The Dissertation Committee for Niels Kornerup
certifies that this is the approved version of the following dissertation:

Time, Space, and Energy in Computation

Committee:

David Soloveichik, Supervisor

Scott Aaronson, Co-supervisor

Paul Beame

David Zuckerman

Time, Space, and Energy in Computation

by

Niels Kornerup

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

August 2025

Acknowledgments

I could not have asked for a better advisor than David Soloveichik, who helped ignite my love of research. David has always made time to collaborate on interesting questions and provide valuable and detailed feedback on my work while also giving me the freedom to pursue my own research directions. His insights on how to conduct and present research have been invaluable to my development as an academic. My co-advisor, Scott, has had profound impact on my research directions. During my tenure as a student at The University of Texas at Austin, I have taken three of Scott's classes, and they have been some of my favorite courses at the University. Scott has opened my eyes to the exciting world of quantum computation, and also given me a keen eye for seeing past the hype behind quantum. I would like to thank Paul for being a wonderful co-author and mentor throughout my PhD journey. I would also like to thank David Zuckerman for taking the time to be a member of my committee.

Next I would like to thank all of my math and science teachers from Gullett, Kealing, and LASA who inspired my love for these subjects. I am extremely privileged to have received such a high quality education that prepared me for my academic journey. Finally, I want to thank all the friends and family who have supported me throughout this process. My father and grandfather, Jacob and Peter respectively, both inspired me to follow in their footsteps and pursue a PhD in computer science. Jacob also took the time to read and give comments on parts of this dissertation, for which I am eternally grateful. Together with my mother and sister, Kat and Eva respectively, I am lucky to have such a supporting family nurturing my love of learning. Last but by no means least I want to thank my fiancée, Elena, for sticking by my side throughout this long journey.

My research in this dissertation was supported by NSF grant CCF-1901025, the Sloan Foundation, and a Schmidt Sciences Polymath award to David Soloveichik.

Preface

This dissertation contains text copied from multiple papers I have made significant contributions to during my PhD. Here I will outline the sources, collaborators, publication venues, and my contributions to each of these papers.

The work in Chapter 2 comes from (Kornerup et al., 2025) in collaboration with Jonathan Sadun and David Soloveichik. I presented an early version of this paper as a poster at QIP 2022 and the full version of the paper was published in the journal Quantum in 2025. I am mostly responsible for our results on the structure of optimal pebbling algorithms, our results on the binary tree, and our PSPACE-hardness result. In addition, I was the main author for most of the exposition as well as the final formulations of all of our proofs and theorems.

The work in Chapter 3 comes from (Beame et al., 2024) in collaboration with Paul Beame and Michael Whitmeyer. I presented this paper as a talk at both STOC 2024 and QIP 2025. Our new lower bounds for matrix-vector products and matrix multiplication were the result of close collaboration with Paul. I am the primary contributor for our lower bounds derived from our main theorems as well as the matching classical query algorithms. I was also substantially involved in the writing of Section 3.3, which was recommended by the anonymous reviewers for an in-progress submission of this paper to SICOMP. While I worked on the Boolean matrix multiplication problem, our final results and presentation were mainly the work of Paul and Michael.

The work in Chapter 4 comes from (Beame and Kornerup, 2023) in close collaboration with Paul Beame. Some of the related results from (Beame et al., 2024) are moved into this chapter in order to improve overall readability. I presented this paper as a talk at ICALP 2023 and a Journal version has been accepted to the ACM Transactions on Computation Theory journal (Beame and Kornerup, 2025). My main contributions to this work include the premise of studying unconditional cumulative

memory lower bounds, our preliminary lower bound on classical sorting (which directly lead to the simple version of our general result), generalizing the quantum sorting bounds to arbitrarily output orders, and using the generic theorems to prove most of our lower bounds. I was also heavily involved in proving our quantum cumulative memory lower bound on sorting, which eventually lead to our fully generic result. I was only minimally involved in the details of the proof of our generic theorems, but have included them for completeness.

The work in Chapter 5 is a heavily modified version of preprint (Doty et al., 2024) coauthored with David Doty, Austin Luchsinger, Leo Orshansky, David Soloveichik, and Damien Woods. The preprint was presented as a poster and short talk at DNA 2023 where it won the best student / postdoc poster presentation award. Since DNA 2023, my coauthors and I have learned significantly more about the thermodynamics of computation, making a lot of the old writing and some of the ideas for this project out of date. This new version of the project fixes many of these issues while adding new content on function computation to supplement our results on sampling. I was the primary author and contributor to both the preprint and the new modified version of this work presented in this dissertation.

Chapter 6 contains new work made in collaboration with Minki Hhan and David Soloveichik. My main intellectual contributions to this project include identifying minimizing entropy flow as a PP-hard problem, proving that approximate optimization is NP hard, and completing most of the writing in this chapter. At the time of writing, this dissertation is the only place where these results appear.

Abstract

Time, Space, and Energy in Computation

Niels Kornerup, PhD
The University of Texas at Austin, 2025

SUPERVISORS: David Soloveichik, Scott Aaronson

Time, Space, and Energy are the three most important measures of cost for computation. In this dissertation, we explore multiple ways of understanding these costs as notions of complexity for classical and quantum computation. This includes traditional ideas that are well established within the field of theoretical computer science alongside new approaches inspired by modern computing environments and recent developments in stochastic thermodynamics. We do this by discussing topics including the spooky pebble game, quantum time-space tradeoffs for matrix problems, cumulative memory complexity, Brownian computation, and the entropy production of Boolean circuits.

The spooky pebble game characterizes time-space trade-offs for the quantum simulation of irreversible classical circuits using intermediate measurements. We show asymptotically tight upper and lower bounds on the number of steps (or time) needed to pebble the line graph with any given pebble (or space) bound. This gives a general technique for simulating any irreversible classical computation on inputs in superposition with a better time-space trade-off than would be possible only using reversible simulation. We also generalize the spooky pebble game to arbitrary DAGs and show that in general finding the minimum number of pebbles required to pebble a graph is PSPACE-hard to approximate.

We introduce a new technique for applying Zhandry's quantum recording query method to prove tight quantum time-space product lower bounds for matrix problems. Using this technique we prove that for any space bound S, there is at most a constant factor speed up between S bit classical algorithms and S qubit quantum algorithms that compute the matrix-vector product function $f(x) = Ax$ or the matrix multiplication function $f(A, B) = AB$. We also introduce a new coloring technique that improves the best known quantum lower bound for Boolean matrix multiplication by a factor of $S^{1/4}$.

Cumulative memory complexity, the sum of the space needed per step of an algorithm, is a notion of time-space complexity originally formulated for password hashing. We justify cumulative memory as a modern measure of complexity for general algorithms that run on a shared device in the cloud or on a high performance computing system. We prove that virtually all known methods for proving *unconditional* time-space product lower bounds for classical and quantum algorithms, including our new bounds for quantum linear algebra, can be extended to give matching asymptotic bounds on the tighter notion of cumulative memory complexity. Thus, current techniques are *insufficient* to prove an unconditional asymptotic gap between cumulative memory and time-space product complexity.

Finally, we explore recent connections between the fields of stochastic thermodynamics and theoretical computer science to evaluate the energetic costs of computation. We provide an overview of how stochastic thermodynamics can be applied to computation and present a general strategy to build devices that can perform computation with an energy upper bound that only scales linearly with the size of the input and output regardless of the time complexity of the computation. We also consider the energetic costs of Boolean circuits, and prove that optimizing the energy efficiency of gates in a Boolean circuit is PP-hard and NP-hard to approximate.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Time and *space* are traditionally viewed as the most important measures of cost in computation. It is highly desirable to design algorithms, be they *classical* or *quantum*, so that they run in few steps (time) and only require a small amount of memory (space). However, simultaneously minimizing time and space costs is impossible for many kinds of computational problems (Cobham, 1966). We say that a problem admits a *time-space tradeoff* if solutions that are more time efficient require additional memory. For such problems, the optimal algorithm might depend on the resources available for computation. Thus it is often desirable to frame a time space tradeoff in the following manor: given a fixed space bound S what is the least time needed to solve a given problem? In the modern era, space-bounded computation is especially relevant within the context of quantum computation. The difficulty in scaling the number of error-corrected *logical qubits* is one of the major obstacles to building general use quantum computers. On the Google Willow processor, achieving a 10^{-6} error rate would require each logical qubit to be represented with $1,457$ physical qubits (AI and Collaborators, 2024). Thus it is important to consider what kinds of computations are possible on a quantum computer with a limited number of qubits.

In this work, we explore how bounds on the number of available qubits change the *time efficiency* of quantum computation. In Chapter 2 we address how a technique known as the *spooky pebble game* originally developed by Gidney (Gidney, 2019) can be used to give better tradeoffs between the number of qubits and the number of operations necessary to run irreversible classical subroutines like those found in Grover's search algorithm (Grover, 1996) on inputs in superposition. We prove *tight tradeoffs* for the spooky pebble game on the line graph, representing the limits of the technique on classical computation composed of sequential black box steps (Kornerup et al., 2025). Next in Chapter 3 we consider tradeoffs between the *query* and space complexity of quantum algorithms for matrix problems (Beame et al., 2024). The

query complexity of an algorithm is how many times that algorithm needs to make a query to its input $x_1, \ldots, x_n \in D^n$ to perform computation. For classical algorithms these queries take the form of an index i and the algorithm then learns the value of x_i. However quantum algorithms can perform queries in superposition to "learn" about multiple inputs in different branches. Since the query complexity of an algorithm is no larger than its time complexity, this gives us a way to describe the time complexity of problems. We prove tight quantum query-space tradeoff lower bounds for a wide range of linear algebra problems including matrix vector products and matrix multiplication. Moreover, most of the quantum lower bounds we prove are matched by *classical* query algorithms with the same tradeoffs. Therefore we can conclude that, for any space bound S, there is no *query advantage* for solving these problems with a quantum computer with S qubits compared to a classical computer with S bits.

Framing time-space complexity of computation in terms of a tradeoff between the maximum number of bits (or qubits) used in the computation and the number of computational steps is extremely natural in historical contexts: an isolated computer only has S memory and should be used to solve the desired problem as quickly a possible. However, we now live in an age where most expensive computations are performed on clusters of computers executing many concurrent programs. In such a setting, the space S available to a "machine" is significantly larger than what is required for any individual task. This change in context necessitates a new way of thinking about space as a resource in such computation. A single task can afford to use significant amounts of memory in short bursts as long as the tasks collectively never exceed S space. A similar observation was first made by Alwen and Serbinenko in the context of designing functions like password hashes that should require large time and space to compute (Alwen and Serbinenko, 2015). In particular, they observed that a password hashing function h only requiring large *maximum space* could be exploited by a special purpose ASIC device that computes k independent evaluations of h in parallel using significantly less than k times as much memory. To address this problem they introduced *cumulative memory complexity* — the sum of the used space per step of an

algorithm — as a more fine-grained way to evaluate the time-space costs of individual tasks on a shared device. Thus it is natural to ask: can we design algorithms that take advantage of spikes in memory to have a cumulative memory complexity lower than the best possible time-space product cost? Subject to cryptographic assumptions like the random oracle model, it turns out that this is possible. There are many cryptographic instantiations of password hashing functions where a clever parallel algorithm that alternates between short memory-intensive and long low-memory phases can compute the function with a cumulative memory complexity lower than the best possible time-space product cost (Alwen and Serbinenko, 2015; Boneh et al., 2016; Alwen et al., 2017a; Blocki and Zhou, 2017). In addition to applications for classical cloud and high performance computing, cumulative memory complexity is a natural concern for quantum algorithms that run on the first general purpose quantum computers. If each qubit in a quantum computer has a fixed rate of experiencing errors per unit time, then the expected number of errors in a quantum computer scales linearly with its cumulative memory complexity, and it is extremely important to optimize quantum algorithms to minimize this cost.

In Chapter 4 we present the first unconditional non-trivial lower bounds on cumulative memory complexity for any problem originally proven in (Beame and Kornerup, 2025). In particular, we prove explicit lower bounds on the classical and quantum cumulative memory complexity of sorting and present a general theorem that can convert virtually all existing classical and quantum time-space product lower bounds for multi-output functions to matching lower bounds on the cumulative memory complexity. Moreover, since many of the prior time-space product lower bounds as well as our new lower bounds in Chapter 3 are matched by existing algorithms, we show that these algorithms are also asymptotically optimal in terms of cumulative memory complexity.

While time and space are by far the best-understood notions of cost for computation, energy consumption is another natural measure of computational complexity. In 2023 United States data centers consumed 176 terawatt hours worth of energy,

representing 4.4% of total consumption (Shehabi et al., 2024). Despite this, we understand significantly less about the energy costs of computation. With current computer architectures, the energy cost of a computation is directly proportional to its time complexity. However, this is not a fundamental constraint on the energetic costs of computation. Landauer first observed that any physical process that erases information must dissipate heat (Landauer, 1961). In particular this is necessary to satisfy the *second law of thermodynamics* which states that the entropy of the universe never decreases. Bennett applied Landauer's observation to computational systems, observing that this heat cost can be avoided at a cost of time and space by making computation logically reversible (Bennett, 1973). Follow-up work has used the number of bit erasures in a computation as a proxy for the energy cost to investigate tradeoffs between time, space, and energy in computation (Li and Vitanyi, 1996; Demaine et al., 2016). However, recent developments in the field of stochastic thermodynamics indicate that these earlier attempts to quantify the energy costs of computation are somewhat misguided. Physicists have identified *entropy production* — the change in the entropy of the universe by a process — as a more precise way to capture energy costs in computation (Esposito and Van den Broeck, 2011; Sagawa, 2014; Strasberg et al., 2015; Wolpert, 2019; Wolpert et al., 2024). For a good summary of the recent progress in this area, we refer the reader to a recent survey on the topic here (Wolpert et al., 2024). Importantly, while Landauer's bound still requires information erasure to release heat, this heat can be later reabsorbed if the process is reversed to generate a fresh random bit (Esposito and Van den Broeck, 2011). On the other hand operations that are logically reversible when viewing the entire state of the machine may still need to release heat even if they are locally irreversible (Ouldridge and Wolpert, 2023). Entropy production is a harder concept to work with than logical reversibility, as it not only depends on the logical behavior of a computation, but also the distribution over inputs and the physical implementation of the device.

Chapter 5 presents a summary of how stochastic thermodynamics can be used to reason about energy costs in computation. Within this framework, we present a general

strategy that can be used to design special purpose energy-efficient computational devices driven entirely by Brownian motion (Doty et al., 2024). With some mild physical assumptions, these devices can sample from desired distributions or perform function computation with an energy cost that only scales linearly with the size of the input and output. In particular, the energy costs of these devices do not scale with the runtime of traditional algorithms that solve these tasks, making it possible to completely decouple *time* and *energy* costs of computation. Finally in Chapter 6, we show that optimizing the energy costs of a Boolean circuit with a fixed (uniform) input distribution is a PP-hard problem. This further justifies the well studied notion of "mismatch costs" — which is a source of entropy production caused by computation whose physical implementation is suboptimal for the target distribution over inputs (Kolchinsky and Wolpert, 2017).

1.1 Preliminaries

Here we present some formal definitions and notation that we will be using throughout this dissertation.

Definition 1.1. Let $f : D^n \to R^m$ be an arbitrary function. Then $f_i : D^n \to R$ denotes the subfunction of f containing only its i'th output.

Definition 1.2. Let $n \in \mathbb{N}$. Then $[n]$ denotes the set of n indices. This is normally $\{1, \ldots, n\}$, but could be $\{0, \ldots, n-1\}$ depending on the context.

Definition 1.3. Let R be a set and $J \subseteq \mathbb{N}$. Then R^J denotes the set of vectors with indices in J and values in R. Equivalently R^J can be viewed as the set of functions $f : J \to R$. Thus an element v of R^J can be indexed by either v_i or $v[i]$ for $i \in J$.

Definition 1.4. Let $y \in R^n$, $J \subseteq [n]$, and $\tau \in R^J$. Then τ agrees with y (or $\tau || y$) if and only if for each $i \in J$ we have that $\tau_i = y_i$.

1.1.1 A brief introduction to quantum computing

Our work in Chapters 2 to 4 features results related to quantum computing. For a more thorough background on this topic, we refer the reader to Scott Aaronson's lecture notes on the topic (Aaronson, 2018) or the Nielsen Chuang textbook (Nielsen and Chuang, 2010). Here we give a brief overview of quantum computing, based on the presentations in (Aaronson, 2018; Nielsen and Chuang, 2010), to make the results in this dissertation a bit more accessible to someone with a classical computer science background.

Quantum states and measurement An n qubit state is a normalized vector (i.e. the vector's two norm is 1) $|\psi\rangle \in \mathbb{C}^{2^n}$. The entry in the j'th index of this vector corresponds to 'amplitude' with which the qubit has the logical value j. We use $|j\rangle$ for $j \in [2^n]$ (or equivalently $j \in \{0,1\}^n$) to denote the quantum state with amplitude one on index j and 0 everywhere else. It is possible to combine two quantum states

with the tensor product. If $|\psi\rangle, |\phi\rangle$ are two quantum states on n and m qubits, then $|\psi\rangle \otimes |\phi\rangle = |\psi\phi\rangle$ is their combined state on $n+m$ qubits. The state $|00\rangle$ can be interpreted as a 2 qubit state where both qubits have the logical value 0 or equivalently as the tensor product of two single qubit states $|0\rangle \otimes |0\rangle$. There are some quantum states, like $(|00\rangle + |11\rangle)/\sqrt{2}$ that cannot be written as a tensor product of their component qubits. Such quantum states are called *entangled* — a special kind of quantum correlation that cannot exist in classical random variables.[1]

When a quantum state

$$|\psi\rangle = \begin{bmatrix} \alpha_0 \\ \vdots \\ \alpha_{2^n-1} \end{bmatrix} = \sum_j \alpha_i |j\rangle$$

is measured (in the standard basis), the probability of it giving the value j is $|\alpha_j|^2$. When the measurement gives the value j, the state 'collapses' to $|j\rangle$.

Let $\langle\psi|$ be the conjugate transpose of $|\psi\rangle$. In other words, if

$$|\psi\rangle = \begin{bmatrix} a_0 + b_0 i \\ \vdots \\ a_{2^n-1} + b_{2^n-1} i \end{bmatrix} = \sum_{j \in [2^n]} (a_j + b_j i) |j\rangle$$

Then

$$\langle\psi| = \begin{bmatrix} a_0 - b_0 i, & \dots & , a_{2^n-1} - b_{2^n-1} i \end{bmatrix} = \sum_{j \in [2^n]} (a_j - b_j i) \langle j|$$

This way for any pair of quantum states $|\psi\rangle, |\phi\rangle$, $\langle\psi|\phi\rangle$ is the inner product between these states. Observe then that $\langle\psi|\psi\rangle = 1$ when $|\psi\rangle$ is a quantum state. Likewise, two states $|\psi\rangle, |\phi\rangle$ are *orthogonal* if $\langle\psi|\phi\rangle = 0$. The inner product between quantum states characterizes their similarity.

Let us define two additional one qubit states:

$$|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}, |-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

[1]The CHSH game (Clauser et al., 1969) is a two player game where players sharing an entangled quantum state can outperform any classical strategy.

These states represent a *superposition* over the values $|0\rangle, |1\rangle$. The states $|+\rangle, |-\rangle$ can be thought of as quantum analogs of a coin flip; when measured (in the standard basis) the state is equally likely to collapse to either $|0\rangle$ or $|1\rangle$.[2] This might lead one to suspect that these states are somehow equivalent to one another. However, $\langle +| |-\rangle = 0$, implying that the states are orthogonal. In fact, using more general measurements, we can see that there is a way to perfectly distinguish state $|+\rangle$ from $|-\rangle$.

More generally (Nielsen and Chuang, 2010) define a quantum measurement as a set of linear operators $M_1, \ldots, M_k$ such that

$$\sum_j M_j^\dagger M_j = I$$

where $M_j^\dagger$ denotes the conjugate transpose of M_i and I is the identity matrix. When a state $|\psi\rangle$ is measured with this set of operators, the probability of outcome j is given by

$$p_j = \langle\psi| M_j^\dagger M_j |\psi\rangle .$$

After obtaining this measurement outcome, the state 'collapses' to

$$M_j |\psi\rangle / \sqrt{p_j}.$$

This more general framework describes, for example, how one could measure a single qubit in a larger system. The standard basis measurement is described with measurement operators $|0\rangle\langle 0|, \ldots, |2^n\rangle\langle 2^n|$. Going back to the states $|+\rangle, |-\rangle$, observe that a measurement described by the pair of operators $|+\rangle\langle +|$, $|-\rangle\langle -|$ gives distinct deterministic measurement outcomes when applied to these states.

The states $|\psi\rangle$ we have described so far are known as *pure states*. However, it is also possible to discuss probability distributions over pure states, forming what are

[2]When $|-\rangle$ is measured in the standard basis and the output value is 1, the state collapses to $-|1\rangle$. However, as there is no way to distinguish states $|1\rangle$ and $-|1\rangle$, we can drop this so-called *global phase*.

known as *mixed states*. Mixed states represent uncertainty in the state of a quantum system. While pure states are naturally defined in terms of normalized vectors in $\mathbb{C}^{2^n}$, mixed states are expressed as Hermitian matrices in $\mathbb{C}^{2^n \times 2^n}$. Let ρ be a mixture over pure states $|\psi_j\rangle$ with probability p_j. Then we can represent ρ as:

$$\rho = \sum_j p_j |\psi_j\rangle \langle\psi_j| .$$

It is possible to obtain the same mixed state from multiple distributions over pure states. For example the maximally mixed state:

$$\begin{aligned} \pi_{mix} &= \sum_{x \in \{0,1\}^n} \frac{1}{2^n} |x\rangle \langle x| = \sum_{y \in \{+,-\}^n} \frac{1}{2^n} |y\rangle \langle y| \\ &= \begin{bmatrix} 1/2^n & & \\ & \ddots & \\ & & 1/2^n \end{bmatrix} \end{aligned}$$

can be created by any uniform distribution over 2^n orthogonal basis states.

Like pure states, we can define measurement of mixed states with linear operators $M_1, \ldots, M_k$ where $\sum_j M_j^\dagger M_j = I$. We get that

$$p_j = \mathrm{Tr}[M_j^\dagger M_j \rho]$$

where Tr is the trace. On measurement outcome j the state collapses to

$$M_j \rho M_j^\dagger / p_j.$$

Unitary operations In addition to measurement, it is possible to manipulate the state of a quantum system by applying *unitary* operations. As a reminder, unitaries are linear operators U whose inverses are their conjugate transpose $U^\dagger$. Applying unitary U to an input $|\phi\rangle$ yields the state $|\psi\rangle = U|\phi\rangle$. For mixed states, applying U to an input σ yields the state $\rho = U\sigma U^\dagger$. It is possible to define unitaries that act on only a subset of the qubits in a quantum system. Such unitaries can be extended

to act as the identity on the remaining qubits. Important unitaries for quantum computation include the following:

$$\mathrm{X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \mathrm{Z} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \mathrm{S} = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}, \mathrm{H} = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix},$$

$$\mathrm{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \mathrm{TOFF} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Intuitively X represents a classical not gate — it swaps the amplitude associated with a qubit having the values 0 or 1. The Z and S gates add a phase of -1 or i respectively to a qubit when it has the logical value 1. The Hadamard gate H maps between $|0\rangle, |1\rangle$ and $|+\rangle, |-\rangle$. The controlled not gate CNOT applies an X gate to the second qubit when the first qubit has the logical value 1. The Toffoli gate applies an X gate to a third qubit when the first two qubits both have the logical value 1. The $\mathrm{X}, \mathrm{CNOT}, \mathrm{TOFF}$ gates represent logically reversible (i.e. invertible) classical operations. In fact, TOFF is a universal gate for classical computation: any classical circuit can be simulated using only TOFF gates.

We say that a set of quantum gates $\mathcal{S}$ is *universal* if an any unitary operator U can be approximated using only gates in $\mathcal{S}$.

Proposition 1.5 ((Shi, 2003))**.** *TOFF, H, S form a Universal gate set for quantum computation.*

While there are many such universal gate sets, picking a particular choice of $\mathcal{S}$ is not important for the performance of quantum circuits by the Solovay-Kitaev theorem.

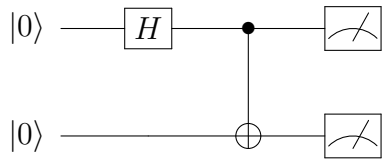

Figure 1.1: A simple quantum circuit that produces the state $(|00\rangle + |11\rangle)/\sqrt{2}$ before measuring.

Proposition 1.6 ((Kitaev, 1997))**.** *For any universal gate set $\mathcal{S}$ closed under inverses (i.e. if $V \in \mathcal{S}$ then $V^{-1} \in \mathcal{S}$), any unitary $V \in \mathbb{C}^{2^n \times 2^n}$ on n qubits can be ε approximated using $O(4^n \mathit{polylog}(1/\varepsilon))$ gates in $\mathcal{S}$.*

Therefore, as long as quantum computation is decomposed into a sequence of unitaries each acting on a constant number of qubits, the choice of universal gate set does not really matter.

Quantum circuits So far, we have described quantum states, measurements, and unitary operations. Often, these elements are combined to describe quantum computation in the language of quantum circuits. In the quantum circuit model, qubits are represented using wires that are acted on by unitary gates and measurements. A simple example of a quantum circuit is shown in Figure 1.1. This circuit starts with two qubits initialized to $|0\rangle$. A Hadamard gate (H) is applied to the first qubit. Next, a controlled not (CNOT) gate is applied from the first qubit to the second. Finally, both qubits are measured in the standard basis. While not shown in this example, classical information can be represented in a quantum circuit using two parallel lines as seen in our Figure 2.1. Such classical information is often used to control the application of a quantum gate. In Figure 2.1, the Z gate is only applied if the classical bit (generated from a measurement earlier in the circuit) has the value one.

Part I

Time and space for quantum and classical computation

Chapter 2: Tight bounds on the spooky pebble game

2.1 Introduction

Pebble games provide a convenient abstraction for reasoning about space and time usage in computation. Pebble games were first used in (Sethi, 1973) to determine optimal register allocation for computing straight line programs. In (Hopcroft et al., 1977) the authors applied the irreversible pebble game to show that any computation running on a Turing machine in time $T(n)$ can be executed on another Turing machine with space $T(n)/\log T(n)$. Since then, pebbling has found uses in establishing time-space trade-offs for computing functions including matrix vector products ((Tompa, 1980)) and memory hard functions based on hashing (eg. (Paul et al., 1976; Lengauer and Tarjan, 1982; Dwork et al., 2005; Ren and Devadas, 2016; Blocki and Zhou, 2017)). In (Bennett, 1989), Bennett used a reversible pebble game to give a general mechanism for reversibly simulating irreversible computation. As an example of more recent work, (Blocki et al., 2022) applied the reversible pebble game to analyze the post-quantum security of these memory hard functions.

