG this can be implemented in $O(1)$ time.
2. **Toggle** Recall (from Chapter 4) that the toggle operation is the quantum analogue of the classical 'insertion/deletion' operation. In the quantum setting we only insert an element (i, x) into $T_{3\text{SUM}}(S)$ if $(i, x) \notin T_{3\text{SUM}}(S)$. Similarly, we only delete an element (i, x) from $T_{3\text{SUM}}(S)$ if $(i, x) \in T_{3\text{SUM}}(S)$ which we can check in $O(1)$ time with the 'find' operation stated above. To insert an element (i, x) into $T_{3\text{SUM}}(S)$ set $\mathtt{b}_x = 1$, $\mathtt{B}_x[i] = 1$, increase the value of $\mathtt{count}_x$ by 1. All of this can be achieved in $O(1)$ amount of time. Thereafter, for every

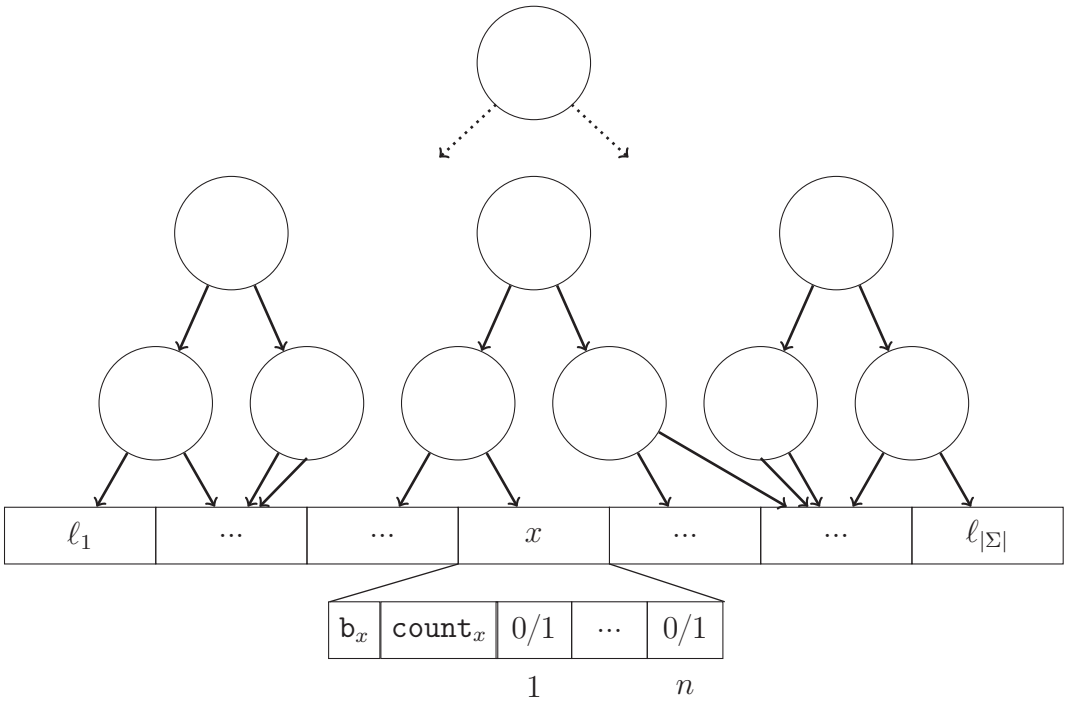

Figure 5.2: Data Structure for Sorted-3SUM and Unique-Sorted-3SUM: each of the circular nodes w is labelled by two variables $\mathtt{count\_children}_w$ and $\mathtt{count\_unique\_children}_w$. Each of the leaf nodes x is labelled by three variables $\mathtt{b}_x$, $\mathtt{count}_x$ and $\mathtt{B}_x$.

ancestor w of x, increase the value of the variable $\mathtt{count\_children}_w$ by 1 and do the same with variable $\mathtt{count\_unique\_children}_w$ only if $\mathtt{b}_x$ was recently changed to 1. All this can be done in $O(\log|\Sigma|)$ time by traversing the path from root node to the leaf node indexed by x. The procedure for deleting an element is just the reverse of the insertion procedure.

3. **Indexing** This can be requested in two ways:

   - For some $k \in [r]$ return the k^{th} largest element of the set S when ordered by the second coordinate of elements in S, or,
   - for some $\bar{k} \in [r]$, return $\bar{k}^{\text{th}}$ largest unique value of the set S again when ordered by the second coordinate of the elements in S.

   For example, if $S = \{(i, x) : x \in X\}$ where $X = [1, 3, 3, 3, 3, 4, 5]$, then the 4^{th} largest element of S is 3, while the 4^{th} largest unique value of S is 5.

   For this data structure, both these types of indexing can be implemented efficiently in the following way: set a pointer to the root node. Clearly the value of $\mathtt{count\_children}_{root}$ (or $\mathtt{count\_unique\_children}_{root}$) is equal to the total number of marked (unique) leaf nodes. Let k_{left} denote the count on the left child of the pointer and similarly let k_{right} denote the count on the right child of the pointer. If $k > k_{left}$, then set the pointer to the right child and $k = k - k_{left}$, and repeat the process recursively with the new k. If $k \leqslant k_{left}$

then set the pointer to the left child and repeat the process recursively with the same k. Repeat until you reach a marked (unique) leaf node. This entire process takes $\log|\Sigma|$ amount of time.

Quantum hardness of Sorted-3SUM and Unique-Sorted-3SUM Using the *3SUM tree* to encode S, we can now efficiently maintain S in a sorted order, additionally one can also implement the following maps efficiently,

$$|i,b\rangle \mapsto |i,b\oplus q_i(S)\rangle, \quad |i,b\rangle \mapsto |i,b\oplus q'_i(S)\rangle,$$

where $q_i(S)$ is query to retrieve the i^{th} largest value in S and $q'_i(S)$ is query to retrieve the i^{th} largest unique value in S. Furthermore, it is history-independent, as required by Theorem 5.2. Hence we may use Theorem 5.2 to prove the following statement.

Corollary 5.6. *If there is a bounded-error quantum algorithm that solves* Sorted-3SUM *or* Unique-Sorted-3SUM *in $O(n^{1-\alpha})$ time, for some constant $\alpha > 0$, using $g(n)$ qubits of memory, then there exist constants $1 > \beta > 0$ and $\delta > 0$ such that* 3SUM *can be solved in $\widetilde{O}(n^{1-\delta})$ quantum time with probability $1 - o(1)$ using at most $\widetilde{O}(n|\Sigma|) + g(n^{\beta})$ qubits of memory.*

Apart from the expensive (in terms of space) data structure, the rest of quantum walk algorithm uses only a poly-logarithmic number of qubits, hence, at most $O(n|\Sigma| + m')$ qubits suffice to implement this algorithm, where m' denotes the number of qubits required by the subroutine for Sorted-3SUM. However, given that the reduction algorithm uses space sparsely, using our compression technique from Chapter 4 we can simulate this space inefficient but r-sparse QRAM algorithm, with $r = n^{\beta}$, in a space-efficient way as well. Therefore, combining Corollary 5.6 and the main theorem 4.1 of Chapter 4 we get the following result.

Corollary 5.7. *If there is a bounded-error quantum algorithm that solves* Sorted-3SUM *or* Unique-Sorted-3SUM *in $\widetilde{O}(n^{1-\alpha})$ time, for some constant $\alpha > 0$, using $g(n)$ qubits of memory, then for every such α there exists constants $1 > \beta > 0$ and $\delta > 0$ such that* 3SUM *can be solved in $\widetilde{O}(n^{1-\delta})$ quantum time with probability $1 - o(1)$ using at most $\widetilde{O}(n^{\beta}) + g(n^{\beta}))$ qubits of memory.*

An immediate implication of Corollary 5.6, consequently also Corollary 5.7 is that a sublinear quantum time algorithm for Sorted-3SUM or or Unique-Sorted-3SUM would imply a sublinear quantum time algorithm for 3SUM, and therefore would contradict the Quantum 3SUM Conjecture.

Corollary 5.8. *Conditioned on the Quantum 3SUM Conjecture, both* Sorted-3SUM *and* Unique-Sorted-3SUM *require $n^{1-o(1)}$ time in the quantum setting.*

Lower bounds conditioned on hardness of Sorted-3SUM Employing the results of Corollary 5.8 we are able to show that conditioned on the Quantum 3SUM Conjecture, the following computational geometry problems and (because of transitivity of quantum fine-grained reductions) many more (refer to Figure 5.1) require $n^{1-o(1)}$ quantum time.

1. The SEPARATOR problem: given a set of n (possibly half-infinite) closed horizontal line segments, is there a non-horizontal separator?

2. The STRIPS-COVER-BOX problem: given a set of strips in the plane does their union contain a given axis-parallel rectangle?

3. The TRIANGLES-COVER-TRIANGLE problem: given a set of triangles in the plane, does their union contain another given triangle?

Having established the quantum hardness of the SORTED-3SUM problem, we can now directly reduce SORTED-3SUM to these problems by a simple adaptation to the quantum setting of the classical reductions presented in [GO95].

Lower bounds conditioned on hardness of UNIQUE-SORTED-3SUM As other implications to Corollary 5.8 we show that conditioned on the Quantum 3SUM Conjecture, the following computational geometry problems also require $n^{1-o(1)}$ quantum time.

1. The 3-POINTS-ON-LINE problem: given a set of points in the plane, is there a line that contains at least three of the points?

2. The POINT-ON-3-LINES problem: given a set of lines in the plane, is there a point that lies on at least three of them?

Both of these problems are computationally equivalent, as the second problem is the exact dual of the first problem under the Point-Line dualization. The classical reduction from 3SUM to these two problems assumes that the input to 3SUM is unique, i.e. there are no duplicate elements in the input. As discussed earlier in Section 5.1.2, the Classical 3SUM Conjecture also trivially holds for this promise version of 3SUM, but such a claim cannot be easily made in the quantum setting. Therefore, we use the results of Corollary 5.8 to establish that both UNIQUE-SORTED-3SUM and 3SUM are equally hard as the original 3SUM problem in the quantum setting as well.

We give an illustration of the relations between the different geometry problems in Figure 5.1 and we point the readers to Section 5.4 for details of some of these reductions.

5.4 3SUM-hard geometry problems

In this section, we present the quantum reductions from (some variants and structured versions of) 3SUM to many problems in Computational Geometry which are adaptations (or in some case, slight modifications) of the classical reductions by [GO95]. Most of these reductions are on-the-fly adaptations of the classical ones, hence, we only present the detailed proofs for only a few of these reductions. The proofs for the rest of the reductions are along the same lines. One can find the summary of these results in Table 5.1 (in page 109).

Recall that, in our quantum model of computation the input is given as an oracle and can be accessed in superposition, therefore it is possible to have strictly sublinear quantum time algorithms even for problems that depend on all elements of the input, for example, Grover's search algorithm [Gro96]. For the same reasons, it

is possible to have quantum reductions which use zero amount of (pre-)computation time as long as query access to the input of the reduced problem is efficiently implementable using the input oracle to the original problem. Therefore, it is possible to quantise these reductions given by [GO95] to run in sublinear quantum time even though they take (at least) linear amount of time classically.

Problem: GeomBase
Given a set of n points with integer coordinates on three horizontal lines $y = 0$, $y = 1$, and $y = 2$, determine whether there exists a non-horizontal line containing three of the points.

Lemma 5.9. $(\text{3SUM-3ListVersion}, n) =_{\text{QFG}} (\text{GeomBase}, n)$.

Proof. The proof of $(\text{3SUM-3ListVersion}, n) \leqslant_{\text{QFG}} (\text{GeomBase}, n)$: for each element $a \in A$ create a point $(a, 0)$, for every $b \in B$ create a point $(b, 2)$, and, for every $c \in C$ create a point $(c/2, 1)$, which means query access to the instance of GeomBase can be directly implemented by using the query oracle of 3SUM-3ListVersion instance. W.l.o.g. we can assume all the elements of the lists A, B, C are even. (If not then multiply each of these elements with 2.) It is easy to see that three points $(a, 0), (b, 2)$ and $(c/2, 1)$ are collinear iff $a + b = c$, hence, a quantum on-the-fly reduction with 0 pre-computation time.

The reduction, $(\text{GeomBase}, n) \leqslant_{\text{QFG}} (\text{3SUM-3ListVersion}, n)$ is also proved in the similar way, for each point $(a, 0)$ create an element $a \in A$, for each point $(b, 2)$ create an element $b \in B$, and, for each point $(c, 1)$ create an element $2c \in C$. □

Problem: 3-Points-on-Line
Given a set of points in the plane, is there a line that contains at least three of the points?

Lemma 5.10. $(\text{Unique-Sorted-3SUM}, n) \leqslant_{\text{QFG}} (\text{3-Points-on-Line}, n)$.

Proof. An input to Unique-Sorted-3SUM is a list S of n unique integers (also sorted but that is not a requirement for this reduction), and the question is whether there exist $a, b, c \in S$ such that $a + b + c = 0$. Let $k = 2\max(\{|x| \mid x \in S)\}$, this can be computed quantumly in $O(\sqrt{n})$ time. Create a list S' of size $3n$ in the following way: for every $x \in S$, put $x + k, x - 3k, x + 2k \in S'$. For every element $y \in S'$ create a point (y, y^3). If there exists $a, b, c \in S$ such that $a + b + c = 0$ then there will exist a triple $a', b', c' \in S'$ such that $a' + b' + c' = 0$ and a', b', c' are all unique.[10] Furthermore, with some elementary calculations we can show that $a' + b' + c' = 0$ iff $(a', (a')^3), (b', (b')^3), (c', (c')^3)$ are collinear. □

[10]The classical reduction from 3SUM to Point-on-3-Lines by [GO95] is slightly incorrect because of the following counterexample: let $S = \{1, -2, 3\}$. There are elements $a, b, c \in S$ such that $a + b + c = 0$, set $a = b = 1, c = -2$. The classical reduction on such an S is going to create three points $(1, 1), (-2, -8), (3, 27)$ and will therefore miss out on the 3SUM solution. We rectify this situation by making three (almost) copies of the original list so that solutions to 3SUM where $a = b$ are not missed. The three copies in the intermediate step are deliberately made nonidentical so that the reduction creates two unique points corresponding to $a, b \in S$ even if $a = b$.

Problem: Point-on-3-Lines
Given a set of lines in the plane, is there a point that lies on at least three of them?

Lemma 5.11. (3-Points-on-Line, n) $=_{\text{QFG}}$ (Point-on-3-Lines, n).

Proof. Both these problems are computationally equivalent as the second problem is the exact dual of the first problem under the Point-Line dualization. □

Lower bounds for the following problems are based on reductions from another promise version of 3SUM (or its variants), namely the Sorted-3SUM. Most of the problems below are reduced from Sorted-GeomBase instead of Sorted-3SUM because they both are computationally equivalent, which directly follows from the result in Lemma 5.9.

Corollary 5.12. (Sorted-3SUM-3ListVersion, n) $=_{\text{QFG}}$ (Sorted-GeomBase, n).

Notice the simple trick that we are employing to adapt these classical reductions in the quantum setting. For all those classical reductions from 3SUM that requires sorting the input, we directly reduce from Sorted-3SUM instead, because we have shown that Quantum 3SUM Conjecture applies to Sorted-3SUM as well. With this result in spotlight, we present the rest of the reductions; all the proofs of their respective theorems follow from 'on-the-fly' adaptation of the classical reductions, hence, will not be stated here.

Problem: Separator
Given a set S of n possible half-infinite, closed horizontal line segments, is there a non-horizontal separator?

Lemma 5.13. (Sorted-GeomBase, n) $\leqslant_{\text{QFG}}$ (Separator, n).

Problem: Strips-Cover-Box
Given a set of strips in the plane, does their union contain a given axis-parallel rectangle?

Lemma 5.14. (Sorted-GeomBase, n) $\leqslant_{\text{QFG}}$ (Strips-Cover-Box, n).

The proof of this statement also directly follows from the (almost) on-the-fly adaptation of the classical reduction. Each query to the input of Strips-Cover-Box can be efficiently computed using the query oracle to the input of Sorted-GeomBase. It is only to compute the boundaries of the rectangle that a constant pre-computation time is required in addition to the on-the-fly reduction.

Problem: Triangles-Cover-Triangle
Given a set of triangles in the plane, does their union contain another given triangle?

Lemma 5.15. (Strips-Cover-Box, n) $\leqslant_{\text{QFG}}$ (Triangles-Cover-Triangle, n).

Problem: Hole-In-Union
Given a set of triangles in the plane, does their union contain a hole?

Lemma 5.16. (Triangles-Cover-Triangle, n) $\leqslant_{\text{QFG}}$ (Hole-In-Union, n).

The relation between Triangles-Cover-Triangle and Hole-In-Union in the other direction is interesting and is captured in the following statement:

Lemma 5.17. *There is a $O(n^{1+o(1)})$ time quantum algorithm to solve* Hole-In-Union.

Proof. Use the classical reduction from Hole-In-Union to Triangles-Cover-Triangle which runs in $O(n \log^2 n)$ time presented by [GO95], and then use the $O(n^{1+o(1)})$ algorithm for Triangles-Cover-Triangle by [AL20] on top of that. □

Problem: Triangle-Measure
Given a set of triangles in the plane, compute the measure of their union.

Lemma 5.18. (Triangles-Cover-Triangle, n) $\leqslant_{\text{QFG}}$ (Triangle-Measure, n).

Problem: Point-Covering
Given a set of n halfplanes and a number k, determine whether there is a point p that is covered by at least k of the halfplanes.

Lemma 5.19. (Strips-Cover-Box, n) $\leqslant_{\text{QFG}}$ (Point-Covering, n).

Problem: Visibility-Between-Segments
Given a set S of n horizontal line segments in the plane and two particular horizontal segments s_1 and s_2, determine whether there are points on s_1 and s_2 that can see each other, that is, such that the open segment between the points does not intersect any segment in S.

Lemma 5.20. (Sorted-GeomBase, n) $\leqslant_{\text{QFG}}$ (Visibility-Between-Segments, n).

Problem: Visibility-From-Infinity
Given a set S of axis-parallel line segments in the plane and one particular horizontal segment s, determine whether there is a point on s that can be seen from infinity, that is, whether there exists an infinite ray starting at the point on s that does not intersect any segment.

Lemma 5.21. (Sorted-GeomBase, n) $\leqslant_{\text{QFG}}$ (Visibility-From-Infinity, n).

Proof. Same as the proof of Lemma 5.20. □

Problem: Visible-Triangle
Given a set S of opaque horizontal triangles, another horizontal triangle t and a viewpoint p, is there a point on t that can be seen from p?

Lemma 5.22. (Triangles-Cover-Triangle, n) $\leqslant_{\text{QFG}}$ (Visible-Triangle, n).

The proof follows from the quantum on-the-fly adaptation of the classical reduction by [GO95] with the only assumption that the point p of the Visible-Triangle problem is a point at infinity.

The result in the other direction is only relevant for us to present a quantum upper bound for the Visible-Triangle problem, hence, we directly use the classical reduction to make the following statement.

Lemma 5.23 (Theorem 7.3 by [GO95])**.** *There is a* $O(n^{1+o(1)})$ *time quantum algorithm to solve* Visible-Triangle*.*

Proof. Reduce Visible-Triangle to Triangles-Cover-Triangle using the $O(n)$ time classical reduction by [GO95], and then use the quantum algorithm for solving Triangles-Cover-Triangle given by [AL20]. □

Problem: Planar-Motion-Planning
Given a set of non-intersecting, non-touching, axis-parallel line segment obstacles in the plane and a line segment robot (a rod or ladder), determine whether the rod can be moved (allowing both translation and rotation) from a given source to a given goal configuration without colliding with the obstacles.

Lemma 5.24. (Sorted-GeomBase, n) $\leqslant_{\text{QFG}}$ (Planar-Motion-Planning, n).

Problem: 3D-Motion-Planning
Given a set of horizontal (that is, parallel to the xy-plane) non-intersecting, non-touching triangle obstacles in 3D-space, and a vertical line segment as a robot, determine whether the robot can be moved, using translations only, from a source to a goal position without colliding with the obstacles.

Lemma 5.25. (Triangles-Cover-Triangle, n) $\leqslant_{\text{QFG}}$ (3D-Motion-Planning, n).

The following problem, General-Covering problem, was introduced by [AL20] for which they presented a $O(n^{1+o(1)})$ quantum algorithm. Additionally they showed that many computational geometry problems from [GO95] can be solved using the algorithm for General-Covering, thereby giving $O(n^{1+o(1)})$ upper bounds for those problems as well. Refer to the summary of these results in Table 5.1.

Problem: General-Covering
We are given a set of n strips and angles (angle is defined as an infinite area between two non-parallel lines in the plane). The task is to find a point X that satisfies the following conditions:

- the point X is an intersection of two angle or strip boundary lines l_1, l_2 (l_1 and l_2 may be boundary lines of two different angles/strips);
- the point X does not belong to the interior of any angle or strip;
- the point X satisfies a given predicate $P(X)$ that can be computed in $O(1)$ time.

Lemma 5.26 ([AL20])**.** (Strips-Cover-Box, n) $\leqslant_{\text{QFG}}$ (General-Covering, n).

Proof. The reduction from Strips-Cover-Box to General-Covering is as follows: recall that the input to the Strips-Cover-Box contain n strips and a axis-parallel rectangle. Let the same n strips be input to the General-Covering problem. Additionally, as a part of our reduction, we set the predicate

$$P(X) = \begin{cases} 1, & \text{if } X \text{ lies in the rectangle,} \\ 0, & \text{otherwise.} \end{cases}$$

If the General-Covering subroutine on this input cannot find such a point X then it implies that the n strips cover the box completely, and, alternatively, if the algorithm for General-Covering finds such a point then its clear that the strips don't fully cover the box. □

5.5 Other 3SUM-hard problems

We will now show that another variant of the 3SUM problem, the Convolution-3SUM problem, also requires linear time in the quantum setting. Having done that, we are also able to quantise the classical reduction from Convolution-3SUM to 0-Edge-Weight-Triangle consequently proving a $n^{1.5-o(1)}$ time lower bound for the latter. Both of these lower bounds are conditioned on the Quantum 3SUM Conjecture.

The 0-Edge-Weight-Triangle problem can be solved in $O(n^{1.3})$ queries using the quantum-walk-based triangle finding algorithm given by [MSS07].[11] In spite of that, the best known time upper bound for this problem is $O(n^{1.5})$. Our results provide an explanation as to why an $O(n^{1.5-\alpha})$ time quantum algorithm for 0-Edge-Weight-Triangle, for an $\alpha > 0$, has not yet been found.

5.5.1 Lower bound for Convolution-3SUM

Consider the 3SUM' problem: given a list S of n elements, is there a $a, b, c \in S$ such that $a + b = c$? We have seen that the Quantum 3SUM Conjecture is equivalent for this version of 3SUM (Section 5.2). The Convolution-3SUM problem, on the other hand, is defined slightly differently, as follows:

Definition 5.27 (Convolution-3SUM)**.** Given an array $A[1..n]$, determine if there exists indices i, j such that $i \neq j$ and $A[i] + A[j] = A[i + j]$.

There is an obvious $O(n^2)$ classical algorithm and equally obvious $O(n)$ quantum algorithm for Convolution-3SUM (and also, less obviously, for 3SUM'). An interesting classical randomized reduction from 3SUM' to Convolution-3SUM by Pătraşcu [Pat10] shows that a subquadratic algorithm for Convolution-3SUM implies a subquadratic algorithm for 3SUM'. We will combine the ideas in that reduction with the quantum-walk-based reduction introduced earlier, to show that the Convolution-3SUM problem requires $n^{1-o(1)}$ quantum time, unless the Quantum 3SUM Conjecture is false.

[11] Note that 0-Edge-Weight-Triangle problem is different from the Triangle-Finding problem. The current best quantum query algorithm for Triangle-Finding uses $\widetilde{O}(n^{5/4})$ queries [Gal14].

The reduction by Pătraşcu In his paper [Pat10], Pătraşcu showed a reduction from 3SUM' to Convolution-3SUM, the intuition of which is as follows. Assume there is an injective hash function $h : S \to [n]$ which is linear in the sense that $h(a)+h(b) = h(c)$ whenever $a+b = c$. If a such a hash function exists then given an instance of 3SUM' one can create the Convolution-3SUM list by hashing every $a \in S$ to $A[h(a)]$. If there is an $a, b, c \in$ such that $a + b = c$ then by linearity of the hash function, $h(a) + h(b) = h(c)$ which would mean that there exists indices $i = h(a), j = h(b), i + j = h(c)$ such that $A[i] + A[j] = A[i + j]$. Thus, the 3SUM' triple will be discovered by the Convolution-3SUM algorithm. Such a well behaving hash function does not exist, however, it is possible to get something close.

The reduction uses a family of hash functions introduced by [Die96] and used by [Pat10; VW13], defined as follows: pick a random odd element z on w bits, where w is going to be fixed later. For any input $a \in S$, the hash function multiplies a with z on w bits (i.e., **mod** 2^w) and then keeps the high order s bits of the result, which can also be visualised in following way: consider the binary representation of za, pick all the bits between index w to $w - s + 1$ (with the lowest significant bit indexed by 1). Formally,

$$h(a) = (za \bmod 2^w) \div 2^{w-s}, \tag{5.7}$$

where $x \div y$ and $x \bmod y$ denote, respectively, the quotient and remainder of the integer division of x and y. This hash family has the following useful properties.

1. **Almost linear** For any two numbers a and b either $h(a) + h(b) = h(a + b)(\mathrm{mod} 2^s)$ or $h(a) + h(b) + 1 = h(a + b)(\mathrm{mod} 2^s)$.

2. **Few false positives** If $a + b = c$, then $h(a) + h(b) + \{0, 1\} = h(c)(\mathrm{mod}\ 2^s)$ (by which we mean $h(a) + h(b) + b = h(c)(\mathrm{mod} 2^s)$ for some $b \in \{0, 1\}$). Additionally, if $a + b \neq c$, the probability (over the choice of h) that $h(a) + h(b) + \{0, 1\} = h(c)(\mathrm{mod} 2^s)$ is $O(1/2^s)$.

3. **Good load balancing** Fix any n elements $z_1, \ldots, z_n$, let us choose a random hash function as above, and let us place z_i into bucket $h(z_i)$. Let $R = 2^s$ denote the total number of buckets. Then, over the choice of h, any fixed bucket will have n/R elements on average. Also, if we say that a bucket is *bad* if it has more than $3n/R$ elements, then the expected total number of elements that are in bad buckets is $O(R)$.

The classical reduction is given an instance S of 3SUM', chooses a hash function h at random from the above family, and thinks the elements of S as being placed in "buckets", so that $a \in A$ is placed in bucket $h(a)$.

The reduction then has two parts. The first part deals with the elements of the *bad* buckets, i.e. the buckets whose load exceeds $3n/R$. The load-balancing property of this hash function promises that the expected total number of elements in bad buckets is $O(R)$. For every element belonging to a bad bucket, one can in $\widetilde{O}(n)$ classical time decide if it is a part of a solution to the 3SUM' problem, as follows [GO95]: suppose the list S is sorted (classically, we can afford to sort S at the start). For every element a belonging to a bad bucket compute $S + a$. Using simultaneous traversal of these two ordered sets $S + a$ and S one can in $O(n)$ time find if there

is any element common to both. Using this trick for every element belonging to a bad bucket, the entire first part of the reduction then takes $\widetilde{O}(nR)$ classical time.

The second part of the classical reduction creates $O((n/R)^3)$ instances to the Convolution-3SUM problem and only deals with elements of the *good* buckets. This part of the reduction is as follows: for every triple $i, j, k \in \{0, ...3n/R\}$ we create an instance $A_{i,j,k}$ of Convolution-3SUM of size $O(R)$. For each good bucket $t \in [R]$, we map the i^{th} element of the t^{th} bucket to index $8t+1$, j^{th} element to $8t+3$ and k^{th} element to $8t+4$. The locations of the array that have no elements mapped to them can have some large value (for e.g., $2\max(S)+1$) stored in them so that they don't participate in the solution to 3SUM'. If there was a triple $a, b, c \in S$ such that $a + b = c$ then because of the "linear"[12] hash function we get $t_a + t_b = t_c$ where $t_a = h(a), t_b = h(b)$ and $t_c = h(c)$. This means there exists a triple $i, j, k \in \{0, ...3n/R\}$ such that these elements a, b, c get mapped to indices $8t_a{+}1, 8t_b{+}3, 8t_c{+}4$ respectively of the Convolution-3SUM array. Hence, the 3SUM' triple a, b, c is discovered by the Convolution-3SUM algorithm.

Clearly, there will be no false-positives. However, there can be false-negatives: firstly, because the construction mentioned until now only takes care of all the elements on which the hash function behaved exactly linearly, but as we have stated above, the hash function can actually be off-linear by 1. The workaround for this is to simply create another set of Convolution-3SUM instances where for every bucket $t \in [R]$, instead of mapping the i^{th} element of the bucket to index $8t+1$ we map it to $8(t+1)+1$. The second source of false-negatives stems from the fact that Convolution-3SUM only checks for $A[i] + A[j] = A[i+j]$, it misses pairs where $h(x) + h(y) \geqslant R$ (a wrap-around happens modulo R). To fix this, double the array size, including two identical copies. This simulates the wrap-around effect.

Why this reduction doesn't directly hold in the quantum setting We notice that the first part of the classical reduction that takes $\widetilde{O}(nR)$ can be sped up quantumly to take only $\widetilde{O}(\sqrt{nR})$ time, using the claw-finding algorithm of Buhrman, Dürr, Høyer, Magniez, Santha, and de Wolf [BDH+00]. However, the input instance to the claw-finding algorithm needs to be sorted in order for the claw-finding algorithm to run in the required time-bound, and this will correspond to sorting the input S to 3SUM'.

The second part of the classical reduction also needs additional structure. The second part of the reduction produces several instances $A_{i,j,k}$ of Convolution-3SUM, and then uses searches for a positive instance among the $A_{i,j,k}$ using an algorithm for Convolution-3SUM. This search can be sped-up using Grover search. However, the Convolution-3SUM algorithm needs to be able to efficiently read any entry $A_{i,j,k}[\ell]$, and this, in turn, can only be done if we are able to index inside the buckets, i.e., we need to be able to quickly access the i^{th} element of the t^{th} bucket, for any given i, t. The classical algorithm achieves this by simply computing the hash function directly and pre-computing a copy of the input sorted by hash value. This is no longer an option to adapt trivially in the quantum setting as doing so requires $\Omega(n)$ time.

However, it is interesting to note in the quantum walk algorithm over Johnson graph $J(n, r)$ (for an $r = n^\beta$ with $\beta \in (0, 1)$ we can now in the "setup" phase use

[12]The hash function is actually almost linear which will be taken care of in the next paragraph.

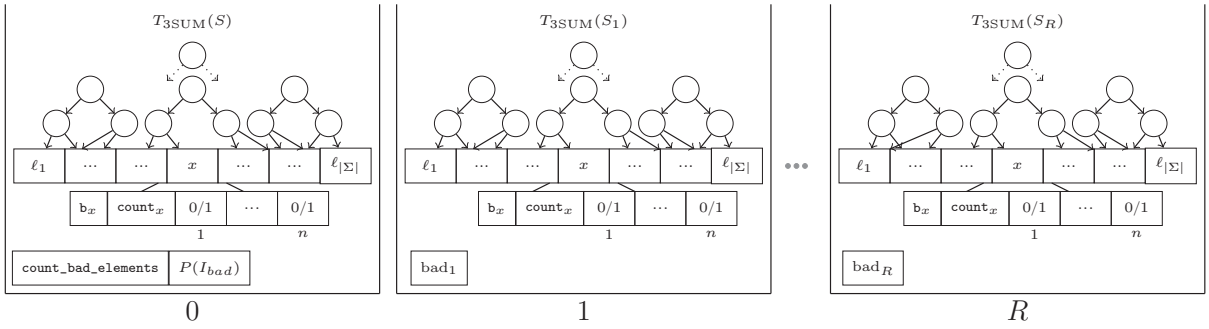

Figure 5.3: Data structure for Convolution-3SUM which stores at most r elements in $R+1$ buckets. The bucket labelled 0 contains the following labels: $T_{3SUM}(S)$ allows us to store the entire set S in a *3SUM tree*, so that we have efficient updates and indexing. The label $P(I_{bad})$ contains the list of indices of all the 'bad' buckets in a prefix tree data structure, and the label `count_bad_elements` stores the total number of 'bad' elements, i.e., the total number elements in all the bad buckets. The buckets indexed by $x \in [R]$ contain two labels: $T_{3SUM}(S_x)$ to store the elements that hashed to value x, and a bit bad_x indicating whether the number of elements in bucket x has exceeded $3r/R$.

additional $\widetilde{O}(r)$ time to store these r sized subsets in dynamic data-structures similar to what is used by Pătraşcu in [Pat10]. What remains is for us to ensure that this data structure allows for efficient updates and queries as required by the quantum walk algorithm. In that way we can then use a quantum-walk plus dynamic data-structure, as in the proof of Theorem 5.2, in order to adapt the classical reduction to the quantum setting.

The data structure In the "setup" phase of our walk algorithm we pick a random odd element z on $w = O(\log n)$ bits and use the hash function as defined in Equation 5.7 on elements $a \in S'$ where $S' \in \Sigma^n$ denotes an n length input to the 3SUM' problem.[13] Having now, picked a value of z we define the data structure that we will use for the 3SUM' to Convolution-3SUM reduction. Note that we are going to use this data structure to store r-sized subsets of S'.

Definition 5.28 (Data Structure for Convolution-3SUM)**.** Let z, w, s be integers as required by Equation 5.7. Let $S \subseteq [n] \times \Sigma$, with $|S| = r$, and such that every $i \in [n]$ appears in at most one pair $(i, x) \in S$. The *convolution-3SUM data structure* that represents S, denoted by $DS_{\text{c-3SUM}}(S)$, is an array of $R+1$ blocks (with $R = r^\alpha$ for any $0 < \alpha < 1$ that can be set later). Each block $x \in [R]$ is labelled by a bit $\texttt{bad}_x \in \{0, 1\}$ and a *3SUM tree* as in Definition 5.5, $T_{3SUM}(S_x)$ representing the set $S_x = \{(i, y) : (i, y) \in S \text{ and } h(y) = x\}$ where

$$h(a) = (za \bmod 2^w) \div 2^{w-s}.$$

[13] To implement this picking of random z one can instead think of the quantum algorithm starting in superposition of different values of z and running our algorithm on this superposed state, as done in Ambainis' walk algorithm for Element Distinctness [Amb07] and also presented in the online version of this work [BLP+22a].

The block 0 is labelled by a number $\texttt{count\_bad\_elements} \in \{0, \ldots, r\}$ and two strings $T_{\text{3SUM}}(S)$ and *prefix-sum tree* $P(I_{bad})$ (as per Definition 4.4) where $I_{bad} = \{i : i \in [R] \text{ and } \texttt{bad}_i = 1\}$. Moreover, the labels obey the following rules:

- For any block x, the label $\texttt{bad}_x = 1$ iff the label $\texttt{count}_{root}$ of $T_{\text{3SUM}}(S_x)$ is greater than $3r/R$.
- Let $\texttt{count}_{i,root}$ denote the value at the label $\texttt{count\_unique\_children}_{root}$ in $T_{\text{3SUM}}(S_i)$, then label $\texttt{count\_bad\_elements} = \sum_{i \in I_{bad}} \texttt{count}_{i,root}$.

See Figure 5.3 for an example of the data structure.

Memory representation A *convolution-3SUM data structure* is represented in the memory by an array of $R+1$ blocks of $\widetilde{O}(n|\Sigma|)$ bits each. Consequently, for every $S \subseteq [n] \times \Sigma$ there is a corresponding binary string of $\widetilde{O}(nR|\Sigma|)$ bits that uniquely encodes $DS_{\text{c-3SUM}}(S)$. Crucially, when $|S| = r$ we have at most $\widetilde{O}(r \log |\Sigma|)$ of these bits are 1. Since $|\Sigma| = \mathsf{poly}(n)$, the encoding is $\widetilde{O}(r)$-sparse.

Implementation of data structure operations We will now verify if all the data structure operations that will be required in our reduction can be done efficiently.

1. **Updates** It is required by the walk algorithm to make efficient updates to $DS_{\text{c-3SUM}}(S)$,i.e. replace an element $(i, x_i) \in S$ with another element (j, x_j) in at most $n^{o(1)}$ time. To achieve that let's look at that the time complexity of the following operations.
   (a) **Find** To find if there exists an element (i, y) in $DS_{\text{c-3SUM}}(S)$ instead check if $(i, y) \in T_{\text{3SUM}}(S)$ contained in block labelled by 0. Using a RAG this can be implemented in constant time.
   (b) **Toggle** To insert (i, y) that is not present in $DS_{\text{c-3SUM}}(S)$, first insert it in $T_{\text{3SUM}}(S)$ in block 0, then having computed the value of $x = h(y)$ insert in $T_{\text{3SUM}}(S_x)$ and also update the value of $\texttt{bad}_x, \texttt{count\_bad\_elements}$ accordingly, if required. If it turns out that $\texttt{bad}_x$ is just set to 1 then insert x into $P(I_{bad})$. All this can be achieved in $O(\log n)$ time. The delete operation is just the reverse of the insertion procedure.
2. **Queries** I.e., the data structure operations that are required by the checking step of the quantum walk. We can efficiently (i.e., in $\mathsf{polylog}(n)$ time) query the i^{th} smallest (by second coordinate) element of S, i^{th} smallest element of I_{bad}, and also the i^{th} smallest element of the t^{th} bucket, for any given i, t.

With this overall data structure, suppose that we were given a structured version of 3SUM', where the input additionally includes the above data structure. Below, using ideas similar to the classical reduction from 3SUM' to Convolution-3SUM, together with a claw-finding algorithm and Grover search, we will reduce this *structured version* of 3SUM' to Convolution-3SUM. This reduction will give us a quantum time lower bound for Convolution-3SUM based on the hardness of this structured version of 3SUM'.

Once this is done, it will suffice to show the hardness of this structured version of 3SUM'. This, in turn, can be done using the same quantum-walk-based reduction that was used in the proof of Theorem 5.2: we do a quantum walk on the Johnson graph, while dynamically preserving at every step the data structure illustrated in Figure 5.3. It should be clear that the data structure can be maintained dynamically, provided we have a data structure for dynamic sorting within each bucket: when inserting or removing an element, we need only compute its hash value to know into which bucket it should be inserted. We will not go into further details on this part of the reduction, and we will now revisit Pătraşcu's reduction from 3SUM' to Convolution-3SUM, in order to show that an analogous reduction can be done, in the quantum setting, from the structured version of 3SUM' to Convolution-3SUM.

Reducing structured 3SUM' to Convolution-3SUM Similar to Pătraşcu's reduction, the first part of our quantum reduction is to deal with *all* the elements of *all* the bad buckets which in expectation are at most $O(R)$ in total. Using the data structure $DS_{\text{c-3SUM}}(S)$ we check the total number of elements in all the bad buckets, if the value exceeds $c' \cdot \log^c \cdot R$, for some constants $c, c' > 1$, then we abort the procedure, otherwise we proceed with the following reduction: suppose that there are k bad buckets in total. We can efficiently find the index of i^{th} bad bucket in $O(\log n)$ time, because of $P(I_{bad})$, the prefix-sum tree storing the set of indices of the bad buckets. Having done that, we then check if there exists any element in this bucket that is part of the solution to the 3SUM' problem. For that we do Grover search over each element a in this bad bucket, and then we need only find a common element in two sorted lists $S + a$ and S of size r each. Such a common element is called a *claw*, and it is known how to find a claw in two sorted lists of size r in quantum time $\widetilde{O}(\sqrt{r})$ [BDH+00]. Let t_i denote the time taken to check if any element in this i^{th} bad bucket is part of the solution to 3SUM', and let s_i denote the number of elements in this i^{th} bad bucket, then $t_i = \widetilde{O}\left(\sqrt{s_i \cdot r}\right)$. Then using variable time search, and because (w.h.p) the total number of bad elements is at most $O(R)$, we can in time

$$O\left(\sqrt{\sum_i t_i^2}\right) = \widetilde{O}\left(\sqrt{\sum_i s_i \cdot r}\right) \leqslant \widetilde{O}\left(\sqrt{R \cdot r}\right),$$

check if any element in the list of *all* bad buckets is part of the 3SUM' solution. Hence, the total time taken for this part of the reduction is $\widetilde{O}(\sqrt{Rr})$.

As explained above, the second part of Pătraşcu's classical reduction from 3SUM' to Convolution-3SUM, creates $O((r/R)^3)$ instances $A_{i,j,k}$ of Convolution-3SUM. Each instance is an array of size $|A_{i,j,k}| = O(R)$, and the different instances are indexed by triples $(i, j, k) \in \{0, ..., 3r/R\}^3$. The algorithm then checks if there is a solution to at least one of the $A_{i,j,k}$, meaning, two indices $\ell_1, \ell_2 \in [|A_{i,j,k}|]$ such that $A[\ell_1] + A[\ell_2] = A[\ell_1 + \ell_2]$. Our quantum reduction will work in the same way, but where we use Grover search to search for a solution among all the triples. For this to be possible, we need to provide fast access to each Convolution-3SUM instance. Formally, given a triple $i, j, k \in \{0, ..., 3r/R\}$ and an index $\ell \in [|A_{i,j,k}|]$, we need to be able to quickly return $A_{i,j,k}[\ell]$. We do the following: let $m = \ell \bmod 8$. If $m \notin \{1, 3, 4\}$ then return a large value such as $2\max(S) + 1$. However if $m \in \{1, 3, 4\}$

then depending on the value of m return i^{th} (if $m = 1$) or j^{th} (if $m = 3$) or k^{th} (if $m = 4$) element of the $q = \lfloor \ell/8 \rfloor^{\text{th}}$ bucket. The data-structure is again used precisely at this point, in order to efficiently obtain the i^{th} (or j^{th} or k^{th}) element of the q^{th} bucket.

As mentioned earlier, the buckets (corresponding to the hash function described in Equation 5.7) along with the support of the data structure (illustrated in Figure 5.3) for each bucket, allow efficient access to the elements contained in these buckets. For example while using the tree data structure, to access i^{th} element of the t^{th} bucket one can simply search for the i^{th} largest element in the *3SUM tree* storing the elements of t^{th} bucket. Additionally, we also know which bucket is *bad* by looking at the value of label $\texttt{bad}_t$. Access to the information about whether a bucket is bad is useful for implementing the first part of the classical reduction which for the choice of $R = r^{\alpha}$ for any $0 < \alpha < 1$ takes strictly sublinear quantum time.

Similarly to the classical reduction, the second part of our quantum reduction checks if a solution exists to at least one of the $O((r/R)^3)$ Convolution-3SUM instances. By doing Grover search over the instances, the quantum time complexity of this part is $\widetilde{O}((r/R)^{1.5} \cdot T_{c\text{-}3SUM}(r'))$ where $T_{c\text{-}3SUM}(r')$ denotes the time taken by a quantum algorithm for Convolution-3SUM on a list of r' elements. In this case $r' = O(R)$.

Therefore, in total, we now have a

$$\sqrt{rR} + (r/R)^{1.5} \cdot T_{c\text{-}3SUM}(R) \tag{5.8}$$

quantum time algorithm for 3SUM', ignoring all the constant and poly-logarithmic factors.

Overview and handling of errors

1. The original quantum walk based query algorithm to solve 3SUM' has a success probability of $1 - o(1)$ [CE05].

2. The 3SUM' subroutine on the r-sized subset (stored on the dynamic data structure), the checking step in the main walk algorithm, could fail. Let the failure probability be p. We will try to reduce p to a small value so that state of the main walk algorithm is very close to state of the algorithm if the checking step was exact (Theorem 2.6 in Chapter 2). Note that, in the actual walk algorithm, this subroutine is repeated $O((n/r)^{1.5})$ times. Therefore setting $p = \frac{1}{3\text{poly}(n)}$ would suffice. We will now see that, this indeed can be done.

   The 3SUM' subroutine on r-sized subsets that are stored on the dynamic data structure (from Figure 5.3) can be reduced to finding a solution in any of the $O(r/R)^3$ instances of Convolution-3SUM for which we supposedly have a bounded-error sublinear quantum algorithm. As we know, given a bounded-error algorithm, we can cheaply reduce this error to any ε by running this subroutine $O(\log(1/\varepsilon))$ many times and taking the majority of the outcomes. With that, the error probability both for the Convolution-3SUM subroutine and the Grover subroutine can be made inverse polynomial in n (of any degree d) by just repeating these subroutines $O(\log n)$ times. However, the data structure, random hash function as mentioned in Equation 5.7, could either fail but with a small probability, or worse it could take longer than

the expected time. We will analyse these two scenarios in the subsequent paragraph.

Failure of data structure operations Let us first revisit the hash function from Equation 5.7 and its properties. The hash function on any element a is,

$$h(a) = (za \bmod 2^w) \div 2^{w-s}, \tag{5.9}$$

where z is a random odd integer of w bits and $s = \Theta(\log w)$. This hash function is probabilistic and has the possible sources of errors:

1. This hash function is *always* (almost) linear, i.e. for any two numbers a and b, $h(a) + h(b) + \{0, 1\} = h(a + b)(\text{mod}2^s)$. Hence, this is not a source of error, in fact similar to the classical case, we only have to create another set of Convolution-3SUM instances with only slight modification to the way the first set of instances are created.

2. However, with probability $O(1/2^s)$, the hash function creates false positive cases, i.e. $h(a) + h(b) + \{0, 1\} = h(c)(\text{mod}2^s)$ even when $a + b \neq c$. For our algorithm to be sublinear in time, the value of R has to be equal to 2^s and r^α, which means $R = n^\zeta$ for some $0 < \zeta < 1$, making the probability of false positive cases equal to $O(1/n^\zeta)$. We propose the following way to deal with this situation.

   The primary goal is to use the Convolution-3SUM subroutine as a black box on instances of size $O(R)$ repeatedly and check if any of these $O(r/R)^3$ instances has a positive solution, for which we use Grover's search subroutine. The result of the Grover's subroutine gives out the details of which instance of size $O(R)$ has the solution, whose validity can be checked in $O(R)$ additional time. This would worsen the complexity to

   $$\sqrt{rR} + (r/R)^{1.5} \cdot T_{c\text{-}3SUM}(R) + R \tag{5.10}$$

   as opposed to what we had in in Equation 5.8. However, given that $R < r$, the calculations in the proof of Corollary 5.29 still goes through.