In (Gidney, 2019) Gidney introduced a new *spooky pebble game* to study time-qubit trade-offs in quantum simulation of classical computation on inputs in superposition. Many quantum algorithms use simulation of classical computation in superposition as a subroutine. For example, applying Grover's search (Grover, 1996) to find an x such that $f(x) = 1$, requires a quantum circuit that implements the following mapping:

$$\sum_x \alpha_x |x\rangle |j_x\rangle \rightarrow \sum_x \alpha_x |x\rangle |j_x \oplus f(x)\rangle$$

which represents the evaluation of f on inputs in superposition. Since quantum gates are unitary (and therefore reversible) operations, $f(x)$ has traditionally been simulated with a reversible circuit. This comes with a time-space overhead that is

often overlooked. The spooky pebble game uses intermediate measurements to make such a simulation more efficient than would be possible using reversible circuits. We expand upon this pebble game and show tight time-space trade-offs for how efficiently it can simulate classical computation.

While the spooky pebble game can reduce the number of qubits needed to perform a computation, it is worth noting that it introduces new classical ancillary bits and does not reduce the total memory (qubits plus classical bits). Nonetheless, qubits are a much more limited resource than classical bits (Oskin et al., 2002). As such, we believe that this trade-off makes the spooky pebble game pragmatic for designing algorithms that run on quantum computers.

Previous work For any $\varepsilon \in (0, 1]$ the reversible pebble game can be used to reversibly simulate any irreversible computation that runs in time T with S space using $O(T^{1+\varepsilon}S^{-\varepsilon})$ steps and $O(S(1 + \log(T/S)))$ space (Bennett, 1989; Levine and Sherman, 1990). However, the asymptotic notation above hides a constant factor cost of approximately $\varepsilon 2^{1/\varepsilon}$ in the space term (Levine and Sherman, 1990).

Using the reversible pebble game, space $O(T^{\varepsilon}S^{1-\varepsilon})$ is sufficient to simulate irreversible computation in linear time, but similarly, this result features a steep but constant $2^{1/\varepsilon}$ cost in the time of the simulation (Král'ovič, 2001). Král'ovič also showed that any simulation via reversible pebbling the line with $O(S \cdot (1 + \log T/S))$ qubits must use $\Omega(T \cdot (1 + \log T/S))$ steps — a lower bound that is not achieved by any known reversible pebbling algorithm (Král'ovič, 2001). In (Blocki et al., 2022) the authors used a parallel version of the reversible pebbling game to show (parallel) time-space efficient algorithms for computing cryptographic objects known as data-independent memory hard functions.

Other works have tried to directly improve the qubit efficiency of running classical subroutines on a quantum computer without going through reversible simulation. In (Perdrix and Jorrand, 2006) the authors showed how a classically controlled

quantum Turing machine can simulate a classical Turing machine with no loss in time or space complexity. In (Ablayev et al., 2002; Cosentino et al., 2013) the authors showed that one qubit is sufficient to simulate NC^1 in polynomial time.

In (Gidney, 2019), Gidney introduced the idea of measurement-based uncomputing with his spooky pebble game. The spooky pebble game extends the reversible pebble game by allowing intermediate measurements that enable irreversible operations. The pebble game is called spooky because these measurements have a chance to produce undesired phases (or ghosts) that need to be removed before completing the computation; otherwise they will obstruct the desired interference patterns generated by subsequent gates. Gidney showed how the spooky pebble game can be used to simulate any classical computation with only a constant factor blow-up in space and a quadratic blow-up in the running time, which is impossible for reversible computation (Král'ovič, 2001). The spooky pebble game was recently extended to work on arbitrary DAGs in (Quist and Laarman, 2023, 2024), similar to the reversible pebble game. These works developed a SAT solver that can compute the minimum runtime necessary to solve the spooky pebble game on an arbitrary DAG. In (Quist and Laarman, 2024) the authors recently proved that deciding if a DAG can be pebbled with s pebbles is a PSPACE-complete problem, although their hard case requires a DAG with maximum in-degree $s - 1$.

Our results We build on the spooky pebble game framework introduced by Gidney (Gidney, 2019) and prove asymptotically tight upper and lower bounds on spooky pebbling the line. For any pebble bound s we show the existence of time-optimal pebbling strategies of a particular form that match our algorithms. We then are able to lower bound the number of steps needed to pebble the line with any algorithm of this form to get a lower bound on the number of steps needed in the spooky pebble game.

We delineate the entire achievable frontier of the spooky pebble game and our tight trade-off bounds can be applied in many different regimes. For example, for

any $\varepsilon \in (0, 1]$, any classical computation that runs in T time with S space can be simulated on a quantum computer using only $O(T/\varepsilon)$ steps and $O(T^{1+\varepsilon}S^{1-\varepsilon})$ qubits. This is an exponential improvement in ε over the reversible bound in (Král'ovič, 2001). We also show that any computation can be simulated in $O(T^{1+\varepsilon}S^{-\varepsilon}/\varepsilon)$ steps with only $O(S/\varepsilon)$ qubits, which is fewer qubit than would be possible (independent of the time bound) with reversible pebbling (Li and Vitanyi, 1996; Li et al., 1998; Král'ovič, 2001). Interestingly, when $\varepsilon = 1/\log(T/S)$, this matches the (unobtained) lower bound for reversible pebbling proved in (Král'ovič, 2001).

We then show that any irreversible pebbling can be converted to a spooky pebbling using only one additional pebble. This gives us that, in general, the number of pebbles needed for the spooky pebble game is PSPACE-hard to approximate. Finally we discuss the spooky pebble game on directed acyclic graphs (DAGs) where we show that it is possible to spooky pebble the complete binary tree of height h (i.e., $n = 2^h - 1$ nodes) using $h + 1$ pebbles and $O(n \log n)$ steps. This is fewer pebbles than is possible in the reversible pebble game, addressing an open question posed in (Quist and Laarman, 2024) regarding efficient algorithms for the spooky pebble game on trees.

2.2 Preliminaries

Reversible computation We say that a computation is *logically reversible* if, after every time step, there is a unique predecessor state for the computation. For example a Turing machine that moves its head from left to right inverting the bits on its tape represents a reversible computation while one that instead overwrites its tape with zeroes is not. When a series of quantum transformations are applied to a quantum system, it results in a unitary transformation U being performed on the system. Since the original state of a measurement free quantum algorithm can be restored by applying the inverse operation $U^\dagger$, measurement-free quantum computation must be reversible (Rieffel and Polak, 2011). This means that implementing classical algorithms

using a quantum computer often implicitly involves the additional step of making the algorithm reversible.

Of the strategies for making classical computation reversible the most widely used method is the reversible pebble game (Bennett, 1989). In the reversible pebble game, steps of an irreversible algorithm are simulated reversibly by placing and removing pebbles on a line, or more generally any directed acyclic graph (DAG). The largest number of pebbles s placed at any time corresponds to the space needed for the simulation and the number of steps τ corresponds to the time of the simulation.

Uncomputation Recycling space is key to space efficiency. But while irreversible algorithms can simply erase values whenever they are no longer needed, such values require more care to dispose of in reversible and quantum computation. In order to erase a value in a reversible manner, it is necessary to end up in a state where it would be possible to efficiently recompute the deleted value. Thus *uncomputation*, the reversible analog of deletion, requires access to the same information needed to compute the deleted value. The importance of uncomputation in quantum circuits it twofold: (1) qubits are an expensive resource for quantum algorithms so it is important to design them so that they can run on as few qubits as possible and (2) failing to uncompute values in a quantum circuit results in entangled garbage that prevents desired interference patterns (Paradis et al., 2021).

We say a quantum circuit *uncomputes* a function f if it can perform the following mapping on any quantum state where the x_i are all distinct:

$$\sum_i \alpha_i |x_i\rangle |f(x_i)\rangle \to \sum_i \alpha_i |x_i\rangle |0\rangle$$

Note that the standard unitary U_f for computing f, which performs the mapping $U_f |x\rangle |b\rangle = |x\rangle |b \oplus f(x)\rangle$ is also a unitary that uncomputes f. Carefully balancing computation and uncomputation is vital for designing space and time efficient quantum algorithms.

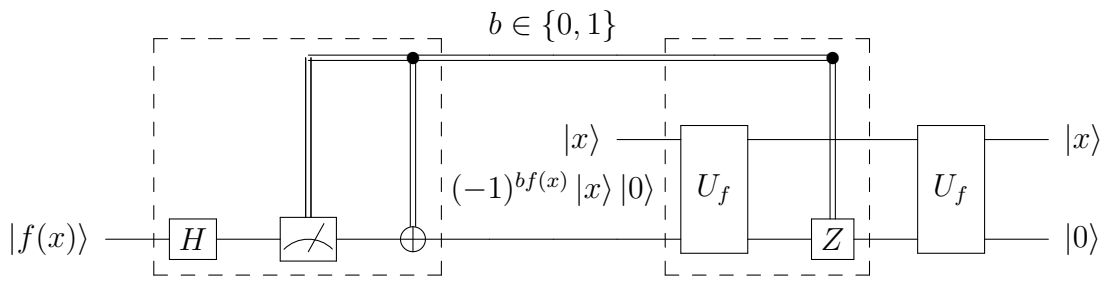

Figure 2.1: Circuit to ghost $f(x)$ and later correct the phase in order to overall uncompute $f(x)$ using measurement-based uncomputing, as described in (Gidney, 2019).

Ghosting Gidney points out that full uncomputation is not always necessary for recycling qubits when simulating classical algorithms in superposition. Instead of full uncomputation, he developed a clever scheme using intermediate measurements to "compress" qubits into classical bits (Gidney, 2019). We say that a quantum circuit $\mathcal{C}$ *ghosts* a register if it can perform the following mapping on any quantum state where the x_i are all distinct:

$$\sum_i \alpha_i |x_i\rangle |y_i\rangle \xrightarrow{\mathcal{C}} \sum_i \alpha_i (-1)^{b \cdot y_i} |x_i\rangle |0\rangle$$

and b is a classical bitstring returned by the circuit. In (Gidney, 2019) Gidney calls this $(-1)^{b \cdot y_i}$ phase a *ghost* of y, as the logical value of this register has been erased and replaced with a phase. Since the logical value of the register has been zeroed out, it can be used as if it were a fresh ancillary register for any subsequent computation acting only in the standard basis. However, for a quantum circuit to behave correctly, this ghost must be removed before acting on this register in another basis. By recreating the state of the y register as it was before ghosting and recalling the classical bitstring b, the ghost can be removed by applying a b-controlled Z gate to the register. Finally, uncomputation can be used to properly uncompute the register. While this might seem like uncomputation with extra steps, we will see that *reusing registers before we could uncompute their values allows for more qubit and gate efficient quantum algorithms.*

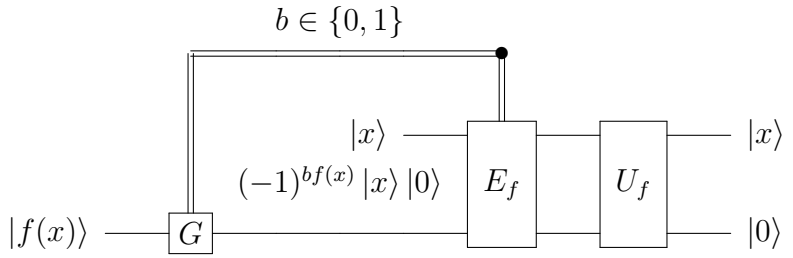

Figure 2.2: The circuit from Figure 2.1 using G and E_f circuit macros. Note that the • on the classical wire above the E_f gate represents a controlled application of the component Z gate rather than a controlled application of an entire module.

Ghosting is not a unitary operation and thus requires intermediate measurements: for example, if you tried to ghost the second qubit in the state $(|00\rangle + |01\rangle)/\sqrt{2}$ and receive $b = 0$, then you would end up with the non-normalized state $(|00\rangle + |00\rangle)/\sqrt{2} = \sqrt{2}\,|00\rangle$. Importantly, note that ghosting is only defined to perform this mapping when the x_i are distinct, as that is sufficient to prevent interference between basis states and make it a norm-preserving operation.

In (Gidney, 2019) Gidney shows that ghosting can be performed by measuring in the Hadamard basis, recording the result as classical output b, and then using b to apply a controlled X-gate to the register. Figure 2.1 shows what this circuit looks like, as the gates on the left "ghost" the $f(x)$ register and the gates on the right recompute $f(x)$ before removing the ghost and uncomputing $f(x)$. Importantly, the gates that ghost the $f(x)$ register do not use the x register, so these operations can be performed even when the x register is holding a different value. However, removing the ghost then requires the presence of x.

For compactness, we compress the highlighted gates on the left and the right of Figure 2.1 into single gates G and E_f as shown in Figure 2.2. When performing a classical subroutine on a quantum computer, the G gate can be applied to any register at any time to reset the logical value of its qubits (ghosting). However, whenever this is done, the E_f gate must be reapplied later in the circuit when it is possible to recompute the original value in the erased register and remove the phase (unghosting).

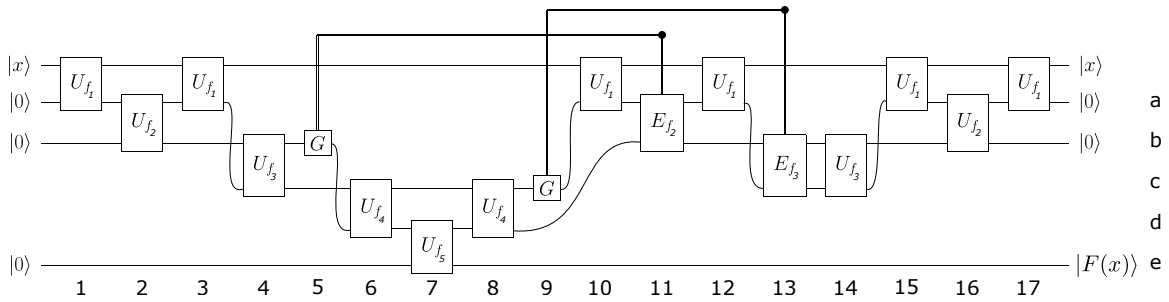

Figure 2.3: Qubit efficient computation of $F(x) = f_5 \circ f_4 \circ f_3 \circ f_2 \circ f_1(x)$ using ghosting. Again the $\bullet$ on the classical wire above the E_{f_i} gates represents a controlled application of the component Z gate rather than a controlled application of an entire module.

Figure 2.3 shows how these G and E_f gates can be applied to compute a function with less qubits than would be possible using reversible simulation. This application is general: we can think of a function f as mapping a computer from its current logical configuration to its configuration at the next time step. Since the measured value b must be stored from the time a value is ghosted until the time when its ghost is removed, we are not able to reduce the overall space complexity of the simulation; we are only able to reduce the number of required qubits.

Gidney's original formulation of the spooky pebble game only produced a ghost when the returned value $b \neq 0$ since when $b = 0$ no phase is added (Gidney, 2019). This version of the spooky pebble game is natural in settings like modular arithmetic circuit where it is desirable to ghost individual bits (Luongo et al., 2024). However in this work, we apply the G gate on registers containing many qubits where the probability that all measured qubits yield zero will be negligibly small. As such, we make the simplifying assumption that ghosts are always created when we apply the G gate.

Pebble games The concept of interweaving computation, uncomputation, and ghosting to efficiently simulate classical computation can be perfectly encapsulated by an appropriate pebble game. In a pebble game we are given a DAG where the nodes represent the variables involved in a computation and the edges show the

dependencies for computing these variables. For example the computation $c = a + b$ can be represented with three nodes for a, b, c and edges $a \to c$ and $b \to c$. A pebble can be placed on a node to indicate that the variable is currently stored in a register for the computation. Thus, a pebble can be placed on node c only when there are already pebbles on a and b. Additional restrictions on when pebbles can be placed or removed create different pebble games that can be used to simulate irreversible, reversible, or quantum computation with ghosting.

Such graph representations naturally characterize computations such as straight line programs, circuits, and data-independent cryptographic functions acting on words (registers) of size w. In this setting, pebble games directly model time-space trade-offs—as a pebbling of the DAG for a function f with s pebbles placed concurrently and τ steps corresponds to a way to compute f with $O(ws)$ bits in $O(\tau)$ steps.

While pebble games are defined for general DAGs, it is possible to think of each pebble as representing a snapshot of the entire state of an arbitrary sequential computation and then represent the computation as a pebble game on a line. If you can construct a pebbling $\mathcal{P}_n$ that works on the line of length n in τ_n steps and uses at most s_n concurrent pebbles, this leads to a naive way to simulate a computation that requires T time and S space in $O(S \cdot \tau_T)$ steps[1] and $O(S \cdot s_T)$ space (or qubits). A more clever way to apply pebbling to simulation involves assigning S states to each pebble, which lets us pebble a shorter line without increasing the asymptotic space needed per pebble or the time needed per step.

Proposition 2.1 (Implicit in (Bennett, 1989))**.** *Let M be an irreversible sequential machine that computes a function f with T steps using S bits. Let $\mathcal{P}_{T/S}$ be a (spooky) pebbling strategy for the line of length T/S that runs in $\tau_{T/S}$ steps and uses at most $s_{T/S}$ concurrent pebbles. Then there exists a quantum circuit that can compute f with $O(S \cdot \tau_{T/S})$ gates and $O(S \cdot s_{T/S})$ qubits.*

[1]There is a multiplicative term of S here because each state must be copied before simulating the next step.

We now more formally define the irreversible, reversible, and spooky pebble games that we will use throughout this chapter.

Definition 2.2. The irreversible pebble game is a one player game on a DAG $G = (V, E)$ where the goal is to place a pebble on exactly the nodes $T \subseteq V$ called *targets* with out-degree zero. A pebbling (strategy) is a list of subsets of V. $\mathcal{P} = [\mathcal{P}_0, \ldots, \mathcal{P}_\tau]$ where $\mathcal{P}_0 = \emptyset$ and $\mathcal{P}_\tau = T$. A strategy is valid as long as

- $|\mathcal{P}_i \triangle \mathcal{P}_{i+1}| = 1$, and
- If $v \in \mathcal{P}_{i+1} \setminus \mathcal{P}_i$, then parents$(v) \subseteq \mathcal{P}_i$.

We can analyze a pebbling by looking at the number of pebbles and steps it requires.

Definition 2.3. The number of steps $T(\mathcal{P})$ in a pebbling strategy $\mathcal{P} = [\mathcal{P}_0, \ldots, \mathcal{P}_\tau]$ is τ.

Definition 2.4. The number of pebbles $S(\mathcal{P})$ in a pebbling strategy $\mathcal{P} = [\mathcal{P}_0, \ldots, \mathcal{P}_\tau]$ is $\max_{i \in [\tau]} |\mathcal{P}_i|$.

When restricted to reversible computation, we can modify Definition 2.2 to get a pebble game that naturally models the restrictions imposed by reversibility.

Definition 2.5. The reversible pebble game has the same setup as the irreversible pebble game in Definition 2.2. A strategy is valid as long as

- $|\mathcal{P}_i \triangle \mathcal{P}_{i+1}| = 1$,
- If $v \in \mathcal{P}_{i+1} \setminus \mathcal{P}_i$, then parents$(v) \subseteq \mathcal{P}_i$, and
- If $v \in \mathcal{P}_i \setminus \mathcal{P}_{i+1}$, then parents$(v) \subseteq \mathcal{P}_i$.

The pebble game capturing ghosting admits better pebbling strategies than are possible in the reversible pebble game, assuming we are concerned with the number of qubits rather than the total space complexity. Given an irreversible pebbling, we can exactly characterize when it produces a ghost.

Definition 2.6. The ghosting sequence $\mathcal{G}(\mathcal{P}) = [\mathcal{G}_0, \ldots, \mathcal{G}_{T(\mathcal{P})}]$ is defined recursively by $\mathcal{G}_0 = \emptyset$ and $\mathcal{G}_{i+1} = (\mathcal{G}_i \cup \{v \in \mathcal{P}_i | \text{parents}(v) \not\subseteq \mathcal{P}_i\}) \setminus \mathcal{P}_{i+1}$.

The ghosting sequence cumulatively tracks the locations where pebbles were removed without access to their parents. These are exactly the steps that prevent a strategy from being a valid strategy for the reversible game; a valid reversible pebble game strategy is exactly a strategy with a ghosting sequence consisting only of empty sets. The spooky pebble game relaxes this condition to only requiring the *last* ghost set to be empty. Note that ghosts can be removed by placing a pebble over them.

Definition 2.7. The spooky pebble game has the same setup as the irreversible pebble game in Definition 2.2. A strategy is valid as long as

- $|\mathcal{P}_i \triangle \mathcal{P}_{i+1}| = 1$,
- If $v \in \mathcal{P}_{i+1} \setminus \mathcal{P}_i$, then $\text{parents}(v) \subseteq \mathcal{P}_i$, and
- In the ghosting sequence $\mathcal{G}(\mathcal{P})$, the final ghost set $\mathcal{G}_\tau = \emptyset$.

Note that by construction, the spooky pebble game always lies somewhere between the reversible and irreversible pebble games. Therefore, it is natural to compare it to these other pebble game and investigate when its behavior is closer to that of reversible or irreversible pebbling.

Figure 2.4 gives an example of a spooky pebbling on the line of length 5 using only 3 pebbles. Note that this models the same computation performed in Figure 2.3. Each numbered step in Figure 2.4 corresponds to the state of the quantum circuit in Figure 2.3 after the corresponding numbered gate is applied; a wire on row c in

Figure 2.3 occurs exactly at the same numbered steps as a pebble at column c in Figure 2.4 and corresponds to storing the value $f_3 \circ f_2 \circ f_1(x)$ in quantum memory within both models. In the reversible pebbling game, this task requires 4 pebbles. The ability to ghost pebbles and remove the ghosts at later steps lets us use one fewer pebble, showing that the spooky pebble game can be used to save qubits.

Rather than describe a pebbling as a list of pebbled nodes, we will sometimes describe algorithms that generate these lists. These algorithms feature the functions "place" and "remove". The t'th call to these functions defines the value of $\mathcal{P}_t$ from $\mathcal{P}_{t-1}$ as follows:

- place(v_i): $(\mathcal{P}_t) = (\mathcal{P}_{t-1} \cup \{v_i\})$
- remove(v_i): $(\mathcal{P}_t) = (\mathcal{P}_{t-1} \setminus \{v_i\})$

Intuitively, the place instruction creates a pebble on a node while the remove instruction destroys that pebble. For the spooky pebble game, ghosts are implicitly created by remove instructions according to Definition 2.6.

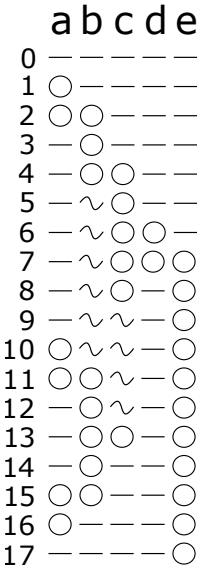

Figure 2.4: A spooky pebbling of the line. Here each ∘ indicates a pebble and each ∼ indicates a ghost.

Given a spooky pebbling algorithm and a node v_i, we define a notation for the first and last time that this node contains a pebble:

Definition 2.8. Let $\text{First}(v_i, \mathcal{P}) = \min(\{t | v_i \in \mathcal{P}_t\})$ and $\text{Last}(v_i, \mathcal{P}) = \max(\{t | v_i \in \mathcal{P}_t\})$.

2.3 Pebbling, unpebbling, and unghosting

Before discussing algorithms and lower bounds for playing the spooky pebble game on the line, we will define a number of subroutines in such pebbling algorithms. These subroutines describe sequence of steps within a pebbling algorithm that seek

to complete some concrete objective. An *unpebbling* subroutine $\mathcal{U}_p(\{v_1, \ldots, v_n\})$ is a sequence of place and remove instructions that starts with a pebble only on v_n and ends with no pebbles or ghosts on $v_1, \ldots, v_n$. Likewise, an *unghosting* subroutine $\mathcal{U}_g(\{v_1, \ldots v_n\})$ starts with no pebbles on $v_1, \ldots, v_n$ but a ghost on v_n and ends with no pebbles or ghosts on $v_1, \ldots, v_n$. Note that the definitions of both of these subroutines do not preclude the existence of ghosts on additional nodes in the initial step. As long as v_n is a sink of $\{v_1, \ldots, v_n\}$ (which is true in the case of the line graph), any unpebbling or unghosting subroutine must place at least one pebble on $v_1, \ldots, v_{n-1}$ in order to remove the pebble or ghost on v_n. In doing so, the subroutine will overwrite any additional ghosts present at the start of its execution.

We are primarily interested in these subroutines as they make it easier to describe pebbling algorithms. In particular pebbling algorithms have a bit of an awkward asymmetry where they first have to place a pebble on the target node(s) and then run a subroutine to remove the remaining ghosts and pebbles. On the other hand, unpebbling and unghosting subroutines have a natural recursive structure: after removing a pebble or ghost from v_n the same subroutine can be used on a smaller input to clean up the rest of the graph.

It turns out that the pebble cost s and time cost τ of pebbling, unpebbling, and unghosting tasks are closely related. Thus, we can reason about easier to work with unpebbling and unghosting tasks to make claims about efficient pebbling algorithms. In this section, we will prove some key lemmas that give us the following connections between the costs of these tasks.

Theorem 2.9. *Let $T_P(n, s), T_{UP}(n, s), T_{UG}(n, s)$ be the minimum number of steps needed to respectively pebble, unpebble, or unghost the line of length n using at most s pebbles. Then:*

$$T_P(n+1, s+1) = T_{UG}(n, s) + 1, T_{UG}(n, s) = T_{UP}(n, s) \pm 1$$

We start by presenting a key lemma showing a time efficient pebbling sequence to place pebbles on any desired set of nodes.