3. Lastly, the hash function has *good load balancing* property which means expected number of elements in the bad buckets is at most $O(R)$ [Die96]. Using Markov's inequality we can see that the probability of the number of bad elements exceeding k times the expected number of bad elements is upper bounded by $\frac{1}{k}$. Therefore, even for a k as small as $\mathsf{polylog}(n)$ the probability of error (in the asymptotic sense) is arbitrarily close to 0. As discussed earlier, we only proceed with the reduction when the total number of elements in the bad bucket is at most $c' \cdot \log^c n \cdot R$ for some constants $c', c > 1$. Therefore, there will be only small amplitude of our state of the algorithm that will be aborted preemptively; see Section 3.2 of the full online version of this work [BLP+22a].

We can now formally state the main result of this section, that is: a sublinear quantum algorithm for Convolution-3SUM would imply a sublinear algorithm for 3SUM'.

Corollary 5.29. *If there exists a quantum algorithm for* Convolution-3SUM *running in time $O(n^{1-\delta})$, for some constant $\delta > 0$, then Quantum 3SUM Conjecture is false.*

Proof. We choose the number of hash values to be $R = r^{\alpha}$ for some $0 < \alpha < 1$ to be chosen later. Then let $T_{c\text{-}3SUM}(R) = R^{1-\delta}$, for some fixed $\delta > 0$, denote the time taken by a bounded-error quantum time algorithm that solves Convolution-3SUM on $O(R)$-sized inputs. The expression in Equation 5.8 then becomes of order

$$r^{\frac{1+\alpha}{2}} + r^{\frac{3}{2}(1-\alpha)} \cdot r^{\alpha(1-\delta)} \tag{5.11}$$

The first additive term in Equation 5.11 is always sublinear, hence can be ignored. Let us analyse the exponent in the second term, i.e., $\frac{3}{2} - \frac{\alpha}{2} - \alpha\delta$. It is easy to see that for every $\delta > 0$, there exists an α such that $0 < \frac{1}{1+2\delta} < \alpha < 1$ and the expression in Equation 5.11 is strictly sublinear in r. Plugging in this in the proof of Theorem 5.2 would now give us a sublinear algorithm for 3SUM'. Note that the reduction algorithm we described uses $\widetilde{O}(nR|\Sigma|)$ bits of memory, but because the reduction is $\widetilde{O}(r)$-sparse, invoking the result Theorem 4.1 from Chapter 4 we can claim the same result space efficiently as well.

Therefore, a sublinear quantum algorithm for Convolution-3SUM, implies a sublinear algorithm for the structured version of 3SUM', which according to the result of Theorem 5.2 is not possible unless Quantum 3SUM Conjecture is false. □

5.5.2 Quantum lower bound for 0-Edge-Weight-Triangle

The quantum reduction from Convolution-3SUM to 0-Edge-Weight-Triangle problem is a straightforward adaptation of the classical local reduction by [VW13], which is as follows: given an input instance of Convolution-3SUM, an array A of n elements, the reduction, for every $i \in [\sqrt{n}]$ creates an instance G_i which is a tripartite graph with a weight function associated with the edges of each graph. We will show that there exists a 0-Edge-Weight-Triangle in any of these $\sqrt{n}$ graphs, if and only if there exists a solution to the Convolution-3SUM.

For every $i \in [\sqrt{n}]$, create a complete tripartite graph G_i of three partitioned sets of nodes L_i, R_i, S_i which contain $\sqrt{n}$ nodes each. Let $L_i[t], R_i[t], S_i[t]$ denote the t^{th} node of the partition L_i, R_i, S_i, respectively. We then set the weights as follows:

1. $w(L_i[s], R_i[t]) = A[(s-1)\sqrt{n} + t]$,
2. $w(R_i[t], S_i[q]) = A[(i-1)\sqrt{n} + q - t]$,
3. $w(L_i[s], S_i[q]) = -A[(s+i-2)\sqrt{n} + q]$.

Clearly, if there is a triangle in a graph G_i having zero total edge weight, then the value of the weights are solution to the Convolution-3SUM problem. The other direction also holds: suppose there is a solution to the Convolution-3SUM at index i_1, i_2, i_3 such that $A[i_1] + A[i_2] = A[i_3]$ then there exists a tripartite graph G_i with a 0-Edge-Weight-Triangle made by the nodes $L_i[s], R_i[t], S_i[i'_2 + t]$ where $i - 1 = i_2 \div \sqrt{n}$ and $i'_2 = i_2 \bmod \sqrt{n}$ are the quotient and rest of the integer division of i_2 by $\sqrt{n}$ (which we are assuming is an integer, without loss of generality), and $s - 1 = i_1 \div \sqrt{n}$, $t = i1 \bmod \sqrt{n}$. It then holds $i_2 = (i-1)\sqrt{n} + i'_2$ with $i'_2 \in \{0, ..., \sqrt{n} - 1\}$ and $i_1 = (s-1)\sqrt{n} + t$ for $0 \leqslant t < \sqrt{n}$.

As the reduction is completely local, given an index $i \in [\sqrt{n}]$ and any three indices $s, t, q \in [\sqrt{n}]$ we can in constant time query the weights $w(L_i[s], R_i[t])$, $w(R_i[t], S_i[q])$, $w(L_i[s], S_i[q])$ associated with nodes $L_i[s], R_i[t], S_i[q]$.

The following now follows:

Corollary 5.30. *There is no quantum algorithm for the* 0-Edge-Weight-Triangle *problem, running in time $O(n^{1.5-\epsilon})$ for an $\epsilon > 0$, unless Quantum 3SUM Conjecture is false.*

Proof. Let $T(v) = v^\beta$, for some $\beta > 0$, denote the time taken quantumly to compute whether a graph $G = (V, E)$ with $|V| = v$ nodes contains a 0-Edge-Weight-Triangle.

Using Grover's subroutine over $\sqrt{n}$ indices, one can in $O(n^{1/4} \cdot T(\sqrt{n}))$ quantum time check if there exists an index i such that the graph G_i contains a 0-Edge-Weight-Triangle. As argued above, this is equivalent to checking for a solution to Convolution-3SUM on n elements. Therefore, by Corollary 5.29, it is required that $\frac{1}{4} + \frac{\beta}{2} \geqslant 1$, which is to say, $\beta \geqslant \frac{3}{2}$, unless the Quantum 3SUM Conjecture is false. □

5.6 Future directions and open questions

The following is a non-exhaustive list of questions which are currently open, and which we hope will benefit from the approach contained in this chapter:

- Table 5.1 contains four problems for which we can prove some quantum lower bound, conditioned on the Quantum 3SUM Conjecture. Is this lower bound tight, i.e., are there matching algorithms? Or can we prove a higher lower bound, perhaps based on a different conjecture?
- The Classical 3SUM Conjecture itself gives various other lower bounds in the classical setting, which we did not study in the quantum setting, namely lower bounds against dynamic data-structure problems. Can these lower bounds be proven in the quantum regime, also?
- More generally, for what other problems can we prove that the known quantum speed-up is optimal, under a reasonable hardness hypothesis such as the Quantum 3SUM Conjecture?

Additionally, the various papers using dynamic data structures in quantum walks, including [Amb07; ACL+20] and our work, give rise to an interesting question in classical data structures. The vast majority of space-efficient dynamic data structures are not history-independent: history-independence is a feature which cannot be properly motivated if one is only interested in classical algorithms, but which is fundamentally necessary for using the dynamic data structure as part of a quantum walk. One can then attempt to understand for which problems do history-independent, memory and time-efficient dynamic data structures exists. For sorting, the only known solution (skip lists) is randomised. Is this necessary? More generally, what dynamic data-structure problems have solutions that are simultaneously deterministic, time-efficient, space-efficient, and history-independent? Can we prove lower bounds against data structures obeying all four criteria simultaneously, which we cannot prove against data structures obeying only three among the four criteria?

Chapter 6

Matching Triangles & Triangle Collection

Chapter summary Classically, for many computational problems one can conclude time lower bounds conditioned on the hardness of one or more of key problems, such as CNF-SAT (or its other variant such as k-SAT), 3SUM and APSP. In the earlier chapters, we discussed and presented similar results that have been derived in the quantum setting conditioned on the quantum hardness of k-SAT and 3SUM [ACL+20; BPS21; BLP+22a].

Interestingly, for Δ-Matching Triangles and Triangle Collection, two graph problems with natural definitions, classical hardness results have been derived [AVY18] conditioned on the hardness of all three key problems. More precisely, it is proven that an $O(n^{3-\epsilon})$ time classical algorithm (for any constant $\epsilon > 0$) for either of these two graph problems would imply faster classical algorithms for k-SAT, 3SUM and APSP, making Δ-Matching Triangles and Triangle Collection worthwhile to study.

In this chapter, analogous to the classical result, we show that an $O(n^{1.5-\epsilon})$ time quantum algorithm (for any constant $\epsilon > 0$) for either of these two graph problems would imply faster quantum algorithms for k-SAT, 3SUM and APSP. To prove these results, we first formulate a quantum hardness conjecture for APSP analogous to the classical one and then present quantum reductions from k-SAT, 3SUM and APSP to Δ-Matching Triangles and Triangle Collection. Additionally, based on the quantum APSP conjecture, we are also able to prove quantum lower bounds for a matrix problem and many other graph problems. The matching upper bounds follow trivially for most of them, except for Δ-Matching Triangles and Triangle Collection for which we present quantum algorithms that require careful use of data structures and Ambainis' variable time search [Amb10] as a subroutine.

This chapter shares results presented in Leijnse's Masters' thesis [Lei22], both of which are built on the following paper:

[ABL+22] Andris Ambainis, Harry Buhrman, Koen Leijnse, Subhasree Patro, Florian Speelman. Matching Triangles and Triangle Collection: Hardness based on a Weak Quantum Conjecture. Preprint available at *arXiv:2207.11068*.

6.1 Introduction

The All Pairs Shortest Path (APSP) problem is defined as follows.

Definition 6.1 (APSP)**.** Given a weighted (directed or undirected) graph $G = (V, E)$ on $n = |V|$ vertices with no negative cycles, for every pair of vertices $(a, b) \in V \times V$, output the shortest distance between vertices a, b if there is a path, else output ∞.

The current fastest known classical algorithm for APSP runs in $n^3/\exp(\sqrt{\log n})$ time [Wil18], and it has been conjectured that no $O(n^{3-\epsilon})$ time classical algorithm, for any constant $\epsilon > 0$, is possible. Based on this conjecture, time lower bounds for a lot of problems, for example, Graph Radius, Graph Median, Negative Triangle and many more, have been concluded. See an excellent survey of these results by Vassilevska Williams [Vas15].

Surprisingly in the quantum setting, as opposed to a quadratic speedup over the classical cubic upper bound, the fastest known quantum algorithm solving APSP runs in $\widetilde{O}(n^{2.5})$ time; one can either use the $\widetilde{O}(n^{1.5})$ time algorithm for solving the *Single Source Shortest Paths* problem for every vertex in the input graph, or (as we will soon see in Section 6.2.2) one could use repeated "squaring" of the weight matrix of the graph, not squaring in the usual sense but under a different definition of matrix multiplication, to output the distance matrix D such that $D[u, v]$ contains the length of the shortest path between nodes u, v.[1]

Despite several efforts through multiple approaches, no significant speedup to the $\widetilde{O}(n^{2.5})$ time quantum upper bound is known for APSP, it is therefore natural to study the consequences of the following conjecture.

Conjecture 6.2 (Quantum APSP Conjecture)**.** There is no bounded error quantum algorithm that solves APSP on a graph of n nodes in $O(n^{2.5-\delta})$ time, for any constant $\delta > 0$.

As a first natural step, we study the classical reductions from APSP to computational problems mentioned in Table 6.1 and (unexcitedly) observe that almost all these reductions can be trivially adapted to the quantum setting; this especially happens because there is now sufficient time to process the input for most of these reductions just as it is done in the classical setting; rest of the classical reductions can be easily modified to behave on-the-fly. Moreover, the matching upper bounds for all these problems in Table 6.1 can be derived using Grover-like speed-ups, *except* for Δ-Matching Triangles and Triangle Collection for which we (excitedly) present quantum algorithms that require careful use of data structures and use Ambainis' variable time search as a subroutine. Additionally, we are able to conclude the optimality of these algorithms based on the conjectured quantum hardness of all the three key problems, CNF-SAT, 3SUM and APSP, just as it can be in the classical setting.[2]

[1] Note that the first algorithm gives the shortest paths tree and the later outputs the shortest distance matrix, but both of them take $\widetilde{O}(n^{2.5})$ time in the quantum setting. See Section 6.2 and Leijnse's Masters' thesis for a discussion on this [Lei22].

[2] For Δ-Matching Triangles, the bounds are tight for $\omega(1) \leqslant \Delta(n) \leqslant n^{o(1)}$.

6.1.1 A weak quantum conjecture

Abboud, Williams, and Yu prove that an $O(n^{3-\epsilon})$ time classical algorithm (for any constant $\epsilon > 0$ and $\omega(1) \leqslant \Delta(n) \leqslant n^{o(1)}$) for either of these two graph problems would imply faster classical algorithms for k-SAT, 3SUM and APSP [AVY18], making Δ-Matching Triangles and Triangle Collection worthwhile to study.

This means one can now make fewer hardness assumptions when it comes to understanding the hardness of Δ-Matching Triangles and Triangle Collection. Towards that, a weaker conjecture was introduced which states that at least one of the classical 3SUM-conjecture, APSP-conjecture or SETH is true. They called it the *extremely popular conjecture*.

Analogous to Abboud et al.'s surprising classical result, we are also able to show that an $O(n^{1.5-\epsilon})$ time quantum algorithm (for any constant $\epsilon > 0$ and the same ranges of Δ) for either of these two graph problems would imply faster quantum algorithms for k-SAT, 3SUM and APSP. Clearly, for problems such as Δ-Matching Triangles and Triangle Collection a weaker quantum hardness assumption can be made to conclude time lower bounds. Hence, we state the following conjecture analogous to the classical case.

Conjecture 6.3. At least one of Conjecture 3.5, 5.1 or 6.2 is true.

Based on Conjecture 6.3, we are able to prove tight quantum time bounds for Δ-Matching Triangles and Triangle Collection. The upper bounds are non-trivial and will be the content of the next subsection.

6.1.2 Upper bounds for Δ-Matching Triangles and Triangle Collection

The Δ-Matching Triangles and Triangle Collection problems are two naturally occurring graph problems, whose definitions are as follows.

Definition 6.4 (Δ-Matching Triangles)**.** Given a graph $G = (V, E)$ with a colouring of the vertices $\gamma : V \to \Gamma$ with $|\Gamma| \leqslant n$, determine if there is a triple of colours $i, j, k \in \Gamma$ such that there are at least Δ triangles $a, b, c \in V$ for which $(\gamma(a), \gamma(b), \gamma(c)) = (i, j, k)$.

Note that the range of Δ can vary between $0 \leqslant \Delta \leqslant n^3$.

Definition 6.5 (Triangle Collection)**.** Given a graph $G = (V, E)$ with a colouring of the vertices $\gamma : V \to \Gamma$ with $|\Gamma| \leqslant n$, determine if for every triple of colours $i, j, k \in \Gamma$ there is at least one triangle $a, b, c \in V$ for which $(\gamma(a), \gamma(b), \gamma(c)) = (i, j, k)$.

Quantum algorithms for many of the graph problems we consider in this work, such as Negative Triangle or 0-Edge-Weight-Triangle, can be found using an easy application of Grover's algorithm. The algorithms for Δ-Matching Triangles and Triangle Collection, however, are less trivial — these will heavily use *Variable Time Search* by Ambainis [Amb10] which can be stated as follows.

Theorem 6.6 ([Amb10])**.** *Given a string $x \in \{0,1\}^n$, the task is to find i such that $x_i = 1$. Additionally, let t_i be the maximum time required to evaluate x_i. There exists a quantum algorithm that with probability at least $\frac{2}{3}$ outputs some $j \in [n]$ such that $x_j = 1$ else outputs 0. The algorithm takes $\widetilde{O}(\sqrt{\sum_{i=1}^n t_i^2})$ time and makes $\widetilde{O}(\sqrt{\sum_{i=1}^n t_i^2})$ queries to x.*

The algorithm for Δ-Matching Triangles is more complicated than the one presented for Triangle Collection, but with the similarity that they both use the Variable Time Search of Theorem 6.6. Hence, we will sketch the algorithm for Triangle Collection first and proceed to discuss the intuition of the algorithm for Δ-Matching Triangles after that. Also, see Section 6.4 for the details of these algorithms.

Algorithm for Triangle Collection Recall its definition: given a graph $G = (V, E)$, we want to know if for every triple of colours there is a triangle in the graph $G = (V, E)$. This is the same as knowing if there is a triple of colours such that there is *no* triangle of that colour triple in G. Making use of this simple observation we do the following: let's assume a subroutine that given a colour triple $(i, j, k) \in \Gamma^3$ outputs yes if there is a triangle of this colour in the input graph G, and no otherwise. Furthermore, let $t_{i,j,k}$ be the time taken for the subroutine on colour triple (i, j, k). Invoking Theorem 6.6 we can now conclude that the total time taken on a graph of n nodes is $T(n) = O(\sqrt{\sum_{(i,j,k)\in\Gamma^3} t^2_{i,j,k}})$. Suppose that with some pre-processing of the input, this is where the data structures come to use, we could *efficiently* access nodes of G coloured by i for any $i \in \Gamma$. Then in $t_{i,j,k} = \widetilde{O}(\sqrt{|V_i| \cdot |V_j| \cdot |V_k|})$ time, where V_i, V_j, V_k denotes the sets of nodes with colours i, j, k respectively, we can find if there is a triangle in G of colour i, j, k. Which means

$$T(n) = O\left(\sqrt{\sum_{i,j,k\in\Gamma^3} t^2_{i,j,k}}\right) = \widetilde{O}\left(\sqrt{\sum_{i,j,k\in\Gamma^3} |V_i| \cdot |V_j| \cdot |V_k|}\right)$$
$$= \widetilde{O}\left(\sqrt{\sum_{i\in\Gamma} |V_i| \sum_{j\in\Gamma} |V_j| \sum_{k\in\Gamma} |V_k|}\right) = \widetilde{O}\left(n^{1.5}\right).$$

On the other hand, the algorithm for Δ-Matching Triangles is slightly more complicated. Apart from using the *Variable Time Grover Search* subroutine, the algorithm makes use of different ways of counting the number of triangles in a graph. To state some of these several methods:

1. Given a graph $G = (V, E)$ of n nodes and access to the adjacency matrix A_G corresponding to G, one can count the number of triangles by computing the trace of A^3_G. Here A^3_G refers to matrix multiplication of A_G with itself three times. This can be achieved classically in $O(n^\omega)$ time, where ω denotes matrix multiplication constant currently known to be at 2.3728.

2. Or, one could use the threshold variant of Grover Search, with which one can check if there are at least k triangles in a graph of n nodes in $O(n^{1.5+\frac{k}{2}})$ time.

Intuition for algorithm for Δ-Matching Triangles Recall the definition of Δ-Matching Triangles: given a graph $G = (V, E)$ with n nodes and a colouring of the nodes $\gamma : V \to \Gamma$, is there a triple of colour $(i, j, k) \in \Gamma^3$ such that there are at least Δ triangles of that colour triple. Clearly, trivial brute forcing (in spite of the Grover-like speedup) over all triples of colours and checking if any of the colour triples has at least Δ triangles is going to cost $O(\sqrt{\Delta n^3})$. While this bound is sub-cubic for small Δ, however when $\Delta \approx n^3$ we get the same upper bound as

Problem		Classical	Quantum
(min, +)-Matrix Multiplication	Lower bound	$n^{3-o(1)}$ [FM71; Mun71]	$n^{2.5-o(1)}$ Lemma 6.9
	Upper bound	$O(n^3)$ $(*)$	$O(n^{2.5})$ $(**)$
All-Pairs Negative Triangle	Lower bound	$n^{3-o(1)}$ [VW18]	$n^{2.5-o(1)}$ Lemma 6.13
	Upper bound	$O(n^3)$ $(*)$	$O(n^{2.5})$ $(**)$
Negative Triangle	Lower bound	$n^{3-o(1)}$ [VW18]	$n^{1.5-o(1)}$ Lemma 6.15
	Upper bound	$O(n^3)$ $(*)$	$O(n^{1.5})$ $(**)$
0-Weight Triangle	Lower bound	$n^{3-o(1)}$ [VW13]	$n^{1.5-o(1)}$ Lemma 6.18
	Upper bound	$O(n^3)$ $(*)$	$O(n^{1.5})$ $(**)$
Δ-Matching Triangles	Lower bound	$n^{3-o(1)}$ [AVY18] (†)	$n^{1.5-o(1)}$ Lemma 6.20 (†)
	Upper bound	$O(n^{3-o(1)})$ [AVY18] (†)	$\widetilde{O}(n^{1.5+o(1)})$ Corollary 6.33 (†)
Triangle Collection	Lower bound	$n^{3-o(1)}$ [AVY18]	$n^{1.5-o(1)}$ Lemma 6.23
	Upper bound	$O(n^3)$ $(*)$	$\widetilde{O}(n^{1.5})$ Theorem 6.34

Table 6.1: Overview of lower bounds based on a hardness conjecture for APSP, both in the classical and in the quantum setting. Corresponding upper bounds are also provided.
$(*)$: These upper bounds are the most straightforward algorithms, like exhaustive search, and therefore have no particular source.
$(**)$: By applying Grover Search, potentially as a subroutine.
(†): Holds only for $\omega(1) \leqslant \Delta \leqslant n^{o(1)}$.

the classical one. What helps is firstly the observation that when Δ is *too big* then there aren't that many triples of colours that one needs to brute force over. Secondly, having fixed a triple of colour, it is more efficient to use brute force search over matrix multiplication if the number of nodes restricted to the triple of colour is small. Using these two simple but crucial observations, we get $\widetilde{O}(n^{1.5}\sqrt{\Delta})$ for $1 \leqslant \Delta(n) \leqslant n^{\omega}$ and $\widetilde{O}(\frac{n^{1.5+\omega}}{\sqrt{\Delta}})$ for $n^{\omega} \leqslant \Delta(n) \leqslant n^3$. We present the details and the exact parameters of the algorithm in Section 6.4.

Structure of this chapter In Section 6.2 we formulate a quantum hardness conjecture for APSP, then study reductions from APSP to several computational problems. See Table 6.1 for an overview of the results and Figure 6.1 captures how these problems computationally relate to each other. In Section 6.3, we present quantum fine-grained reductions for Δ-Matching Triangles and Triangle Collection from all three key problems. In Section 6.4, we give non-trivial quantum time upper bounds for Δ-Matching Triangles and Triangle Collection. Finally, we conclude in Section 6.5.

6.2 Quantum fine-grained reductions from APSP

6.2.1 All Pairs Shortest Path (APSP) problem

Definition 6.1 (APSP)**.** Given a weighted (directed or undirected) graph $G = (V, E)$ on $n = |V|$ vertices with no negative cycles, for every pair of vertices $(a, b) \in V \times V$, output the shortest distance between vertices a, b if there is a path, else output ∞.

Note that, there is one other version of the APSP problem where the algorithm solving it is required to output the shortest *path* between every pair of vertices in the

graph. Classically, both these versions of APSP are conjectured to require $n^{3-o(1)}$ time; Hardness is conjectured for the distance version of APSP [Vas15; VW18], which immediately implies that the path version of APSP also requires $n^{3-o(1)}$ time. Quantumly, APSP can be solved in $\widetilde{O}(n^{2.5})$ time for both these mentioned versions of APSP, except that for the path version of the APSP problem the algorithm outputs a *shortest path tree* for every vertex instead of outputting shortest path between every pair of vertices in the graph. It is easy to see that otherwise achieving such an upper bound would not be feasible: for example, consider the n-cycle graph $G = (V, E)$: for every node $v \in V$ in G, we will require a total of $\widetilde{\Omega}(n^2)$ bits to write down all the paths from v to nodes $w \in V$. Doing this for every node in the graph would then require $\widetilde{\Omega}(n^3)$ bits, hence $\widetilde{\Omega}(n^3)$ time.

One can solve APSP by using the quantum algorithm for *single source shortest path* (SSSP) problem by [DHH+06] on every vertex in the graph.

Theorem 6.7 (Implicit in Theorem 18 [DHH+06])**.** *There is a bounded error* $\widetilde{O}(n^{1.5})$ *time quantum algorithm to solve* SSSP *on a graph of* n *nodes.*[3]

Consequently, this gives a $\widetilde{O}(n^{2.5})$ time quantum algorithm for APSP both for the path and the distance versions of the problem. Moreover, they also show that single source shortest path problem requires $\Omega(n^{1.5})$ queries, hence also $\Omega(n^{1.5})$ time, in the quantum model of computation we are considering. While there might not be sufficient reason to believe that the only way to solve APSP is through solving n instances of SSSP (hence APSP must require $n^{2.5-o(1)}$ time), we do have additional reasons to present the Quantum APSP Conjecture the way it is; especially because there are no faster algorithms known for APSP or for any of the problems in the web of reductions stemming from APSP.

In this thesis, we primarily study the *distance* version of this problem as stated in Definition 6.1 and conjecture the following.

Conjecture 6.2 (Quantum APSP Conjecture)**.** There is no bounded error quantum algorithm that solves APSP on a graph of n nodes in $O(n^{2.5-\delta})$ time, for any constant $\delta > 0$.

6.2.2 Reductions from APSP

The quantum lower bounds stated here follow trivially from the classical reductions presented in the following papers: [FM71; Mun71; AVY18; VW18; VW13]. Some adjustments are necessary, however, for boosting the accuracy of our subroutines. We sketch the reductions here sometimes without going into much detail about the correctness of these reductions. See Leijnse's Masters' thesis for a more comprehensive overview of all these classical reductions [Lei22].

The first couple of results are quite straightforward, we can use the classical reductions with occasional on-the-fly methods, and we derive many quantum lower bounds based on the quantum hardness of APSP. We are also able to retain the hardness-equivalence with $(\min, +)$-Matrix Multiplication just as it was first discovered in [FM71; Mun71] for the classical setting. These two reductions are somewhat considered folklore, for detailed descriptions we again refer to [Lei22].

[3]Note that, while their theorem was originally stated for the adjacency array model of input, this upper bound holds for the case where the access to the input $G = (V, E)$ is given via a matrix $M \in \{0,1\}^{n\times n}$, with $M[u, v] = 1$ iff $(u, v) \in E$ for all pairs $(u, v) \in V \times V$.

Definition 6.8 ((min, +)-Matrix Multiplication)**.** Given two $n \times n$ matrices $M, N \in \mathcal{R}^{n \times n}$ with $\mathcal{R} = [-n^c, n^c]$, compute the distance product $M \star N$ defined as follows:

$$(M \star N)[ij] := \min_{k \in [n]}(M[ik] + N[kj]).$$

Lemma 6.9. $(\text{APSP}, n^{2.5}) \leqslant_{\text{QFG}} ((\min, +)\text{-Matrix Multiplication}, n^{2.5})$.

Proof. Let $G = (V, E)$ be an input to the APSP problem with $n = |V|$ nodes and weight function $w : E \to \mathbb{R}$ such that there are no negative cycles in the graph. Let W denote the weight matrix corresponding to G, i.e., for all $u, v \in V$ we set

$$W[uv] = \begin{cases} 0 & \text{if } u = v, \\ w(u, v) & \text{if } (u, v) \in E, \\ \infty & \text{if } (u, v) \notin E. \end{cases}$$

In the context of this proof, let $W^k := W \star W \star \ldots \star W$, with k copies of W in the sequence, defined for all $k \in [n]$. Interestingly, W^n is the *shortest distance matrix* of G, i.e., $W^n[uv]$ contains the length of the shortest path between vertices u and v; the proof follows using induction. See Theorem 3.2.1 [Lei22] for the same.

Let $T(k)$ denote the number of calls to the (min, +)-Matrix Multiplication subroutine we make for computing W^k; with that, we have $T(1)$=0. We can now compute W^n using the algorithm for (min, +)-Matrix Multiplication and repeated squaring method as follows: to compute $W^n = W^{n/2} \star W^{n/2}$, first compute $A = W^{n/2}$ and then compute $A \star A$. Therefore we have $T(n) = T(n/2) + 1$. Solving this recurrence relation gives us $T(n) = O(\log n)$. Therefore, with $O(\log n)$ calls to (min, +)-Matrix Multiplication we can solve APSP. Additionally, to boost the success probability of each of these $O\log(n)$ subroutine calls we need to repeat each of them $O(\log(\log(n)))$ times. Therefore, if (min, +)-Matrix Multiplication can be solved in $O(n^{2.5-\alpha})$ time, for some $\alpha > 0$, then APSP can be solved in $\widetilde{O}(n^{2.5-\alpha})$ time as well. Hence, $(\text{APSP}, n^{2.5}) \leqslant_{\text{QFG}} ((\min, +)\text{-Matrix Multiplication}, n^{2.5})$ as per Definition 2.16 in Chapter 2. □

Interestingly, a similar result holds in the other direction as well.

Lemma 6.10. $((\min, +)\text{-Matrix Multiplication}, n^{2.5}) \leqslant_{\text{QFG}} (\text{APSP}, n^{2.5})$.

Proof. Let M and N be two $n \times n$ matrices. We will construct a tripartite graph G in a way that we can read the product $M \star N$ from its distance matrix D that is the output of the algorithm for APSP on input G.

Construct $G = (V, E)$ with a vertex set $V = A \cup B \cup C$ s.t. $A = \{a_1, , , a_n\}$, $B = \{b_1, , , b_n\}$ and $C = \{c_1, , , c_n\}$. For every $M[ij]$ we add edge (a_i, b_j) to E with weight $w(a_i, b_j) = M[ij]$, and for every $N[ij]$ we add edge (b_i, c_j) to E with weight $w(b_i, c_j) = N[ij]$. For a pair of nodes a_i, c_j it holds that the distance of its shortest path is $\min_{b_k \in B}(w(a_i, b_k) + w(b_k, c_j)) = \min_{1 \leqslant k \leqslant n}(M[ik] + N[kj]) = (M \star N)[ij]$.

Therefore, we can compute all the entries $(M \star N)[ij]$ first by running the APSP algorithm on G and then in the distance matrix D which is its output, look for the entries corresponding to nodes $a_i, c_j \in V$. Do this for all $i, j \in [n]$ to generate the matrix $M \star N$. All this processing takes $O(n^2)$ time; hence, an $O(n^{2.5-\alpha})$ time algorithm for APSP (for an $\alpha > 0$) implies a $O(T(n))$ time algorithm for (min, +)-Matrix Multiplication where $T(n) = n^{\max(2.5-\alpha, 2)}$, therefore proving the statement of Lemma 6.9. □

Both these reductions from Lemma 6.10 and Lemma 6.9 were straightforward implementations of their respective classical counterparts. Moreover, by combining both these results we obtain the following.

Corollary 6.11. $(\text{APSP}, n^{2.5}) =_{\text{QFG}} ((\min,+)\text{-Matrix Multiplication}, n^{2.5})$.

What follows now is a chain of reductions from $(\min,+)$-Matrix Multiplication. We first define All-Pairs Negative Triangle and then prove a conditional quantum lower bound for this problem in the same way as it was shown in the classical setting [VW18]; this problem is rather contrived, but it is useful in connecting APSP and $(\min,+)$-Matrix Multiplication to other naturally defined graph problems.

Definition 6.12 (All-Pairs Negative Triangle)**.** Given a tripartite weighted graph $G = (A \cup B \cup C, E)$ over $O(n)$ nodes and with an associated weight function $w : E \to [-n^c, n^c]$, for every pair of nodes a, b such that $a \in A$ and $b \in B$ determine whether there exists $c \in C$ such that nodes a, b, c form a triangle of negative weight in G.

Lemma 6.13. $(\min,+)$-Matrix Multiplication *is quantum* $(n^{2.5}, n^{2.5})$*-reducible to* All-Pairs Negative Triangle.

Proof. Consider matrices M, N as an input to the $(\min,+)$-Matrix Multiplication problem. Let $M, N \in \mathcal{R}^{n\times n}$ with $\mathcal{R} = [-n^c, n^c]$. Just as it is in the classical reduction given by [VW18], construct a tripartite graph $G = (A \cup B \cup C)$ of $O(n)$ nodes with $A = \{a_1, \ldots, a_n\}$, $B = \{b_1, \ldots, b_n\}$ and $C = \{c_1, \ldots, c_n\}$. Then $\forall j, k \in [n]$ add an edge (b_j, c_k) to E with weight $w(b_j, c_k) = -N[jk]$ and $\forall k, i \in [n]$ add an edge (c_k, a_i) to E with weight $w(c_k, a_i) = -M[ki]$.

Let S be a $n \times n$ matrix with $S[ij]$ denoting a value in $[-2n^c, 2n^c]$. We will soon show how to fix these values. For every i, j, add an edge (a_i, b_j) in E with weight $w(a_i, b_j) = S[ij]$. Initially, $\forall i, j$ set $S[ij] = 2n^c$. Call the subroutine for All-Pairs Negative Triangle on graph G. At first there will be no i, j such that $a_i, b_j \in V \times V$ have a negative triangle. For each i, j we binary search overall values in $[-2n^c, 2n^c]$ to set the value of $S[ij]$ in such a way that the All-Pairs Negative Triangle subroutine on those indices has a negative triangle, once found we do not alter the value of that $S[ij]$ any more. We continue this process until the All-Pairs Negative Triangle algorithm has negative triangles for all $a_i, b_j \in V \times V$. It is then not hard to see that the final matrix S is actually $M \star N$. Therefore, with $O(\log n)$ applications of the All-Pairs Negative Triangle subroutine we are able to compute $(\min,+)$-Matrix Multiplication. Additionally, to boost the success probability of the individual All-Pairs Negative Triangle calls, we apply each one of them $O(\log(\log(n)))$ times. Therefore, we can solve $(\min,+)$-Matrix Multiplication in $\widetilde{O}(n^{2.5-\alpha} + n^2)$ time using a $O(n^{2.5-\alpha})$ time algorithm for All-Pairs Negative Triangle. Hence, proved. □

For all the computational problems we have seen so far in this chapter, the (conditional) time complexities have been n^3 classically and $n^{2.5}$ quantumly. Interestingly, we now notice a divergence between the classical and quantum time complexities of Negative Triangle. See Figure 6.1.

Definition 6.14 (Negative Triangle)**.** Given a graph $G = (V, E)$ of $n = |V|$ nodes with weight function $w : E \to [-n^c, n^c]$, determine if G has a triangle with negative weight, i.e., is there a triple of nodes $a, b, c \in V$ such that $w(a, b) + w(b, c) + w(c, a) < 0$.

The classical reduction from All-Pairs Negative Triangle to Negative Triangle given by [VW18] is as follows: given a weighted tripartite graph $G = (A \cup B \cup C, E)$ over $n = |V|$ nodes and a weight function $w : E \to [-n^c, n^c]$ construct N graphs as inputs to the Negative Triangle problem, an N which we will soon fix. First split the sets A, B, C into n^α sets of $n^{1-\alpha}$ vertices each, for some $\alpha > 0$. For every triple of these sets, we are going to create a sub-graph of G. So in total there will be $N = n^{3\alpha}$ tripartite sub-graphs of size $3n^{1-\alpha}$ each. For each of these N sub-graphs, run the subroutine that solves Negative Triangle and if the subroutine outputs 'yes' then *find* the triangle, let's say the triangle found has nodes a, b, c originally belonging to sets A, B, C, respectively, and then remove the edge (a, b) from all the sub-graphs, and make a note that the nodes a, b did have a negative weight triangle. Repeat until no new triangles are detected in any of the remaining sub-graphs. For those leftover nodes in set A, B output that no negative triangle exists for these nodes.

Note that, we only have access to the subroutine for Negative Triangle that detects a negative triangle but doesn't necessarily find it. Using the subroutine for Negative Triangle we can also find the triangle with a $O(\log n)$ overhead in time: split the set of vertices into four (roughly) equal parts. We know that there must be at least one combination of three vertex sets that contains the nodes of a negative triangle. Run Negative Triangle on sub-graphs induced by each of these $\binom{4}{3}$ combinations of these sets separately. As soon as we have determined such a triple of three vertex sets using the Negative Triangle algorithm, we eliminate the remaining fourth of the vertex set. We repeat this step, eliminating a fourth of the remaining vertex sets at every iteration until only three vertices remain. If the Negative Triangle subroutine takes $T(n)$ time on n sized set then the total run time to find a negative triangle is

$$O\left(T(n) + T(\frac{3n}{4}) + T(\frac{9n}{16}) + \dots\right) = O(T(n)\log n), \tag{6.1}$$

whenever $T(n) = \mathsf{poly}(n)$. Additionally, for each graph we need to make $\log(n)$ calls to the Negative Triangle subroutine and to boost the success probability of each call, we apply each one of them $\log(\log(n))$ times.

Therefore the total time taken for solving All-Pairs Negative Triangle is $\widetilde{O}(T(n^{1-\alpha})(n^{3\alpha} + n^2))$ because there are at most $O(n^2)$ edges to be removed in the input graph G. Plugging in $\alpha = 2/3$ we get the time complexity as $\widetilde{O}(T(n^{1/3}) \cdot n^2)$. Hence, if $T(n) = n^{1.5-\epsilon}$ for any $\epsilon > 0$ then we have a $\widetilde{O}(n^{\frac{1.5-\epsilon}{3}+2}) = \widetilde{O}(n^{2.5-\frac{\epsilon}{3}})$ time quantum algorithm for All-Pairs Negative Triangle. With that, we can now claim the following result.

Lemma 6.15. All-Pairs Negative Triangle *is quantum* $(n^{2.5}, n^{1.5})$*-reducible to* Negative Triangle.

Note that, in the classical setting we had cubic lower bounds for all problems from APSP to Δ-Matching Triangles and Triangle Collection, however,

quantumly we can only conclude a $n^{1.5-o(1)}$ lower bound for Negative Triangle for graphs with n nodes. While a lower bound of $n^{1.5-o(1)}$ does not imply that a better lower bound of e.g. $n^{2.5-o(1)}$ does not exist for Negative Triangle, however, given that a simple Grover search of all triples of nodes results in a matching upper bound, it *does* imply that a larger lower bound for Negative Triangle would be impossible.

One could argue that it may come as more of a surprise that Negative Triangle is as hard as All-Pairs Negative Triangle or APSP classically than that we find a gap in computational complexity in the quantum case. We look for a single triangle in Negative Triangle and for n^2 potential triangles in All-Pairs Negative Triangle. The quantum model highlights these gaps in the difficulty of the problems in a way that the classical model could not. In this sense working in a quantum model on itself is already providing us with useful insights into the complexity of these problems, without requiring that we have a physical quantum algorithm to implement our reductions. An unfortunate consequence of this gap in lower bound complexities is that we lose the classical reduction from Negative Triangle to $(\min,+)$-Matrix Multiplication, since this reduction 'as-is' does not let us go *up* in complexity. In fact, the definition of fine-grained reductions makes it very challenging to prove computational lower bounds conditioned on smaller computational lower bounds, especially when the instances are similarly structured.

Less surprisingly, using the reduction from [VW13] we find a similar lower bound of $n^{1.5-o(1)}$ for the 0-Edge-Weight-Triangle problem.

Definition 6.16 (0-Edge-Weight-Triangle)**.** Given a weighted graph $G = (V, E)$ over $n = |V|$ nodes and with a weight function $w : E \to [-n^c, n^c]$, determine if there is a triangle in G with weight 0, i.e., is there a triple of nodes $a, b, c \in V$ such that $w(a, b) + w(b, c) + w(c, a)=0$.

Using the following property of integers, which is a corrected version of Proposition 3.4 by [VW13], and the classical reduction from Negative Triangle to 0-Edge-Weight-Triangle also by [VW13] we then show a quantum reduction from the Negative Triangle problem to the 0-Edge-Weight-Triangle problem.

Lemma 6.17. *For all integers x, y, z we have that $x + y > z$ if and only if there exists a k such that either:*

$$\lfloor \frac{x}{2^k} \rfloor + \lfloor \frac{y}{2^k} \rfloor = \lfloor \frac{z}{2^k} \rfloor + 1$$

or it holds that

$$\lfloor \frac{x}{2^k} \rfloor + \lfloor \frac{y}{2^k} \rfloor = \lfloor \frac{z}{2^k} \rfloor \quad \textit{and} \quad \lfloor \frac{x+y}{2^k} \rfloor = \lfloor \frac{x}{2^k} \rfloor + \lfloor \frac{y}{2^k} \rfloor + 1.$$

Proof. We first prove that if one of the two above-mentioned conditions holds then $x + y > z$.

1. There exists a k such that
$$\lfloor \frac{x}{2^k} \rfloor + \lfloor \frac{y}{2^k} \rfloor = \lfloor \frac{z}{2^k} \rfloor + 1.$$
For any integers x, y, k it holds that $\frac{x+y}{2^k} \geqslant \lfloor \frac{x+y}{2^k} \rfloor \geqslant \lfloor \frac{x}{2^k} \rfloor + \lfloor \frac{y}{2^k} \rfloor$. Furthermore, for any z, it holds that $\lfloor \frac{z}{2^k} \rfloor + 1 > \frac{z}{2^k}$. Combining all these inequalities we get $\frac{x+y}{2^k} > \frac{z}{2^k}$ for that k, therefore, $x + y > z$.

2. There exists a k such that

$$\lfloor\frac{x}{2^k}\rfloor + \lfloor\frac{y}{2^k}\rfloor = \lfloor\frac{z}{2^k}\rfloor \quad \text{and} \quad \lfloor\frac{x+y}{2^k}\rfloor = \lfloor\frac{x}{2^k}\rfloor + \lfloor\frac{y}{2^k}\rfloor + 1.$$

Which means for that k,

$$\lfloor\frac{x+y}{2^k}\rfloor = \lfloor\frac{x}{2^k}\rfloor + \lfloor\frac{y}{2^k}\rfloor + 1 = \lfloor\frac{z}{2^k}\rfloor + 1.$$

But for all x, y, z, k it holds that $\frac{x+y}{2^k} \geqslant \lfloor\frac{x+y}{2^k}\rfloor$ and $\lfloor\frac{z}{2^k}\rfloor + 1 > \frac{z}{2^k}$. Therefore, we again get $\frac{x+y}{2^k} > \frac{z}{2^k}$ for that k, therefore, $x + y > z$.

Now we prove the other direction, that is if $x + y > z$ then one of the conditions mentioned in the statement of Lemma 6.17 must hold; having $x + y > z$ implies $\frac{x+y}{2^k} > \frac{z}{2^k}$ for all k. Alternatively, this also means there exists a k such that $\lfloor\frac{x+y}{2^k}\rfloor = \lfloor\frac{z}{2^k}\rfloor + 1$. But $\lfloor\frac{x+y}{2^k}\rfloor \geqslant \lfloor\frac{x}{2^k}\rfloor + \lfloor\frac{y}{2^k}\rfloor$ for any k. Therefore there exists a k such that $\lfloor\frac{z}{2^k}\rfloor + 1 \geqslant \lfloor\frac{x}{2^k}\rfloor + \lfloor\frac{y}{2^k}\rfloor$. We have two cases to analyse from this.

1. If $\lfloor\frac{z}{2^k}\rfloor + 1 = \lfloor\frac{x}{2^k}\rfloor + \lfloor\frac{y}{2^k}\rfloor$ then we have the first condition in the statement of Lemma 6.17. Hence, partially proved.

2. If $\lfloor\frac{z}{2^k}\rfloor + 1 > \lfloor\frac{x}{2^k}\rfloor + \lfloor\frac{y}{2^k}\rfloor$ while $\lfloor\frac{x+y}{2^k}\rfloor = \lfloor\frac{z}{2^k}\rfloor + 1$, which is the scenario we are in, then these two statements can simultaneously hold only when $\lfloor\frac{x+y}{2^k}\rfloor = \lfloor\frac{x}{2^k}\rfloor + \lfloor\frac{y}{2^k}\rfloor + 1$ which means $\lfloor\frac{x}{2^k}\rfloor + \lfloor\frac{y}{2^k}\rfloor = \lfloor\frac{z}{2^k}\rfloor$.

Hence, proved. □

Additionally, we will assume that the input to Negative Triangle is tripartite, and that is a fine assumption because the lower bound we get in Lemma 6.15 also holds under this promise that inputs to Negative Triangle are tripartite.

Lemma 6.18. (Negative Triangle, $n^{1.5}$) $\leqslant_{\text{QFG}}$ (0-Edge-Weight-Triangle, $n^{1.5}$).

Proof. Let $G = (V, E)$, a tripartite graph with weight function $w : E \to [-n^c, n^c]$ and $V = A \cup B \cup C$, be an input to the Negative Triangle problem.

We will now invoke the 0-Edge-Weight-Triangle subroutine on $2\log(n^c)$ new instances of G, namely on $G_{i,j}$ for every $i \in [\log(n^c)], j \in \{0, 1\}$. The new instances $G_{i,j}$ are constructed in the following way: $G_{i,j}$ has the same vertices and edges but different weight function $w_{i,j} : E \to [-n^c, n^c]$. For every edge $(a, b), (b, c), (c, a) \in E$ with $a \in A, b \in B, c \in C$, we set $w_{i,j}(a, b) = \lfloor\frac{-w(a,b)}{2^i}\rfloor$, $w_{i,j}(b, c) = \lfloor\frac{-w(b,c)}{2^i}\rfloor$ and $w_{i,j}(c, a) = \lfloor\frac{w(c,a)}{2^i}\rfloor + j$.

For each of these graphs we run the 0-Edge-Weight-Triangle algorithm. Because of the result from Lemma 6.17 we can make the following claim: whenever we detect a 0-weight triangle in a graph $G_{i,1}$ we know there must be a negative triangle in G. If we detect a 0-weight triangle a, b, c in a graph $G_{i,0}$ for some i we use the triangle finding algorithm just as it was done before statement of Lemma 6.15 to locate it and check whether $\lfloor\frac{x+y}{2^k}\rfloor = \lfloor\frac{x}{2^k}\rfloor + \lfloor\frac{y}{2^k}\rfloor + 1$ holds. This way we determine if there is indeed a negative triangle in G. We therefore have to run the 0-Edge-Weight-Triangle subroutine for $O(\log n)$ times.