Lemma 2.10. *Let $\mathcal{G} = (V, E)$ be a line and $S \subseteq V$ be such that $|S| = s$. Then there exists a valid sequence of pebbling instructions that places pebbles on exactly S that is time-optimal and uses at most $s + 1$ pebbles.*

Proof. Consider the following pebbling sequence:

$\mathcal{I}(\{v_1, \ldots, v_n\}, S)$:
    $k = \texttt{argmax}_i v_i \in S$
    `place(`v_1`)`
    `for` $i \in [2, k]$:
        `place(`v_i`)`
        `if(`$v_{i-1} \notin S$`)`:
            `remove(`v_{i-1}`)`

The above sequence uses at most $s + 1$ pebbles and takes exactly $2k - s$ steps. Since placing a pebble on v_k requires placing pebbles on each node before it and we want to only end with pebbles on S, any pebbling sequence that completes this task must place at least k pebbles and remove $k - s$ pebbles. Therefore, the above pebbling sequence uses the time-optimal number of steps. □

Now we can use this result to give a tight relationship between pebbling and unghosting the line. We first show how an unghosting subroutine can be converted into a pebbling algorithm.

Lemma 2.11. *Let $\mathcal{U}_g$ be an unghosting algorithm for the line of length n that uses s pebbles and takes τ steps. Then there exists a pebbling algorithm on the line of length $n + 1$ that uses $(s + 1)$ pebbles and takes at most $(\tau + 1)$ steps.*

Proof. We take $\mathcal{U}_g$ and modify it by adding a pebble instruction on v_{n+1} immediately after step $\text{First}(v_n, \mathcal{U}_g)$ — which must have been a pebble instruction on node v_n. Since v_n starts with a ghost, $\mathcal{U}_g$ must place a pebble at v_n during some step. The resulting algorithm is a valid pebbling that uses at most $\tau + 1$ steps and $s + 1$ pebbles. □

Next we show that Lemma 2.11 is tight.

Lemma 2.12. *There exists a pebbling algorithm $\mathcal{P}$ on the line of length $n+1$ that uses $s+1$ pebbles and takes $\tau+1$ steps if and only if there exists an unghosting algorithm $\mathcal{U}_g$ on the line of length n that uses s pebbles and takes τ steps.*

Proof. In Lemma 2.11, we showed how an unghosting can be modified to give a pebbling. Now we show that any pebbling can be modified to give an unghosting.

Let t be the last step of $\mathcal{P}$ operating on v_{n+1}. This must be a pebbling step, so both v_n and v_{n+1} are in $\mathcal{P}_t$. Divide $\mathcal{P}$ into two components around this point, $\mathcal{P}_{\text{pre}}$ for the first t steps and $\mathcal{P}_{\text{post}}$ for the remaining steps. Since $\mathcal{P}_{\text{pre}}$ must have spent at least one step on pebbling each of v_n and v_{n+1}, there must be some algorithm $\mathcal{P}^*_{\text{pre}}$ running in at most $t-2$ steps that ends in $\mathcal{P}_t \setminus \{v_n, v_{n+1}\}$. Moreover, by Lemma 2.10, $\mathcal{P}^*_{\text{pre}}$ may be constructed to require at most $|\mathcal{P}_t \setminus \{v_n, v_{n+1}\}| + 1 \leq s$ pebbles.

Let $\mathcal{P}^*_{\text{post}}$ be a sequence of pebbling instructions derived from $\mathcal{P}_{\text{post}}$ by removing any instructions operating on v_n before $\text{Last}(v_n, \mathcal{P}_{\text{post}})$, then inserting an instruction placing a pebble on v_n immediately before that last removal of v_n. This is valid, as the new placement occurs immediately before a removal (so the predecessor of v_n is pebbled), and no operations occur on v_{n+1} that would be affected by changing whether a pebble is present on v_n.

$\mathcal{P}^*_{\text{pre}}$ followed by $\mathcal{P}^*_{\text{post}}$ is then a valid unghosting of the line graph up to v_n, excluding v_{n+1}. Since $\mathcal{P}^*_{\text{pre}}$ is at least 2 steps shorter than $\mathcal{P}_{\text{pre}}$ and $\mathcal{P}^*_{\text{post}}$ is at most 1 step longer than $\mathcal{P}_{\text{post}}$, the composition takes at most τ steps. Moreover, the composition requires at most s pebbles, $\mathcal{P}^*_{\text{pre}}$ by Lemma 2.10 and $\mathcal{P}^*_{\text{post}}$ because it operates as in $\mathcal{P}_{\text{post}}$ but without an extra pebble sitting on v_{n+1}. □

We can also characterize the relationship between unghosting and unpebbling.

Lemma 2.13. *Let $\mathcal{U}_g$ be any unghosting subroutine for the line of length n that uses s pebbles and requires τ steps. Then there exists an unpebbling subroutine using at most s pebbles and $\tau+1$ steps.*

Proof. Define $\mathcal{U}_p$ by first removing the pebble at v_n to create a ghost and then running $\mathcal{U}_g$. □

Lemma 2.14. *Let $\mathcal{U}_p$ be any unpebbling subroutine for the line of length n that uses s pebble. Then there exists an unghosting subroutine that uses at most s pebbles and $\tau + 1$ steps.*

Proof. Define $\mathcal{U}_g$ by running $\mathcal{U}_p$, but before $\text{Last}(v_n, \mathcal{U}_p)$, skip all preexisting operations on v_n, and immediately before $\text{Last}(v_n, \mathcal{U}_p)$, add a new step to place a pebble on v_n. $\mathcal{U}_g$ only differs from $\mathcal{U}_p$ by sometimes not having a pebble on v_n when $\mathcal{U}_p$ does, so it uses at most as many pebbles, and it inserts only a single new step. □

2.4 Algorithms for spooky pebbling the line

First we show asymptotically tight results for playing the spooky pebble game on the line graph, which characterizes the achievable time-qubit tradeoffs for simulating arbitrary irreversible classical computation on inputs in superposition using intermediate measurements. We start by constructing a time-efficient pebbling strategy for any bound $s \geq 3$ on the number of allowed pebbles.

Lemma 2.15. *For any $s, m \in \mathbb{N}$ such that $n \leq \binom{m+s-1}{s-1}$, there exists an unghosting algorithm for the line of length n that uses at most s pebbles and $O(mn)$ steps.*

Proof. We consider the following unghosting algorithm that uses s pebbles for when the line has length exactly $n = \binom{m+s-1}{s-1}$ for some value of m:

```
U_g({v_1, ..., v_n}, s): //Let n = C(m+s-1, s-1) for some value of m.
    if (n ≤ s):
        for i ∈ [1, n]:
            place(v_i)
        for i ∈ [0, n − 1]:
            remove(v_{n−i})
    else:
        let k = C(m+s−2, s−1)
```

```
place(v_1)
for i ∈ [2,k]:
    place(v_i)
    remove(v_{i-1})
run U_g({v_{k+1},... v_n}, s-1) //Line of length n-k = C(m+s-2, s-2).
remove(v_k)
run U_g({v_1,...,v_k}, s) //Line of length k = C(m+s-2, s-1).
```

Note that if $m' = m - 1$ then $k = \binom{m'+s-1}{s-1}$, so our assumption on n holds in the recursive cases. While the recursive calls to $\mathcal{U}_g$ may run on lines that contain ghosts on nodes other than the target, removing the ghost at the target will require placing pebbles on all other nodes. This will remove any additional ghosts present at the start of the recursive call. To upper bound the number of steps required to run $\mathcal{U}_g$ on a line of length $n = \binom{m+s-1}{s-1}$ with s pebbles, we will upper bound the number of steps that act on any node and multiply that by the total number of nodes. Let $T_i(m, s)$ be the number of times $\mathcal{U}_g$ acts on node v_i and $T^*(m, s) = \max_i T_i(m, s)$. Then directly from the construction of $\mathcal{U}_g$ we have:

$$T^*(m,s) \leq \begin{cases} 2 & s \geq \binom{m+s-1}{s-1} \\ \max\left(2 + T^*(m-1,s), T^*(m,s-1)\right) & s < \binom{m+s-1}{s-1}. \end{cases}$$

In other words, when $s \geq n$, each node has a pebble placed and removed at most once. Otherwise, when $s < n$, the node visited the most times either is before v_{k+1} and gets a pebble placed and removed and is part of the call $\mathcal{U}_g(\{v_1, \ldots, v_k\}, s)$ or is after k and is part of the call $\mathcal{U}_g(\{v_{k+1}, \ldots v_n\}, s-1)$.

When m or s is 1, we observe that $s \geq \binom{m+s-1}{s-1}$, and so the base case $T^*(1, s) = T^*(m, 1) = 2$. Since every recursive step at most increases T^* by 2 every time that m increases by 1, it follows that $T^*(m, s) \leq 2m$. This lets us conclude that when the line has length $n = \binom{m+s-1}{s-1}$, we can unghost it using $O(mn)$ steps.

When $n < \binom{m+s-1}{s-1}$ we observe that the above unghosting algorithm can be truncated to the shorter line of length n by ignoring steps on nodes after v_n. Again

the largest number of steps on any node is at most $2m$ so we can conclude that the unghosting algorithm takes at most $2mn$ steps. □

Finally, we can convert this to a pebbling algorithm with Lemma 2.11.

Theorem 2.16. *For any $s, m \in \mathbb{N}$ such that $n \leq \binom{m+s-2}{s-2} + 1$, there exists a pebbling algorithm for the line of length n that uses at most s pebbles and $O(mn)$ steps.*

As an immediate corollary of this result, we are able to obtain a strategy for pebbling a line of arbitrary length using only 3 pebbles. This algorithm was first presented in (Gidney, 2019) and our Theorem 2.16 is a generalization of this algorithm to more pebbles.

Corollary 2.17. *There exists a pebbling algorithm for the line of length n that uses 3 pebbles and $O(n^2)$ steps.*

While the formal statement of Theorem 2.16 is hard to parse, it is a strict improvement over the best possible tradeoffs between pebbles and steps in the reversible pebble game. Corollary 2.17 already improves on the $\Omega(\log n)$ lower bound on the number of pebbles needed to reversibly pebble the line of length n. Here we state some corollaries of Theorem 2.16 that help compare the performance of our spooky pebbling algorithm to known results for the reversible pebble game.

Corollary 2.18. *There exists spooky pebbling algorithms on the line of length n that use s pebbles and run in $O(sn^{1+1/(s-2)})$ steps for all $s \geq 3$.*

Proof. Set $m = \left\lceil (s-2)(2n^{1/(s-2)} - 1) \right\rceil$ and observe that:

$$\begin{aligned}\binom{m+s-2}{s-2} + 1 &\geq \left(\frac{m+s-2}{s-2}\right)^{s-2} \\ &= \left(\frac{\left\lceil (s-2)(2n^{1/(s-2)} - 1) \right\rceil + s - 2}{s-2}\right)^{s-2}\end{aligned}$$

$$
\begin{aligned}
&= \left(\frac{\left\lceil (s-2)2n^{1/(s-2)} \right\rceil}{s-2} \right)^{s-2} \\
&\geq \left(2n^{1/(s-2)} \right)^{s-2} \\
&\geq n
\end{aligned}
$$

Therefore by Theorem 2.16 there exists a pebbling on the line of length n using s pebbles and $O(sn^{1+1/(s-2)})$ steps. □

We can compare Corollary 2.18 to the following lower bounds on the reversible pebble game.

Proposition 2.19 ((Li et al., 1998))**.** *Any reversible pebbling on the line of length n requires $\Theta(\log n)$ pebbles.*

When s is $o(\log n)$ then Corollary 2.18 shows the existence of a spooky pebbling for parameters where there cannot be a reversible pebbling. Setting s to $\Theta(\log n)$ gives a spooky pebbling algorithm where the number of steps matches a known lower bound on what is possible with the same number of pebbles in the reversible pebble game.

Proposition 2.20 ((Král'ovič, 2001))**.** *Any reversible pebbling on the line of length n using the minimum number of pebbles requires $\Omega(n \log n)$ steps.*

The best known reversible pebbling using this many pebbles requires $O(n^{\log_2(3)})$ steps (Li and Vitanyi, 1996). We conjecture that the bound in Proposition 2.20 is not tight for reversible pebbling and that Corollary 2.18 gives a spooky pebbling algorithm that is more time efficient than the best possible reversible pebbling with this number of pebbles. Král'ovič gives another upper bound on reversible pebbling algorithms that use $O(n^{1/k})$ pebbles.

Proposition 2.21 ((Král'ovič, 2001))**.** *For any fixed k, there is a reversible pebbling algorithm for the line of length n that uses $O(n^{1/k})$ pebbles and $O(2^k n)$ steps.*

We are able to improve this result by an exponential factor in k within the spooky pebble game.

Corollary 2.22. *For any fixed k, there is a spooky pebbling algorithm for the line of length $n \geq (2k)^{2k}$ that uses $O(n^{1/k})$ pebbles and $O(kn)$ steps.*

Proof. Set $m = 2k$ and $s = \left\lceil n^{1/k} \right\rceil$. Then

$$\begin{aligned}
\binom{m+s-2}{s-2} + 1 &\geq \binom{2k + \left\lceil n^{1/k} \right\rceil - 2}{\lceil n^{1/k} \rceil - 2} \\
&\geq \left(\frac{2k + \left\lceil n^{1/k} \right\rceil - 2}{2k} \right)^{2k} \\
&\geq \frac{n^2}{(2k)^{2k}} \\
&\geq n
\end{aligned}$$

Therefore by Theorem 2.16 there exists a spooky pebbling on the line of length n using $O(n^{1/k})$ pebbles and $O(kn)$ steps. □

2.5 Lower bounds for spooky pebbling the line

To show that our construction in Section 2.4 is optimal for any pebble bound, we prove an asymptotically matching lower bound that applies to any sequential spooky pebbling algorithm on the line graph. We do this by first proving, for any pebble bound s, the existence of a time-optimal spooky pebbling strategy of a particular form. Then we use a tree based argument to prove that any such algorithm with s pebbles that spooky pebbles the line of length $n \geq \binom{m+s-2}{s-2} + 1$ requires $\Omega(mn)$ steps.

Theorem 2.23. *For any line of length n, any pebble bound s, and any positive integer m where $n \geq \binom{m+s-2}{s-2} + 1$, a spooky pebbling of that line with at most s pebbles uses at least $\Omega(mn)$ steps.*

2.5.1 The existence of well-structured optimal pebbling algorithms.

Let $\mathcal{P} = [\mathcal{P}_0, \ldots, \mathcal{P}_\tau]$ be an arbitrary spooky pebbling of the line with nodes $v_1, \ldots v_n$. We will show that $\mathcal{P}$ uses at least as many steps as a spooky pebbling algorithm matching $\mathcal{P}'$ in Lemma 2.24.

Lemma 2.24. *Let $\mathcal{P}$ be an arbitrary spooky pebbling of the line using $T(\mathcal{P}) = \tau$ steps and $S(\mathcal{P}) = s$ pebbles. Then there exists k, pebbling algorithm $\mathcal{P}^*$, and unpebbling algorithm $\mathcal{U}_p^*$ such that the pebbling algorithm $\mathcal{P}'$ described below has $T(\mathcal{P}') \leq \tau$ and $S(\mathcal{P}') \leq s$.*

$\mathcal{P}'(\{v_1, \ldots, v_n\})$:
    `place(`v_1`)`
    `for` $i \in [2, k]$:
        `place(`v_i`)`
        `remove(`v_{i-1}`)`
    `run` $\mathcal{P}^*(\{v_{k+1}, \ldots v_n\})$
    `run` $\mathcal{U}_p^*(\{v_1, \ldots, v_k\})$

In other words, $\mathcal{P}'$ is the same algorithm as in Theorem 2.16, but with the choice of k determined by the values of n and s in some arbitrary way. We prove Lemma 2.24 over the course of Section 2.5.1.

Lemma 2.25. $\forall i \in [2, n-1]$, *Last*$(v_i, \mathcal{P}) <$ *Last*$(v_{i-1}, \mathcal{P})$.

Proof. Let $t = \text{Last}(v_i, \mathcal{P})$, which implies that there is never a pebble placed on v_i after $\mathcal{P}_t$. By the definition of a spooky pebbling we require that $\mathcal{G}_\tau = \emptyset$, so $v_i \notin \mathcal{G}_{t+1}$ as we never place a pebble on v_i after step t. Since this is a valid pebbling, we can conclude that $v_{i-1} \in \mathcal{P}_t$ and thus $\text{Last}(v_i, \mathcal{P}) < \text{Last}(v_{i-1}, \mathcal{P})$. □

Definition 2.26. Let $\text{First}(v_n, \mathcal{P}) = t$. We call $\mathcal{P}$ *sweep-first* if a pebble is placed at v_n only once and each v_i has at most one pebble placed on it before step t.

Note that $\mathcal{I}$ in the proof of Lemma 2.10 is sweep first subroutine and only uses s pebbles when $v_{k-1} \in S$. Using this result, we can turn any pebbling into one that is sweep first.

Lemma 2.27. *Let $\mathcal{P}$ be any spooky pebbling algorithm where $T(\mathcal{P}) = \tau$ and $S(\mathcal{P}) = s$. Then there exists a sweep-first pebbling algorithm $\mathcal{P}'$ where $T(\mathcal{P}') \leq \tau$ and $S(\mathcal{P}') \leq s$.*

Proof. Let t be the last step where $\mathcal{P}$ places a pebble on v_n. We know that $t < \text{Last}(v_{n-1}, \mathcal{P})$. Thus, by Lemma 2.25 we know that $t < \text{Last}(v_i, \mathcal{P})$ for all $i < n$. $\mathcal{P}'$ will place pebbles on exactly $\mathcal{P}_t$ in $t' = 2n - |\mathcal{P}_t|$ steps by running $\mathcal{I}(\{v_1, \ldots v_n\}, \mathcal{P}_t)$. We know that $t' \leq t$ since each v_i has a pebble placed and each node not in $\mathcal{P}_t$ must have had a pebble removed before step t in $\mathcal{P}$. We define $\mathcal{P}' = [\mathcal{P}_0, \mathcal{P}'_1, \ldots, \mathcal{P}'_{t'} = \mathcal{P}_t, \mathcal{P}_{t+1}, \ldots, \mathcal{P}_\tau]$. Since $t' \leq t$ we know that $T(\mathcal{P}') \leq \tau$. We can additionally conclude that $S(\mathcal{P}') \leq s$ since $\mathcal{P}_t$ must contain a pebble at v_{n-1} and so $\mathcal{I}$ only uses s pebbles. Note that while $\mathcal{P}$ and $\mathcal{P}'$ may have different ghosting sequences, $\mathcal{P}'$ is still a valid spooky pebbling. This is because $\mathcal{P}$ must place a pebble (and subsequently remove it without creating a ghost) at every location where $\mathcal{P}'$ had a ghost during step t' due to Lemma 2.25. □

Now we are ready to prove Lemma 2.24.

Proof of Lemma 2.24. By Lemma 2.27 we can assume that $\mathcal{P}$ is sweep-first. Without loss of generality, let $\mathcal{P}$ be the algorithm with $T(\mathcal{P}) = \tau$ and $S(\mathcal{P}) = s$ that maximizes over all algorithms the least value of k where $v_k \in \mathcal{P}_{\text{First}(v_n, \mathcal{P})}$. Let $t = \text{First}(v_n, \mathcal{P})$. By our choice of $\mathcal{P}$ and k, for all $i < k$, $\mathcal{P}$ must place and remove a pebble at v_i; thus we can rearrange these steps from $\mathcal{P}$ to match lines 2 through 5 of $\mathcal{P}'$. Since $\mathcal{P}$ is a sweep-first pebbling we know that these are the only instructions acting on $v_1, \ldots, v_k$ before step t. We now partition the remaining instructions of $\mathcal{P}$ into $\mathcal{P}^*$, the set of all instructions acting on nodes $v_{k+1}, \ldots, v_n$, and $\mathcal{U}_p^*$, the set of all instructions acting on $v_1, \ldots, v_k$ — while maintaining their relative order within $\mathcal{P}$.

We will now show that $\mathcal{P}^*$ is a valid pebbling of $\{v_{k+1}, \ldots, v_n\}$ that uses at most $s - 1$ pebbles. Since (i) $\mathcal{P}$ is a valid pebbling, (ii) $\mathcal{P}'$ as constructed above maintains a pebble at v_k during the execution of $\mathcal{P}^*$, and (iii) instructions acting on vertices $v_1, \ldots, v_{k-1}$ cannot change the validity of an instruction on $v_{k+1}, \ldots, v_n$, it follows that $\mathcal{P}^*$ is a valid pebbling. Now assume for the sake of contradiction that $\mathcal{P}^*$

placed s pebbles during some step. Then since $\mathcal{P}$ uses only s pebbles, there must be some first step t' in $\mathcal{P}$ where $\{v_1, \ldots, v_k\} \cap \mathcal{P}_{t'} = \emptyset$ before the s'th pebble is placed on vertices $\{v_{k+1}, \ldots, v_n\}$. Let ℓ be the least value where $v_\ell \in \mathcal{P}_{t'}$. We will construct another sweep-first pebbling algorithm $\mathcal{P}^\perp$ that has $\ell > k$ as the least value where $v_\ell \in \mathcal{P}^\perp_{\text{First}(v_n, \mathcal{P}^\perp)}$ while maintaining $T(\mathcal{P}^\perp) \leq \tau$ and $S(\mathcal{P}^\perp) \leq s$, contradicting our choice of $\mathcal{P}$. Intuitively $\mathcal{P}^\perp$ will behave like $\mathcal{P}$ except that it has a pebble at v_ℓ instead of v_k during step t. $\mathcal{P}^\perp$ has $\mathcal{P}^\perp_{\text{First}(v_n, \mathcal{P}^\perp)} = (\mathcal{P}_t \cup \{v_\ell\}) \setminus \{v_k\}$ and reaches this state by following the first t instructions of $\mathcal{P}$, except that $\mathcal{P}^\perp$ ghosts v_k after a pebble is placed on v_{k+1} and ignores any instruction to remove v_ℓ before placing a pebble on v_n.

After this, $\mathcal{P}^\perp$ continues following the instructions of $\mathcal{P}$ on all vertices excluding $v_k, \ldots v_\ell$ until step t' of $\mathcal{P}$. At this point we know that $\mathcal{P}_{t'}$ has empty intersection with $v_k, \ldots, v_{\ell-1}$ and therefore $\mathcal{P}$ and $\mathcal{P}^\perp$ have pebbles in the same positions and ghosts on the same positions excluding $v_k, \ldots, v_{\ell-1}$. However, $\mathcal{P}_{t'}$ has a pebble at v_ℓ and therefore must place pebbles on all of $v_k, \ldots, v_{\ell-1}$. So if $\mathcal{P}^\perp$ copies all steps of $\mathcal{P}$ after step t', it will also remove any ghosts it had in $v_k, \ldots v_{\ell-1}$. Thus, $\mathcal{P}^\perp$ is a valid spooky pebbling that uses at most s pebbles and τ steps and violates our choice of $\mathcal{P}$ since $\ell > k$.

We know that $\mathcal{U}^*_p$ must be a valid unpebbling by the same reasoning as $\mathcal{P}^*$. All that remains is to show that $\mathcal{U}^*_p$ uses at most $s - 1$ pebbles. As $\mathcal{P}$ is a sweep-first algorithm that uses at most s pebbles, we know that for all steps $t' > t$, $|\mathcal{P}_{t'} \cap \{v_1, \ldots, v_k\}| \leq s - 1$. Thus executing the same instructions in $\mathcal{U}^*_p$ cannot use more than $s - 1$ pebbles and $\mathcal{P}'$ is a valid spooky pebbling that uses no more pebbles or steps than $\mathcal{P}$. □

By Lemma 2.24 we can assume that for any fixed pebble count, there is a minimum step algorithm that matches the structure of $\mathcal{P}'$. By recursively applying this argument, we get a recurrence relation for the minimum number of steps needed to pebble the line of length n using at most s pebbles. In general this recurrence has

a tree like structure based on the choice of k at each level. However, the value of k that minimizes the expression is the optimal choice.

Corollary 2.28. *Let $T_P(n,s)$ ($T_{UP}(n,s)$) be the minimum number of steps needed to pebble (unpebble) a line of length n using at most s pebbles. Then:*

$$T_P(n,s) = \begin{cases} 1 & s \geq 1 \text{ and } n = 1 \\ \infty & s < 3 \text{ and } s < n \\ \min_{k \in [1,n)} 2k - 1 + T_{UP}(k, s-1) + T_P(n-k, s-1) & o.w. \end{cases}$$

The ∞ comes from observing that there is no way to spooky pebble a line longer than s with s pebbles when $s < 3$. Unfortunately the above recurrence features a call to T_{UP} and the non-obvious choice of k makes it hard to analyze.

However, Theorem 2.9 gives us a way to relate the costs of various pebbling subroutines. To match the upper bounds we proved in Section 2.4, we can restate Corollary 2.28 in the language of unghosting.

Corollary 2.29. *Let $T_{UG}(n,s)$ ($T_{UP}(n,s)$) be the minimum number of steps needed to unghost (unpebble) a line of length n using at most s pebbles. Then:*

$$T_{UG}(n,s) = \begin{cases} 2 & s \geq 1 \text{ and } n = 1 \\ \infty & s < 2 \text{ and } s < n \\ \min_{k \in [1,n]} 2k - 1 + T_{UP}(k, s) + T_{UG}(n-k, s-1) & o.w. \end{cases}$$

Note that we cannot apply Theorem 2.9 to the base case in Corollary 2.28. Instead, the base case in the above recurrence can be verified by inspection. By Theorem 2.9 we know that $T_{UP}(k,s) \geq T_{UG}(k,s) - 1$. We therefore define the following recurrence relation that is a lower bound on $T_{UG}(n,s)$:

$$T(n,s) = \begin{cases} 2 & s \geq 1 \text{ and } n = 1 \\ \infty & s < 2 \text{ and } s < n \\ \min_{k \in [1,n]} 2k - 2 + T(k,s) + T(n-k, s-1) & \text{o.w.} \end{cases} \quad (2.1)$$

While $T(n, s)$ is not exactly the minimum number of steps needed to unghost the line of length n with s pebbles, it is a sufficiently tight lower bound. A function similar to our $T(n, s)$ appears in (Wheeler and Hughey, 2000; Newberg, 2008) as the solution to the backtracking problem in dynamic programming. They provide efficient algorithms that can be used to compute the optimal value of k in each recursive call. Another similar recurrence appears in the problem of sequence reversing, where there is a proven lower bound similar to the one we present here (Dumas, 1995; Grimm et al., 1996). In the rest of this section we use this recurrence relation to prove that unghosting with at most s pebbles requires $\Omega(mn)$ steps.

2.5.2 Analysis of T(n,s)

Theorem 2.30. *For any line of length n, any pebble bound s, and any positive integer m where $n \geq \binom{m+s-2}{s-2} + 1$, a spooky pebbling of that line with at most s pebbles uses at least $\Omega(mn)$ steps.*

One difficulty in analyzing our recurrence relation is that the choice of k that minimizes this expression is not obvious, making it difficult to directly compute this function. Instead, we will describe a family of related recurrence relations for a fixed choice of n and s using trees with n leaf nodes and right-depth $s - 1$ to determine the choices of k. The lowest valued recurrence in this family gives us the value of $T(n, s)$.

Definition 2.31. Let $\mathcal{T}$ be a binary tree, $\mathcal{T}.l$ be the left subtree, and $\mathcal{T}.r$ be the right subtree. Let $|\mathcal{T}|_L$ be the number of leaves in a given binary tree. If $|\mathcal{T}|_L = n$ then we can define the recurrence relation:

$$T_{\mathcal{T}}(n, s) = \begin{cases} 2 & s \geq 1 \text{ and } n = 1 \\ \infty & s < 2 \text{ and } s < n \\ 2\,|\mathcal{T}.l|_L - 2 + T_{\mathcal{T}.l}(|\mathcal{T}.l|_L, s) + T_{\mathcal{T}.r}(|\mathcal{T}.r|_L, s-1) & \text{o.w.} \end{cases} \tag{2.2}$$

We can think of this as being Equation (2.1) except that k is selected as the number of leaves in the left sub-tree rather than the value that minimizes the

expression. This lets us fix n and s in order to see how different choices of k lead to different values for the recurrence. Then the minimum of $T_{\mathcal{T}}(n, s)$ over all trees must equal to $T(n, s)$. Instead of directly computing the recurrence, we can compute $T_{\mathcal{T}}(n, s)$ for a specific tree $\mathcal{T}$ by assigning a cost function to binary trees that is equal to $T_{\mathcal{T}}(n, s)$.

Definition 2.32. Let v be any node in a tree. We let $R(v)$ denote the right-depth of v, which is equal to the number of times you must take right children to reach v from the root. We likewise define $L(v)$ as the left-depth of v.

Definition 2.33. For any depth bound s, the *cost* of a binary tree $C_s(\mathcal{T})$ is equal to the sum of the costs of its nodes. Each non-leaf node of a tree has cost -2. Any leaf node v where $R(v) < s$ has a cost of $2L(v) + 2$ while a leaf node where $R(v) \geq s$ has a cost of ∞.

Lemma 2.34. *The cost of the tree $C_s(\mathcal{T})$ is equal to $T_{\mathcal{T}}(|\mathcal{T}|_L, s)$.*

Proof. We prove this by structural induction on the tree. In the base case we have a tree that is a single leaf. Then $C_s(\mathcal{T}) = T_{\mathcal{T}}(n, s) = 2$ as desired. Now consider an arbitrary tree $\mathcal{T}$. By the inductive hypothesis we can assume that $C_s(\mathcal{T}.l) = T_{\mathcal{T}.l}(|\mathcal{T}.l|_L, s)$ and $C_{s-1}(\mathcal{T}.r) = T_{\mathcal{T}.r}(|\mathcal{T}.r|_L, s-1)$. Given that the cost of a tree is the sum of the costs of its nodes, $C_s(\mathcal{T}) = 2\,|\mathcal{T}.l|_L - 2 + C_s(\mathcal{T}.l) + C_{s-1}(\mathcal{T}.r)$, as all nodes in $\mathcal{T}.l$ have their left depth increased by one, all nodes in $\mathcal{T}.r$ has their right depth increased by one, and the root of $\mathcal{T}$ has a cost of -2. By our inductive hypothesis, this implies that

$$C_s(\mathcal{T}) = 2\,|\mathcal{T}.l|_L - 2 + T_{\mathcal{T}.l}(|\mathcal{T}.l|_L, s) + T_{\mathcal{T}.r}(|\mathcal{T}.r|_L, s-1) = T_{\mathcal{T}}(|\mathcal{T}|_L, s).$$

If we have a tree with a leaf node where $R(v) \geq s$ then the tree has a cost of infinity and $T_{\mathcal{T}}(n, s)$ must reach the base case where $s \leq 1$ and $s < n$ when evaluating that node. □

Corollary 2.35. *Let $\mathcal{T}_{(n,s)}$ be the tree with n leaf nodes that minimizes $C_s(\mathcal{T}_{(n,s)})$. Then $T(n, s) = C_s(\mathcal{T}_{(n,s)})$.*

It is easier to reason about $C_s(\mathcal{T}_{(n,s)})$ than $T(n,s)$ as we can start with an arbitrary tree with n leaf nodes and show how it would be possible to mutate that tree and reduce its cost without computing the full value of the recurrence relation.

Now that we have established the equivalence between the cost of a tree $\mathcal{T}_{(n,s)}$ and $T(n,s)$, we want to prove some things about the structure of this tree. When $s < 2$ and $s < n$ we know that $T(n,s) = \infty$, so whenever possible a tree will never have a sub-tree with such parameters as nodes. This means that any tree $\mathcal{T}_{(n,2)}$ must have $\mathcal{T}_{(1,1)}$ as its right child. By construction, this means that the right-depth of any node in the tree $\mathcal{T}_{(n,s)}$ is at most $s-1$. Note that the cost of all the intermediate nodes is only a function of the total number of nodes in the tree.

What follows is a collection of technical lemmas that give the number of nodes in $\mathcal{T}_{(n,s)}$ with specific left and right depths. Together they let us characterize the number of leaves in $\mathcal{T}_{(n,s)}$ with each cost. This lets us construct a lower bound on the cost of $\mathcal{T}_{(n,s)}$, which in turn is a lower bound on the total number of steps needed to unghost a line of length n using at most s pebbles.

Lemma 2.36. *Let $\mathcal{T}_{(n,s)}$ be the tree in Corollary 2.35. Let x be a leaf of $\mathcal{T}_{(n,s)}$ with the largest value for $L(x)$, $m = L(x)$, and x' be any other leaf where $L(x') < m-1$. Then $R(x') = s-1$.*

Proof. Assume that $R(x') < s-1$. Note that x must be a left child, as it is the node with the largest left depth. Then we could replace the parent of x with its other child and replace x' with an intermediate node whose left child is x and right child is x'. Doing so must reduce the left depth of x by at least one and does not change the left depth of any other node in the tree, so the new tree has a lower cost. But this is a contradiction since $\mathcal{T}_{(n,s)}$ is supposed to be the tree with the lowest cost. □

The above lemma tells us a surprising amount about the structure of $\mathcal{T}_{(n,s)}$. Since all nodes x where $L(x) < m-1$ and $R(x) < s-1$ cannot be leaves, these nodes

all have left and right children. We can therefore derive exactly how many leaves $\mathcal{T}_{(n,s)}$ must have with each left depth less than m.

Lemma 2.37. *Let $\mathcal{T}_{(n,s)}$ be the tree in Corollary 2.35 and m be the largest value of $L(v)$ for $v \in \mathcal{T}_{(n,s)}$. If $m > 1$, then we know that the number of leaf nodes $\mathcal{T}_{(n,s)}$ possesses with left depth $L < m$ is $\binom{L+s-2}{s-2}$.*

Proof. By Lemma 2.36 we know that no node with a left depth less than $m-1$ and a right depth less than $s-1$ can be a leaf. Thus all such nodes must have left and right children. We will identify each node with a string of l's and r's indicating which directions are taken from the root of the tree to arrive at that node. Since all nodes with right depth s or larger have cost infinity, we know that all nodes with right depth $s-1$ must be leaf nodes. Thus, each leaf node with left depth less than $m-1$ must be a right child (otherwise, their sibling would have right depth s) and be identified with a string ending in an r. There are $\binom{L+s-2}{s-2}$ strings containing L copies of l and $s-1$ copies of r that end in an r. This exactly corresponds to the number of leaf nodes with left depth $L < m-1$; each of these leaf nodes must exist in the tree because none of their ancestors can be leaves by Lemma 2.36, and there can be no other leaves with this left depth unless there is a leaf with cost ∞.

Since, by Lemma 2.36, all nodes with left depth $L = m-2$ and right depth $R < s-1$ must exist in the tree and cannot be leaves, each of these nodes has a left child. There is a bijection between the number of leaves with left depth $L = m-1$ and these children, as their rightmost descendants are exactly the leaf nodes with left depth $m-1$. Since there are $\binom{m+R-2}{R}$ nodes with left depth $m-2$ and right depth $R < s-1$, there are

$$\sum_{R=0}^{s-2} \binom{m+R-2}{R} = \sum_{R=0}^{s-2} \binom{m+R-2}{m-2} = \binom{m+s-3}{m-1} = \binom{L+s-2}{s-2}$$

leaf nodes with left depth $m-1$. □

Lemma 2.38. *Let $\mathcal{T}_{(n,s)}$ be the tree in Corollary 2.35 and m be the largest left depth of any node in $\mathcal{T}_{(n,s)}$ and m' be the largest left-depth of any node in $\mathcal{T}_{(n',s)}$. If $m' > m$ then $n' > n$.*

Proof. Assume that $n' \leq n$. By Lemma 2.37 for all $L < m$, $\mathcal{T}_{(n,s)}$ and $\mathcal{T}_{(n',s)}$ must have the same number of leaf nodes with left depth L. This means that $\mathcal{T}_{(n,s)}$ must have at least $n - n'$ more leaves with left depth m than $\mathcal{T}_{(n',s)}$ has with left depth at least m. We construct a new tree $\mathcal{T}^*_{(n',s)}$ by taking $\mathcal{T}_{(n,s)}$ and removing $n - n'$ leaves with left depth m by replacing their parent with its right child. The tree $\mathcal{T}^*_{(n',s)}$ represents a valid binary tree and its cost is less than that of $\mathcal{T}_{(n',s)}$ since the two trees have the same total number of nodes, the same number of leaves with any left depth less than m, and $\mathcal{T}^*_{(n',s)}$ has no leaves with left depth larger than m. This implies that $\mathcal{T}_{(n',s)}$ was not constructed according to Corollary 2.35, giving us a contradiction. □

Lemma 2.39. *Let $\mathcal{T}_{(n,s)}$ be the tree in Corollary 2.35 and n be the smallest value such that $\mathcal{T}_{(n,s)}$ contains a node with left depth $m + 1$. Then $\mathcal{T}_{(n-1,s)}$ contains exactly $\binom{m+s-2}{s-2}$ leaves with left depth m and $n = 1 + \binom{m+s-1}{s-1}$.*

Proof. Assume for the sake of contradiction that $\mathcal{T}_{(n,s)}$ contained at least two nodes with left depth $m + 1$. By our choice of n we know that $\mathcal{T}_{(n-1,s)}$ contains no nodes with left depth $m + 1$. By Lemma 2.37 we know that $\mathcal{T}_{(n,s)}$ and $\mathcal{T}_{(n-1,s)}$ have the same number of leaves with each left depth less than m. This means the difference in cost between $\mathcal{T}_{(n,s)}$ and $\mathcal{T}_{(n-1,s)}$ must be at least m.[2] But there is another tree $\mathcal{T}^*_{(n,s)}$ that costs only $m - 1$ more than $\mathcal{T}_{(n-1,s)}$ constructed by replacing the leftmost leaf of $\mathcal{T}_{(n-1,s)}$ with an intermediate node whose children are both leaves. Since $\mathcal{T}^*_{(n,s)}$ has cost less than $\mathcal{T}_{(n,s)}$, we get a contradiction.

[2] $\mathcal{T}_{(n,s)}$ must have two leaves with left depth $m + 1$ while $\mathcal{T}_{(n-1,s)}$ has one fewer node that must have left depth m. Since $\mathcal{T}_{(n,s)}$ has one more intermediate node, its cost is at least m more than $\mathcal{T}_{(n-1,s)}$.

Now $\mathcal{T}_{(n,s)}$ must contain exactly one node with left depth $m+1$. By Lemma 2.37 this means that it contains exactly $\binom{L+s-2}{s-2}$ leaves with left depth L, so the total number of leaves in $\mathcal{T}_{(n,s)}$ is:

$$\begin{aligned} n &= 1 + \sum_{L=0}^{m} \binom{L+s-2}{s-2} \\ &= 1 + \binom{m+s-1}{s-1} \end{aligned}$$

This means that $\mathcal{T}_{(n-1,s)}$ must have $\binom{m+s-1}{s-1}$ leaves. All of these leaves have left depth at most m by Lemma 2.38. So Lemma 2.37 gives us that $\mathcal{T}_{(n-1,s)}$ has:

$$\begin{aligned} &\binom{m+s-1}{s-1} - \left(\sum_{L=0}^{m-1} \binom{L+s-2}{s-2}\right) \\ &= \binom{m+s-1}{s-1} - \binom{m+s-2}{s-1} \\ &= \binom{m+s-2}{s-2} \end{aligned}$$

leaves with left depth m. □

Lemma 2.40. *Let $\mathcal{T}_{(n,s)}$ be the tree in Corollary 2.35 and $\binom{m+s-1}{s-1} \leq n$. Then the cost of $\mathcal{T}_{(n,s)}$ is $\Omega(mn)$.*

Proof. By Lemma 2.37, Lemma 2.38, and Lemma 2.39, we know that for all $L \in \{0, 1, 2, \ldots, m\}$ the number of leaf nodes with left depth L is $\binom{L+s-2}{s-2}$. Thus, the cost of these leaf nodes is at least:

$$\sum_{L=0}^{m} (2L+2) \binom{L+s-2}{s-2}.$$

Expanding the above summation yields:

$$\frac{2(m+1)(m(s-1)+s)}{s(s-1)} \binom{m+s-1}{s-2}.$$

Using that $\binom{n}{k-1} = \frac{k}{n-k+1}\binom{n}{k}$ and $s \geq 2$, this is at least:

$$\frac{2(m(s-1)+s)}{s} \binom{m+s-1}{s-1} = 2\left(\frac{m(s-1)}{s} + 1\right)\binom{m+s-1}{s-1}$$

$$\geq (m+2)\binom{m+s-1}{s-1}.$$

By Lemma 2.37 and Lemma 2.39, if $n = \binom{m+s-1}{s-1} + k$ then the remaining k leaves must all have left depth larger than m. Thus, the total cost of the leaves is at least:

$$(m+2)\binom{m+s-1}{s-1} + 2(m+1)k \geq (m+2)n.$$

There are $n-1$ intermediate nodes with cost -2 so the total cost of the tree must be at least:

$$(m+2)n - 2(n-1) = mn - 2.$$

Which is $\Omega(mn)$ as desired. □

By applying Lemma 2.40 to lower bound the cost of trees and Lemma 2.34 to apply this lower bound to Equation (2.1), we obtain the following corollary:

Corollary 2.41. *The number of steps needed to unghost a line of length* $n \geq \binom{m+s-1}{s-1}$ *with* s *pebbles is* $\Omega(mn)$.

By applying Theorem 2.9 to Corollary 2.41, we get Theorem 2.30.

2.6 Spooky pebbling beyond the line graph

In Section 2.4 and Section 2.5 we discussed how the spooky pebble game can be applied on a line graph, which simulates unstructured irreversible sequential computation on a quantum computer. While this gives us very general results, additional structure on the classical computation can lead to more efficient simulation. Here we consider computation that can be expressed using a dependency graph — such as Boolean circuits, straight line (or oblivious) programs, and data-independent memory hard functions (e.g. (Paul and Tarjan, 1978; Pippenger, 1978; Lengauer and Tarjan, 1979; Tompa, 1980; Alwen and Serbinenko, 2015; Ren and Devadas, 2016; Alwen et al., 2017a)). When considering a pebbling of a DAG, each pebble represents only a single word rather than an entire copy of the computation's state.

2.6.1 Hardness of approximation

Despite having a tight characterization of optimal pebbling strategies on the line graph, finding optimal pebbling strategies is generally intractable. In fact, we will show that even approximating the number of pebbles needed in the spooky pebble game is PSPACE-hard.

There is a strong connection between the number of pebbles needed in the irreversible and spooky pebble games. A similar observation was independently made in (Quist and Laarman, 2023, 2024).

Lemma 2.42. *If a DAG $\mathcal{G}$ with n nodes and one target can be pebbled in the irreversible pebble game using τ steps and s pebbles, then it can be spooky pebbled using $O(n\tau)$ steps and at most $s+1$ pebbles.*

Proof. Let $\mathcal{P}$ be an irreversible pebbling of $\mathcal{G}$ that uses s pebbles. We will construct the following spooky pebbling $\mathcal{P}_{\text{spook}}$ that uses at most $s+1$ pebbles. $\mathcal{P}_{\text{spook}}$ first simulates $\mathcal{P}$, which since we are operating in the spooky pebble game, may leave some ghosts behind. Additionally, the first time that $\mathcal{P}_{\text{spook}}$ places a pebble at any node v_i, we push v_i onto an initially empty stack. Once $\mathcal{P}_{\text{spook}}$ has placed a pebble at the target node v_t, it removes all other pebbles (leaving many ghosts) and repeats the following procedure until the stack is empty: We pop the top element off of our stack and call it v_g. If v_g does not contain a ghost, we continue to the next element in the stack. Once we find a v_g that contains a ghost, we have $\mathcal{P}_{\text{spook}}$ simulate $\mathcal{P}$ until v_g is pebbled again, leaving ghosts on some nodes. $\mathcal{P}_{\text{spook}}$ then removes the pebble at v_g (without creating a ghost) and then removes all pebbles other than v_t, possibly addling ghosts. By our choice of v_g, future iterations of this procedure cannot result in a ghost on a prior v_g, so this procedure terminates in less than n iterations with pebbles only on v_t and no ghosts. Since $\mathcal{P}_{\text{spook}}$ is replaying steps from $\mathcal{P}$ with at most one additional pebble on v_t, it will use at most $s+1$ pebbles. □

Since any spooky pebbling is also an irreversible pebbling, the above lemma is sufficient to give us a nearly tight two-sided bound on the minimum number of pebbles needed in the spooky pebble game.

Corollary 2.43. *Let $\mathcal{G}$ be a DAG that requires s pebbles in the irreversible pebble game. The fewest pebbles needed for the spooky pebble game on $\mathcal{G}$ is either s or $s+1$.*

In (Demaine and Liu, 2017a) the authors already showed that even approximating the number of pebbles in the irreversible pebble game is PSPACE-hard. Demaine and Liu proved that determining the fewest pebbles needed in the irreversible pebble game is PSPACE-hard to approximate.

Proposition 2.44 (Theorem 1[3] in (Demaine and Liu, 2017a))**.** *The minimum number of pebbles needed in the irreversible pebble game on DAGs with n nodes, a single target, and maximum in-degree 2 is PSPACE-hard to approximate to within an additive $n^{1/3-\varepsilon}$ for any $\varepsilon > 0$.*

By combining Proposition 2.44 with Corollary 2.43, we see that finding even approximate minimum pebble spooky pebbling algorithms for arbitrary DAGs solves a PSPACE-hard problem. Thus unless $P = \text{PSPACE}$, there can be no polynomial time algorithm that generates minimum pebble spooky pebbling strategies on arbitrary DAGs. Quist and Laarman independently[4] showed the PSPACE-completeness of this problem in (Quist and Laarman, 2024) through a different reduction; however, going through Proposition 2.44 extends this result to a restricted subset of DAGs (those with maximum in-degree two with a single sink node) and shows that even approximating the minimum number of pebbles to an additive factor of $n^{1/3-\varepsilon} + 1$ is PSPACE-hard.

[3]Although Theorem 1 as stated in (Demaine and Liu, 2017a) does not refer to the number of targets, the full construction presented in section 2.3.2 of (Demaine and Liu, 2017b) is a DAG with a single target node.

[4]We had written down a proof of this result 2022; however, we did not release the result publicly until shortly after (Quist and Laarman, 2024) was posted to arXiv.

Theorem 2.45. *The minimum number of pebbles needed in the spooky pebble game on DAGs with n nodes, a single sink, and maximum in-degree 2 is PSPACE-hard to approximate to within an additive $n^{1/3-\varepsilon} + 1$ for any $\varepsilon > 0$.*

2.6.2 Efficient pebbling of the tree

Even though approximating the number of required pebbles is hard in general, we can still hope to find efficient pebbling algorithms for specific DAGs. The binary tree with directed edges from the leaves to the root is a natural topology for algorithms like divide and conquer, where a problem is solved by combining the results of two smaller sub-problems. Concrete algorithms with this structure include floating point pairwise summation and computing Merkle trees (McCracken and Dorn, 1964; Merkle, 1988).

It is known that $h+1$ pebbles and $\Theta(n)$ steps are both necessary and sufficient to pebble a binary tree of height h with $n = 2^h - 1$ nodes in the irreversible pebble game (Cook, 1973). In (Král'ovič, 2001), Král'ovič showed that $(h+\Theta(\log^* h))$ pebbles are necessary and sufficient this task in the reversible pebble game, where $\log^*$ is the iterated logarithm function. Despite this there are no known polynomial time pebbling strategies for the binary tree using the optimal number of pebbles; the best known tradeoff is a strategy that uses $(1+\varepsilon)h$ pebbles and runs in $n^{O(\log 1/\varepsilon)}$ steps (Komarath et al., 2015). We show the existence of a spooky pebbling strategy for the binary tree that uses the same number of pebbles as the most efficient irreversible pebbling while only being an $O(\log n)$ factor slower. Thus on binary trees, the spooky pebble game is almost as efficient as the irreversible pebble game and provably more efficient than the reversible pebble game.

Theorem 2.46. *There exists a spooky pebbling algorithm on the complete binary tree with $n = 2^h - 1$ nodes that uses $h + 1$ pebbles and $O(n \log n)$ steps.*

Proof. Before getting to the algorithm, we present the following subroutine:

```
fast_pebble(𝒯, h): //𝒯 is the root and h is the height
    if(h = 1):
        place(𝒯)
    else:
        run fast_pebble(𝒯.l, h − 1)
        run fast_pebble(𝒯.r, h − 1)
        place(𝒯)
        remove(𝒯.l)
        remove(𝒯.r)
```

The above subroutine uses at most $h+1$ pebbles and $2^{h+1}-1$ steps to place a pebble at the root of the tree while leaving ghosts at all other internal nodes. We will now use fast_pebble to construct a recursive unpebbling algorithm for a complete binary tree with root $\mathcal{T}$ and depth h:

$\mathcal{U}_p(\mathcal{T}, h)$:

```
    remove(𝒯)
    run fast_pebble(𝒯.l, h − 1)
    run fast_pebble(𝒯.r, h − 1)
    place(𝒯)
    remove(𝒯)
    remove(𝒯.r)
    run 𝒰(𝒯.l, h − 1)
    run 𝒰(𝒯.r, h − 1)
```

We note that $\mathcal{U}_p(\mathcal{T}, h)$ uses at most $h+1$ pebbles; we do not need to remove $\mathcal{T}.l$ before step 8 as the pebble on this node is removed at the beginning of the recursive call. The number of steps this algorithm takes follows the recurrence relation:

$$T(h) \leq 2T(h-1) + 2^{h+2} + 2$$

Unrolling the recurrence gives us that $T(h)$ is $O(h2^h)$, which in turn is $O(n \log n)$. We then note that $\mathcal{U}_p$ can be converted into a pebbling algorithm $\mathcal{P}$ by removing step 6. Doing this does not change the number of pebbles used and reduces the total number of steps by one, so $\mathcal{P}$ also uses at most $h+1$ pebbles and only $O(n \log n)$ steps. □

Chapter 3: Quantum time-space tradeoffs for matrix problems

3.1 Introduction

Matrix computations are among the most fundamental computational problems and are critically important in areas such as numerical and scientific computing, optimization, and machine learning. If quantum computers can be shown to have a significant advantage over classical computations for these types of problems then it would open up a wide range of applications for such devices.

Prior work has shown that non-standard versions of matrix problems may indeed admit exponential or large polynomial quantum advantage: For any efficiently implementable operator M, the HHL algorithm of Harrow, Hassidim, and Lloyd (Harrow et al., 2009) (with the improvements of (Childs et al., 2017)) can efficiently ε-approximate the value of $x^\dagger Mx$ for the solution x of a well-conditioned linear system. However, it is important to note that this algorithm requires the input to be presented in an unconventional format.

Many extensions of the HHL algorithm have also been proposed that can be elegantly described in the quantum singular value transform (qSVT) framework first described in (Low and Chuang, 2019) and popularized by (Gilyén et al., 2019). Despite initial hope of exponential speed-up, a series of papers by Tang and co-authors, and others (e.g. (Tang, 2019; Chia et al., 2020a,b; Gilyén et al., 2022; Bakshi and Tang, 2024; Chepurko et al., 2022)) has shown that, by providing classical algorithms a comparable input format to the HHL algorithm, these quantum algorithms can be replaced by classical ones with only a polynomial blowup in the running time, although this polynomial is not always small.

This body of work still begs the question: What is the conventional quantum complexity of standard classical problems like explicitly computing linear-system

solutions, multiplying or inverting matrices, computing matrix-vector products, and computing the low rank approximation of a matrix?

By the polynomial method, we know that computing a single inner product (or parity) of n-bit vectors requires $\Omega(n)$ quantum queries (Beals et al., 2001), but linear algebra computations generally involve $\Omega(n)$ or $\Omega(n^2)$ such computations. Sherstov (Sherstov, 2012), generalizing results of Klauck, Špalek, and de Wolf (Klauck et al., 2007) for the OR function, gave a strong direct product lower bound for quantum query complexity proved using the polynomial method, which yields strong lower bounds for inner products involving many *disjoint* input vectors. However, the matrix problems in linear algebra are very far from direct product problems: The vectors involved are highly correlated with each other, so this prior work does not shed light on the key question of whether quantum algorithms provide any advantage for general linear algebra.

In this chapter, we resolve these questions for quantum computation of a wide array of linear algebra problems, proving lower bounds for quantum computation that are asymptotically the same as the best classical lower bounds. Since many of the problems also have deterministic algorithms whose resource usage matches the lower bounds, our results show that there is provably no asymptotic quantum advantage at all in solving these linear algebra problems!

As with the study of classical computation involving super-linear time lower bounds, we consider quantum algorithms in which we limit the number of qubits of memory and hence produce quantum time-space tradeoffs. That is, for each fixed bound on the amount of memory allowed, we derive asymptotically the same time lower bound for the quantum algorithm as one would get for the time lower bound on classical algorithms with the same number of classical bits. In many ways, quantum memory is an even more critical resource than classical memory since it is a measure of the maximum number of qubits that maintain coherence at any time during the algorithm's execution. For this reason the first general-purpose fault-tolerant quantum

computers will likely have very limited memory and only be able to execute low depth quantum circuits. As such, it is crucial to consider both the time and space complexity for quantum algorithms.