Also, the quantum reduction has to take strictly less than $n^{1.5-O(1)}$ time, therefore we cannot use the classical reduction as it is, however we can make it on-the-fly: for any i, j, we can compute any entry of the adjacency matrix for graph

$G_{i,j}$ in $\widetilde{O}(1)$ time via the input oracle to G. Additionally, to boost success probability of the individual calls to the 0-Edge-Weight-Triangle subroutine, we incur another $O(\log(\log(n)))$ factor overhead in time. This means we can solve Negative Triangle in $\widetilde{O}(n^{1.5-O(1)})$ time using a $O(n^{1.5-O(1)})$ time algorithm for 0-Edge-Weight-Triangle. Hence, proved. □

Like in the classical case, we find no reduction from 0-Edge-Weight-Triangle up the chain of reductions towards APSP or 3SUM. Note that for all problems for which we have proven quantum lower bound up to this point, we can easily match this lower bound by a simple application of Grover search.

The reduction from 0-Edge-Weight-Triangle to Δ-Matching Triangles by [AVY18] however highlights some interesting aspects about the complexity of Δ-Matching Triangles.

Definition 6.4 (Δ-Matching Triangles)**.** Given a graph $G = (V, E)$ with a colouring of the vertices $\gamma : V \to \Gamma$ with $|\Gamma| \leqslant n$, determine if there is a triple of colours $i, j, k \in \Gamma$ such that there are at least Δ triangles $a, b, c \in V$ for which $(\gamma(a), \gamma(b), \gamma(c)) = (i, j, k)$.

The reduction from 0-Edge-Weight-Triangle to Δ-Matching Triangles requires another lemma which was used in [ALW14] and later reformulated in [AVY18].

Lemma 6.19 (Lemma 2.1 by [AVY18])**.** *For all integers $p, d, s, n, c \geqslant 1$ if $p \geqslant 3n^{\lceil \frac{c}{d} \rceil}$, there is a set of $s = 2^{O(d)}$ functions $f_1, \ldots, f_s : [-n^c, n^c] \to [-\frac{p}{3}, \frac{p}{3}]^d$ and s many target vectors $t_1, \ldots, t_s \in [-p, p]^d$, computable in $O(\log(n))$ time, such that for all numbers $x, y, z \in [-n^c, n^c]$ it holds that*

$$x + y + z = 0 \text{ iff } \exists j \in [s] \text{ s.t. } f_j(x) + f_j(y) + f_j(z) = t_j$$

The reduction from 0-Edge-Weight-Triangle to Δ-Matching Triangles and Triangle Collection requires us to turn a weighted graph instance into a coloured graph instance. In order to encode the edge weights we use Lemma 6.19 to encode the weight space $[-n^c, n^c]$ of a graph into d-dimensional vectors and construct a node for each possible edge weight. We see it in more detail in the following reduction which is a straightforward adaptation of the classical reduction presented in Theorem 3.2.8 by [Lei22] but originally given by [AVY18].

Lemma 6.20. (0-Edge-Weight-Triangle, $n^{1.5}$) $\leqslant_{\text{QFG}}$ (Δ-Matching Triangles, $n^{1.5}$) *for $\omega(1) \leqslant \Delta(n) \leqslant o(\log(n))$.*

Proof. Let $G(V, E)$ be a weighted graph with weight function $w : E \to [-n^c, n^c]$ and assume that it is tripartite with vertex partition $V = A \cup B \cup C$ and let $A = \{a_1, \ldots, a_n\}$, $B = \{b_1, \ldots, b_n\}$ and $C = \{c_1, \ldots, c_n\}$. We use Lemma 6.19 setting $d = \Delta, p = O(n^{\frac{c}{\Delta}})$ to construct $s = 2^{O(\Delta)}$ functions $f_i : [-n^c, n^c] \to [-n^{\frac{c}{\Delta}}, n^{\frac{c}{\Delta}}]^\Delta$ for our $n = |V|$. Now we use f_i to construct unweighted coloured graphs $G_i = (A_i \cup B_i \cup C_i, E_i)$ for each $i \in [s]$.

For every $a \in A$ add Δ copies labelled a_j for $j \in [\Delta]$ to A_i and let their color be a. For each $b \in B$ add $\Delta \cdot 2n^{\frac{c}{\Delta}}$ copies to B_i labelled $b_{j,x}$ for $j \in [\Delta]$ and $x \in [-n^{\frac{c}{\Delta}}, n^{\frac{c}{\Delta}}]$ and with colour b. Similarly, add $\Delta \cdot 2n^{\frac{c}{\Delta}}$ copies $c_{j,x}$ with colour c for each $c \in C$

to C_i. That is, we add a node to B and C for every dimension up to Δ and every possible vector entry value in $[-n^{\frac{c}{\Delta}}, n^{\frac{c}{\Delta}}]$.

Now for the edges: for each $(a,b) \in A \times B$ add the edges $(a_j, b_{j,f_i(w(a,b))[j]})$ to E_i for every $j \in [\Delta]$. For each $(b,c) \in B \times C$ add the edges $(b_{j,x}, c_{j,x+f_i(w(b,c))[j]})$ to E_i for every $j \in [\Delta]$. Finally, for each $(c,a) \in C \times A$, add edges $(c_{j,t_i[j]-f_i(w(c,a))[j]}, a_j)$ to E for every $j \in [\Delta]$. That is, we have edges going only from nodes in the same dimension, with the specific value for the b and c nodes given by our constructed function f_i. Every a_j has one outgoing edge and every b_j, x has one outgoing edge, even for the values of x that don't have an edge incoming from a_j. For every c only Δ of the $\Delta 2n^{\frac{c}{\Delta}}$ nodes $c_{j,x}$ has an outgoing edge.

We now claim that G has a 0-weight triangle if and only if there exists some $i \in [s]$ such that G_i has Δ matching triangles; suppose G has a 0-weight triangle a, b, c. It follows from Lemma 6.19 that there is some $i \in [s]$ such that $f_i(w(a,b)) + f_i(w(b,c)) + f_i(w(c,a)) = t_i$. In the following set for ease of notation $x = f_i(w(a,b)), y = f_i(w(b,c))$ and $z = f_i(w(c,a))$. We know that there exists an $i \in [s]$ such that

$$x[j] + y[j] + z[j] = t_i[j],$$

for each $j \in [\Delta]$. By our construction it always holds for a graph G_i that $(a_j, b_{j,x[j]})$, $(b_{j,x[j]} c_{j,(x+y)[j]})$, $(c_{j,-z[j]}, a_j) \in E_i$ for each $j \in [\Delta]$. Since $(x+y)[j] = (t_i - z)[j]$ by the previous observation, it follows that $c_{j,(x+y)[j]} = c_{j,t_i[j]-z[j]}$ and the triple of nodes $(a_j, b_{j,x[j]}, c_{j,-z[j]})$ forms a triangle for every $j \in [\Delta]$. Furthermore, since the triangle $(a_j, b_{j,x[j]}, c_{j,-z[j]})$ is the same colour for every $j \in [\Delta]$, we have Δ matching triangles.

Conversely, suppose there are Δ matching triangles in G_i for some $i \in [s]$. We know from the construction of our graphs G_i that the colours must correspond to a triple of nodes $(a,b,c) \in A \times B \times C$ in the graph G. Every $a_j \in A_i$ has only one outgoing edge $(a_j, b_{j,x})$ and so it must be that $x = f_i(w(a,b))[j]$. The node $b_{j,x}$ also only has one outgoing edge $(b_{j,x}, c_{j,x+y})$ and it must be that $y = f_i(w(b,c))[j]$. Lastly, since a_j has only one incoming edge $(c_{j,t_i[j]-z}, a_j)$ it must be that $z = t_i[j] - f_i(w(c,a))[j]$ and as a result $f_i(w(a,b))[j] + f_i(w(b,c))[j] = t_i[j] - f_i(w(c,a))[j]$ for each $j \in [\Delta]$. It follows by Lemma 6.19 that $w(a,b) + w(b,c) + w(c,a) = 0$.

We can use the above algorithm to solve 0-Edge-Weight-Triangle using a Δ-Matching Triangles oracle. We construct $2^{O(\Delta)}$ graphs of $O(\Delta n \cdot n^{\frac{c}{\Delta}})$ nodes and $O(\Delta m n^{\frac{c}{\Delta}})$ edges each. To keep the number of graphs small enough, the reduction only works for providing a $n^{1.5-O(1)}$ time algorithm for 0-Edge-Weight-Triangle if $\Delta = o(\log n)$. On the other hand, to ensure that the individual graph sizes are small enough, we need to ensure that $\Delta = \omega(1)$.

Computing the edges of our graphs requires functions computed using Lemma 6.19; each of these functions are computable in $O(\log n)$ time. It follows that for $\omega(1) \leqslant \Delta \leqslant o(\log(n))$ we can use $O(n^{1.5-\epsilon})$ time Δ-Matching Triangles algorithm to solve 0-Edge-Weight-Triangle in $\widetilde{O}(n^{1.5-\epsilon})$ time. The additional logarithmic factor we incur is to boost the success probability of calls made to the Δ-Matching Triangles oracle. Thus the lemma follows. □

Additionally, the following classical result from [AVY18] also holds in the quantum setting. Using this we can prove quantum hardness of Δ-Matching Triangles for a larger range of Δ as mentioned in Corollary 6.22.

Lemma 6.21 (Lemma 2.4 by [AVY18])**.** *If* Δ-Matching Triangles *on graphs with* n *nodes can be solved by a* $O(n^{1.5-\epsilon})$ *time algorithm for some* $\epsilon > 0$ *and for* $\omega(1) \leqslant \Delta \leqslant n^{o(1)}$*, then* Δ'-Matching Triangles *can be solved in* $\tilde{O}(n^{1.5-\epsilon})$ *time for* $\omega(1) \leqslant \Delta' \leqslant o(\log(n))$.

Proof. Given an instance of Δ'-Matching Triangles with a graph G on n nodes with $\omega(1) \leqslant \Delta' \leqslant o(\log(n))$, we add $\Delta - \Delta'$ nodes to each of the colour. Then take the i^{th} newly added node in all colours, make them a complete graph. It adds exactly $\Delta - \Delta'$ triangles to every triple of colours. Then run the $O(n^{1.5-\epsilon})$ time quantum algorithm for solving Δ-Matching Triangles on the new graph. The total running time is $O(((\Delta - \Delta')n)^{1.5-\epsilon})$; which is $\tilde{O}(n^{1.5-\epsilon})$ for the stated ranges of Δ, Δ'. □

With that we get the following result.

Corollary 6.22. (0-Edge-Weight-Triangle, $n^{1.5}$) $\leqslant_{\text{QFG}}$ (Δ-Matching Triangles, $n^{1.5}$) *for* $\omega(1) \leqslant \Delta(n) \leqslant n^{o(1)}$.

Proof. Follows from Lemma 6.20 and Lemma 6.21. □

Here it is less clear whether the lower bound of $n^{1.5-o(1)}$ is the best lower bound we can find for Δ-Matching Triangles. On an intuitive level, this definitely seems more complex than problems like the Negative Triangle problem or the 0-Edge-Weight-Triangle problem. It is important to note that the above reduction only holds for the specified values of Δ: $\omega(1) \leqslant \Delta(n) \leqslant n^{o(1)}$. We will see in Section 6.4 that for these ranges of Δ, there is indeed a matching upper bound. This leaves open the question of whether we can find a reduction from 0-Edge-Weight-Triangle to Δ-Matching Triangles for ranges of Δ that are polynomial or constant in the number of nodes in the graph. For polynomial values of Δ we are faced with the same challenge as in reducing Negative Triangle to $(\min, +)$-Matrix Multiplication in the quantum case: we would be trying to increase the complexity of the lower bound through fine-grained reduction, going from $n^{1.5-o(1)}$ to potentially $n^{2.5-o(1)}$, depending on values of Δ. In the next section we will see why $n^{2.5-o(1)}$ could be a reasonable quantum lower bound for Δ-Matching Triangles for unrestricted values of Δ.

The reduction from 0-Edge-Weight-Triangle to Triangle Collection by [AVY18], on the other hand, makes use of the construction from the reduction from 0-Edge-Weight-Triangle to Δ-Matching Triangles for Δ values of $O(\sqrt{\log n})$, which is in the regime of Δ where we found an $n^{1.5-o(1)}$ lower bound for Δ-Matching Triangles. As a consequence, we get a similar result for Triangle Collection.

Definition 6.5 (Triangle Collection)**.** Given a graph $G = (V, E)$ with a colouring of the vertices $\gamma : V \to \Gamma$ with $|\Gamma| \leqslant n$, determine if for every triple of colours $i, j, k \in \Gamma$ there is at least one triangle $a, b, c \in V$ for which $(\gamma(a), \gamma(b), \gamma(c)) = (i, j, k)$.

Lemma 6.23 (Theorem 3.2.10 by [Lei22])**.** 0-Edge-Weight-Triangle *is quantum* $(n^{1.5}, n^{1.5})$*-reducible to* Triangle Collection.

Proof. We will use the same graph construction as in the proof of Lemma 6.20 but use it to construct new graphs $G'_i = (A'_i \cup B'_i \cup C'_i, E'_i)$ such that there is a 0-weight triangle in G if and only if there is a G'_i where we can't collect all colour triples.

From each G_i from Lemma 6.20, we construct the graph G'_i by inverting all the edges in G_i. In the case that there is no 0-weight triangle in G, we want all colour triples to be collected in every G'_i, which includes colour triples with colours of the same parts in G, e.g. $(a, b, b') \in A \times B \times B$. To do so, we finish the constructions of the graphs G'_i by adding nodes and edges to G'_i.

For every $a \in A$, add two nodes a_B, a_C to A'_i with colour a. For every $b \in B$ add nodes b_A, b_C to B'_i with colour b. Lastly, for $c \in C$ add c_A, c_B to C'_i with colour c.

For any pair of nodes $a, a' \in A$ we add edges (a_B, a'_B) and (a_C, a'_C) to E'_i. Similarly, for $b, b' \in B$ and $c, c' \in C$ we add edges $(b_A, b'_A), (b_C, b'_C), (c_A, c'_A), (c_B, c'_B)$ to E'_i. Finally, for pair $(a, b) \in A \times B$, we add edge (a_B, b_A) for pair $(b, c) \in B \times C$, we add (b_C, c_B) and for pair $(c, a) \in C \times A$ we add (c_A, a_C) to E'_i.

This finishes off our construction of graphs G'_i, now we prove that there is no 0-weight triangle in G if and only if there is no triangle collection in G'_i for some $i \in [2^{O(\Delta)}]$.

Suppose $(a, b, c) \in A \times B \times C$ forms a 0-weight triangle in G. Let G_i be the graph containing Δ-matching triangles from the proof of Lemma 6.20. We saw that every a_j in G_i can only be part of one triangle. Since there are Δ of each a_j, every such a_j must be part of a triangle in G_i if there is a 0-weight triangle in G. In our inverted graph G'_i, there can therefore be no triangle with colour a. By adding the extra nodes and edges to G'_i, we have not made any new triangles that have a node in the three different parts of G'_i and there is therefore no triangle collection for graph G'_i.

Conversely, suppose there is no 0-weight triangle in G. Following the reasoning in the previous paragraph and the proof of Lemma 6.20, it must be that we collect every possible triple of colours $(a, b, c) \in A \times B \times C$ in every graph G'_i. We now show that we also collect every triple of colours that is not in $A \times B \times C$. Suppose we have three colours corresponding to three nodes from the same part in G, e.g. $a, a', a'' \in A$. Then nodes a_B, a'_B, a''_B from a triangle in G'_i for every i. If two colours come from one part of G and the third from another, e.g. $(a, a', b) \in A \times A \times B$, the nodes a_B, a'_B, b_a form a triangle in G'_i for every i.

Complexity-wise we have the bounds to Δ as in the proof of Lemma 6.20 and we can instantiate our construction by setting $\Delta = \sqrt{\log n}$. □

Here the question now really comes down to whether we can find a $O(n^{1.5})$ matching upper bound for Triangle Collection, which we do in the Section 6.4.

6.3 Matching Triangles & Triangle Collection: lower bounds

Through results from the previous section, and the quantum reductions from 3SUM to 0-Edge-Weight-Triangle from Chapter 5 we now have lower bounds on Δ-Matching Triangles and Triangle Collection from the disjunction of hardness conjectures for APSP and 3SUM. To complete the picture, we need to verify that the classical reductions from k-SAT, both for Δ-Matching Triangles and Triangle Collection taken from [AVY18], continue to hold in the quantum case. For k-SAT we go through the Sparsification Lemma given by [IP01], arriving at the same lower bound we found through 0-Edge-Weight-Triangle for

$\omega(1) \leqslant \Delta \leqslant n^{o(1)}$ in Δ-Matching Triangles. We observe that their reductions also directly hold for k-SAT and CNF-SAT, however, the lower bound implied is only for a smaller range of $\omega(\mathsf{poly}(\log n)) \leqslant \Delta \leqslant n^{o(1)}$ as shown in Figure 6.1.

Let us first look at the reduction from SAT to Δ-Matching Triangles given by [AVY18]. They proved the following result.

Lemma 6.24 (Lemma 2.3 by [AVY18])**.** *If* Δ-Matching Triangles *on a graph of* N *nodes can be solved in* N^{c_Δ} *time, then* CNF-SAT *on* n *variables and* m *clauses can be solved in* $O((\Delta 2^{\frac{n}{3}+\frac{m}{3\Delta}})^{c_\Delta})$ *time.*

The only thing we need from their proof is that their reduction takes an instance of CNF-SAT on n variables and m clauses and constructs a graph G on $N = O(\Delta 2^{\frac{n}{3}+\frac{m}{3\Delta}})$ nodes. To make sure that their classical reduction holds *as it is* in the quantum setting, we first need to ensure that the values of Δ are in the range that lets us keep $N = O(2^{\frac{n}{2}(1-O(1))})$. We need a Δ which ensures that $\frac{m}{3\Delta} = o(1)$ and is also strictly sub-exponential; setting $\Delta = n^d$ where $m = n^{<d}$ works. With this value of $\Delta = \omega(\mathsf{poly}(n))$ we get that N is at most $O(2^{\frac{n}{2}(1-O(1))})$.

What we now observe is the following. One can solve CNF-SAT on n variables and m clauses quantumly in $O(N + N^{c_\Delta})$ time using a $O(N^{c_\Delta})$ time quantum algorithm for Δ-Matching Triangles, where $N = \Delta \cdot 2^{\frac{n}{3}+o(1)}$.

Having done that, we now plug in the values of $c_\Delta = (1.5 - \epsilon)$ for some $\epsilon > 0$. It is not hard to see that for every such ϵ there exists a corresponding $\epsilon' > 0$ such that we can solve CNF-SAT in $O(2^{\frac{n}{2}(1-\epsilon')})$ using the reduction from Lemma 6.24. However, note that $\Delta = \omega(\mathsf{poly}(n)) = \omega(\mathsf{poly}(\log N))$ in this reduction and cannot be greater than $N^{o(1)}$. Therefore we get the following result.

Lemma 6.25. $(\text{CNF-SAT}, 2^{n/2}) \leqslant_{\text{QFG}} (\Delta\text{-Matching Triangles}, n^{1.5})$ *for* $\omega(\mathsf{poly}(\log n)) \leqslant \Delta \leqslant n^{o(1)}$.

Fortunately, using the *sparsification lemma* as stated below, we can further extend this range of Δ.

Lemma 6.26 (Sparsification Lemma by [IP01; IPZ01]; value of c achieved by [CIP06])**.** *Let* ϕ *be a* k*-CNF formula over* n *variables and with* m *clauses for* $k \geqslant 3$*. For any* $\epsilon' > 0$ *there is an* $O^*(2^{\epsilon' n})$ *time algorithm that produces* $O(2^{\epsilon' n})$ k*-CNF formulas* $\phi_1, \ldots, \phi_{O(2^{\epsilon' n})}$ *over* n *variables and at most* cn *clauses where* $c = (\frac{k}{\epsilon'})^{O(k)}$*. It then holds that* ϕ *is in* SAT *if and only if there is a satisfying assignment to* $\bigvee_{i=1}^{O(2^{\epsilon' n})} \phi_i$.

For any $\epsilon' < \frac{1}{2}$ the sparsification lemma can be applied as it for quantum reductions from k-SAT. We will now use this sparsification lemma with the classical reduction CNF-SAT to Δ-Matching Triangles as stated in Lemma 6.24 to prove the following result.

Lemma 6.27. $(k\text{-SAT}, 2^{n/2}) \leqslant_{\text{QFG}} (\Delta\text{-Matching Triangles}, n^{1.5})$ *for* $\omega(1) \leqslant \Delta \leqslant n^{o(1)}$.

Proof. Let ϕ be a k-CNF instance over n variables and m clauses. Fix an $\epsilon' < \frac{1}{2}$ and then apply the sparsification lemma, Lemma 6.26, to compute $2^{\epsilon' n}$ k-CNF formulas ϕ_i with at most cn clauses each, for some constant value c, which is only dependent on k, ϵ', in $O^*(2^{\epsilon' n})$ time.

Suppose that we have a $O(N^{\frac{3}{2}-\epsilon})$ time Δ-Matching Triangles algorithm for graphs with N nodes and $\omega(1) \leqslant \Delta(N) \leqslant N^{o(1)}$. We then apply the reduction from [AVY18] to decide whether ϕ_i is satisfiable for each $i \in [2^{\epsilon' n}]$ in $O((\Delta 2^{\frac{n}{3}+\frac{nc}{3\Delta}})^{\frac{3}{2}-\epsilon})$ time. Since $\omega(1) \leqslant \Delta(N) \leqslant N^{o(1)}$ it follows that $O((\Delta 2^{\frac{n}{3}+\frac{nc}{3\Delta}})^{\frac{3}{2}-\epsilon}) = O(2^{n(\frac{1}{2}-\frac{\epsilon}{3})+o(1)})$. The total time to evaluate the disjunction over all the sparse formulas will be $O(2^{n(\frac{1}{2}-\frac{\epsilon}{3}+\epsilon')+o(1)}) = O(2^{\frac{n}{2}(1-\frac{2}{3}\epsilon+2\epsilon')+o(1)})$.

For every $\epsilon > 0$ there exists a value of $\epsilon' < \min\{\frac{1}{2}, \frac{\epsilon}{3}\}$ such that k-SAT can have a $O^*(2^{\frac{n}{2}(1-O(1))})$ time quantum algorithm. Hence, proved. □

The classical reduction from CNF-SAT to Triangle Collection in Theorem 3.3.5 by [Lei22] can be used as it is to give us the same quantum lower bound as we get via 0-Edge-Weight-Triangle. Their reduction takes as input a CNF formula over n variables and m clauses and constructs a graph of $N = O(m2^{\frac{n}{3}})$ nodes as an input to Triangle Collection. As $m = \mathsf{poly}(n)$, we can use the classical reduction in exactly same way and then run a presumably $N^{1.5-\epsilon}$ time quantum algorithm to solve the Triangle Collection problem. This way we can solve solve CNF-SAT in $O^*(2^{\frac{n}{2}(1-\frac{2\epsilon}{3})} + 2^{\frac{n}{3}}) = 2^{\frac{n}{2}(1-O(1))}$ time quantumly. Therefore, we can claim the following.