We prove our lower bounds for quantum computation in a query model where algorithms are able to perform arbitrary input-independent unitary transformations on their state between quantum queries to their input. This is a sufficiently general model that our lower bounds also apply to any reasonable model of quantum computation—including quantum circuits where the (classical) input is stored in quantum-readable read only memory (QROM).

The keys to proving our time-space tradeoffs are new results proving much stronger lower bounds than strong direct product theorems for matrix-vector products and matrix multiplication. While our bounds have the same form as strong direct product theorems (the success probability decays exponentially with the number of outputs), they also apply with almost completely overlapping sets of inputs, in contrast to the disjoint inputs that are necessary to apply direct product theorems.

While there is a large body of work proving strong classical time-space tradeoffs (e.g. (Tompa, 1978; Borodin et al., 1979; Yesha, 1984; Borodin and Cook, 1982; Abrahamson, 1990, 1991; Mansour et al., 1993)) and a large body of work analyzing unrestricted quantum query algorithms versus their classical randomized counterparts (e.g. (Deutsch and Jozsa, 1992; Bernstein and Vazirani, 1997; Simon, 1997; Beals et al., 2001; Ambainis, 2002; Spalek and Szegedy, 2006; Spalek, 2008; Sherstov, 2012)), there are just a few previous papers that analyze the quantum memory required to make use of these quantum queries. Klauck, Špalek, and de Wolf (Klauck et al., 2007) extended the classical method of Borodin and Cook (Borodin and Cook, 1982) for proving time-space tradeoffs to quantum circuits using a new strong direct product theorem for quantum query algorithms computing the OR function. They showed that algorithms making T quantum queries and using S qubits of quantum memory require $T = \Theta(n^{1.5}/S^{1/2})$ to sort lists of length n, and require $T = \Omega(n^{2.5}/S^{1/2})$ to compute

$n \times n$ Boolean matrix product. Ambainis, Špalek, and de Wolf (Ambainis et al., 2009) extended this direct product approach to 2-sided error algorithms computing k-threshold functions which allowed them to produce similar trade-off lower bounds for systems of linear inequalities/equalities (though these have the drawback, unlike the other results, that the hard function for space S depends on the space bound). This approach, based on an extension of the adversary method using eigenspace analysis, was very difficult to apply.

As a result, further study of quantum time-space tradeoff lower bounds languished until it was enabled by an idea of Zhandry (Zhandry, 2019) who, motivated by understanding quantum algorithms interacting with random function oracles, developed an approach to understanding quantum query algorithms using a *compressed oracle* and Fourier analysis. This views computations in a *recording query* basis that allow one to keep track of a quantum query algorithm as a superposition of basis states that have a natural classical query interpretation. It has been applied to finding multi-way collisions (Liu and Zhandry, 2019) and to inverting a random permutation (Rosmanis, 2022). This greatly simplifies the analysis of quantum query algorithms and can be applied to many lower bound methods that use randomly chosen inputs rather than being limited to cryptographic applications.

Extending Zhandry's approach, Hamoudi and Magniez (Hamoudi and Magniez, 2021) applied an even cleaner expression of the method, using phase oracles with the recording query basis rather than Fourier analysis, and extended it using biased random inputs to derive query lower bounds in a regime of exponentially small success probability. They used this to obtain time-space tradeoff lower bounds, proving that any quantum algorithm that finds K disjoint collisions in an input of length n with T quantum queries and S qubits of memory must have $T = \Omega(KN^{1/3}/S^{1/3})$. They also re-derived the earlier sorting lower bound using this method.

Our linear algebra lower bounds and methods Time-space trade-off lower bounds for linear algebraic problems were among the first to be studied for classical

computation (Yesha, 1984) after the first bounds for sorting. The strongest classical results are due to Abrahamson (Abrahamson, 1991) who developed a powerful general method based on matrix rigidity. This yields state-of-the-art lower bounds for computation of Fourier transforms, convolution, matrix-vector products, matrix multiplication, matrix inversion, matrix powering, and linear system solving. The lack of any analogous results for quantum computation has been a substantial gap in our understanding [1].

Our results show that all the linear algebraic time-space tradeoff lower bounds shown by Abrahamson (Abrahamson, 1991) also apply to quantum computation even when the quantum circuit can adaptively decide when to produce output based on the observed input. Since many of these classical lower bounds are tight, our results directly imply that there is no hybrid classical-quantum algorithms with a polynomial advantage for these problems unlike the query bounds for search and collision finding in (Hamoudi et al., 2024). We include a table of our time-space tradeoff lower bounds in Table 3.1.

As discussed already, we need a much stronger lower bound method than any derivable from strong direct product theorems. We do this by the adding new ideas to the compressed oracle/recording query approach of Zhandry (Zhandry, 2019) as extended and applied by Magniez and Hamoudi (Hamoudi and Magniez, 2021). Thus far, the compressed oracle method has used a two-step pattern: First, identify a notion of unusual progress of a quantum algorithm towards a solution (i.e., the partial information so far is more determinative of the answer than one might expect) and show that the total amplitude of states where this occurs is small, Second, show that the total amplitude of the quantum states where many outputs are produced without unusual progress can be bounded; this latter part has used ideas with classical analogs

[1] Over a field of $> n$ elements one can reduce $n \times n$ Boolean matrix multiplication to ordinary multiplication of 0-1 matrices but the lower bound is inherently too weak because in the Boolean case each output bit is a disjointness function of its inputs and hence can be computed using only $O(\sqrt{n})$ quantum queries using Grover's algorithm ((Grover, 1996)).

Problem	Lower Bound	Source
Matrix Multiplication $f(A,B) = AB$	$T = \Theta(n^3\sqrt{\log d\ /S})$	Theorem 3.36
Matrix Squaring $f(A) = A^2$	$T = \Theta(n^3\sqrt{\log d\ /S})$	Corollary 3.40
Matrix Triple Product $f(A,B,C) = ABC$	$T = \Theta(n^4 \log d\ /S)$	Corollary 3.32
Matrix Cubing $f(A) = A^3$	$T = \Theta(n^4 \log d\ /S)$	Corollary 3.33
Matrix Inversion $f(A) = A^{-1}$	$T = \Omega(n^4 \log d\ /S)$	Corollary 3.34
System of Linear Eqns $f(A,y) = A^{-1}y$	$T = \Omega(n^3 \log d\ /S)$	Corollary 3.35
Matrix-Vector Product $f(x) = Ax$	$T = \Theta(n^2 \log d\ /S)$	Theorem 3.21
Discrete Fourier Transform $f(x) = Wx$	$T = \Theta(n^2 \log d\ /S)$	Corollary 3.26
Convolution $f(u,v) = u * v$	$T = \Theta(n^2 \log d\ /S)$	Corollary 3.28
Binary Integer Multiplication	$T = \Omega(n^2/(S \log^2 n))$	Corollary 3.29
Bool. Matrix Mult. $f(A,B) = A \bullet B$	$T = \Omega(n^{2.5}/S^{0.5})$	(Klauck et al., 2007)
	$T = \Omega(n^{2.5}/S^{0.25})$	Theorem 3.45
Classical	$T = \Omega(n^3/S)$	(Abrahamson, 1990)
Classical	$T = \Omega(n^{3.5}/S)$ for $S \geq cn$	(Abrahamson, 1990)
Classical	$T = \Theta(n^3/S^{0.5})$	Theorem 3.54
Boolean Matrix Squaring $f(A) = A \bullet A$	$T = \Omega(n^{2.5}/S^{0.25})$	Corollary 3.55

Table 3.1: Summary of our quantum lower bounds, along with prior work. Inputs are assumed to be of length n vectors or $n \times n$ matrices. Our linear algebra bounds apply for input elements coming from any fixed subset D of a field with $d = |D|$. These are the first quantum time-space lower bounds for all of these problems other than Boolean matrix multiplication. Problems with deterministic classical query algorithms given in (JáJá and Simon, 1982) and (Abrahamson, 1991) that match our quantum query lower bounds are denoted with Θ notation instead of Ω. Constructions of the matching query algorithms can be found in Section 3.7.

that can be applied by breaking the algorithm's final state into mutually orthogonal components, each with small amplitude on the correct answers.

However, in our case with linear algebra problems, there is no form of unusual progress and also no clear way to break up the problem into mutually orthogonal basis states. Thus, neither part of the pattern seems to work. Instead, we can use the recording query framework to characterize how much a quantum circuit can know about its input. We use the triangle inequality to bucket amplitude from the algorithm's state into a small number of non-orthogonal components (or buckets) that share some set of inputs that they know nothing about. We can then apply a classical argument showing that each component must have small amplitude on the correct answers. By finding a way to divide the state into a small number of buckets that each have small amplitude on correct answers, we can obtain tight lower bounds. The properties required of this division become more subtle as we move to the problem of matrix multiplication, where in order to get small amplitude, we need to contend with a partition featuring significantly more parts.

Improved bounds for Boolean matrix operations Here we improve the previous lower bound for quantum algorithms computing Boolean matrix multiplication given in (Klauck et al., 2007) from $T = \Omega(n^{2.5}/S^{1/2})$ to $T = \Omega(n^{2.5}/S^{1/4})$. We do this using a more sophisticated embedding of the k-fold direct product of OR functions into an arbitrary subset of k outputs of Boolean matrix multiplication. The embedding hinges on the number of colors needed for a certain kind of partial coloring of subsets E of the $n \times n$ grid. The exponents of n and S in our lower bound are optimal for the general quantum circuit model to which it applies.

Our lower bounds also lead to improving the classical lower bound tradeoff of $T = \Omega(n^3/S)$ for circuits shown in (Klauck et al., 2007) to $T = \Omega(n^3/S^{1/2})$. (In these bounds, T is circuit depth and S is circuit width.) Just as with our quantum lower bound, this has optimal exponents for n and S, achieving the goal of Klauck, Špalek,

and de Wolf (Klauck et al., 2007) who suggested that $T^2S = \Omega(n^6)$ was a likely tight tradeoff for classical computation of Boolean matrix multiplication. It is strictly larger almost everywhere than a classical lower bound of $T = \Omega(n^3/S)$ for $S \leq n^{0.5}$ and $T = \Omega(n^{3.5}/S)$ for $S \geq n$ for Boolean matrix multiplication on branching programs (a more general model than circuits) due to Abrahamson (Abrahamson, 1990) that is tight almost surely for input matrices whose entries are 1 with probability $1/\sqrt{n}$ independently.

Finally, we make a small adjustment to convert the Boolean matrix-vector lower bounds and lower bounds for systems of inequalities given in (Klauck et al., 2007) and (Ambainis et al., 2009), respectively, so that the problems that are shown hard for space S do not depend on S.

3.2 Preliminaries

We define the binary entropy function $H_2 : [0, 1] \to \mathbb{R}$, by $H_2(p) = -p \log_2 p - (1 - p) \log_2(1 - p)$.

Proposition 3.1 (Shannon)**.** *The number of subsets of* $[k]$ *of size at most* αk *is at most* $2^{H_2(\alpha)\, k}$.

Definition 3.2. An $m \times n$ matrix is (g, h, c)-*rigid* iff every $k \times w$ submatrix where $k \leq g$ and $w \geq n - h$ has rank at least ck. We call $(g, h, 1)$-rigid matrices (g, h)-rigid.

Matrix rigidity is a robust notion of rank and is an important property for proving time-space lower bounds for linear algebra. Fortunately, Yesha gives an explicit example of such a matrix and Abrahamson proved that there are many rigid square matrices.

Proposition 3.3 (Lemma 3.2 in (Yesha, 1984))**.** *The* $n \times n$ *Discrete Fourier Transform (DFT) matrix is* $(n/4, n/4, 1/2)$ *rigid.*

Proposition 3.4. *[Lemma 4.3 in (Abrahamson, 1991)] There is a constant $\gamma \in (0, \frac{1}{2})$ such that at least a $1 - d^{-1}(2/3)^{\gamma n}$ fraction of the matrices over $D^{n\times n}$ with $|D| = d$ are $(\gamma n, \gamma n)$-rigid.*

3.2.1 Time space tradeoffs for multi-output functions

Unitary quantum circuits with oracle states Throughout this dissertation, we consider quantum circuits that seek to compute target functions $f : D^n \to R^m$ (or functions $f : D^n \to \mathcal{P}(R)$ where the requirement is to output at least m elements of $f(x)$ if they exist). Let $d = |D|$ and assume the existence of some canonical bijective map $\nu : D \to \{\, 0, \ldots, d-1 \,\}$ that gives us an ordering on the elements of D. A T-query quantum circuit $\mathcal{C}$ is specified using input independent unitaries $U_0, \ldots, U_T$. These unitaries define a sequence of quantum states $|\psi_1\rangle_{\mathcal{C}}, \ldots |\psi_T\rangle_{\mathcal{C}}$ that an algorithm enters during its execution. When it is ambiguous, we use the subscript $\mathcal{C}$ to denote the partial trace of $|\psi_t\rangle$ that keeps only the qubits involved in the state of the query algorithm. Note that even though $|\psi_t\rangle$ is always a pure state, $|\psi_t\rangle_{\mathcal{C}}$ is often a mixed state. We can think of each of these states $|\psi_t\rangle_{\mathcal{C}}$ as a linear combination of basis vectors $|i, p, w\rangle$ where i represents an index to query, p represents a phase for the query, and w contains all the remaining qubits of the state.

Similar to (Ambainis, 2002; Zhandry, 2019; Hamoudi and Magniez, 2021), we define a general oracle operator $\mathcal{O}$ that interacts with an input register that starts in a state $|\psi_0\rangle_{\mathcal{O}}$. When it is ambiguous, we use the subscript $\mathcal{O}$ to denote the partial trace of $|\psi_t\rangle$ that keeps only the qubits involved in the state of the oracle containing the input. Given a distribution $\mathcal{D}$ over D^n, we can make $|\psi_0\rangle_{\mathcal{O}} = \sum_{X\in D^n} \sqrt{\Pr_{X'\sim\mathcal{D}}[X' = X]}\,|X\rangle$ to represent an input sampled from $\mathcal{D}$. We define our oracle operator $\mathcal{O}$ as

$$\mathcal{O}\,|i, p, w\rangle\,|X\rangle = \omega_d^{x_i p}\,|i, p, w\rangle\,|X\rangle\,.$$

Thus the joint state of the input and quantum circuit at the end of the computation is given by $|\psi_T\rangle = U_T\mathcal{O}\ldots\mathcal{O}U_0\,|\psi_0\rangle$ where $|\psi_0\rangle = |0\rangle_{\mathcal{C}} \otimes |\psi_0\rangle_{\mathcal{O}}$.

The output of the quantum circuit is determined by measuring the work register of $|\psi_T\rangle_{\mathcal{C}}$ in the standard basis and applying some input-independent post-processing function q to interpret the result as an output $\tau \in R^J$ where $J \subseteq [m]$. The correctness of these output values is then determined by measuring the input registers in the standard basis to obtain the input X and evaluating whether τ is consistent with $f(X)$, which we denote by writing $\tau \| f(X)$. In general we can define the projector Π_k where:

$$\Pi_k = \sum_{\substack{i,p,w,x_1,\ldots,x_n \\ \text{s.t. } q(w)\| f(x_1,\ldots,x_n) \\ \text{and } |q(w)| \geq k}} |i,p,w,x_1,\ldots,x_n\rangle \langle i,p,w,x_1,\ldots,x_n| \tag{3.1}$$

The probability that the circuit produces a correct partial assignment of at least k output values is given by $\|\Pi_k |\psi_T\rangle\|^2$. For a given partial assignment $q(w)$ to some outputs, we can define $\Pi_{q(w)}$ to be the projection onto the values of $|X\rangle$ where $q(w)\|f(X)$. More specifically we have that:

$$\Pi_{q(w)} = \sum_{\substack{x_1,\ldots,x_n \\ \text{s.t. } q(w)\| f(x_1,\ldots,x_n)}} |x_1,\ldots,x_n\rangle \langle x_1,\ldots,x_n| \tag{3.2}$$

By construction when q always produces a partial assignment of at least k elements we have that $\Pi_k = \sum_{i,p,w} |i,p,w\rangle \langle i,p,w| \otimes \Pi_{q(w)}$.

Space-bounded quantum computation As described above, we think of space-bounded quantum circuits as starting in the all $|0\rangle$ state and cycling between applying input queries $\mathcal{O}$, and arbitrary input-independent computation U_t. Unlike in the unitary circuit model, we allow our space-bounded quantum circuits to make intermediate measurements after applying each U_t as shown in Figure 3.1. Adopting the notation of (Beame and Kornerup, 2025), we will consider the set of consecutive $\mathcal{O}$, U_t and measurement gates as layer L_t. As was done in (Hamoudi and Magniez, 2021), we will assume that the quantum query circuit has a dedicated register containing a Boolean flag and a potential output $(i, y_i) \in [m] \times R$. After each query $\mathcal{O}$ and subsequent unitary operation U_t, the flag register is measured in the standard basis. Should the

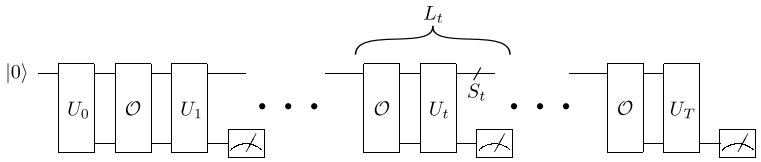

Figure 3.1: A general quantum circuit with T queries.

outcome 1 be obtained, the output register is measured in the standard basis and interpreted as an output pair (i, y_i) which is written to a write-only tape. Otherwise, the circuit produces no output during this layer. The space of layer L_t is the number of qubits that are passed from layer L_t to L_{t+1} and is denoted S_t. We define the space of a circuit as the maximum space of any layer, the time as the total number of layers. Thus the space needed to store the input and output is not included in this model.

Intermediate measurements enable circuits to produce parts of their output early and discard unnecessary ancillary qubits. Similar to the disjoint collisions bound in (Hamoudi and Magniez, 2021), our results in Sections 3.4 and 3.5 apply to quantum circuits without any required structure on their output order. Thus, as long as the circuits produce the correct output value for each index i, they may do so during arbitrary layers of the circuit that may depend on the chosen input. However, as was the case in (Klauck et al., 2007), our results for quantum Boolean matrix multiplication in Section 3.6 apply to a more restricted model of computation where the choice of when to produce each output value is independent of the input. In this *output-oblivious* model, quantum circuits do not have a flag register. Instead, on predefined layers the quantum circuit measures the output register in the standard basis and interprets the result as an element of R corresponding to a fixed output index. This output-oblivious ordering restricts the set of allowed algorithms and is necessary to prove our key lemmas associated with Boolean matrix multiplication.

Space-bounded classical computation One can view our classical lower bounds in Section 3.6 as applying to layered *branching programs* (Borodin and Cook, 1982) where the space bound corresponds to the logarithm of the width of the program

and the time corresponds to the number of layers. Output in a branching program is produced along the edges and written to a write-only output tape. Thus the space bound of a classical computation only considers the S bits of internal state maintained by the device and not the size of its read-only input or write-only output. Our results for classical Boolean matrix multiplication in Section 3.6 apply to an output-oblivious model, which corresponds to branching programs that must produce outputs for the same output index regardless of which edge is taken between two layers.

The Borodin-Cook method The Borodin-Cook method provides a general framework for proving time-space tradeoff lower bounds for multi-output problems, those for which every input vector in D^n is associated with some fixed set of possible output values from set R and the objective is to output at least m of these output values. As discussed earlier these can be functions $f : D^n \to R^m$, or $f : D^n \to \mathcal{P}(R)$ where the requirement is to produce at least m elements of $f(D^n)$, if they exist[2].

The property of the function f that enables the Borodin-Cook method to be used is the following[3] for some well-behaved function $h(k, n)$:

(*) Let $c = c(D) > 1$. Any classical query algorithm that makes at most $t \leq h = h(k, n)$ queries for an input distribution $\mathcal{D}$ on D^n, correctly produces k correct output values of f with probability at most c^{-k}.

With this property, Borodin and Cook showed that one directly obtains a classical time-space tradeoff for computing f of the form $T \cdot S = \Omega(m\, h(S/(\log c), n)\, \log c)$ for time T that is $n^{O(1)}$ and space S as follows:

Proof sketch. Choose k with $\log n \leq k \leq m$ such that $2^S \cdot T \cdot c^{-k} < 1$; then k is roughly $S/(\log c)$.

[2]There is a more general version where the query algorithm is only required to produce these m outputs with some sufficiently high probability but we focus on the simpler form

[3]We do not specify an upper limit on the possible $k \leq m$ in this informal statement. The exact range for which it holds will impact the space bounds for which the tradeoff holds.

Divide the T query steps into disjoint blocks of $h = h(k, n)$ queries each and assume that T is a multiple of h, without loss of generality. Since m outputs must be produced on all inputs in D^n and there are T/h blocks, for $T < mh/k$, which is $\Theta(m\, h \log c\ /S)$, for every execution on every input there must some block where at least k correct outputs are produced.

However, since the space is at most S there are at most 2^S configurations of the states that the algorithm could have been in at the beginning of each time block. Since **(*)** says that any fixed block can produce at least k output values correctly with probability at most c^{-k} under $\mathcal{D}$, by a union bound the total probability that some fixed block produces at least k correct output values is at most $2^S c^{-k} < 1/T$ by our choice of k. Since there are only T/h blocks, the probability that there is one of them that produces k correct answers is < 1.

Therefore T must be $\Omega(mh \log c\ /S)$ as required. □

For quantum algorithms, Klauck et al. (Klauck et al., 2007) observed that one could use a result by Aaronson in place of the union bound over the 2^S classical state configurations at the start of each block in the Borodin-Cook method.

Proposition 3.5 ((Aaronson, 2005))**.** *Let $\mathcal{C}$ be a quantum circuit, ρ be an S-qubit (possibly mixed) state, and π_{mix} be the S-qubit maximally mixed state. If $\mathcal{C}$ starting in initial state ρ produces some output z with probability p, then $\mathcal{C}$ starting in state π_{mix} will produce z with probability q which is at least $p/2^S$.*

We include a stand-alone derivation here for completeness.

Proof. Without loss of generality we can assume $\mathcal{C}$ performs no measurements until the end of the circuit. Thus we can think of $\mathcal{C}$ as representing a unitary operator U. Let Π_z be the projection onto output states of $\mathcal{C}$ that cause the circuit to output the value z. Then $p_z = \mathrm{Tr}[\Pi_z U \rho U^\dagger]$. By the spectral decomposition theorem we can represent ρ as a convex combination of some set of orthogonal pure states $\rho = \sum_{i \in [2^S]} \lambda_i \left|\varphi_i\right\rangle \left\langle\varphi_i\right|$.

Since the maximally mixed state can be represented as $\pi_{mix} = \sum_{i\in[2^S]}(1/2^S)\left|\varphi_i\right\rangle\left\langle\varphi_i\right|$ we have that:

$$\begin{aligned}
q &= \mathrm{Tr}[\Pi_z U \pi_{mix} U^\dagger] \\
&= \mathrm{Tr}[\Pi_z U \left(\sum_{i\in[2^S]} \frac{1}{2^S}\left|\varphi_i\right\rangle\left\langle\varphi_i\right|\right) U^\dagger] \\
&= \frac{1}{2^S}\mathrm{Tr}[\sum_{i\in 2^S} \left\langle\varphi_i\right| U^\dagger \Pi_z U \left|\varphi_i\right\rangle] \\
&\geq \frac{1}{2^S}\mathrm{Tr}[\sum_{i\in 2^S} \lambda_i \left\langle\varphi_i\right| U^\dagger \Pi_z U \left|\varphi_i\right\rangle] \\
&= \frac{1}{2^S}\mathrm{Tr}[\Pi_z U \left(\sum_{i\in[2^S]} \lambda_i \left|\varphi_i\right\rangle\left\langle\varphi_i\right|\right) U^\dagger] \\
&= \frac{1}{2^S}\mathrm{Tr}[\Pi_z U \rho U^\dagger] = p/2^S
\end{aligned}$$

Where the inequality comes from the fact that $\left\langle\varphi\right| U^\dagger \Pi_z U \left|\varphi\right\rangle \geq 0$ for any state $\left|\varphi\right\rangle$. □

With this they showed that essentially the same paradigm could be used to give similar time-space tradeoff lower bounds for quantum algorithms if one can prove a quantum analog of **(*)**. One subtlety that arises from the quantum version of the Borodin-Cook method is that often the quantum version of **(*)** is proven in a non-space-bounded unitary circuit model without intermediate measurements. By using the deferred measurement principle, we can see that lower bounds on the success probability of short quantum circuits in this model imply equally tight lower bounds in the space-bounded model where we directly apply the Borodin-Cook method.

3.2.2 The quantum recording query technique

Here we review the methods developed in (Zhandry, 2019; Hamoudi and Magniez, 2021) that allow us to analyze what a quantum circuit learns about its input by making quantum queries. We will assume that the input state $\left|\psi_0\right\rangle_{\mathcal{O}}$ is the equal superposition state over all inputs, although (Zhandry, 2019; Hamoudi and Magniez, 2021; Rosmanis, 2022) generalize this method to other input distributions.

We can exchange the general query operator $\mathcal{O}$ for the uniform input distribution with a recording query operator $\mathcal{R}$ that we define as follows:

Definition 3.6 (adapted from (Hamoudi and Magniez, 2021))**.** Let D be the input alphabet, $d = |D|$, and ν be our choice of canonical bijection between D and $\{0, \ldots, d-1\}$. We define $\mathcal{S}_1$ to be the unitary operator that maps

$$\mathcal{S}_1 : \begin{cases} |\perp\rangle & \longrightarrow \frac{1}{\sqrt{d}} \sum_{y \in D} |y\rangle \\ \frac{1}{\sqrt{d}} \sum_{y \in D} |y\rangle & \longrightarrow |\perp\rangle \\ \frac{1}{\sqrt{d}} \sum_{y \in D} \omega_d^{p\,\nu(y)} |y\rangle & \longrightarrow \frac{1}{\sqrt{d}} \sum_{y \in D} \omega_d^{p\,\nu(y)} |y\rangle \;\; \forall p \in \{1, \ldots, d-1\}. \end{cases}$$

Let $\mathcal{S} = (I)_{i,p,w} \otimes (\mathcal{S}_1^{\otimes n})_{x_1,\ldots,x_n}$ and $\mathcal{O}$ be the standard oracle operator that maps the basis state

$$|i, p, w, x_1, \ldots, x_n\rangle \longrightarrow \omega_d^{p\,\nu(x_i)} |i, p, w, x_1, \ldots, x_n\rangle .$$

Then the *recording query oracle operator* $\mathcal{R}$ is defined as $\mathcal{S}\mathcal{O}\mathcal{S}$.