Lemma 6.28. $(\text{CNF-SAT}, 2^{\frac{n}{2}}) \leqslant_{\text{QFG}} (\text{Triangle Collection}, n^{1.5})$.

Also note that the result of Lemma 6.28 straightforwardly applies to k-SAT as well. Also mentioned in Figure 6.1.

Using transitivity of quantum fine-grained reductions we are able to the present the same quantum time lower bounds for Δ-Matching Triangles and Triangle Collection from all the three, CNF-SAT, 3SUM and APSP, key problems. Therefore, we can now state the following.

Theorem 6.29. *If* Triangle Collection *or* Δ-Matching Triangles *for* $\omega(\mathsf{poly}(\log n)) \leqslant \Delta \leqslant n^{o(1)}$ *can be solved in time* $O(n^{1.5-\epsilon})$ *for some* $\epsilon > 0$*, then Conjecture 6.3 must be false.*

Furthermore, if we conjecture that the k-SAT problem also requires $2^{\frac{n}{2}(1-o(1))}$ time in the quantum setting as stated below

Conjecture 6.30 ([ACL+20])**.** For all $\epsilon > 0$, there exists some $k \in \mathbb{N}$ such that there is no quantum algorithm solving k-SAT in time $O(2^{\frac{n}{2}(1-\epsilon)})$.

We can then claim the following.

Theorem 6.31. *If* Triangle Collection *or* Δ-Matching Triangles *for* $\omega(1) \leqslant \Delta \leqslant n^{o(1)}$ *can be solved in time* $O(n^{1.5-\epsilon})$ *for some* $\epsilon > 0$*, then at least one of following Conjectures 6.30, 5.1 or 6.2 must be false.*

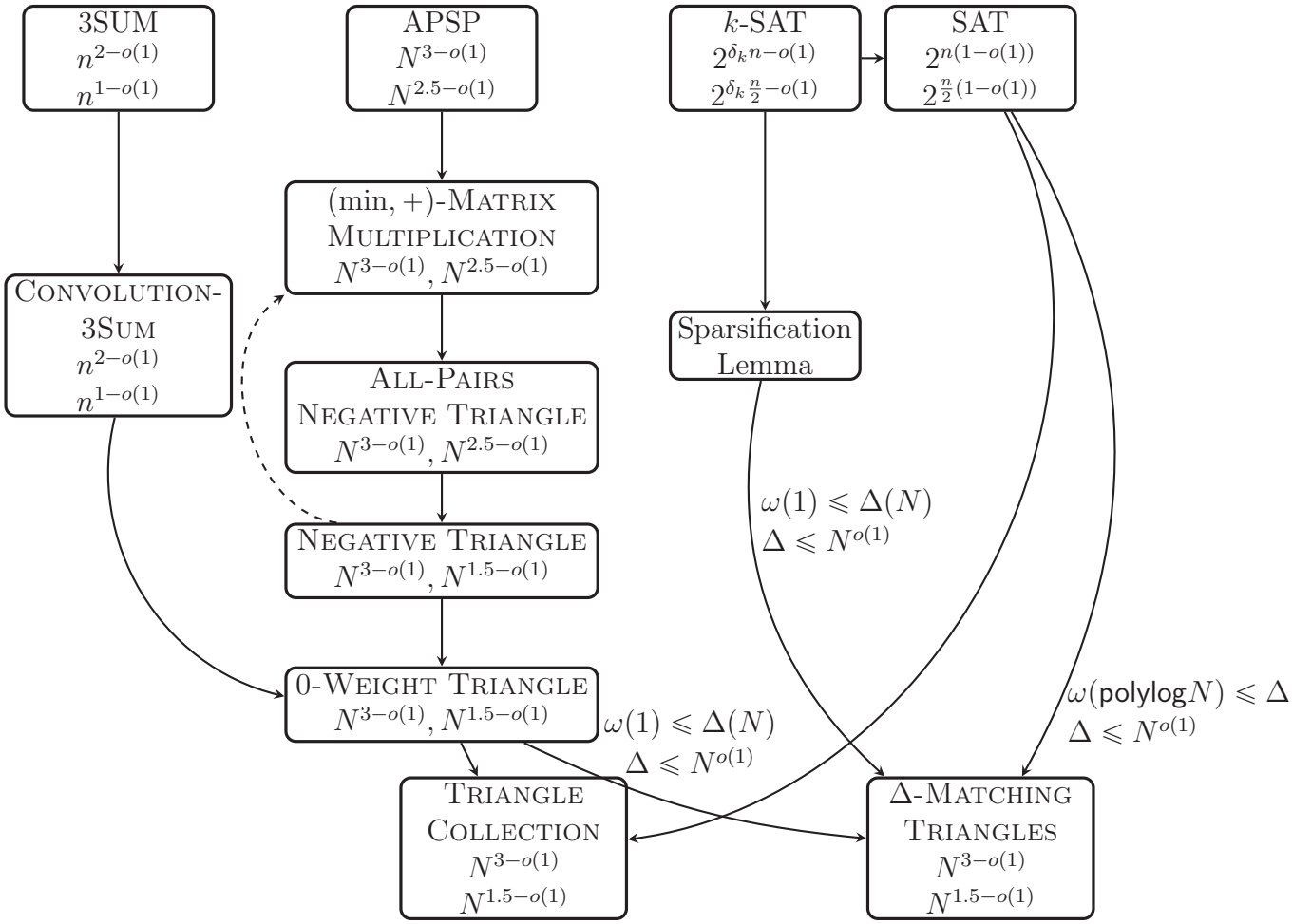

Figure 6.1: Quantum fine-grained reductions to Δ-Matching Triangles and Triangle Collection. The parameter n denotes the size of the sets in the case of 3SUM and Convolution-3SUM and the number of variables in the case of SAT and k-SAT. The parameter N denotes the number of nodes in a graph. The first lower bound in each node in the figure denotes the known classical lower bound and the second lower bound denotes the new quantum lower bound. In the case where multiple edges arrive at one node, the lower bound in the second line of the node is the same for all reductions. For the dashed edge we only know of a classical reduction. Lastly, the lower bounds for Δ-Matching Triangles hold only for the Δ values denoted by the labels on the incoming edges.

6.4 Matching Triangles & Triangle Collection: upper bounds

In the previous two sections we presented a series of lower bounds, conditioned on our quantum hardness conjectures. For most of these problems the matching upper bounds follow from trivial applications of Grover Search, except for Δ-Matching Triangles and Triangle Collection, for which we will now present upper bounds by making clever use of data structures and Ambainis' Variable Time Grover Search [Amb10] stated in Theorem 6.6.

6.4.1 Quantum algorithm for Δ-Matching Triangles

Recall that for Δ-Matching Triangles, given a graph as an input we want to find out whether there exists a colour triple for which there are at least Δ triangles. Depending on the value of $\Delta = n^\alpha$, we exhibit an algorithm that invokes one of the following two subroutines: for small α, it uses brute force with a variant of Grover Search, while for large α, our subroutine uses matrix multiplication. Additionally, for the first case our algorithm requires fast access to the list of vertices coloured by i for every colour $i \in \Gamma$. For that we use a data structure, which with the help of quantum random-access gates RAGs (defined in Section 2.3 of Chapter 2), allows us to index into any memory location in $O(1)$ time.[4] On the other hand, for large α, first notice that as α increases the amount of colour triples for which there are even enough nodes in the graph to form n^α triangles decreases because the number of triples of nodes any graph G (of n nodes) can have is n^3. Therefore, number of colour triples $(i, j, k) \in \Gamma^3$ that can have at least Δ triangles, must have at least Δ different triples of vertices, is at most $\frac{n^3}{\Delta}$. Moreover, as we have computed the values of $|V_i|$ for every $i \in \Gamma$ in the pre-processing step we can easily filter out the un-promising candidates and only check if there exist Δ triangles (using matrix multiplication) for the colour triples with enough nodes of that colour triple.

Theorem 6.32. *There is a quantum algorithm that solves* Δ-Matching Triangles *on a graph of* n *nodes in* $\widetilde{O}(\min\{n^{1.5+\frac{\alpha}{2}}, n^{1.5+\omega-\frac{\alpha}{2}}\})$ *time with* $\Delta = n^\alpha$ *for* $0 \leqslant \alpha \leqslant 3$.

Proof. Let $G = (V, E)$ be the input, a coloured graph with $|V| = n$ and colours given by $\gamma : V \to \Gamma$. Let $\Delta = n^\alpha$ for a given $\alpha \in [0, 3]$. Given a colour $i \in \Gamma$, let $V_i \subseteq V$ denote the subset containing only i-coloured nodes, and use $|V_i|$ to denote the number of vertices in V_i.

Pre-processing step In $O(n)$ time one can compute values $|V_i|$ for all $i \in \Gamma$. Having computed the $|V_i|$ for all $i \in \Gamma$, create a hash-table of $|\Gamma|$ buckets. Each bucket is indexed by $i \in |\Gamma|$ containing an array of size $|V_i|$, respectively. We go over all the vertices $v \in V$ and place them in the hash table corresponding to the bucket indexed by $\gamma(v)$, i.e., the colour of vertex v. Each hash table consists of a prefix tree of n leaves, as in Definition 4.4 of Chapter 4, so that one can efficiently (i.e., in poly-logarithmic time) access the k^{th} node of any bucket for any $k \in [n]$. This entire process takes $O(n)$ time and uses $O(n^2)$ space.

[4] The constant time is in the word-RAM model.

Subroutine for small α We use VTGS on the set Γ^3. For every colour triple $(i,j,k) \in \Gamma^3$, let $t_{i,j,k}$ denote the time taken to determine whether G contains Δ triangles of colour triple (i,j,k). This can be done in $t_{i,j,k} = \widetilde{O}(\sqrt{\Delta|V_i||V_j||V_k|}) = \widetilde{O}(\sqrt{n^\alpha|V_i||V_j||V_k|})$ time using (a variant of) Grover Search where the queries to this Grover-like subroutine are indexed by $(a,b,c) \in [|V_i|] \times [|V_j|] \times [|V_k|]$. Note that, the $\widetilde{O}(\cdot)$ hides the $\mathsf{poly}(\log(n))$ factors that arise because we want the probability of failure to be reduced to $\frac{1}{\mathsf{poly}(n)}$. Moreover, every such query to the Grover-like subroutine can be implemented in $\widetilde{O}(1)$ time (because of the data structure) and three queries to the adjacency matrix given to us as input, in the following way: access the a^{th} element of hash bucket containing V_i; let us denote that vertex by a'. Do the same for accessing the b^{th} element of hash bucket containing V_j and c^{th} element of hash bucket containing V_k, let us denote those vertices with b' and c', respectively. Having done that, use three queries to input of G to check if there is a triangle labelled by nodes a', b', c' in G.

Since we can use VTGS, the total time taken for this algorithm is $T(n) = O(\sqrt{\sum_{i,j,k\in\Gamma} t^2_{i,j,k}})$, and moreover, because we have $\sum_{i\in\Gamma}|V_i| = n$ we find the following.

$$\begin{aligned} T(n) &= O(\sqrt{\sum_{i,j,k\in\Gamma} t^2_{i,j,k}}) = \widetilde{O}(\sqrt{\sum_{i,j,k\in\Gamma} n^\alpha|V_i||V_j||V_k|}) \\ &= \widetilde{O}(\sqrt{n^\alpha \sum_{i\in\Gamma}|V_i| \sum_{j\in\Gamma}|V_j| \sum_{k\in\Gamma}|V_k|}) = \widetilde{O}(\sqrt{n^\alpha n^3}) = O(n^{1.5+\frac{\alpha}{2}}). \end{aligned}$$

Subroutine for large α For every triple $(i,j,k) \in \Gamma^3$, let $t_{i,j,k}$ denote the time taken to compute the following:

- Step 1: check if $|V_i| \cdot |V_j| \cdot |V_k| < \Delta$. If yes then do nothing more, in that case $t_{i,j,k} = O(1)$. Else, check Step 2 below.
- Step 2: check if there are at least Δ triangles of colour triple (i,j,k) in G. As we are in the large α regime we do not want to use threshold version of Grover search, instead we use matrix multiplication to compute the cube of the adjacency matrix restricted to the entries only coloured by coloured by i,j,k and taking trace of the diagonal. A *yes* instance is when the computed trace value is greater than equal to Δ, else it is a *no* instance. The total time taken in Step 2 is $t_{i,j,k} = \widetilde{O}(n^\omega)$. Here the $\widetilde{O}(\cdot)$ hides the $\mathsf{poly}(\log(n))$ factors that arise because we want the probability of failure to be $\frac{1}{\mathsf{poly}(n)}$.

Now to analyse the time taken for the large α case: using VTGS we again have

$$\begin{aligned} T(n) &= O(\sqrt{\sum_{i,j,k\in\Gamma} t^2_{i,j,k}}) = O(\sqrt{\sum_{\substack{i,j,k\in\Gamma \\ \text{s.t. } |V_i||V_j||V_k|\geqslant\Delta}} t^2_{i,j,k} + \sum_{\substack{i,j,k\in\Gamma \\ \text{s.t. } |V_i||V_j||V_k|<\Delta}} t^2_{i,j,k}}) \\ &= \widetilde{O}(\sqrt{\sum_{\substack{i,j,k\in\Gamma \\ \text{s.t. } |V_i||V_j||V_k|\geqslant\Delta}} n^{2\omega} + \sum_{\substack{i,j,k\in\Gamma \\ \text{s.t. } |V_i||V_j||V_k|<\Delta}} O(1)}) \leqslant \widetilde{O}(\sqrt{\frac{n^{2\omega}\cdot n^3}{\Delta} + n^3}) \\ &= \widetilde{O}(\sqrt{\frac{n^{2\omega}\cdot n^3}{\Delta}}) = \widetilde{O}(n^{\omega+1.5-\frac{\alpha}{2}}). \end{aligned}$$

Combined approach Given an input instance of Δ-Matching Triangles, we first compute which of the two approaches is faster based on $\Delta = n^\alpha$ and then apply that approach. We find $T(n) = \widetilde{O}(\min\{n^{1.5+\frac{\alpha}{2}}, n^{1.5+\omega-\frac{\alpha}{2}}\})$.

Setting $1.5 + \frac{\alpha}{2} = 1.5 + \omega - \frac{\alpha}{2}$, we see that for $\alpha < \omega$ the small α algorithm will be faster while for $\alpha > \omega$ our large α algorithm is faster. Therefore, Δ-Matching Triangles for any $\Delta = n^\alpha$ can be solved in $\widetilde{O}(\min\{n^{1.5+\frac{\alpha}{2}}, n^{1.5+\omega-\frac{\alpha}{2}}\})$ time quantumly. □

From Theorem 6.32 we get a worst-case corollary and a corollary for the range of Δ for which we found reductions in Sections 6.2 and 6.2.2.

Corollary 6.33. *There exists a quantum algorithm that solves* Δ-Matching Triangles *on graphs of n nodes*

- *in $\widetilde{O}(n^{1.5+\frac{\omega}{2}})$ time for any Δ, and*
- *in $\widetilde{O}(n^{1.5+o(1)})$ time for $\Delta \leqslant n^{o(1)}$.*

Note that this means that for the current value of $\omega \approx 2.3728$, the matrix-multiplication constant, we can solve Δ-Matching Triangles in sub-cubic time on a quantum computer for any range of Δ.

The commonly conjectured lower bound for ω is 2, in that case we have a worst case complexity for Δ-Matching Triangles of $\widetilde{O}(n^{2.5})$, matching the quantum complexity of other problems encountered in our reductions from APSP. That is not to say that a faster algorithm is not possible of course. Pushing the $n^{1.5-o(1)}$ lower bound for Δ-Matching Triangles up for polynomial Δ is challenging with current techniques and a matching upper bound of $\widetilde{O}(n^{1.5})$ for unrestricted Δ is not impossible, albeit unlikely.

6.4.2 Quantum algorithm for Triangle Collection

Using a similar strategy as for the case of small Δ in the proof of Theorem 6.32 we also find a tight upper bound for Triangle Collection. Recall that for Triangle Collection we want to know whether there is a triangle for every possible colour triple in a given graph. Equivalently, we may ask whether there exists a colour triple in a graph for which there is no triangle.

Theorem 6.34. *There exists a quantum algorithm that solves* Triangle Collection *in $\widetilde{O}(n^{1.5})$ time.*

Proof. Let $G = (V, E)$ be a coloured graph with $|V| = n$ and colours given by $\gamma : V \to \Gamma$. We use the Variable-Time Grover Search algorithm to the set of all colour triples Γ^3, to determine whether there is a colour triple for which there is no triangle in G. Let (i, j, k) be a triple of colours and let the time it takes to check whether G contains a triangle in these colours be $t_{i,j,k}$. Then the algorithm takes $T(n) = O(\sqrt{\sum_{i,j,k\in\Gamma} t_{i,j,k}^2})$ time. Given a colour i, let $V_i \subseteq V$ be the subset containing only nodes in that colour and let $|V_i|$ be the number of vertices in V_i. With some pre-processing of the input, just like it is done in the proof of Theorem 6.32, for any triple of colours (i, j, k), we can Grover Search over the set $V_i \times V_j \times V_k$ to determine

whether there is a triangle in the induced sub-graph in $\widetilde{O}(\sqrt{|V_i| \cdot |V_j| \cdot |V_k|})$ time, and we have that $t^2_{i,j,k} = \widetilde{O}(|V_i| \cdot |V_j| \cdot |V_k|)$. Since it holds that $\sum_{i \in \Gamma} |V_i| = n$ we have

$$T(n) = O\left(\sqrt{\sum_{i,j,k \in \Gamma} t^2_{i,j,k}}\right) = \widetilde{O}\left(\sqrt{\sum_{i,j,k \in \Gamma} |V_i| \cdot |V_j| \cdot |V_k|}\right)$$
$$= \widetilde{O}\left(\sqrt{\sum_{i \in \Gamma} |V_i| \sum_{j \in \Gamma} |V_j| \sum_{k \in \Gamma} |V_k|}\right) = \widetilde{O}\left(n^{1.5}\right).$$

Therefore, we can solve Triangle Collection in $\widetilde{O}(n^{1.5})$ time quantumly. □

6.5 Discussions, future directions and open questions

Discussions In the course of proving quantum time lower bounds for Δ-Matching Triangles and Triangle Collection, we also prove conditional lower bounds for many other related problems in the process and were able to make some interesting observations. In the classical setting, it can be seen as surprising that seemingly simple problems such as Negative Triangle and 0-Edge-Weight-Triangle are at least as hard as problems like APSP and (min, +)-Matrix Multiplication, which seem more complex. With our results in the quantum setting we find that this intuition regarding the difference in complexity of these problems (to a reasonable degree) is well-founded, because here a quadratic gap in complexity in both the lower and the upper bounds of these problems is witnessed instead. A consequence of this gap in complexity was the loss of the reduction from Negative Triangle back to (min, +)-Matrix Multiplication.

Future directions and open questions This chapter leaves us with the following open questions.

1. Classically, it is possible to give a reduction from Negative Triangle to (min, +)-Matrix Multiplication, but it seems much harder to find an equivalent reduction in the quantum case. To pick up where this work has left off would be to face the challenges of finding such a reduction.

2. The Δ-Matching Triangles problem presents another series of worthwhile challenges.

    - Due to the dependency of the upper bound on the matrix-multiplication exponent ω, faster algorithms for Δ-Matching Triangles could be found, either by improving ω, or by finding an algorithm with no or lower dependency on ω. The question remains then whether there exist quantum algorithms that have a better than $\widetilde{O}(n^{2.5})$ run-time for all ranges of Δ; we only achieve this complexity with $\omega = 2$. To reinforce the likelihood of $\widetilde{O}(n^{2.5})$ being the best upper bound for Δ-Matching Triangles, we could look for reductions to Δ-Matching Triangles that work for polynomial ranges of Δ.

- In the classical setting, we see in [AVY18] that when Δ is a constant, the Δ-Matching Triangles problem permits a faster than cubic algorithm; this makes us wonder whether a faster than $\widetilde{O}(n^{1.5})$ time quantum algorithm exists for Δ-Matching Triangles for $\Delta = O(1)$.

3. Much work remains to be done in quantum fine-grained complexity with APSP as a key problem; for a good starting point see survey articles by Williams [Vas15; Vas19]. It is also worthwhile to consider reductions that are only possible in the quantum case — to find a true quantum complexity web of fine-grained reductions, guiding the possibilities and impossibilities of quantum algorithmic design.

Chapter 7

The Last Chapter

The study of quantum fine-grained complexity is just beginning. Classically, there are many fine-grained reductions laying out the structure of P, but only a few such reductions have been established for BQP. It is possible that some of these reductions can be easily adapted to the quantum setting, as we saw in the case of APSP, or only with a lot of technical effort, as we witness in the case of CNF-SAT and 3SUM. Either way, this forms an appealing avenue for future work as not only is the topic very much unexplored, any tight lower bounds given by quantum fine-grained reductions will allow us to understand how much quantum speed-up is possible.

In this thesis we study the consequences of some *worst-case quantum conjectures*, and as a result present worst-case time bounds for several problems in the quantum setting. It will be interesting to use similar conjectures towards establishing *average-case lower bounds* as it has been done classically [BRS+17; BRS+18; GR18; LLV19; BBB19]. These results are of much interest in the context of fine-grained cryptography and towards that several worst-case to average-case reductions have been presented in the classical setting. For an example, see Proof of Work schemes [BRS+17; BRS+18]; In Chapter 3 we were able to show that the Useful Proof of Work scheme by [BRS+18] also holds in the quantum setting, conditioned on QSETH. But that is just one such result and there are several more already known in the classical setting, which haven't been explored in the quantum realm yet. Alternatively, there are problems that are both worst-case and average-case fine-grained hard classically [DLV20]; it will be interesting to check if these hardness results hold in the quantum setting as well.

Another relatively new area close to quantum fine-grained complexity is *fixed parameter tractability* where one wishes to study under what parameter a problem is hard. There are examples of QMA-hard problems, when parameterized by some parameter k of the input can be solved in $g(k)\mathsf{poly}(n)$ time, where n is the size of the input [ABN+22; BJM+22]. However, this idea does not have to be restricted to QMA-hard problems. One can pose similar questions for problems with polynomial-time quantum algorithms; consider a problem that can be solved in $O(n^d)$ time for some constant d and is believed to also require $n^{d-o(1)}$ time. Then, one can ask, for what parameters k is this problem solvable in $g(k) \cdot n^{d-\varepsilon}$ time for an $\varepsilon > 0$? Hardly any such results are known for problems in BQP. Also in this case, the existing classical results can be helpful. First as a good starting point to prove similar results

in the quantum setting, and secondly, by allowing a separation of concerns that could potentially give us faster quantum algorithms — for example, conditioned on SETH as we have seen in this thesis it requires $n^{2-o(1)}$ time classically to compute the edit distance between two strings of length n, and we show an analogous $n^{1.5-o(1)}$ time lower bound conditioned on QSETH in this thesis. There is a classical algorithm that computes the edit distance in $O(n+k^2)$ time for inputs with edit distance k; this means finding an $n + k^{2-\alpha}$ time classical algorithm or an $n + k^{1.5-\alpha'}$ time quantum algorithm for Edit Distance (for any constants $\alpha, \alpha' > 0$) is highly unlikely. As with other examples, one can ask *under what parameters can the* 3SUM*-hard problems discussed in this thesis have fixed-parameter sublinear time quantum algorithms?* Such a question is of course interesting for all hard problems in BQP.

Part II
The Closing Matters

Bibliography

[ABI+20] Andris Ambainis, Kaspars Balodis, Janis Iraids, et al. *Quantum Lower and Upper Bounds for 2D-Grid and Dyck Language*. Proceedings of the 45th International Symposium on Mathematical Foundations of ComputerScience (MFCS 2020). Vol. 170. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 8:1–8:14. DOI: 10.4230/LIPIcs.MFCS.2020.8.

[ABL+22] Andris Ambainis, Harry Buhrman, Koen Leijnse, Subhasree Patro, and Florian Speelman. *Matching Triangles and Triangle Collection: Hardness based on a Weak Quantum Conjecture*. 2022. DOI: 10.48550/ARXIV.2207.11068.

[ABN+22] Srinivasan Arunachalam, Sergey Bravyi, Chinmay Nirkhe, and Bryan O'Gorman. *The Parametrized Complexity of Quantum Verification*. Proceedings of the 17th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2022). Vol. 232. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 3:1–3:18. DOI: 10.4230/LIPIcs.TQC.2022.3.

[ABV15] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. *Tight Hardness Results for LCS and Other Sequence Similarity Measures*. Proceedings of the 56th Annual Symposium on Foundations of Computer Science (FOCS 2015). IEEE Computer Society, 2015, pp. 59–78. DOI: 10.1109/FOCS.2015.14.

[ACL+14] Amihood Amir, Timothy M. Chan, Moshe Lewenstein, and Noa Lewenstein. *On Hardness of Jumbled Indexing*. 41st International Colloquium on Automata, Languages, and Programming - ICALP. Vol. 8572. Lecture Notes in Computer Science. Springer, 2014, pp. 114–125. DOI: 10.1007/978-3-662-43948-7_10.

[ACL+20] Scott Aaronson, Nai-Hui Chia, Han-Hsuan Lin, Chunhao Wang, and Ruizhe Zhang. *On the Quantum Complexity of Closest Pair and Related Problems*. Proceedings of the 35th Computational Complexity Conference (CCC 2020). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2020. DOI: 10.4230/LIPIcs.CCC.2020.16.

[AGS19] Scott Aaronson, Daniel Grier, and Luke Schaeffer. *A Quantum Query Complexity Trichotomy for Regular Languages*. Electronic Colloquium on Computational Complexity **26** (2019), p. 61. URL: https://eccc.weizmann.ac.il/report/2019/061.

[AGV15] Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. *Subcubic Equivalences between Graph Centrality Problems, APSP and Diameter*. Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015). Society for Industrial and Applied Mathematics, 2015, pp. 1681–1697.

[AHV+16] Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. *Simulating Branching Programs with Edit Distance and Friends Or: a Polylog Shaved is a Lower Bound Made*. Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC 2016). ACM, 2016, pp. 375–388. DOI: 10.1145/2897518.2897653.

[AIK+04] Andris Ambainis, Kazuo Iwama, Akinori Kawachi, et al. *Quantum Identification of Boolean Oracles*. Proceedings of the 21st Annual Symposium on Theoretical Aspects of Computer Science (STACS 2004). Vol. 2996. Lecture Notes in Computer Science. Springer, 2004, pp. 105–116. DOI: 10.1007/978-3-540-24749-4_10.

[AL20] Andris Ambainis and Nikita Larka. *Quantum Algorithms for Computational Geometry Problems*. Proceedings of the 15th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2020). Vol. 158. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 9:1–9:10. DOI: 10.4230/LIPIcs.TQC.2020.9.

[ALW14] Amir Abboud, Kevin Lewi, and Ryan Williams. *Losing Weight by Gaining Edges*. Proceedings of the 22th Annual European Symposium on Algorithms (ESA 2014). Vol. 8737. Lecture Notes in Computer Science. Springer, 2014, pp. 1–12. DOI: 10.1007/978-3-662-44777-2_1.

[Amb02] Andris Ambainis. *Quantum Lower Bounds by Quantum Arguments*. Journal of Computer and System Sciences **64**, 4 (2002), pp. 750–767. DOI: 10.1006/jcss.2002.1826. Earlier version in STOC'00.

[Amb06] Andris Ambainis. *Polynomial Degree vs. Quantum Query Complexity*. Journal of Computer and System Sciences **72**, 2 (2006), pp. 220–238. DOI: 10.1016/j.jcss.2005.06.006. Earlier version in FOCS'03.

[Amb07] Andris Ambainis. *Quantum Walk Algorithm for Element Distinctness*. SIAM Journal on Computing **37**, 1 (2007), pp. 210–239. DOI: 10.1137/S0097539705447311. Earlier version in FOCS'04.

[Amb10] Andris Ambainis. *Quantum Search with Variable Times*. Theory of Computing Systems **47**, 3 (2010), pp. 786–807. DOI: 10.1007/s00224-009-9219-1. Earlier version in STACS'08.

[AV14] Amir Abboud and Virginia Vassilevska Williams. *Popular Conjectures Imply Strong Lower Bounds for Dynamic Problems*. Proceedings of the 55th Annual Symposium on Foundations of Computer Science (FOCS 2014). 2014, pp. 434–443. DOI: 10.1109/FOCS.2014.53.

[AVW14] Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. *Consequences of Faster Alignment of Sequences*. Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP 2014). Vol. 8572. Lecture Notes in Computer Science. Springer, 2014, pp. 39–51. DOI: 10.1007/978-3-662-43948-7_4.

[AVW16] Amir Abboud, Virginia Vassilevska Williams, and Joshua Wang. *Approximation and Fixed Parameter Subquadratic Algorithms for Radius and Diameter in Sparse Graphs*. Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016). Society for Industrial and Applied Mathematics, 2016, pp. 377–391.

[AVY18] Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. *Matching Triangles and Basing Hardness on an Extremely Popular Conjecture*. SIAM Journal on Computing **47**, 3 (2018), pp. 1098–1122. DOI: 10.1137/15M1050987. Earlier version in STOC'15.

[Bar89] David A. Barrington. *Bounded-width polynomial-size branching programs recognize exactly those languages in NC1*. Journal of Computer and System Sciences **38**, 1 (1989), pp. 150–164. DOI: https://doi.org/10.1016/0022-0000(89)90037-8.

[BBB+97] Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. *Strengths and Weaknesses of Quantum Computing*. SIAM Journal on Computing **26**, 5 (1997), pp. 1510–1523. DOI: 10.1137/S0097539796300933.

[BBB19] Enric Boix-Adserà, Matthew Brennan, and Guy Bresler. *The Average-Case Complexity of Counting Cliques in Erdős-Rényi Hypergraphs*. Proceedings of the 60th Annual Symposium on Foundations of Computer Science (FOCS 2019). 2019, pp. 1256–1280. DOI: 10.1109/FOCS.2019.00078.

[BBC+01] Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. *Quantum Lower Bounds by Polynomials*. Journal of the ACM **48**, 4 (2001), pp. 778–797. DOI: 10.1145/502090.502097.

[BBH+98] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. *Tight bounds on quantum searching*. Fortschritte der Physik: Progress of Physics **46**, 4-5 (1998), pp. 493–505.

[BDH+00] Harry Buhrman, Christoph Durr, Peter Høyer, et al. *Quantum Algorithms for Finding Claws, Collisions and Triangles* (2000).

[BDP08] Ilya Baran, Erik D. Demaine, and Mihai Pătraşcu. *Subquadratic Algorithms for 3SUM*. Algorithmica **50**, 4 (2008), pp. 584–596. Earlier version in WADS'05.

[BEG+18] Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, Mohammad Taghi Hajiaghayi, and Saeed Seddighin. *Approximating Edit Distance in Truly Subquadratic Time: Quantum and MapReduce*. Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2018). SIAM, 2018, pp. 1170–1189. DOI: 10.1137/1.9781611975031.76.

[Ben89] Charles H. Bennett. *Time/Space Trade-Offs for Reversible Computation*. SIAM Journal on Computing **18**, 4 (1989), pp. 766–776. DOI: 10.1137/0218053.

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, et al. *On the (im) possibility of obfuscating programs*. Journal of the ACM **59**, 2 (2012), pp. 1–48.

[BGS75] Theodore Baker, John Gill, and Robert Solovay. *Relativizations of the P=?NP question*. SIAM Journal on computing **4**, 4 (1975), pp. 431–442.

[BI18] Arturs Backurs and Piotr Indyk. *Edit Distance Cannot Be Computed in Strongly Subquadratic Time (Unless SETH is False)*. SIAM Journal on Computing **47**, 3 (2018), pp. 1087–1097. DOI: 10.1137/15M1053128. Earlier version in STOC'15.

[BJL+13] Daniel J. Bernstein, Stacey Jeffery, Tanja Lange, and Alexander Meurer. *Quantum Algorithms for the Subset-Sum Problem*. Proceedings of the 5th International Workshop on Post-Quantum Cryptography (PQCrypto 2013). Vol. 7932. Lecture Notes in Computer Science. Springer, 2013, pp. 16–33. DOI: 10.1007/978-3-642-38616-9_2.

[BJM+22] Michael J. Bremner, Zhengfeng Ji, Ryan L. Mann, et al. *Quantum Parameterized Complexity*. 2022. DOI: 10.48550/ARXIV.2203.08002.

[BK15] Karl Bringmann and Marvin Kunnemann. *Quadratic Conditional Lower Bounds for String Problems and Dynamic Time Warping*. Proceedings of the 56th Annual Symposium on Foundations of Computer Science (FOCS 2015). IEEE Computer Society, 2015, pp. 79–97. DOI: 10.1109/FOCS.2015.15.

[BLP+22a] Harry Buhrman, Bruno Loff, Subhasree Patro, and Florian Speelman. *Limits of Quantum Speed-Ups for Computational Geometry and Other Problems: Fine-Grained Complexity via Quantum Walks*. Proceedings of the 13th Innovations in Theoretical Computer Science Conference (ITCS 2022). Vol. 215. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 31:1–31:12. DOI: 10.4230/LIPIcs.ITCS.2022.31. Full version at arXiv:2106.02005.

[BLP+22b] Harry Buhrman, Bruno Loff, Subhasree Patro, and Florian Speelman. *Memory Compression with Quantum Random-Access Gates*. Proceedings of the 17th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2022). Vol. 232. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 10:1–10:19. DOI: 10.4230/LIPIcs.TQC.2022.10.

[BMN+21] Ryan Babbush, Jarrod R McClean, Michael Newman, et al. *Focus beyond quadratic speedups for error-corrected quantum advantage*. PRX Quantum **2**, 1 (2021).

[BPS21] Harry Buhrman, Subhasree Patro, and Florian Speelman. *A Framework of Quantum Strong Exponential-Time Hypotheses*. Proceedings of the 38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021). Vol. 187. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 19:1–19:19. DOI: 10.4230/LIPIcs.STACS.2021.19. Full version at arXiv:1911.05686.

[Bri14] Karl Bringmann. *Why Walking the Dog Takes Time: Frechet Distance Has No Strongly Subquadratic Algorithms Unless SETH Fails*. Proceedings of the 55th Annual Symposium on Foundations of Computer Science (FOCS 2014). IEEE Computer Society, 2014, pp. 661–670. DOI: 10.1109/FOCS.2014.76.

[BRS+17] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. *Average-Case Fine-Grained Hardness*. Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2017). Association for Computing Machinery, 2017, pp. 483–496. ISBN: 9781450345286. DOI: 10.1145/3055399.3055466.

[BRS+18] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. *Proofs of Work From Worst-Case Assumptions*. Proceedings of the 38th Annual International Cryptology Conference (CRYPTO 2018). Vol. 10991. Lecture Notes in Computer Science. Springer, 2018, pp. 789–819. DOI: 10.1007/978-3-319-96884-1_26. Earlier version of this work appeared with the title 'Proofs of Useful Work'.

[BŠ13] Aleksandrs Belovs and Robert Špalek. *Adversary Lower Bound for the k-Sum Problem*. Proceedings of the 4th Conference on Innovations in Theoretical Computer Science (ITCS 2013). Association for Computing Machinery, 2013, pp. 323–328. ISBN: 9781450318594. DOI: 10.1145/2422436.2422474.

[BSS03] H. Barnum, M. Szegedy, and M. Saks. *Quantum query complexity and semi-definite programming*. Proceedings of the 18th IEEE Annual Conference on Computational Complexity (CCC 2003). IEEE Computer Society, 2003, p. 179. DOI: 10.1109/CCC.2003.1214419.

[BV97] E. Bernstein and U. Vazirani. *Quantum Complexity Theory*. SIAM Journal on Computing **26**, 5 (1997), pp. 1411–1473. DOI: 10.1137/S0097539796300921.

[CDG+18] Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael E. Saks. *Approximating Edit Distance within Constant Factor in Truly Sub-Quadratic Time*. Proceedings of the 59th Annual Symposium on Foundations of Computer Science (FOCS 2018). IEEE Computer Society, 2018, pp. 979–990. DOI: 10.1109/FOCS.2018.00096.

[CDL+16] Marek Cygan, Holger Dell, Daniel Lokshtanov, et al. *On Problems As Hard As CNF-SAT*. ACM Transactions on Algorithms **12**, 3 (2016), 41:1–41:24. DOI: 10.1145/2925416.

[CE05] Andrew M. Childs and Jason M. Eisenberg. *Quantum Algorithms for Subset Finding*. Quantum Information and Computation **5**, 7 (2005), pp. 593–604.

[CIP06] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. *A Duality Between Clause Width and Clause Density for SAT*. Proceedings of the 21st Annual IEEE Conference on Computational Complexity (CCC 2006). IEEE Computer Society, 2006, pp. 252–260. DOI: 10.1109/CCC.2006.6.

[CLR+14] Shiri Chechik, Daniel H. Larkin, Liam Roditty, et al. *Better Approximation Algorithms for the Graph Diameter*. Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 14). Society for Industrial and Applied Mathematics, 2014, pp. 1041–1052. ISBN: 9781611973389.

[CMP22] Arjan Cornelissen, Nikhil S. Mande, and Subhasree Patro. *Improved Quantum Query Upper Bounds Based on Classical Decision Trees*. 2022. DOI: 10.48550/ARXIV.2203.02968.

[DH96] Christoph Durr and Peter Høyer. *A Quantum Algorithm for Finding the Minimum*. 1996. DOI: 10.48550/ARXIV.QUANT-PH/9607014.

[DHH+06] Christoph Durr, Mark Heiligman, Peter Høyer, and Mehdi Mhalla. *Quantum query complexity of some graph problems*. SIAM Journal on Computing **35**, 6 (2006), pp. 1310–1328. DOI: 10.1137/050644719.

[DHM+14] Holger Dell, Thore Husfeldt, Dániel Marx, Nina Taslaman, and Martin Wahlén. *Exponential time complexity of the permanent and the Tutte polynomial*. ACM Transactions on Algorithms **10**, 4 (2014), p. 21.

[Die96] Martin Dietzfelbinger. *Universal Hashing and K-Wise Independent Random Variables via Integer Arithmetic without Primes*. Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science (STACS 1996). Springer-Verlag, 1996, pp. 569–580. ISBN: 3540609229.

[DLV20] Mina Dalirrooyfard, Andrea Lincoln, and Virginia Vassilevska Williams. *New Techniques for Proving Fine-Grained Average-Case Hardness*. Proceedings of the 61st Annual Symposium on Foundations of Computer Science (FOCS 2020). IEEE, 2020, pp. 774–785. DOI: 10.1109/FOCS46700.2020.00077.

[FGG+98] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. *Limit on the Speed of Quantum Computation in Determining Parity*. Physical Review Letters **81** (24 1998), pp. 5442–5444. DOI: 10.1103/PhysRevLett.81.5442.

[FM71] M. J. Fischer and A. R. Meyer. *Boolean matrix multiplication and transitive closure*. Proceedings of the 12th Annual Symposium on Switching and Automata Theory (SWAT 1971). 1971, pp. 129–131. DOI: 10.1109/SWAT.1971.4.

[Gal14] François Le Gall. *Improved Quantum Algorithm for Triangle Finding via Combinatorial Arguments*. Proceedings of the 55th Annual Symposium on Foundations of Computer Science (FOCS 2014). IEEE Computer Society, 2014, pp. 216–225. DOI: 10.1109/FOCS.2014.31.

[Gil14] András Pál Gilyén. *Quantum walk based search methods and algorithmic applications*. Masters thesis. Budapest University of Technology and Economics, 2014. URL: https://web.cs.elte.hu/blobs/diplomamunkak/msc_mat/2014/gilyen_andras_pal.pdf.

[GO95] Anka Gajentaan and Mark H Overmars. *On a class of $O(n^2)$ problems in computational geometry*. Computational Geometry **5**, 3 (1995), pp. 165–185. DOI: https://doi.org/10.1016/0925-7721(95)00022-2.

[GR18] Oded Goldreich and Guy Rothblum. *Counting t-Cliques: Worst-Case to Average-Case Reductions and Direct Interactive Proof Systems*. Proceedings of the 59th Annual Symposium on Foundations of Computer Science (FOCS 2018). 2018, pp. 77–88. DOI: 10.1109/FOCS.2018.00017.

[Gro96] Lov K. Grover. *A Fast Quantum Mechanical Algorithm for Database Search*. Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC 1996). Association for Computing Machinery, 1996, pp. 212–219. DOI: 10.1145/237814.237866.

[HLŠ07] Peter Høyer, Troy Lee, and Robert Špalek. *Negative Weights Make Adversaries Stronger*. Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC 2007). Association for Computing Machinery, 2007, pp. 526–535. DOI: 10.1145/1250790.1250867.

[HMW03] Peter Høyer, Michele Mosca, and Ronald de Wolf. *Quantum Search on Bounded-Error Inputs*. Proceedings of the 30th International Colloquium on Automata, Languages, and Programming (ICALP 2003). Vol. 2719. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2003, pp. 291–299. DOI: 10.1007/3-540-45061-0_25.

[HNS01] Peter Høyer, Jan Neerbek, and Yaoyun Shi. *Quantum Complexities of Ordered Searching, Sorting, and Element Distinctness*. Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP 2001). Vol. 2076. Lecture Notes in Computer Science. Springer, 2001, pp. 346–357. DOI: 10.1007/3-540-48224-5_29.

[HNS18] Cupjin Huang, Michael Newman, and Mario Szegedy. *Explicit lower bounds on strong quantum simulation* (2018). arXiv: 1804.10368.

[IKK+17] Russell Impagliazzo, Valentine Kabanets, Antonina Kolokolova, Pierre McKenzie, and Shadab Romani. *Does Looking Inside a Circuit Help?* Proceedings of the 42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2017.

[IP01] Russell Impagliazzo and Ramamohan Paturi. *On the Complexity of k-SAT*. Journal of Computer and System Sciences **62**, 2 (2001), pp. 367–375. DOI: https://doi.org/10.1006/jcss.2000.1727.

[IPZ01] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. *Which Problems Have Strongly Exponential Complexity?* Journal of Computer and System Sciences **63**, 4 (2001), pp. 512–530. DOI: https://doi.org/10.1006/jcss.2001.1774.

[Jef14] Stacey Jeffery. *Frameworks for Quantum Algorithms*. PhD thesis. University of Waterloo, 2014.

[Kot14] Robin Kothari. *An optimal quantum algorithm for the oracle identification problem*. Proceedings of the 31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014). Vol. 25. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014, pp. 482–493. DOI: 10.4230/LIPIcs.STACS.2014.482.

[Kup05] Greg Kuperberg. *A Subexponential-Time Quantum Algorithm for the Dihedral Hidden Subgroup Problem*. SIAM Journal on Computing **35**, 1 (2005), pp. 170–188. DOI: 10.1137/S0097539703436345.

[Lei22] K. Leijnse. *On the Quantum Hardness of Matching Colored Triangles*. Masters thesis. Universiteit van Amsterdam, 2022.

[LLV19] Rio LaVigne, Andrea Lincoln, and Virginia Vassilevska Williams. *Public-Key Cryptography in the Fine-Grained Setting*. Proceedings of the 39th Annual International Cryptology Conference (CRYPTO 2019). Vol. 11694. Lecture Notes in Computer Science. Springer, 2019, pp. 605–635. DOI: 10.1007/978-3-030-26954-8_20.

[LM08] Sophie Laplante and Frédéric Magniez. *Lower Bounds for Randomized and Quantum Query Complexity Using Kolmogorov Arguments*. SIAM Journal on Computing **38**, 1 (2008), pp. 46–62. DOI: 10.1137/050639090. Earlier version in CCC'04.

[LMR+11] Troy Lee, Rajat Mittal, Ben W. Reichardt, Robert Špalek, and Mario Szegedy. *Quantum Query Complexity of State Conversion*. Proceedings of the 52nd Annual Symposium on Foundations of Computer Science (FOCS 2011). IEEE Computer Society, 2011, pp. 344–353. DOI: 10.1109/FOCS.2011.75.

[MP80] William J. Masek and Michael S. Paterson. *A faster algorithm computing string edit distances*. Journal of Computer and System Sciences **20**, 1 (1980), pp. 18–31. DOI: https://doi.org/10.1016/0022-0000(80)90002-1.

[MSS07] Frédéric Magniez, Miklos Santha, and Mario Szegedy. *Quantum Algorithms for the Triangle Problem*. SIAM Journal on Computing **37**, 2 (2007), pp. 413–424. DOI: 10.1137/050643684. Earlier version in SODA'05.

[MT19] Tomoyuki Morimae and Suguru Tamaki. *Fine-grained quantum computational supremacy*. Quantum Information & Computation **19**, 13&14 (2019), pp. 1089–1115. URL: http://www.rintonpress.com/xxqic19/qic-19-1314/1089-1115.pdf.

[Mun71] Ian Munroe. *Efficient Determination of the Transitive Closure of a Directed Graph*. Information Processing Letters, 1 (1971). ISSN: 0020-0190, pp. 56–58. DOI: 10.1016/0020-0190(71)90006-8.

[NC00] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[NS20] María Naya-Plasencia and André Schrottenloher. *Optimal Merging in Quantum -Xor and -Xor-Sum Algorithms*. Proceedings of the 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2020). Springer-Verlag, 2020, pp. 311–340. DOI: 10.1007/978-3-030-45724-2_11.

[Pat10] Mihai Patrascu. *Towards Polynomial Lower Bounds for Dynamic Problems*. Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC 2010). Association for Computing Machinery, 2010, pp. 603–610. DOI: 10.1145/1806689.1806772.

[PP20] Subhasree Patro and Álvaro Piedrafita. *An Overview of Quantum Algorithms: From Quantum Supremacy to Shor Factorization*. IEEE International Symposium on Circuits and Systems (ISCAS 2020). 2020, pp. 1–5. DOI: 10.1109/ISCAS45731.2020.9180793.

[PPS+05] Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. *An Improved Exponential-time Algorithm for k-SAT*. Journal of the ACM **52**, 3 (2005), pp. 337–364. DOI: 10.1145/1066100.1066101.