$\mathcal{S}_1$ introduces $\perp$ as a new value for the input registers. Intuitively, the $\perp$ symbol indicates that the algorithm does not know anything about that register of the oracle. Hence by adding and correctly manipulating the $\perp$ symbols in the oracle's registers, we can record what the algorithm knows about the input. Since $\mathcal{S}^2 = I$, we can exactly characterize how the states of quantum circuits with oracles $\mathcal{O}$ and $\mathcal{R}$ relate to one another.

Proposition 3.7 (Theorem 3.3 in (Hamoudi and Magniez, 2021))**.** *Let $\mathcal{C}$ be a quantum circuit that for each $j \leq t$ applies unitary U_j after the j-th query. Let $\mathcal{S}$ be the unitary operation and $\mathcal{R}$ be the recording query oracle from Definition 3.6. Let*

$$|\psi_t\rangle = U_t \mathcal{O} U_{t-1} \ldots U_1 \mathcal{O} U_0 \left(|0\rangle_{i,p,w} \otimes \frac{1}{d^{n/2}} \sum_{x_1,\ldots,x_n \in D} |x_1, \ldots, x_n\rangle_{x_1,\ldots,x_n} \right)$$
$$|\phi_t\rangle = U_t \mathcal{R} U_{t-1} \ldots U_1 \mathcal{R} U_0 \left(|0\rangle_{i,p,w} \otimes |\perp\rangle_{x_1,\ldots,x_n} \right)$$

be the states of $\mathcal{C}$ with oracle $\mathcal{O}$ or $\mathcal{R}$ respectively. Then $|\psi_t\rangle = \mathcal{S} |\phi_t\rangle$.

In other words, it is impossible to distinguish the final state $|\psi_T\rangle$ of a circuit with standard oracle $\mathcal{O}$ from the output with recording oracle $\mathcal{R}$ if we apply $\mathcal{S}$ to the registers of $\mathcal{R}$ after the final query. Thus we can conclude that the success probability of a quantum circuit with T queries producing a partial assignment of k correct output values is given by $\|\Pi_k |\psi_T\rangle\|^2 = \|\Pi_k \mathcal{S} |\phi_T\rangle\|^2$. Note that while $|\phi_T\rangle$ may have inputs in the $\perp$ state, Proposition 3.7 tells us that $\mathcal{S} |\phi_T\rangle$ will never have an input in the $\perp$ state. This means that when considering recording query oracles, it is safe to keep our current definitions of Π_k and $\Pi_{q(w)}$ which will always project out any basis state where an input is assigned to $\perp$. We will leverage the following property of $|\phi_T\rangle$ to bound the success probability of quantum circuits with at most T queries.

Definition 3.8. Let Γ_t be the set of all elements $(D \cup \{\perp\})^n$ with at most t non-$\perp$ elements. This is the set of indices for all recording query basis states associated with quantum algorithms that make at most t queries.

Proposition 3.9 (Fact 3.2 in (Hamoudi and Magniez, 2021))**.** *The state $|\phi_t\rangle$ from Proposition 3.7 is a linear combination of basis states $|i, p, w, x_1, \ldots, x_n\rangle$ where variables $(x_1, \ldots, x_n) \in \Gamma_t$.*

For the bounds in (Hamoudi and Magniez, 2021) it is essential to bound how the state of $|\phi\rangle_{\mathcal{O}}$ can change after each query. For our use of the recording query technique, this detailed analysis is not necessary. Nevertheless, we state the following proposition here for completeness.

Proposition 3.10 (Lemma 4.1 in (Hamoudi and Magniez, 2021))**.** *Let D be the input alphabet, $d = |D|$, and ν be our choice of canonical bijection between D and $\{0, \ldots, d-1\}$. If the recording query operator $\mathcal{R}$ is applied to a basis state $|i, p, w, x_1, \ldots, x_n\rangle$ where $p \neq 0$ then the register $|x_i\rangle$ is mapped to*

$$\begin{cases} \sum_{y \in D} \frac{\omega_d^{p\,\nu(y)}}{\sqrt{d}} |y\rangle & \text{if } x_i = \perp \\ (1 - \frac{2}{d})\omega_d^{p\,\nu(x_i)} |x_i\rangle + \frac{1}{d} |x_i\rangle + \frac{\omega_d^{p\,\nu(x_i)}}{\sqrt{d}} |\perp\rangle + \sum_{y \in D \setminus \{x_i\}} \frac{1 - \omega_d^{p\,\nu(y)} - \omega_d^{p\,\nu(x_i)}}{d} |y\rangle & \text{otherwise.} \end{cases} \tag{3.3}$$

If $p = 0$ then the register remains unchanged.

3.3 Our bucketing methods

The Borodin-Cook method with recording queries Paraphrasing **(*)** from our earlier description of the Borodin-Cook method, to derive a time-space tradeoff lower bound for a function $f : D^n \to R^m$ or $f : D^n \to \mathcal{P}(R)$, we need to prove that any quantum query algorithm making at most some $h(k, n)$ queries can correctly produce at least k of the m output values only with a probability that decays exponentially in k (over the random choice of the input and the quantum measurements). In the recording query method, both the input and the state of the quantum algorithm are encoded in quantum states where measuring the local state of the algorithm determines the produced outputs (both indices and values) and measuring the local state of the input determines the classical input to the problem instance. Which k output positions are produced may depend on the input, so the paradigm proceeds by fixing both the quantum query algorithm and the k output values produced, and arguing that those k output values are correct for a fraction of the amplitude of the input state that is exponentially small in k.

Before discussing our bucketing method to do this, we first give some more detail about the method of Hamoudi and Magniez (Hamoudi and Magniez, 2021), as expressed in their lower bound for the m-disjoint collision problem:

The method of Hamoudi and Magniez operates in two parts: They show that

- for any quantum query algorithm (making at most $\varepsilon k\sqrt{n}$ queries), only an exponentially small fraction in k of the total amplitude of the input is on recording query basis states with at least $k/2$ disjoint collisions explicitly represented in the state (and hence for which at least $k/2$ outputs would automatically be correct), and
- for any fixed partial assignment of k output values (disjoint collisions), the fraction of the total amplitude on recording query basis states that do not

explicitly represent at least $k/2$ of those output values as collisions on which all k output values are correct is exponentially small in k.

The first part has most of the quantum flavor of the argument since the growth in the number of disjoint collisions observed is much faster in the quantum case than in the corresponding classical case. Because the k-disjoint collisions problem involves explicit local properties of the input, the second part involves many orthogonal components and hence is a rather straightforward adaptation of a classical argument, with a Cauchy-Schwartz calculation replacing a union bound.

In all the matrix problems we consider, correctness of the output values depends on the input values in highly non-local ways that do not yield the kind of orthogonality properties that Hamoudi and Magniez are able to exploit. There also is no analog of the kind of progress argument from the first part available. We have to work simply from a bound on the total number of queries that the algorithm makes. To handle this we introduce a bucketing approach.

3.3.1 Bucketing

Throughout this section we assume a fixed function f defined on D^n with m output values; all of our definitions are implicitly defined relative to this fixed function. The bucketing processes we define apply to the state of a quantum query algorithm after it has made t queries to a recording query oracle. By Proposition 3.9, such a state is a linear combination of basis states $|i, p, w, x\rangle$ with $x \in \Gamma_t$. Then for any partial assignment q of k output values that the query algorithm could have produced, we wish to prove that the fraction of the total amplitude on which the recording query input leads to an output that agrees with q is exponentially small in k.

Definition 3.11. Let q be a partial assignment of k output values and Π_q be the projector defined in Equation (3.2) for this partial assignment. For $c > 1$, we define a *c-admissible bucket* B for q to be a subset of $(D \cup \{\perp\})^n$ with the property that,

for *any* quantum state over the inputs that is spanned by the elements of B (i.e. $|\phi\rangle = \sum_{x \in B} \alpha_x |x\rangle$) such that:

$$\|\Pi_q \mathcal{S} |\phi\rangle\|^2 \leq c^{-k}$$

In our definitions of admissible buckets, the exponentially small bound will follow from the fact that for some fixed set of a $k' \geq c'k$ of the k output values, any state spanned by the elements of B will have a squared amplitude for those k' outputs of q being correct of exactly $d^{-k'}$; i.e., that of a completely random guess. In this case, $c = d^{c'}$.

Definition 3.12. Let A be a subset of $(D \cup \{\perp\})^n$. Then $\Pi_A = \sum_{x \in A} |x\rangle \langle x|$.

In Section 3.2.1 we defined projectors Π_k where $k \in \mathbb{N}$ and $\Pi_q(w)$ where $q(w) \in R^J$ for $J \subseteq [m]$ as ways to project onto basis states associated with the quantum circuit producing k correct output values or associated with an assignment $q(w)$ being correct for the value in the input register. Here, we use projectors Π_A to keep track of the contributions associated with various sets of recording query basis states in the analysis of our bucketing methods.

Definition 3.13. A *c-admissible bucket t-family* of size ℓ for a partial assignment q of k output values is a collection $\mathcal{B}$ of subsets of $(D \cup \{\perp\})^n$ such that

- $|\mathcal{B}| \leq \ell$,
- each $B \in \mathcal{B}$ is a c-admissible bucket for q, and
- every element of Γ_t is in at least one c-admissible bucket $B \in \mathcal{B}$.

The simple version of bucketing recording queries that we use to prove our lower bound for matrix-vector products works by showing that for $t \leq h(k, n)$ and each partial assignment of k output values q, there is a c-admissible bucket t-family $\mathcal{B}$ whose size is not too large. The amplitudes of the recording query basis states

indexed by Γ_t can be partitioned arbitrarily by assigning each element of Γ_t to some c-admissible bucket in the family that contains it. We obtain that the total squared amplitude associated with successfully producing output q is at most $|\mathcal{B}|^2/c^k$ for $c > 1$. For matrix-vector product with suitable fixed matrices, we are able to show in this case that $|\mathcal{B}|^2$ is at most b^k for some $b < c$ and hence obtain an upper bound on the overall success probability that is exponentially small in k.

In the case of matrix multiplication, the admissible bucket t-families we can produce are much too large. The basic approach above does not tailor the choice of buckets to the specific final state of the recording query input. We will instead need to produce an association of basis states with buckets that depends on the final state of the recording query input.

Definition 3.14. The *total amplitude* of a state $|\phi_t\rangle$ on recording query basis states in a set $A \subseteq (D \cup \{\perp\})^n$ is $\|\Pi_A |\phi_t\rangle\|$.

Definition 3.15. A *weighted c-admissible bucket t-scheme* of total weight w for a partial assignment q of k output values is a mapping that takes any quantum state $|\phi_t\rangle$ defined over recording query basis state indexed by Γ_t to a c-admissible bucket t-family $\mathcal{B}$ and an assignment of weights $w_B \in [0, 1]$ to sets $B \in \mathcal{B}$ such that

- for every $B \in \mathcal{B}$ we have $\|\Pi_B |\phi_t\rangle\| \leq w_B$ and
- $\sum_{B \in \mathcal{B}} w_B \leq w$.

When we want to emphasize the dependence of $\mathcal{B}$ on $|\phi_t\rangle$ we write it as $\mathcal{B}_{|\phi_t\rangle}$.

A c-admissible bucket t-family $\mathcal{B}'$ of size ℓ can always be interpreted as a weighted c-admissible bucket t-scheme of total weight ℓ by always setting $\mathcal{B}_{|\phi_t\rangle} = \mathcal{B}'$ and weight $w_B = 1$ for each $B \in \mathcal{B}'$.

Lemma 3.16. *Let q be a partial assignment of k output values and Π_q be the projector defined in Equation* (3.2) *for this partial assignment. If there is a weighted c-admissible*

bucket t-scheme of total weight w for q then there is a constant $c > 0$ such that for any quantum state $|\phi_t\rangle$ defined over recording query basis states indexed by Γ_t:

$$\|\Pi_q \mathcal{S} |\phi_t\rangle\|^2 \leq w^2 \cdot c^{-k}$$

Proof. Let $|\phi_t\rangle = \sum_{x \in \Gamma_t} \alpha_x |x\rangle$ be a quantum state defined over recording query basis elements indexed by Γ_t. Let $\mathcal{B}_{|\phi_t\rangle}$ with associated weights w_B for $B \in \mathcal{B}_{|\phi_t\rangle}$ be given by the weighted c-admissible bucket t-scheme for state $|\phi_t\rangle$. For $B \in \mathcal{B}_{|\phi_t\rangle}$, by definition, we have

$$\Pi_B |\phi_t\rangle = \sum_{x \in B} \alpha_x |x\rangle$$

and

$$\|\sum_{x \in B} \alpha_x |x\rangle\| = \|\Pi_B |\phi_t\rangle\| \leq w_B.$$

Then, since every $x \in \Gamma_t$ is contained in some $B \in \mathcal{B}_{|\phi_t\rangle}$, we have

$$\begin{aligned}
\|\Pi_q \mathcal{S} |\phi_t\rangle\|^2 &\leq \|\Pi_q \mathcal{S} \sum_{B \in \mathcal{B}_{|\phi_t\rangle}} \Pi_B |\phi_t\rangle\|^2 \\
&= \|\Pi_q \mathcal{S} \sum_{B \in \mathcal{B}_{|\phi_t\rangle}} \sum_{x \in B} \alpha_x |x\rangle\|^2 \\
&\leq \left(\sum_{B \in \mathcal{B}_{|\phi_t\rangle}} \|\Pi_q \mathcal{S} \sum_{x \in B} \alpha_x |x\rangle\| \right)^2 \\
&= \left(\sum_{B \in \mathcal{B}_{|\phi_t\rangle}} w_B \|\Pi_q \mathcal{S} \sum_{x \in B} \frac{\alpha_x}{w_B} |x\rangle\| \right)^2.
\end{aligned}$$

By definition, $\sum_{x \in B} \frac{\alpha_x}{w_B} |x\rangle$ is a vector whose 2-norm is at most 1 and the fact that B is a c-admissible bucket for q, implies that

$$\left(\sum_{B \in \mathcal{B}_{|\phi_t\rangle}} w_B \|\Pi_q \mathcal{S} \sum_{x \in B} \frac{\alpha_x}{w_B} |x\rangle\| \right)^2 \leq \left(\sum_{B \in \mathcal{B}_{|\phi_t\rangle}} w_B \cdot c^{-k/2} \right)^2 = w^2 \cdot c^{-k}$$

which yields our claimed bound on $\|\Pi_q \mathcal{S} |\phi_t\rangle\|^2$. □

Note that setting each w_B to 1 regardless of the quantum state gives us that $\|\Pi_q \mathcal{S} |\phi_t\rangle\| \leq |\mathcal{B}|^2 c^{-k}$, which is the simpler bound we use for the matrix-vector product

case. Though weighted schemes are much more flexible than such simple families and can yield bounds where those may not, choosing good weights presents an additional challenge. The follow concept will allow us to define these weights implicitly and is what we use when we analyze matrix product algorithms.

Definition 3.17. A *c-admissible bucket t-reduction scheme* of size ℓ for a partial assignment q of k output values is a mapping that takes any quantum state that involves a combination of recording query basis elements indexed by Γ_t, $|\phi_t\rangle = \sum_{z\in\Gamma_t} \delta_Z |z\rangle$, and outputs a collection of buckets $\mathcal{B}$ and a subset $\Gamma'_t \subseteq \Gamma_t$ such that

- $|\mathcal{B}| \leq \ell$,
- each $B \in \mathcal{B}$ is a c-admissible bucket for q,
- every element of Γ'_t is in at least one c-admissible bucket in $\mathcal{B}$, and
- the total amplitude of $|\phi_t\rangle$ on recording query basis states in $\Gamma_t \setminus \Gamma'_t$ is at most $1/2$. In other words, $\|\Pi_{\Gamma_t\setminus\Gamma'_t} |\phi_t\rangle\| \leq 1/2$.

Lemma 3.18. *The existence of a c-admissible bucket t-reduction scheme of size ℓ implies the existence of a weighted c-admissible bucket t-scheme of total weight 2ℓ.*

Proof. Fix q as the partial assignment to k output values and $|\phi_t\rangle$ as any input state over recording query basis states indexed by Γ_t. We apply the reduction scheme with the full state to begin with. This yields a collection of c-admissible buckets $\mathcal{B}$ of size ℓ and a set Γ'_t. Without loss of generality we can assume that $\Gamma'_t = \cup_{B\in\mathcal{B}} B$, as otherwise we could always define another c-admissible bucket t-reduction scheme of size ℓ with this choice of Γ'_t to use instead. We assign weight 1 to all those buckets.

The remaining total amplitude of states in Γ_t is at most $1/2$. We apply the reduction scheme inductively to the state $|\phi'_t\rangle = \Pi_{\Gamma_t\setminus\Gamma'_t} |\phi_t\rangle / \|\Pi_{\Gamma_t\setminus\Gamma'_t} |\phi_t\rangle\|$ which is a renormalized state for the portion of $|\phi_t\rangle$ defined on $\Gamma_t \setminus \Gamma'_t$. This yields a new set of at most ℓ buckets, each of which we assign weight $1/2$. Each iteration of this procedure

results in a renormalized state whose support is a smaller subset of Γ_t. We repeat in this way until we have exhausted all of Γ_t and produced a weighted c-admissible bucket t-scheme. The total weight of this scheme is at most $\sum_{i\geq 0} \ell/2^i \leq 2\ell$. □

Corollary 3.19. *Let q be a partial assignment of k output values and Π_q be the projector defined in Equation* (3.2) *for this partial assignment. The existence of a c-admissible bucket t-reduction scheme of size ℓ for q implies that for any quantum state $|\phi_t\rangle = \sum_{x\in\Gamma_t} \alpha_x |x\rangle$ there is a constant $c > 0$ such that:*

$$\|\Pi_q \mathcal{S} |\phi_t\rangle\|^2 \leq 4\ell^2 \cdot c^{-k}$$

3.3.2 When do good bucket reduction schemes exist?

Our definition of c-admissible t-reduction schemes requires that they are defined for all possible states $|\phi_t\rangle$ defined over Γ_t. In this section we show that a much simpler combinatorial property is sufficient to yield such schemes. As noted earlier, none of the lower bounds for specific functions that we prove use the methods of this section.

To motivate this property, for any $G \subseteq \Gamma_t$, we can consider a state $|\phi_t^G\rangle = \sum_{x\in G} \frac{1}{\sqrt{|G|}} |x\rangle$, the uniform superposition over G. A c-admissible t-reduction scheme for q of size ℓ must yield a set $\mathcal{B} = \mathcal{B}_G$ of c-admissible buckets for q of size ℓ such that $\|\Pi_{\bigcup_{B\in\mathcal{B}_G} B} |\phi_t^G\rangle\| \geq 1/2$. In particular, since $|\phi_t^G\rangle$ is a uniform superposition, $\bigcup_{B\in\mathcal{B}_G} B$ must contain at least a $1/\sqrt{2}$ fraction of the elements of G. In other words, we must have the following (c, ℓ, t) *admissible bucket covering property for q*:

- For every subset G of elements of Γ_t, there is a set of at most ℓ c-admissible buckets for q that contain at least a $1/\sqrt{2}$ fraction of the elements of G.

We will see that merely having such a property is sufficient to yield a reduction scheme that is not much larger and works for any state $|\phi_t\rangle$ defined over Γ_t.

Lemma 3.20. *Let f be a function defined on D^n. Let $c > 1$ and q be a partial assignment of k output values of f. If the (c, ℓ, t) admissible bucket covering prop-*

erty holds for q then there is a c-admissible bucket t-reduction scheme for q of size $O(t\ell \log(ne|D|/t))$.

Proof. Let $|\phi_t\rangle = \sum_{x\in\Gamma_t} \alpha_x |x\rangle$ be a quantum state. For $\lambda = 2^{1/12}$, partition Γ_t into subsets $\Gamma_t^1, \Gamma_t^2, \ldots$ such that Γ_t^i contains the $x \in \Gamma_t$ such that $|\alpha_x| \in (\lambda^{-i}, \lambda^{-(i-1)}]$.

Let $\kappa = 24 + \lceil 6t \ \log_2(ne|D|/t)\rceil$ and let $E = \bigcup_{i>\kappa} \Gamma_t^i$ be the portion of $|\phi_t\rangle$ not associated with the first κ sets of elements of Γ_t. The norm of the projection of $|\phi_t\rangle$ on E (in other words $\|\Pi_E |\phi_t\rangle\|$) is at most

$$\sqrt{|E|}\cdot\lambda^{-\kappa} \le \sqrt{|\Gamma_t|}\cdot\lambda^{-\kappa} = \left(\sum_{j=0}^{t}\binom{n}{j}|D|^j\right)^{1/2} 2^{-\kappa/12} \le \left(\frac{ne|D|}{t}\right)^{t/2} 2^{-\kappa/12} \le 1/4 \quad (3.4)$$

by our definition of κ. The reduction we construct will definitely leave this "error" portion of $|\phi_t\rangle$ uncovered.

Associated with each Γ_t^i for $i \in [\kappa]$, we apply the $c, \ell, t)$ admissible bucket property for q twice. The first yields at set of ℓ c-admissible buckets for q that together contain at least $1/\sqrt{2}$ fraction of the elements of Γ_t^i; we then apply the property to the $\le 1 - 1/\sqrt{2}$ fraction of elements of Γ_t^i that were not covered by the first application. Together we obtain a family $\mathcal{B}_i$, of size at most 2ℓ that contains a subset G_i consisting of at least a $\frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}}(1 - \frac{1}{\sqrt{2}}) = \sqrt{2} - 1/2$ fraction of the elements of Γ_t^i. The set of c-admissible buckets for q associated with $|\phi_t\rangle$ in the t-reduction scheme is $\mathcal{B} = \bigcup_{i\in[\kappa]} \mathcal{B}_i$. The size of this family is at most $2\kappa\ell$ which is $O(t\ell \log(ne|D|/t))$ as claimed.

It remains to prove that at most amplitude $1/2$ is left after removing the portion of $|\phi_t\rangle$ covered by $\mathcal{B}$. For each $\mathcal{B}_i$, which contains the elements of G_i for $i \le \kappa$, we have the following:

$$\begin{aligned}\sum_{x\in G_i} |\alpha_x|^2 &\ge (\sqrt{2} - 1/2)\cdot|\Gamma_t^i|\lambda^{-2i} \\ &= \lambda^{-2}(\sqrt{2} - 1/2)\cdot|\Gamma_t^i|\lambda^{-2(1-i)} \\ &\ge \lambda^{-2}(\sqrt{2} - 1/2)\cdot\sum_{x\in\Gamma_t^i}|\alpha_x|^2\end{aligned}$$

$$= (2^{1/3} - 2^{-7/6}) \cdot \sum_{x \in \Gamma_t^i} |\alpha_x|^2.$$

Now let $\Gamma'_t = \bigcup_{B \in \mathcal{B}} B$. Since $|\phi_t\rangle$ is a normalized vector supported by basis state in Γ_t:

$$\begin{aligned}
\|\Pi_{\Gamma'_t} |\phi_t\rangle\|^2 &= \sum_{\substack{i \in [\kappa] \\ x \in G_i}} |\alpha_x|^2 \\
&\geq (2^{1/3} - 2^{-7/6}) \sum_{\substack{i \in [\kappa] \\ x \in \Gamma_t^i}} |\alpha_x|^2 \\
&= (2^{1/3} - 2^{-7/6})(1 - \|\Pi_E |\phi_t\rangle\|^2) \\
&\geq (2^{1/3} - 2^{-7/6})15/16 \\
&> 3/4
\end{aligned}$$

This directly implies that $\|\Pi_{\Gamma_t \setminus \Gamma'_t} |\phi_t\rangle\| < 1/2$ as required. □

3.4 Quantum matrix vector products

In this section, we consider the task of — for a fixed matrix $A \in \mathbb{F}^{m \times n}$ — computing the function $f_A(x) = Ax$ for inputs $x \in D^m$ (where D is a fixed subset of $\mathbb{F}$) using a quantum circuit. We note that this is a fundamentally harder task than is considered in many quantum machine learning papers (for example (Harrow et al., 2009)) as we require the circuit to output a classical vector $y \in \mathbb{F}^n$ rather than either a quantum state encoding the entries of y in the amplitudes or an estimate of $y^\dagger My$. Also unlike many prior quantum time-space tradeoffs, including sorting (Klauck et al., 2007; Hamoudi and Magniez, 2021; Beame and Kornerup, 2023) and Boolean matrix multiplication (Klauck et al., 2007) (and our Theorem 3.45), our matrix vector product and matrix multiplication lower bounds apply to circuits that can adaptively decide when to produce each output based on the observed inputs. Time-space lower bounds against such quantum circuits were first described in (Hamoudi and Magniez, 2021) for the multiple disjoint collisions problem, although they were not able to show such a result for sorting. Similar to (Hamoudi and Magniez, 2021) we are able to lower

bound these circuits by identifying a single hard distribution over the inputs that applies to any set of outputs.

Theorem 3.21. *Let $m \leq n^r$ for some constant r and $2 \leq d \leq n^n$. There is a constant $C > 0$ such that the following holds: Let A be an $m \times n$ matrix over a field $\mathbb{F}$ that is (g, h, c)-rigid. Then any quantum circuit using time T and space $S < \frac{c}{6(r+6)} g \log_2 d$ that computes the function $f_A : D^n \to \mathbb{F}^m$ for $D \subseteq \mathbb{F}$ with $d = |D|$ given by $f_A(x) = Ax$ with success probability larger than 2^{-S} requires that $T \geq Cmh \log d \,/S$.*

When the fixed matrix A is sufficiently rigid, for example when both g and h are linear in n as is the case with the DFT matrix per Proposition 3.3 or a random matrix with high probability per Proposition 3.4, this lower bound becomes $\Omega(mn \log d)$ provided that S is at most some constant times $n \log d$ which is essentially a trivial constraint for the problem. This bound is tightly matched by a classical query algorithm in Proposition 3.62.

This theorem follows from the following key lemma, proven in Section 3.4.1, which lets us bound the number of correct output values produced by a shallow quantum circuit.

Lemma 3.22. *Let A be any (k, h, c)-rigid $m \times n$ matrix over a finite field $\mathbb{F}$ and let $f_A : D^n \to \mathbb{F}^m$ for $D \subseteq \mathbb{F}$ be defined by $f_A(x) = Ax$. Then for $\alpha > 0$ and for input x sampled uniformly from D^n and any quantum circuit $\mathcal{C}$ with at most αh queries to x, the probability that $\mathcal{C}$ produces k correct output values of $f_A(x)$ is at most $\lceil h/(ck) \rceil^2 \, (4^{H_2(\alpha)}/|D|^{1-\alpha})^{ck}$.*

Note: For $\alpha \leq 0.0737$ we have $1 - \alpha - 2H_2(\alpha) > 1/6$ and hence the bound is at most $\lceil h/(ck) \rceil^2 |D|^{-ck/6}$ for $d \geq 2$.

Proof of Theorem 3.21 from Lemma 3.22. Let $\mathcal{C}$ be a quantum circuit with T queries and space S that computes $f_A(x)$ with success probability larger than 2^{-S}. Since $h \leq n$, $m \leq n^r$ and $S \geq \log_2 n$ we only need to consider the case that $T \leq n^{r+1} \log_n d \leq n^{r+2}$.

Let $\alpha = 0.0737$. We partition $\mathcal{C}$ into $\lceil T/(\alpha h)\rceil$ sub-circuits that each have at most αh queries. By combining Proposition 3.5 and Lemma 3.22, we know that each sub-circuit can produce $k \leq g$ correct output values with probability at most $2^S \lceil h/(ck)\rceil^2 d^{-ck/6} \leq h^2\, 2^S d^{-ck/6}$.

By assumption, we have $d^{-cg/6} \leq 2^{-(r+6)S} \leq n^{-(r+4)} 2^{-2S} \leq h^{-2} 2^{-2S}/T$ since $S \geq \log_2 n$, $T \leq n^{r+2}$, and $h \leq n$. In particular, this implies that $h^2 d^{-cg/6} < 2^{-S}$ so we must have $T > \alpha h$ by Lemma 3.22. Set $k \leq g$ to be the smallest integer such that $h^2\, 2^S d^{-ck/6} \leq 2^{-S}/T$. Then the probability that a sub-circuit produces k correct output values is at most $2^{-S}/T$. This gives $k = \lceil [6\log_2(hT) + 12S]/(c \log_2 d)\rceil$. We note that k is at most $c^* S/\log_2 d$ for some constant $c^* > 0$ since $\log_2(hT) \leq (r+3)\log_2 n \leq (r+3)S$.

Taking a union bound over the sub-circuits, the probability that any of them produces k correct output values is at most 2^{-S}. Since f_A has m outputs, this means that

$$\lceil T/(\alpha h)\rceil\,(k-1) \geq m$$

Since $T \geq \alpha h$, we have

$$2Tk \geq \alpha m h.$$

Plugging in our upper bound on k we have that

$$2c^* TS/\log_2 d \geq \alpha m h$$

and hence $T \cdot S$ is at least $\frac{\alpha}{2c^*} mh \log d$ as claimed. □

3.4.1 Success probability of small depth quantum circuits

We first give an overview of the argument, which involves an initial uniform distribution over the inputs $x \in D^n$. This begins by decomposing the state after $t \leq \alpha h$ queries into orthogonal components based on the values of working qubits $|i, p, w\rangle$, which also determine the set of k output values produced. It then suffices to prove that for each fixed $|i, p, w\rangle$ the total fraction of the squared amplitude for any

state that is spanned by recording query basis states with at most t non-$\perp$ items can be on inputs for which the fixed output values are correct is exponentially small in k.

If we knew which $t \leq \alpha h$ input indices were queried, as we would with classical algorithms in the analysis of (Abrahamson, 1991), then things would be easy: Since the fixed matrix A is (k, h, c) rigid, the sub-matrix of A with rows corresponding to these k outputs, and with the $\geq n - \alpha h$ "unqueried" columns has rank at least ck, so any fixed output can be correct with probability at most d^{-ck} over the choice of inputs. However, the quantum state after t queries is a superposition of recording query basis states that could involve all possible subsets of $\leq t$ non-$\perp$ indices which is at least $\binom{n}{t}$ possibilities.

To handle this we use the basic version of our bucketing method for recording query basis states and find a relatively small collection of admissible buckets (whose size will be a sufficiently small exponential in k) that allows us to run the quantum analogue of the classical argument within each bucket. We now give the proof in detail.

Proof of Lemma 3.22. Let $d = |D|$. For simplicity we will assume that $q(w)$—the output as a function of the measured value of the work register—always produces k outputs.[4] Let A be a (k, h, c)-rigid matrix. By Proposition 3.9 after $t \leq \alpha h$ queries in the recording query oracle model, the state $|\phi_t\rangle$ is a linear combination of basis states $|i, p, w, x_1, \ldots, x_n\rangle$ where $(x_1, \ldots, x_n) \in \Gamma_t$. It will be useful to be more explicit in our discussion of Γ_t. Each element of Γ_t consists of an assignment $y \in D^I$ for some subset $I \subseteq [n]$ with $|I| \leq t$ and value $\perp$ on all coordinates in $[n] \setminus I$. Therefore, we can write the state as:

$$|\phi_t\rangle = \sum_{\substack{i,p,w \\ I \subseteq [n],\ |I| \leq t \\ y \in D^I}} \alpha_{i,p,w,I,y} \, |i, p, w\rangle \, |y\rangle_I \, |\perp\rangle_{[n] \setminus I} \tag{3.5}$$

[4] If in general $q(w)$ produces more than k outputs, we only consider its first k outputs.

for some $\alpha_{i,p,w,I,y}$ with $\sum_{i,p,w,I,y} |\alpha_{i,p,w,I,y}|^2 = 1$. Thus by Proposition 3.7, the final state of the algorithm (after $t \leq \alpha h$ queries) in the non-recording query oracle setting is given by:

$$|\psi_t\rangle = \mathcal{S}\,|\phi_t\rangle = \mathcal{S} \sum_{\substack{i,p,w \\ I\subseteq[n],\ |I|\leq t \\ y\in D^I}} \alpha_{i,p,w,I,y}\,|i,p,w\rangle\,|y\rangle_I\,|\perp\rangle_{[n]\setminus I}$$

Since $\mathcal{S}$ behaves as the identity on $|\phi_t\rangle_{\mathcal{C}}$ and the $|i,p,w\rangle$ are orthogonal basis states, we can rewrite this as:

$$\sum_{i,p,w} \beta_{i,p,w}\,|i,p,w\rangle \otimes \left[\mathcal{S}_1^{\otimes n} \sum_{\substack{I\subseteq[n],\ |I|\leq t \\ y\in D^I}} \beta_{I,y}^{i,p,w}\,|y\rangle_I\,|\perp\rangle_{[n]\setminus I}\right]$$

for some $\beta_{i,p,w}$ and $\beta_{I,y}^{i,p,w}$ such that $\alpha_{i,p,w,I,y} = \beta_{i,p,w}\,\beta_{I,y}^{i,p,w}$, $\sum_{i,p,w} |\beta_{i,p,w}|^2 = 1$ and for each choice of i,p,w, we have that $\sum_{I,y} |\beta_{I,y}^{i,p,w}|^2 = 1$. With this decomposition, using the definition in Equation (3.1), the success probability of producing k correct output values is given by:

$$\begin{aligned}
\left\|\Pi_k \mathcal{S}\,|\phi_t\rangle\right\|^2 &= \left\|\Pi_k \sum_{i,p,w} \beta_{i,p,w}\,|i,p,w\rangle \otimes \left[\mathcal{S}_1^{\otimes n} \sum_{\substack{I\subseteq[n],\ |I|\leq t \\ y\in D^I}} \beta_{I,y}^{i,p,w}\,|y\rangle_I\,|\perp\rangle_{[n]\setminus I}\right]\right\|^2 \\
&= \left\|\sum_{i,p,w} \beta_{i,p,w}\,|i,p,w\rangle \otimes \left[\Pi_{q(w)}\mathcal{S}_1^{\otimes n} \sum_{\substack{I\subseteq[n],\ |I|\leq t \\ y\in D^I}} \beta_{I,y}^{i,p,w}\,|y\rangle_I\,|\perp\rangle_{[n]\setminus I}\right]\right\|^2
\end{aligned}$$

where $\Pi_{q(w)}$ is defined as in Equation (3.2) and is the projection of Π_k onto fixed values of $q(w)$. Since the basis states $|i,p,w\rangle$ are orthogonal and $\sum_{i,p,w} |\beta_{i,p,w}|^2 = 1$, we have

$$\left\|\Pi_k \mathcal{S}\,|\phi_t\rangle\right\|^2 \leq \max_{i,p,w} \left\|\Pi_{q(w)}\mathcal{S}_1^{\otimes n} \sum_{\substack{I\subseteq[n],\ |I|\leq t \\ y\in D^I}} \beta_{I,y}^{i,p,w}\,|y\rangle_I\,|\perp\rangle_{[n]\setminus I}\right\|^2 \tag{3.6}$$

We now fix i,p,w and let $A_{q(w)}$ be the submatrix of A restricted to the rows defined by the set of the k output values U associated with $q(w)$. We can describe $\Pi_{q(w)}$ as a

projection onto basis states $|x_1, \ldots, x_n\rangle$ such that:

$$A_{q(w)} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = q(w).$$

Since the basis states $|y\rangle_I |\perp\rangle_{[n]\setminus I}$ for distinct I are orthogonal in the recording query basis, they remain orthogonal in the standard basis after the $\mathcal{S}$ operator is applied. However, the subsequent application of the $\Pi_{q(w)}$ projector makes these vectors no longer orthogonal.

To handle this, we bucket the sets $I \subseteq [n]$ with $|I| \leq t$ into a small number of buckets, $\mathcal{B}_1, \ldots$, so that for each bucket $\mathcal{B}_\ell$ we can bound:

$$\mu_\ell = \left\| \Pi_{q(w)} \mathcal{S}_1^{\otimes n} \sum_{I \in \mathcal{B}_\ell, y \in D^I} \beta_{I,y}^{i,p,w} |y\rangle_I |\perp\rangle_{[n]\setminus I} \right\|^2$$

and then we can use the triangle inequality to bound the success probability as a sum of the μ_ℓ.

In particular, our key observation is that if a bucket of recording query basis states completely misses querying a fixed set of input variables that could completely scramble the value of a set of r output values, then one cannot do better than randomly guess those output values. More precisely, we show that the contribution to success from that bucket of basis states has amplitude at most $\frac{1}{\sqrt{d^r}}$.

Lemma 3.23. *Let $U \subseteq [m]$ be a set of output indices and $V \subseteq [n]$ be a set of input indices with $|V| = |U| = r$ such that the submatrix $A_{U,V}$ is full rank. Fix $q \in \mathbb{F}^U$ and define Π_q to be the projection map onto the span of the set of basis states $|x_1, \ldots, x_n\rangle$ with $x_1 \ldots x_n \in D$ such that $A_U x = q$. Then for any collection $\mathcal{B}$ of sets $I \subseteq [n] \setminus V$ and any quantum state $\sum_{I \in \mathcal{B},\ y \in D^I} \eta_{I,y} |y\rangle_I |\perp\rangle_{[n]\setminus I}$ we have*

$$\left\| \Pi_q \mathcal{S}_1^{\otimes n} \sum_{I \in \mathcal{B},\ y \in D^I} \eta_{I,y} |y\rangle_I |\perp\rangle_{[n]\setminus I} \right\|^2 \leq \frac{1}{d^r}.$$

Proof. By definition each $I \in \mathcal{B}$ satisfies $I \cap V = \varnothing$, so

$$\begin{aligned}
&\Pi_q \mathcal{S}_1^{\otimes n} \sum_{I\in\mathcal{B},\ y\in D^I} \eta_{I,y} \left|y\right\rangle_I \left|\perp\right\rangle_{[n]\setminus I} \\
&= \Pi_q \mathcal{S}_1^{\otimes n} (\left|\perp\right\rangle_V \otimes \sum_{I\in\mathcal{B},\ y\in D^I} \eta_{I,y} \left|y\right\rangle_I \left|\perp\right\rangle_{[n]\setminus(I\cup V)} \\
&= \Pi_q (\mathcal{S}_1^{\otimes r} \left|\perp\right\rangle_V \otimes \mathcal{S}_1^{\otimes(n-r)} \sum_{I\in\mathcal{B},\ y\in D^I} \eta_{I,y} \left|y\right\rangle_I \left|\perp\right\rangle_{[n]\setminus(I\cup V)}) \\
&= \Pi_q (\sum_{y'\in D^V} \frac{1}{\sqrt{d^r}} \left|y'\right\rangle_V \otimes \mathcal{S}_1^{\otimes(n-r)} \sum_{I\in\mathcal{B},\ y\in D^I} \eta_{I,y} \left|y\right\rangle_I \left|\perp\right\rangle_{[n]\setminus(I\cup V)})
\end{aligned}$$

since $\mathcal{S}_1(\left|\perp\right\rangle) = \sum_{y'\in D} \frac{1}{\sqrt{d}} \left|y'\right\rangle$. Now

$$\mathcal{S}_1^{\otimes(n-r)} \sum_{I\in\mathcal{B},\ y\in D^I} \eta_{I,y} \left|y\right\rangle_I \left|\perp\right\rangle_{[n]\setminus(I\cup V)} = \sum_{z\in(D\cup\{\perp\})^{[n]\setminus V}} \delta_z \left|z\right\rangle_{n\setminus V}$$

for some amplitudes δ_z satisfying $\sum_{z\in(D\cup\{\perp\})^{[n]\setminus V}} |\delta_z|^2 = 1$.

For each value of $z \in D^{[n]\setminus V}$, since the sub-matrix $A_{U,V}$ is invertible, there is a unique value $y_z \in D^V$ such that $A_U(y_z \cup z) = q$ so we get that

$$\begin{aligned}
&\left\| \Pi_q \mathcal{S}_1^{\otimes n} \sum_{I\in\mathcal{B},\ y\in D^I} \eta_{I,y} \left|y\right\rangle_I \left|\perp\right\rangle_{[n]\setminus I} \right\|^2 \\
&= \left\| \Pi_q \left[\sum_{y'\in D^V} \frac{1}{\sqrt{d^r}} \left|y'\right\rangle_V \otimes \sum_{z\in(D\cup\{\perp\})^{n-r}} \delta_z \left|z\right\rangle_{[n]\setminus V} \right] \right\|^2 \\
&= \left\| \frac{1}{\sqrt{d^r}} \cdot \Pi_q \left[\sum_{y'\in D^V} \left|y'\right\rangle_V \sum_{z\in D^{n-r}} \delta_z \left|z\right\rangle_{n\setminus V} \right] \right\|^2 \\
&= \left\| \frac{1}{\sqrt{d^r}} \cdot \Pi_q \sum_{z\in D^{[n]\setminus V}} \delta_z \sum_{y'\in D^V} \left|y'\right\rangle_V \left|z\right\rangle_{n\setminus V}) \right\|^2 \\
&= \left\| \frac{1}{\sqrt{d^r}} \sum_{z\in D^{[n]\setminus V}} \delta_z \left|y_z\right\rangle_V \left|z\right\rangle_{n\setminus V}) \right\|^2 \\
&\leq \frac{1}{d^r}
\end{aligned}$$

since $\sum_{z\in D^{[n]\setminus V}} |\delta_z|^2 \leq 1$. □

Next we decompose the set of all I with $|I| \leq t$ into buckets where we can apply the above with r equal to a constant fraction of k. (This decomposition of the

sets I into buckets automatically implies a decomposition of Γ_t into buckets, each of which will be a c'-admissible bucket for some constant c' by Lemma 3.23 and the value of r, yielding a c'-admissible bucket t-family corresponding to the basic version of our bucketing methods as discussed in Section 3.3.)

Lemma 3.24. *Let A be a (k, h, c)-rigid matrix and let $k' = \lceil ck \rceil$. Then for every subset U of k rows of A, there is a collection of disjoint k'-subsets of columns from $[n]$, $V_1, \ldots, V_\ell$ for $\ell = \lceil h/k' \rceil \leq \lceil h/(ck) \rceil$ and corresponding sets of rows $U_1, \ldots, U_\ell \subseteq U$ such that for each $j \in [\ell]$, the $k' \times k'$ submatrix A_{U_j, V_j} is full rank. (In particular the union, W, of the sets V_j has size at least h.) If $c = 1$ then all $U_j = U$.*

Proof. Fix $U \in [m]$ with $|U| = k$. The following procedure constructs such a collection, one set at a time. We maintain a subset of W columns that is the union of the V_j constructed so far. Suppose that $|W| < h$. Then, by the (k, h, c)-rigidity of A, the submatrix $A_{U,[n]\setminus W}$ has rank at least k'. Hence there is a $k' \times k'$ submatrix A_{U_j, V_j} of $A_{U,[n]\setminus W}$ that has full rank k'. We now add V_j to the collection of k'-sets of columns, record its corresponding row set U_j, and set $W \leftarrow W \cup V_j$. This produces exactly $\lceil h/k' \rceil$ subsets. □

Fix the collection of sets $V_1, \ldots, V_\ell$ given by Lemma 3.24. Let $k'' = \lfloor \alpha k' \rfloor$. Suppose that $V_j = \{i_1, \ldots, i_{k'}\} \subseteq [n]$ with $i_1 \leq \cdots \leq i_{k'}$. For each $\lambda \in \binom{[k']}{k''}$, define the set V_j^λ to be the subset of V_j that has the k'' elements of V_j indexed by λ removed. (That is, $i_{j'} \notin V_j^\lambda$ iff $j' \in \lambda$.) Then $|V_j^\lambda| = k' - k'' \geq c(1-\alpha)k$. There are a total of $\binom{k'}{k''} \leq 2^{H_2(\alpha)\, k'}$ possible values of λ and hence $\lceil h/k' \rceil \cdot 2^{H_2(\alpha)\, k'}$ sets of the form V_j^λ. These sets have two useful properties: first any subset of $[n]$ with size at most αh must miss some V_j^λ and second if the entries of x corresponding to some V_j^λ are uniformly random, then for any set of k indices in Ax, at least $c(1-\alpha)k$ of these values are also uniformly random.

Lemma 3.25. *For $t \leq \alpha h$ and every $I \subseteq [n]$ with $|I| \leq t$, there is some $j \leq \lceil h/k' \rceil$ and $\lambda \in \binom{[k']}{k''}$ such that $I \subseteq [n] \setminus V_j^\lambda$.*

Proof. Fix such a set I with $|I| \leq t$. Since $t \leq \alpha h$, $|\bigcup_{j\in[\ell]} V_j| \geq h$, and the sets V_j are disjoint, by averaging there is some set V_j that has at most an α fraction of its elements in I. Hence V_j has at most $k'' \leq \alpha k'$ elements of I. Choose a set $\lambda \in \binom{[k']}{k''}$ that contains the indices within V_j of all of the elements of $V_j \cap I$. Then by construction $I \cap V_j^\lambda = \varnothing$. □

(Lemmas 3.23 and 3.25 together give us all the ingredients we need to yield a c'-admissible bucket t-family with $c' = d^{c(1-\alpha)}$ as defined in Section 3.3; each element of the family is determined by a pair (j, λ) as follows:) By applying Lemma 3.25 we can associate each $I \subseteq [n]$ with $|I| \leq t$ with a pair (j, λ) such that $I \in [n] \setminus V_j^\lambda$ and define bucket $\mathcal{B}_j^\lambda$ to consist of all such sets I associated with pair (j, λ).[5] Further, define a set $U_j^\lambda \subseteq U_j \subseteq [m]$ of the rows of $A_{q(w)}$ with $|U_j^\lambda| = k' - k''$ such that the submatrix $A_{U_j^\lambda, V_j^\lambda}$ is full rank. Such a subset of rows must exist since A_{U_j, V_j^λ} is a full rank matrix. Then let $q_j^\lambda = q(w)|_{U_j^\lambda}$ be the portion of the assignment $q(w)$ on the rows of U_j^λ.

We are now ready to provide an upper bound on the success probability from Equation (3.6) using our admissible bucket family.

$$\begin{aligned}
&\left\| \Pi_{q(w)} \mathfrak{S}_1^{\otimes n} \sum_{\substack{I\subseteq[n],\ |I|\leq t \\ y\in D^I}} \beta_{I,y}^{i,p,w} \left|y\right\rangle_I \left|\perp\right\rangle_{[n]\setminus I} \right\|^2 \\
&= \left\| \Pi_{q(w)} \mathfrak{S}_1^{\otimes n} \sum_{j\in[\ell]} \sum_{\lambda\in\binom{[k']}{k''}} \sum_{I\in\mathcal{B}_j^\lambda,\ y\in D^I} \beta_{I,y}^{i,p,w} \left|y\right\rangle_I \left|\perp\right\rangle_{[n]\setminus I} \right\|^2 \\
&\leq \left\| \sum_{j\in[\ell]} \sum_{\lambda\in\binom{[k']}{k''}} \Pi_{q_j^\lambda}\ \mathfrak{S}_1^{\otimes n} \sum_{I\in\mathcal{B}_j^\lambda,\ y\in D^I} \beta_{I,y}^{i,p,w} \left|y\right\rangle_I \left|\perp\right\rangle_{[n]\setminus I} \right\|^2 . \qquad (3.7)
\end{aligned}$$

Applying Lemma 3.23 with $r = k' - k''$, $q = q_j^\lambda$, $U = U_j^\lambda$, $V = V_j^\lambda$, and $\mathcal{B} = \mathcal{B}_j^\lambda$, we

[5]Note that while some sets I could be associated with multiple pairs (j, λ) in the admissible bucket family, since we only require one bucket per recording query basis element for the analysis, we will choose only one such pair for each I.

have that

$$\left\| \Pi_{q_j^\lambda} \, \mathcal{S}_1^{\otimes n} \sum_{I \in \mathcal{B}_j^\lambda, \; y \in D^I} \beta_{I,y}^{i,p,w} \, |y\rangle_I \, |\perp\rangle_{[n]\setminus I} \right\|^2 \leq 1/d^{k'-k''} \leq 1/d^{(1-\alpha)\,k'}.$$

and hence using Equation (3.7) we obtain that

$$\left\| \Pi_k \mathcal{S} \, |\phi_t\rangle \right\|^2 \leq \ell^2 \binom{k'}{k''}^2 / d^{(1-\alpha)\,k'} \leq \lceil h/k' \rceil^2 \, 4^{H_2(\alpha)\,k'} / d^{(1-\alpha)\,k'} = \lceil h/k' \rceil \, (4^{H_2(\alpha)} / d^{(1-\alpha)})^{k'}.$$

Without loss of generality in our desired bound we can assume that $4^{H_2(\alpha)}/d^{(1-\alpha)} < 1$. Therefore, the bound still applies when we replace k' by the potentially smaller ck which is what we needed to show. □

3.4.2 Related time-space tradeoffs

Following the same arguments as for classical computation (Abrahamson, 1991), we use Theorem 3.21 to obtain a collection of time-space lower bounds for problems that are closely related to matrix vector products. Our proofs are identical to their classical counterparts proven in(Abrahamson, 1991, Sections 5-6) and are duplicated here for completeness. Many of these lower bounds are tightly matched by classical query algorithms. Constructions of matching upper bounds can be found in Section 3.7.

Corollary 3.26. *Let $\mathbb{F}$ be a field and $D \subseteq \mathbb{F}$ such that $d = |D|$. Any quantum circuit that computes the discrete Fourier transform (DFT) of vectors in D^n in time T and space S with probability at least 2^{-S} requires T to be $\Omega(n^2 \log(d) \; / S)$.*

Proof. Applying Theorem 3.21 with the rigidity of the DFT from Proposition 3.3 directly gives us the lower bound. □

Proposition 3.27 ((Abrahamson, 1991))**.** *There is a constant $\gamma \in (0, 1/2)$ such that at least a $1 - |D|^{-1}(2/3)^{\gamma n}$ fraction of the Toeplitz (diagonal constant) matrices over $D^{n \times n}$ are $(\gamma n, \gamma n)$-rigid.*

Recall that the convolution of two vectors $w = u * v$ is $w_k = \sum_{i \in [n]} u_i v_{k-i}$ where the indices are reduced modulo n, where we identify n with 0.

Corollary 3.28. *Let $\mathbb{F}$ be a field and $D \subseteq \mathbb{F}$ such that $d = |D|$. Any quantum query algorithm computing the convolution of two vectors in D^n in time T and space S with probability at least 2^{-S} requires T to be $\Omega(n^2 \log(d)\ /S)$*

Proof. For simplicity assume that n is even. Let

$$U = \begin{bmatrix} u_n & u_{n-1} & \dots & u_2 & u_1 \\ u_1 & u_n & \dots & u_3 & u_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ u_{n-2} & u_{n-3} & \dots & u_n & u_{n-1} \\ u_{n-1} & u_{n-2} & \dots & u_1 & u_n \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

Where A, B, C and D are $n/2 \times n/2$ submatrices. Then Uv is the convolution between vectors u and v. Observe that U is a Toeplitz matrix and by picking u to be a uniform vector over D, Proposition 3.27 tells us that for sufficiently large n, there is a constant $\gamma \in (0, 1/2)$ such that both A and B are $(\gamma n, \gamma n/2)$-rigid with probability at least $1/2$. This lets us restrict our input to such choices for u and observe that the matrix $U' = \begin{bmatrix} A & B \end{bmatrix}$ is $(\gamma n, \gamma n/2)$-rigid, so Theorem 3.21 gives us that computing $U'v$ requires T that is $\Omega(n^2 \log(d)\ /S)$. Since U' is a subfunction of U, convolution also requires T that is $\Omega(n^2 \log(d)\ /S)$. □

Corollary 3.29. *A quantum circuit that multiplies two n bit binary numbers in time T and space S with probability at least 2^{-S} requires T to be $\Omega(n^2/(S \log^2 n))$.*

Proof. Let u, v be arbitrary vectors over $\mathbb{F}_2$. Define the binary number

$$u' = 0^{\lceil \log_2 n \rceil - 1} u_n \dots 0^{\lceil \log_2 n \rceil - 1} u_1 0^{\lceil \log_2 n \rceil - 1} u_n \dots 0^{\lceil \log_2 n \rceil - 1} u_1$$

and similarly define v'. Then observe that the product $u' \cdot v'$ contains all entries of the convolution between u and v encoded in blocks of $\lceil \log_2 n \rceil$ bits each. By Corollary 3.28 this requires T to be $\Omega(n^2/(S \log^2 n))$. □

Proposition 3.30 ((Abrahamson, 1991))**.** *Let $A, B, C, Y \in D^{n\times n}$. Let $\mathcal{B}$ (and $\mathcal{Y}$) be the vectors in D^{n^2} formed by stacking the transposes of the rows of B (and Y) into a column vector. If D is a commutative ring, then the following conditions are equivalent:*

$$Y = ABC$$
$$\mathcal{Y} = (A \otimes C^T)\mathcal{B}$$

Where $\otimes$ is the standard tensor (Kronecker) product.