[PPV+21] Dhrumil Patel, Subhasree Patro, Chiranjeevi Vanarasa, Indranil Chakrabarty, and Arun Kumar Pati. *Impossibility of cloning of quantum coherence*. Physical Review A **103** (2 2021), p. 022422. DOI: 10.1103/PhysRevA.103.022422.

[Pre98] John Preskill. *Lecture Notes for Physics 229: Quantum Information and Computation* (1998).

[Rei11] Ben W. Reichardt. *Reflections for Quantum Query Algorithms*. Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2011). Society for Industrial and Applied Mathematics, 2011, pp. 560–569.

[Ren19] Jorg Van Renterghem. *The implications of breaking the strong exponential time hypothesis on a quantum computer*. Masters thesis. Ghent University, 2019. URL: https://lib.ugent.be/fulltxt/RUG01/002/787/416/RUG01-002787416_2019_0001_AC.pdf.

[RZ11] Liam Roditty and Uri Zwick. *On Dynamic Shortest Paths Problems*. Algorithmica **61**, 2 (2011), pp. 389–401. DOI: 10.1007/s00453-010-9401-5. Earlier version in ESA'04.

[Sch22] Daan Schoneveld. *Quantum Fine-Grained Complexity: Hitting-Set and Related Problems*. Bachelors thesis. Universiteit van Amsterdam, 2022.

[Sho97] Peter W. Shor. *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*. SIAM Journal on Computing **26**, 5 (1997), pp. 1484–1509. DOI: 10.1137/S0097539795293172. Earlier version in FOCS'94.

[ŠS06] Robert Špalek and Mario Szegedy. *All Quantum Adversary Methods are Equivalent*. Theory of Computing **2**, 1 (2006), pp. 1–18. DOI: 10.4086/toc.2006.v002a001. Earlier version in ICALP'05.

[Vas15] Virginia Vassilevska Williams. *Hardness of Easy Problems: Basing Hardness on Popular Conjectures such as the Strong Exponential Time Hypothesis (Invited Talk)*. Proceedings of the 10th International Symposium on Parameterized and Exact Computation (IPEC 2015). Vol. 43. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015, pp. 17–29. DOI: 10.4230/LIPIcs.IPEC.2015.17.

[Vas19] Virginia Vassilevska Williams. *On some fine-grained questions in algorithms and complexity*. Proceedings of the International Congress of Mathematicians (ICM 2018). World Scientific, 2019, pp. 3447–3487. DOI: 10.1142/9789813272880_0188.

[VW04] Farrokh Vatan and Colin Williams. *Optimal quantum circuits for general two-qubit gates*. Physical Review A **69** (3 2004), p. 032315. DOI: 10.1103/PhysRevA.69.032315.

[VW13] Virginia Vassilevska Williams and Ryan Williams. *Finding, Minimizing, and Counting Weighted Subgraphs*. SIAM Journal on Computing **42**, 3 (2013), pp. 831–854. DOI: 10.1137/09076619X. Earlier version in STOC'09.

[VW18] Virginia Vassilevska Williams and R. Ryan Williams. *Subcubic Equivalences Between Path, Matrix, and Triangle Problems*. Journal of the ACM **65**, 5 (2018), pp. 1–38. DOI: 10.1145/3186893. Earlier version in FOCS'10.

[Wil05] Ryan Williams. *A New Algorithm for Optimal 2-constraint Satisfaction and Its Implications*. Theoretical Computer Science **348**, 2 (2005), pp. 357–365. DOI: 10.1016/j.tcs.2005.09.023. Earlier version in ICALP'04.

[Wil18] R. Ryan Williams. *Faster All-Pairs Shortest Paths via Circuit Complexity*. SIAM Journal on Computing **47**, 5 (2018), pp. 1965–1985. DOI: 10.1137/15M1024524. Earlier version in STOC'14.

[Wol21] Ronald de Wolf. *Quantum Computing: Lecture Notes*. 2021. arXiv: 1907.09415.

[Zha04] Shengyu Zhang. *On the Power of Ambainis's Lower Bounds*. Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP 2004). Vol. 3142. Lecture Notes in Computer Science. Springer, 2004, pp. 1238–1250. DOI: 10.1007/978-3-540-27836-8_102.

Abstract

Quantum Fine-Grained Complexity

There is considerable excitement around quantum computing because of so-called *quantum speedups*: quantum algorithms can solve many computational problems faster than their classical counterparts. However, the amount of speedup that is possible varies among different computational problems. It is expected that quantum computers will remain an expensive resource for a long time, and the extent to which a quantum speedup is possible, or not possible, may one day be a key factor in deciding whether or not to invest in the use of quantum computation, for example in an industrial setting. Therefore it is essential to understand how much quantum speedup is possible for a specific computational problem, and for this purpose we need to not only find classical and quantum algorithms but also understand their limits in the form of lower bounds.

Sadly, one of the major challenges in the field of complexity theory is the inability to prove unconditional time lower bounds, including for practical problems within the complexity class P. One way around this is the study of fine-grained complexity where we use special reductions to prove time lower bounds for many diverse problems in P based on the conjectured hardness of some key problems. For example, computing the edit distance between two strings, a problem that has a practical interest in the comparing of DNA strings, has an algorithm that takes $O(n^2)$ time. Using a fine-grained reduction it can be shown that faster algorithms for computing edit distance also imply a faster algorithm for the Boolean Satisfiability (SAT) problem (that is believed to not exist) — strong evidence that it will be very hard to solve the edit distance problem faster. In addition to SAT, 3SUM and APSP are other such key problems that are very suitable to use as a basis for such reductions, since they are natural to describe and well studied.

The situation in the quantum regime is no better; almost all known lower bounds for quantum algorithms are defined in terms of query complexity, which does not help much for problems for which the best known algorithms take superlinear time, or for problems whose best known query algorithms aren't time efficient. Therefore, employing fine-grained reductions in the quantum setting seems a natural way forward. However, translating the classical fine-grained reductions directly into the quantum regime is not always possible for various reasons. In this thesis, we discuss some of these challenges and present results in which we circumvent these challenges. As a consequence we prove quantum time lower bounds for many problems in BQP conditioned on the conjectured quantum hardness of SAT (and its variants), 3SUM, and APSP problems.

Nederlandse samenvatting

Fijnmazige Kwantumcomplexiteit

De wereld investeert in kwantumcomputers vanwege zogenaamde *kwantumversnellingen*: kwantumalgoritmen kunnen veel rekenproblemen sneller oplossen dan hun klassieke tegenhangers. In hoeverre zo'n versnelling mogelijk is, is echter voor ieder rekenprobleem anders. Naar verwachting zullen kwantumcomputers nog voor lange tijd een duur hulpmiddel blijven, en de mate waarin een kwantumversnelling mogelijk of onmogelijk is, kan ooit de sleutelfactor worden bij de beslissing om wel of niet te investeren in kwantumcomputers, bijvoorbeeld in een industriële context. Zodoende is het essentieel te begrijpen hoeveel kwantumversnelling te behalen valt bij een specifiek rekenprobleem, en hiertoe zijn scherpe boven- en ondergrenzen voor zowel klassieke als kwantumalgoritmen noodzakelijk.

Jammer genoeg is een van de grootste uitdagingen in het vakgebied van de complexiteitstheorie ons onvermogen om onvoorwaardelijke bovengrenzen voor de benodigde rekentijd te bewijzen, ook bij praktische problemen in de complexiteitsklasse P. Eén manier om deze uitdaging uit de weg te gaan, is de studie van *fine-grained* complexiteit waarbij we speciale reducties gebruiken om tijdsondergrenzen te bewijzen voor vele diverse problemen in P, op basis van de vermoede moeilijkheid van enkele kernproblemen. Bijvoorbeeld: het berekenen van de bewerkingsafstand tussen twee tekenreeksen, een probleem dat van belang is bij de vergelijking van twee DNA-strengen, kent een algoritme dat $O(n^2)$ tijd nodig heeft. Door een *fine-grained* reductie te gebruiken, kan worden aangetoond dat snellere algoritmen voor het berekenen van de bewerkingsafstand ook een sneller algoritme impliceren voor het booleaanse vervulbaarheidsprobleem (SAT, naar het Engelse *satisfiability problem*) — waarvan men gelooft dat het niet bestaat — en dit is een sterke aanwijzing dat het zeer moeilijk is om bewerkingsafstanden sneller te berekenen. Naast SAT, 3SUM en APSP bestaan er andere dergelijke kernproblemen die zeer geschikt zijn om te gebruiken als basis te voor zulke reducties, omdat ze een natuurlijke beschrijving kennen en goed bestudeerd zijn.

De situatie in het kwantumregime is niet veel beter: vrijwel alle ondergrenzen voor kwantumalgoritmen zijn gedefineerd in termen van *query*-complexiteit, wat niet heel nuttig is bij problemen waarvoor de best bekende algoritmen superlineaire tijd vereisen, of bij problemen waarvoor de beste *query*-algoritmen niet tijdsefficiënt zijn. Om deze reden lijkt de inzet van *fine-grained* reducties het meest voor de hand liggend in de kwantumcontext. Niettemin is een directe vertaling van klassieke *fine-grained* reducties naar het kwantumregime niet altijd mogelijk, om uiteenlopende redenen. In dit proefschrift beschouwen we enkele van deze moeilijkheden en presenteren we resultaten waarin we deze weten te omzeilen. Bijgevolg bewijzen we kwantum-tijdsondergrenzen voor vele problemen in BQP, onder de voorwaarde van de vermoede kwantummoeilijkheid van de problemen SAT (en diens varianten), 3SUM en APSP.

Acknowledgements

First and foremost I would like to thank my supervisor Harry for thinking of this project and then giving me the chance to be a part of this project. I am indebted to his constant support and guidance throughout the last 4.25 years.

I would also like to express my gratitude towards my co-supervisor Florian. I am thankful for his immense amount of patience with my numerous unplanned knocks at his office door, always followed by the question *'do you have a minute?'*; I am also sorry because none of those discussions ever finished within a minute. Without his support (especially during the early years of my PhD) I wouldn't have learnt half as much.

I have been extremely fortunate to have collaborated with Bruno. Not only is his passion for research highly contagious but he always found the time to discuss ideas irrespective of the time of the day. Some of my favourite results from this thesis are with Bruno and I am very happy that I got to work with him towards those results.

I would also like to thank my other collaborators. In particular my co-authors Álvaro, Andris, Arjan, Koen (Leijnse) and Nikhil. Thank you Koen; I had a lot of fun bouncing off ideas with you. I learnt a lot about a lot of things from Álvaro. I would like to express my amazement towards my co-author and office mate Arjan; seeing him work at the office was no different from watching a superhero movie. There was hardly anything he couldn't solve with his *Flash-like* brain and his *python* superpower. I also would like to express my gratitude towards my co-author and dear friend Nikhil. I learnt a great deal about research (even quantum stuff which he will deny) and academia from him. I can safely say that it is because of him that I could survive the stress from the last crucial years of my PhD. I also thank Rajendra, Yilei and Yanlin for understanding my lack of participation over the last few months.

I would also like to thank my other colleagues at QuSoft who together make QuSoft a pleasant place of work; thanks Adam, Akshay, Alex, András, Arie, Bas, Chanelle, Chris (Cade), Crownie, Daan, Davi, Dmitry, Doutzen, Dyon, Emiel, Farrokh, Fran, Freek, Galina, Garazi, Harold, Ido, Jan, Jana, Jelena, Jeroen, Jonas, Jop, Joran, Jordi, Joris, Kareljan, Koen (Groenland), Léo (Colisson), Llorenç, Ludo, Lynn, Marten, Mathys, Mehrdad, Mert, Michael, Niels, Peter (van der Gulik), Philip, Quinten, Randy, René, Sander, Sebastian, Seenivasan, Simon (Apers), Srinivasan, Stacey, Susanne, Tom, Victor, Yanlin, Yaroslav, Yfke, Yinan and Yvonne for making QuSoft wonderful both at a social and scientific level. Thanks to Bikkie, Erik, Irma, Karin, Maarten, Minnie, Nada, Remco, Rob and Vera for making working at CWI smooth and pleasant. Thank you, Doutzen for always cheering me for all my (even though tiny) achievements. I thank the CWI activity committee for organising several fun events over the last 4.25 years. I would also like to thank Anurudh, Kfir, Simona and Subhayan for our discussions.

I would also like to thank the members of my PhD committee, Andris, Bruno, Chris (Schaffner), Maris and Ronald, for taking the time to review my thesis (and also for approving it). In particular, I would like to thank Ronald first for his detailed comments not only on my thesis but also on several talks and poster presentations and secondly for his valuable time for all the discussions over the last 4.25 years. His door (whenever open) was always open for questions. I also thank Dyon for translating my abstract into Dutch. I thank Koen (Groenland) for letting me use

his thesis template. I thank Garazi and Yanlin for agreeing to be my paranymphs.

I would also like to thank my friends Adu, Akshita, Anee, Anu, Boddu, Dolly, Gannu, Hema, Monika, Mythili, Paapi, Tiqvah, Ujwala and Vicky who kept me sane these last few years. Friends who drove me insane are not acknowledged in this thesis. I would also like to thank Aishwarya, Bharadwaj, Jelle, John, Julian, Kuldeep, Marie-Louise, Ritsya, Sandeep, Soumya and Stephanie who made me and Nimerah feel at home in the Netherlands. It wouldn't have been possible for me to handle research and parenting if not for you all. I would also like to thank my therapist Hazel and Nimerah's babysitter Kate for their support and care. I would also like to thank the Dutch government for the *kinderopvangtoeslag* (the childcare benefit) without which I wouldn't have been able to take care of Nimerah here in the Netherlands. I would also like to thank WIQD for the funding I have received towards childcare during conferences and work visits. I thank my ex-husband Joseph for a smooth divorce and emotional support through the process. I would like to thank the *weekend crew*, Chinmay, Neety, Nishant, Samruddhi and Visu, for the fun board game afternoons. I would like to thank my extended family, Anu, Chandan, Jaya, Neha, Rebathi, Sai, Siddhu and Srinu, for their love and support. I would like to thank my teacher Praveen for being a wonderful inspiration to his students.

Lastly, I would like to express my gratitude towards my family (both living and dead). My love for research stems from the scientific discussions I used to have with my father Mahendra. I am sure he would have been proud of this thesis. None of this work would have been possible without the support (and occasional nagging) of my mother Swarupa. I thank my sister Sunayana for *always* being there for me (even while writing this acknowledgement section making sure I don't miss out on thanking loved ones such as herself). I would also like to thank my daughter Nimerah for her constant love and support; this once squishy bundle of joy is no longer squishy but is still joyful and delightful.

Titles in the ILLC Dissertation Series:

ILLC DS-2016-01: **Ivano A. Ciardelli**
Questions in Logic

ILLC DS-2016-02: **Zoé Christoff**
Dynamic Logics of Networks: Information Flow and the Spread of Opinion

ILLC DS-2016-03: **Fleur Leonie Bouwer**
What do we need to hear a beat? The influence of attention, musical abilities, and accents on the perception of metrical rhythm

ILLC DS-2016-04: **Johannes Marti**
Interpreting Linguistic Behavior with Possible World Models

ILLC DS-2016-05: **Phong Lê**
Learning Vector Representations for Sentences - The Recursive Deep Learning Approach

ILLC DS-2016-06: **Gideon Maillette de Buy Wenniger**
Aligning the Foundations of Hierarchical Statistical Machine Translation

ILLC DS-2016-07: **Andreas van Cranenburgh**
Rich Statistical Parsing and Literary Language

ILLC DS-2016-08: **Florian Speelman**
Position-based Quantum Cryptography and Catalytic Computation

ILLC DS-2016-09: **Teresa Piovesan**
Quantum entanglement: insights via graph parameters and conic optimization

ILLC DS-2016-10: **Paula Henk**
Nonstandard Provability for Peano Arithmetic. A Modal Perspective

ILLC DS-2017-01: **Paolo Galeazzi**
Play Without Regret

ILLC DS-2017-02: **Riccardo Pinosio**
The Logic of Kant's Temporal Continuum

ILLC DS-2017-03: **Matthijs Westera**
Exhaustivity and intonation: a unified theory

ILLC DS-2017-04: **Giovanni Cinà**
Categories for the working modal logician

ILLC DS-2017-05: **Shane Noah Steinert-Threlkeld**
Communication and Computation: New Questions About Compositionality

ILLC DS-2017-06: **Peter Hawke**
The Problem of Epistemic Relevance

ILLC DS-2017-07: **Aybüke Özgün**
Evidence in Epistemic Logic: A Topological Perspective

ILLC DS-2017-08: **Raquel Garrido Alhama**
Computational Modelling of Artificial Language Learning: Retention, Recognition & Recurrence

ILLC DS-2017-09: **Miloš Stanojević**
Permutation Forests for Modeling Word Order in Machine Translation

ILLC DS-2018-01: **Berit Janssen**
Retained or Lost in Transmission? Analyzing and Predicting Stability in Dutch Folk Songs

ILLC DS-2018-02: **Hugo Huurdeman**
Supporting the Complex Dynamics of the Information Seeking Process

ILLC DS-2018-03: **Corina Koolen**
Reading beyond the female: The relationship between perception of author gender and literary quality

ILLC DS-2018-04: **Jelle Bruineberg**
Anticipating Affordances: Intentionality in self-organizing brain-body-environment systems

ILLC DS-2018-05: **Joachim Daiber**
Typologically Robust Statistical Machine Translation: Understanding and Exploiting Differences and Similarities Between Languages in Machine Translation

ILLC DS-2018-06: **Thomas Brochhagen**
Signaling under Uncertainty

ILLC DS-2018-07: **Julian Schlöder**
Assertion and Rejection

ILLC DS-2018-08: **Srinivasan Arunachalam**
Quantum Algorithms and Learning Theory

ILLC DS-2018-09: **Hugo de Holanda Cunha Nobrega**
Games for functions: Baire classes, Weihrauch degrees, transfinite computations, and ranks

ILLC DS-2018-10: **Chenwei Shi**
Reason to Believe

ILLC DS-2018-11: **Malvin Gattinger**
New Directions in Model Checking Dynamic Epistemic Logic

ILLC DS-2018-12: **Julia Ilin**
Filtration Revisited: Lattices of Stable Non-Classical Logics

ILLC DS-2018-13: **Jeroen Zuiddam**
Algebraic complexity, asymptotic spectra and entanglement polytopes

ILLC DS-2019-01: **Carlos Vaquero**
What Makes A Performer Unique? Idiosyncrasies and commonalities in expressive music performance

ILLC DS-2019-02: **Jort Bergfeld**
Quantum logics for expressing and proving the correctness of quantum programs

ILLC DS-2019-03: **András Gilyén**
Quantum Singular Value Transformation & Its Algorithmic Applications

ILLC DS-2019-04: **Lorenzo Galeotti**
The theory of the generalised real numbers and other topics in logic

ILLC DS-2019-05: **Nadine Theiler**
Taking a unified perspective: Resolutions and highlighting in the semantics of attitudes and particles

ILLC DS-2019-06: **Peter T.S. van der Gulik**
Considerations in Evolutionary Biochemistry

ILLC DS-2019-07: **Frederik Möllerström Lauridsen**
Cuts and Completions: Algebraic aspects of structural proof theory

ILLC DS-2020-01: **Mostafa Dehghani**
Learning with Imperfect Supervision for Language Understanding

ILLC DS-2020-02: **Koen Groenland**
Quantum protocols for few-qubit devices

ILLC DS-2020-03: **Jouke Witteveen**
Parameterized Analysis of Complexity

ILLC DS-2020-04: **Joran van Apeldoorn**
A Quantum View on Convex Optimization

ILLC DS-2020-05: **Tom Bannink**
Quantum and stochastic processes

ILLC DS-2020-06: **Dieuwke Hupkes**
Hierarchy and interpretability in neural models of language processing

ILLC DS-2020-07: **Ana Lucia Vargas Sandoval**
On the Path to the Truth: Logical & Computational Aspects of Learning

ILLC DS-2020-08: **Philip Schulz**
Latent Variable Models for Machine Translation and How to Learn Them

ILLC DS-2020-09: **Jasmijn Bastings**
A Tale of Two Sequences: Interpretable and Linguistically-Informed Deep Learning for Natural Language Processing

ILLC DS-2020-10: **Arnold Kochari**
Perceiving and communicating magnitudes: Behavioral and electrophysiological studies

ILLC DS-2020-11: **Marco Del Tredici**
Linguistic Variation in Online Communities: A Computational Perspective

ILLC DS-2020-12: **Bastiaan van der Weij**
Experienced listeners: Modeling the influence of long-term musical exposure on rhythm perception

ILLC DS-2020-13: **Thom van Gessel**
Questions in Context

ILLC DS-2020-14: **Gianluca Grilletti**
Questions & Quantification: A study of first order inquisitive logic

ILLC DS-2020-15: **Tom Schoonen**
Tales of Similarity and Imagination. A modest epistemology of possibility

ILLC DS-2020-16: **Ilaria Canavotto**
Where Responsibility Takes You: Logics of Agency, Counterfactuals and Norms

ILLC DS-2020-17: **Francesca Zaffora Blando**
Patterns and Probabilities: A Study in Algorithmic Randomness and Computable Learning

ILLC DS-2021-01: **Yfke Dulek**
Delegated and Distributed Quantum Computation

ILLC DS-2021-02: **Elbert J. Booij**
The Things Before Us: On What it Is to Be an Object

ILLC DS-2021-03: **Seyyed Hadi Hashemi**
Modeling Users Interacting with Smart Devices

ILLC DS-2021-04: **Sophie Arnoult**
Adjunction in Hierarchical Phrase-Based Translation

ILLC DS-2021-05: **Cian Guilfoyle Chartier**
A Pragmatic Defense of Logical Pluralism

ILLC DS-2021-06: **Zoi Terzopoulou**
Collective Decisions with Incomplete Individual Opinions

ILLC DS-2021-07: **Anthia Solaki**
Logical Models for Bounded Reasoners

ILLC DS-2021-08: **Michael Sejr Schlichtkrull**
Incorporating Structure into Neural Models for Language Processing

ILLC DS-2021-09: **Taichi Uemura**
Abstract and Concrete Type Theories

ILLC DS-2021-10: **Levin Hornischer**
Dynamical Systems via Domains: Toward a Unified Foundation of Symbolic and Non-symbolic Computation

ILLC DS-2021-11: **Sirin Botan**
Strategyproof Social Choice for Restricted Domains

ILLC DS-2021-12: **Michael Cohen**
Dynamic Introspection

ILLC DS-2021-13: **Dazhu Li**
Formal Threads in the Social Fabric: Studies in the Logical Dynamics of Multi-Agent Interaction

ILLC DS-2022-01: **Anna Bellomo**
Sums, Numbers and Infinity: Collections in Bolzano's Mathematics and Philosophy

ILLC DS-2022-02: **Jan Czajkowski**
Post-Quantum Security of Hash Functions

ILLC DS-2022-03: **Sonia Ramotowska**
Quantifying quantifier representations: Experimental studies, computational modeling, and individual differences

ILLC DS-2022-04: **Ruben Brokkelkamp**
How Close Does It Get?: From Near-Optimal Network Algorithms to Suboptimal Equilibrium Outcomes

ILLC DS-2022-05: **Lwenn Bussière-Carae**
No means No! Speech Acts in Conflict