Proposition 3.31 ((Abrahamson, 1991))**.** *Let $\gamma \in (0, 1/2)$. If A and B are $(\gamma n, \gamma n)$-rigid, then $A \otimes B$ is $(\gamma^2 n^2, \gamma^2 n^2, \gamma^2)$-rigid.*

Corollary 3.32. *Let $\mathbb{F}$ be a field and $D \subseteq \mathbb{F}$ such that $d = |D|$. Any quantum circuit that computes the product ABC on inputs $A, B, C \in D^{n\times n}$ in time T and space S with probability at least 2^{-S} requires T that is $\Omega(n^4 \log(d)\ /S)$.*

Proof. We use Proposition 3.30 to view this as a matrix-vector product problem where $\mathcal{B}$ is the input and $\mathcal{Y}$ is the output. By Proposition 3.4 there is a constant $\gamma \in (0, 1/2)$ such that both A and C are γ rigid with constant probability, so we can assume such without increasing the expected cost by more than a constant factor. Then Proposition 3.31 gives us that $A \otimes C$ is $(\gamma^2 n^2, \gamma^2 n^2, \gamma^2)$-rigid and we can apply Theorem 3.21 to get that T must be $\Omega(n^4 \log(d)\ /S)$ as desired. □

Corollary 3.33. *Let $\mathbb{F}$ be a field and $D \subseteq \mathbb{F}$ such that $d = |D|$. Any quantum circuit that computes A^3 on inputs in $D^{n\times n}$ in time T and space S with probability at least 2^{-S} requires T that is $\Omega(n^4 \log(d)\ /S)$.*

Proof. Let $A, B, C \in D^{n\times n}$. Then construct the $4n \times 4n$ matrix:

$$M = \begin{bmatrix} 0 & A & 0 & 0 \\ 0 & 0 & B & 0 \\ 0 & 0 & 0 & C \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Observe that the top right $n \times n$ sub-matrix of M^3 is equal to the product ABC. Thus we get a reduction to matrix-matrix-matrix product and can apply Corollary 3.32 to get our lower bound. □

Corollary 3.34. *Let $\mathbb{F}$ be a field and $D \subseteq \mathbb{F}$ such that $d = |D|$. Any quantum circuit that computes A^{-1} on unit upper triangular inputs in $D^{n \times n}$ in time T and space S with probability at least 2^{-S} requires T that is $\Omega(n^4 \log(d)/S)$.*

Proof. Let $A, B, C \in D^{n \times n}$. Then construct the $4n \times 4n$ matrix:

$$M = \begin{bmatrix} I & -A & 0 & 0 \\ 0 & I & -B & 0 \\ 0 & 0 & I & -C \\ 0 & 0 & 0 & I \end{bmatrix}$$

Where I is the $n \times n$ identity submatrix. Then observe that M^{-1} has the product ABC as its top right $n \times n$ submatrix. We can again use Theorem 3.21 to get our lower bound. □

Corollary 3.35. *Let $\mathbb{F}$ be a field and $D \subseteq \mathbb{F}$ such that $d = |D|$. Any quantum circuit that solves any $n \times n$ system of linear equations over D in time T and space S with probability at least 2^{-S} requires T that is $\Omega(n^3 \log(d)\ /S)$*

Proof. It is possible to invert a matrix by solving n systems of n linear equations. By a reduction Corollary 3.34 gives us that solving these equations requires T that is $\Omega(n^4 \log(d)\ /S)$. Thus least one of these equations must require T that is $\Omega(n^3 \log(d)\ /S)$ to solve. □

3.5 Quantum matrix multiplication

While many of the applications so far, including the matrix triple product lower bound discussed in the previous section, are derived from the matrix-vector product lower bound, our matrix multiplication lower bound requires a separate argument using ideas from the classical lower bound for the problem in (Abrahamson, 1991).

Implementing this requires a much more subtle way of applying our bucketing method for states that allows us to concentrate on just a subset of the buckets containing most of the total amplitude and ignore the others. As in Section 3.4, our lower bounds in this section apply to a more general model of quantum circuits that can decide which outputs they want to produce in a given layer based on the inputs that they have queried.

Here we consider the matrix multiplication problem $f(A, B) = AB$ where both A and B are considered input. If we could fix a choice of A, we would be able to make our proof somewhat simpler. However, as Abrahamson pointed out in (Abrahamson, 1991), there is a classical algorithm that can compute the function $f(B) = AB$ for any fixed matrix A in $O(n^2)$ queries and $O(n \log d)$ space. Thus our lower bound requires both A and B to be inputs to the function.

Theorem 3.36. *Let $\mathbb{F}$ be a field and $D \subseteq \mathbb{F}$ with $d = |D|$. Then any quantum circuit $\mathcal{C}$ that uses time T and space S and computes the function $f : D^{2n^2} \to \mathbb{F}^{n^2}$ given by $f(A, B) = AB$ with success probability larger than $1/T$ must have T that is $\Omega(n^3\sqrt{\log d \ /S})$.*

Again this theorem follows from the following key lemma, proven in Section 3.5.1, which lets us bound the number of correct output values produced by a shallow quantum circuit.

Lemma 3.37. *Let $\gamma \in (0, 1/2)$ and $f : D^{n^2} \times D^{n^2} \to \mathbb{F}^{n^2}$ for $D \subseteq \mathbb{F}$ with $|D| = d$ be defined by $f(A, B) = AB$. Then for any constant $\beta > 0$ and quantum circuit $\mathcal{C}$ with at most $h = \beta\gamma n\sqrt{k/2}$ queries to input matrices A, B sampled uniformly from D^{n^2}, the probability that A and B are $(\gamma n, \gamma n)$-rigid and $\mathcal{C}$ produces k correct output values of $f(A, B)$ is at most $16 \min(k, n)^{\sqrt{2k}}(4^{H_2(4\beta)}/d^{1-4\beta})^{k/4}$*

Note that for $\beta \leq 0.0184$ we have $1 - 4\beta - 2H_2(4\beta) > 1/6$ so the bound is at most $16 \min(k, n)^{\sqrt{2k}} d^{-k/24}$.

Proof of Theorem 3.36 from Lemma 3.37. Let $\gamma \in (0, 1/2)$ be the constant given by Proposition 3.4. By that proposition, the probability that either of two matrices A and B chosen uniformly randomly from D^{n^2} is not $(\gamma n, \gamma n)$-rigid is at most $2d^{-1}(2/3)^{\gamma n}$. Let $\mathcal{C}$ be a quantum circuit with T queries and space S. Let $\beta = 0.0429$, $d = |D|$, and set $k = \lceil 48(6S + 4)/\log_2 d \rceil$. We partition $\mathcal{C}$ into $\left\lceil T/(\beta\gamma n\sqrt{k/2}) \right\rceil$ sub-circuits that each have at most $\beta\gamma n\sqrt{k/2}$ queries. Without loss of generalities there are at most n^2 such sub-circuits. By combining Proposition 3.5 with Lemma 3.37, we know that for a uniformly random input, the probability that A and B are $(\gamma n, \gamma n)$-rigid matrices and a fixed sub-circuit can produce k outputs is at most $16 \min(k, n)^{\sqrt{2k}} 2^S d^{-k/24} \leq 16k^{\sqrt{2k}} 2^S d^{-k/24}$. Therefore the probability that A and B are $(\gamma n, \gamma n)$-rigid matrices and one of the sub-circuits produces k correct output values is at most $16k^{\sqrt{2k}} 2^S d^{-k/24} n^2$. Combining this with the probability that one of A or B is not $(\gamma n, \gamma n)$-rigid, the probability that there is a sub-circuit that correctly produces k output values is at most

$$16k^{\sqrt{2k}} 2^S d^{-k/24} n^2 + 2d^{-1}(2/3)^{2\gamma n}.$$

Since we can assume without loss of generality that $T \leq n^3$, for sufficiently large n, $2d^{-1}(2/3)^{2\gamma n} \leq 1/(2T)$ and $k^{\sqrt{2k}} \leq 2^{k/48} \leq d^{k/48}$. Plugging in our value of k and the fact that $S \geq \log_2 n$ without loss of generality gives a probability of at most

$$\begin{aligned} 16k^{\sqrt{2k}} 2^{2S} d^{-k/24} n^2 + 2d^{-1}(2/3)^{2\gamma n} &\leq 16 2^S d^{-k/48} n^2 + 1/(2T) \\ &\leq 1/(2T) + 1/(2T) = 1/T. \end{aligned}$$

Since $\mathcal{C}$ must be correct with probability larger than $1/T$, this implies that

$$(k - 1) \left\lceil T/(\beta\gamma n\sqrt{k/2}) \right\rceil \geq n^2.$$

Plugging in our value of k gives us that

$$T \text{ is } \Omega(n^3\sqrt{\log d}/\sqrt{S + \log T}).$$

Since $S \geq \log_2 n$ and our bound trivially holds when T is $\omega(n^3\sqrt{\log d})$ there is a constant $c > 0$ such that $cS \geq \log_2 T$. This implies that T is $\Omega(n^3\sqrt{\log d/S})$ as desired. □

Our quantum lower bound is tightly matched by a classical query algorithm in Proposition 3.66.

3.5.1 The success probability of small depth quantum circuits

We first give an overview of the argument which assumes a uniform distribution over all input matrices A and B in $D^{n\times n}$. Unlike the matrix-vector product proof, in addition to the requirement of k correct output values, for success we also include the extra condition that both matrices must be $(\gamma n, \gamma n)$ rigid. As in the case of the matrix-vector product proof, we decompose the state after $t \leq h = \beta\gamma n\sqrt{k/2}$ steps into orthogonal components based on different values $|i, p, w\rangle$ which determines the k output values produced, though this now can be up to quadratic in n. However, unlike that proof, we need to use the weighted version of our bucketing method. It again suffices to show that for each such $|i, p, w\rangle$ the total fraction of the squared amplitude for any state that is spanned by recording query basis states with at most t non-$\perp$ items can correspond to inputs where there is success is exponentially small in k.

The output values produced determine a set of rows of the matrix A and columns of the matrix B that are relevant. For classical algorithms, where we can determine a set of input locations queried, the lower bound of (Abrahamson, 1991) shows that either at least $k/4$ of the output values lie in rows where few elements of A are queried or $k/4$ lie in columns where few elements of B are queried. For each of these cases ("light" rows or "light" columns). The corresponding output values in those rows or columns are hard to produce in that the requirement that the other matrix is rigid means that the algorithm is exponentially unlikely in k to be correct on those entries.

In the quantum case, when viewed in the recording query basis, the state involves a superposition over all possible assignments to subsets of indices for the relevant rows of A and columns of B with at most t non-$\perp$ entries. For convenience, we first split these basis states depending on whether there are many outputs in light

rows or many in light columns; and then on which rows/columns those are; each determines a set of $k/4$ output values to consider hard and whether to focus on matrix A or B. The number of such possibilities is not too large so the total is not too much larger than the maximum over all such choices. We further consider a fixed choice of the other rigid matrix that maximizes the resulting probability that the hard outputs produced have correct values. The number of consistent recording query basis states in each such superposition is still enormous.

We need to apply bucketing where either A or B is fixed as a rigid matrix and the other can be interpreted as a having a collection of light columns (or rows) such that the output values are the results of a matrix-vector products involving vectors with few queries. However, repeatedly applying the basic bucketing method for basis states we used for matrix-vector products fails because the total number of buckets would be too large since it would end up being the product over the number of choices for each row or column.

Instead, we show that among these potential buckets we can find a small number of admissible buckets that together capture a large portion of the amplitude associated with the state, yielding an admissible bucket t-reduction scheme that lets us derive the final lower bound. We now give the details of this argument.

Proof of Lemma 3.37. Let $C = AB$, $\Pi_{\text{rigid}(A)}$ (and $\Pi_{\text{rigid}(B)}$) be the projection onto inputs where A (and B) are $(\gamma n, \gamma n)$-rigid matrices, and define $\Pi_{\text{rigid}} = \Pi_{\text{rigid } A}\Pi_{\text{rigid } B}$. Assume that $q(w)$ — the output as a function of the measured value of the work register — produces exactly k outputs; we ignore anything it produces after the first k. We will use $[A]$ to denote the set of indices of elements in A and likewise for $[B]$ and $[C]$. By Proposition 3.9, after $t \leq h$ queries in the recording query basis, the state $|\phi_t\rangle$ is a linear combination of basis states $|i, p, w, x_1, \ldots, x_n\rangle$ where $(x_1, \ldots, x_n) \in \Gamma_t$. As in our analysis of the case of matrix-vector products, it will be necessary to be more explicit in our discussion of Γ_t. Each element of Γ_t consists of an assignment $x \in D^E$ and $y \in D^F$ for some subsets $E \subseteq [A]$ and $F \subseteq [B]$ with $|E| + |F| \leq t$ and value $\perp$

on all coordinates in $[A] \setminus E$ and $[B] \setminus F$. Therefore, our state can be written as:

$$|\phi_t\rangle = \sum_{\substack{i,p,w \\ E\subseteq[A],F\subseteq[B] \\ |E|+|F|\leq t \\ x\in D^E, y\in D^F}} \alpha_{i,p,w,E,F,x,y} |i,p,w\rangle |x\rangle_E |\perp\rangle_{[A]\setminus E} |y\rangle_F |\perp\rangle_{[B]\setminus F}$$

for some $\alpha_{i,p,w,E,F,x,y}$ with $\sum_{i,p,w,E,F,x,y} |\alpha_{i,p,w,E,F,x,y}|^2 = 1$. We first apply an analogous series of observations and decompositions to those that allowed us to derive (3.6) from (3.5) in the case of matrix-vector product. By Proposition 3.7, we note that the final state of the algorithm in the standard oracle setting is given by:

$$|\psi_t\rangle = \mathcal{S}|\phi_t\rangle = \mathcal{S} \sum_{\substack{i,p,w \\ E\subseteq[A],F\subseteq[B] \\ |E|+|F|\leq t \\ x\in D^E, y\in D^F}} \alpha_{i,p,w,E,F,x,y} |i,p,w\rangle |x\rangle_E |\perp\rangle_{[A]\setminus E} |y\rangle_F |\perp\rangle_{[B]\setminus F}$$

Because $\mathcal{S}$ behaves as the identity on $|\phi_t\rangle_{\mathcal{C}}$ and each distinct choice of $|i,p,w\rangle$ gives an orthogonal basis state, this equals:

$$\sum_{i,p,w} \beta_{i,p,w} |i,p,w\rangle \otimes \left[S_1^{\otimes 2n^2} \sum_{\substack{E\subseteq[A],F\subseteq[B] \\ |E|+|F|\leq t \\ x\in D^E, y\in D^F}} \beta^{i,p,w}_{E,F,x,y} |x\rangle_E |\perp\rangle_{[A]\setminus E} |y\rangle_F |\perp\rangle_{[B]\setminus F} \right]$$

for some $\beta_{i,p,w}$ and $\beta^{i,p,w}_{E,F,x,y}$ such that $\sum_{i,p,w} |\beta_{i,p,w}|^2 = 1$ and $\sum_{E,F,x,y} |\beta^{i,p,w}_{E,F,x,y}|^2 = 1$ for each i,p,w. Now the probability over the choices of the input matrices and the result of the quantum algorithm making t queries that the matrices A and B are both $(\gamma n, \gamma n)$-rigid and the algorithm produces k correct output values from $C = AB$ is at most:

$$\begin{aligned}
&\left\| \Pi_k \Pi_{\text{rigid}} \mathcal{S} |\phi_t\rangle \right\|^2 \\
&= \left\| \Pi_k \Pi_{\text{rigid}} \sum_{i,p,w} \beta_{i,p,w} |i,p,w\rangle \otimes \left[S_1^{\otimes 2n^2} \sum_{\substack{E\subseteq[A],F\subseteq[B] \\ |E|+|F|\leq t \\ x\in D^E, y\in D^F}} \beta^{i,p,w}_{E,F,x,y} |x\rangle_E |\perp\rangle_{[A]\setminus E} |y\rangle_F |\perp\rangle_{[B]\setminus F} \right] \right\|^2 \\
&= \left\| \sum_{i,p,w} \beta_{i,p,w} |i,p,w\rangle \otimes \left[\Pi_{q(w)} \Pi_{\text{rigid}} S_1^{\otimes 2n^2} \sum_{\substack{E\subseteq[A],F\subseteq[B] \\ |E|+|F|\leq t \\ x\in D^E, y\in D^F}} \beta^{i,p,w}_{E,F,x,y} |x\rangle_E |\perp\rangle_{[A]\setminus E} |y\rangle_F |\perp\rangle_{[B]\setminus F} \right] \right\|^2
\end{aligned}$$

$$= \sum_{i,p,w} |\beta_{i,p,w}|^2 \left\| \left[\Pi_{q(w)} \Pi_{\text{rigid}} S_1^{\otimes 2n^2} \sum_{\substack{E\subseteq[A],F\subseteq[B] \\ |E|+|F|\leq t \\ x\in D^E, y\in D^F}} \beta^{i,p,w}_{E,F,x,y} |x\rangle_E |\perp\rangle_{[A]\setminus E} |y\rangle_F |\perp\rangle_{[B]\setminus F} \right] \right\|^2$$

$$\leq \max_{i,p,w} \left\| \Pi_{q(w)} \Pi_{\text{rigid}} \mathcal{S}_1^{\otimes 2n^2} \sum_{\substack{E\subseteq[A],F\subseteq[B] \\ |E|+|F|\leq t \\ x\in D^E, y\in D^F}} \beta^{i,p,w}_{E,F,x,y} |x\rangle_E |\perp\rangle_{[A]\setminus E} |y\rangle_F |\perp\rangle_{[B]\setminus F} \right\|^2. \tag{3.8}$$

For the rest of the proof we fix an i, p, w to achieve the maximum value in Equation (3.8) and prove an upper bound on the resulting probability. This fixes the output values $q(w)$; we write $G \subseteq [C]$ with $|G| = k$ for the set of indices of the outputs given by $q(w)$. To keep notations simpler in the remainder of the proof we observe that Equation (3.8) is upper bounded by the maximum of

$$\left\| \Pi_{q(G)} \Pi_{\text{rigid}} \mathcal{S}_1^{\otimes 2n^2} \sum_{\substack{E\subseteq[A],F\subseteq[B] \\ |E|,|F|\leq t \\ x\in D^E, y\in D^F}} \beta_{E,F,x,y} |x\rangle_E |\perp\rangle_{[A]\setminus E} |y\rangle_F |\perp\rangle_{[B]\setminus F} \right\|^2 \tag{3.9}$$

over all $\beta_{E,F,x,y}$ with $\sum_{E,F,x,y} |\beta_{E,F,x,y}|^2 = 1$, all sets $G \subseteq [C]$ with $|G| = k$ and all assignments $q(G)$ to G.

We will split the sum in Equation (3.9) over the different sets E and F of queried input indices depending on how they relate to the set of output indices given by G. Let $r(G)$ be the set of rows containing elements of G and $c(G)$ be the set of columns containing elements of G.[6]

Recall our bound $h = \beta\gamma n\sqrt{k/2}$ on the number of queries. We define a *light row of* E to be an element of $r(G)$ that contains at most $\beta\gamma n$ elements of E and define a *light column of* F to be an element of $c(G)$ that contains at most $\beta\gamma n$ elements of F. Since $|E| + |F| \leq t \leq \beta\gamma n\sqrt{k/2}$ we have $\leq \sqrt{k/2}$ rows of E in $r(G)$ and $\leq \sqrt{k/2}$ columns of F in $c(G)$ that are not light. We define $\mathcal{L}(E) \subseteq r(G)$, to be the set of light rows of E and $\mathcal{L}'(F) \subseteq c(G)$ to be the set of light columns of F. Therefore

[6]We will think of $r(G)$ and $c(G)$ as being subsets of indices in $[n]$ that correspond to rows in A and columns of B, respectively, that are relevant for the outputs in G.

$|\{(i',j') \in G \mid i' \notin \mathcal{L}(E),\ j' \notin \mathcal{L}'(F)\}| \leq k/2$ so at least $k/2$ elements of G are in light rows of E or in light columns of F. Therefore for every pair (E,F) at least one of the sets of outputs $G^r_{\mathcal{L}(E)} = \{(i',j') \in G \mid i' \in \mathcal{L}(E)\}$ or $G^c_{\mathcal{L}'(F)} = \{(i',j') \in G \mid j' \in \mathcal{L}'(F)\}$ has size $\geq k/4$.

Let $\mathcal{E}$ be the set of all $E \subseteq [A]$ with $|E| \leq t$ such that G has at least $k/4$ outputs in light rows and $\mathcal{F}$ be the set of all $F \subseteq [B]$ with $|F| \leq t$ such that G has at least $k/4$ outputs in light columns. We separately bound the contribution to Equation (3.9) from pairs (E,F) with $E \in \mathcal{E}$ or $F \in \mathcal{F}$. The analyses of the two cases are completely symmetric up to matrix transposition. It will be convenient to focus on the case $F \in \mathcal{F}$ representing basis states where there are many outputs of G in light columns and compute an upper bound on

$$\left\| \Pi_{q(G)} \Pi_{\text{rigid}} \mathcal{S}_1^{\otimes 2n^2} \sum_{\substack{E \subseteq [A] \\ |E| \leq t \\ x \in D^E}} \sum_{\substack{F \in \mathcal{F} \\ y \in D^F}} \beta_{E,F,x,y} \left|x\right\rangle_E \left|\perp\right\rangle_{[A]\setminus E} \left|y\right\rangle_F \left|\perp\right\rangle_{[B]\setminus F} \right\|^2. \tag{3.10}$$

Basis states where $E \in \mathcal{E}$ give exactly the same upper bound as Equation (3.10) by applying the argument to the transposed product $B^T A^T$ and corresponding transposed sets F^T, E^T, and G^T. Hence, the quantity in Equation (3.9) is at most 4 times that of Equation (3.10).

To upper bound Equation (3.10), we first remove the projection operator $\Pi_{\text{rigid } B}$ from $\Pi_{q(G)}\Pi_{\text{rigid}} = \Pi_{q(G)}\Pi_{\text{rigid } A}\Pi_{\text{rigid } B}$ to get $\Pi_{q(G)}\Pi_{\text{rigid } A}$. We then rewrite this combined projection operator as $\Pi_{q(G)}\Pi_{\text{rigid } A} = \sum_{A\ (\gamma n, \gamma n)\textbf{-rigid}} \Pi_A \otimes \Pi^A_{q(G)}$ where Π_A is the projection onto the specific matrix A and for each A, $\Pi^A_{q(G)}$ is the projection onto the choices for matrix B such that $C = AB$ agrees with $q(w)$. We therefore obtain that Equation (3.10) is at most

$$\left\| \sum_{A\ (\gamma n, \gamma n)\text{-rigid}} (\Pi_A \otimes \Pi^A_{q(G)}) \mathcal{S}_1^{\otimes 2n^2} \sum_{\substack{E \subseteq [A] \\ |E| \leq t \\ x \in D^E}} \sum_{\substack{F \in \mathcal{F} \\ y \in D^F}} \beta_{E,F,x,y} \left|x\right\rangle_E \left|\perp\right\rangle_{[A]\setminus E} \left|y\right\rangle_F \left|\perp\right\rangle_{[B]\setminus F} \right\|^2$$

$$= \left\| \sum_{A\ (\gamma n, \gamma n)\text{-rigid}} (\Pi_A \otimes \Pi^A_{q(G)} \mathcal{S}_1^{\otimes n^2}) \sum_{A' \in (D \cup \{\perp\})^{[A]}} \sum_{\substack{F \in \mathcal{F} \\ y \in D^F}} \beta_{A'} \beta^{A'}_{F,y} \left|A'\right\rangle_{[A]} \left|y\right\rangle_F \left|\perp\right\rangle_{[B]\setminus F} \right\|^2$$

$$= \left\| \sum_{A \ (\gamma n,\gamma n)\text{-rigid}} \beta_A \left|A\right\rangle_{[A]} \otimes [\Pi^A_{q(G)} \mathcal{S}_1^{\otimes n^2} \sum_{\substack{F\in\mathcal{F} \\ |F|\leq t \\ y\in D^F}} \beta^A_{F,y} \left|y\right\rangle_F \left|\perp\right\rangle_{[B]\setminus F}] \right\|^2 \tag{3.11}$$

for some β_A and $\beta^A_{F,y}$ such that $\sum_{A\in(D\cup\{\perp\})^{n^2}} |\beta_A|^2 = 1$ and $\sum_{F\in\mathcal{F}, y\in D^{F0}} |\beta^A_{F,y}|^2 = 1$ for each A. Since $\Pi^A_{q(G)}$ only projects onto the $[B]$ input registers, each distinct choice of $\left|A\right\rangle_{[A]}$ gives orthogonal states so Equation (3.11) equals

$$\sum_{A \ (\gamma n,\gamma n)\text{-rigid}} |\beta_A|^2 \left\| \Pi^A_{q(G)} \mathcal{S}_1^{\otimes n^2} \sum_{\substack{F\in\mathcal{F} \\ |F|\leq t \\ y\in D^F}} \beta^A_{F,y} \left|y\right\rangle_F \left|\perp\right\rangle_{[B]\setminus F} \right\|^2$$

$$\leq \max_{A \ (\gamma n,\gamma n)\text{-rigid}} \left\| \Pi^A_{q(G)} \mathcal{S}_1^{\otimes n^2} \sum_{\substack{F\in\mathcal{F} \\ y\in D^F}} \beta^A_{F,y} \left|y\right\rangle_F \left|\perp\right\rangle_{[B]\setminus F} \right\|^2 \tag{3.12}$$

We fix a $(\gamma n, \gamma n)$-rigid matrix A that maximizes (3.12) and partition the set $\mathcal{F}$ based on the set $\mathcal{L}'(F)$ which contains all but at most $\left\lfloor \sqrt{k/2} \right\rfloor$ columns in $c(G)$. Therefore we can rewrite (3.12) as

$$\left\| \sum_{\substack{H\subseteq c(G) \\ \text{s.t. } |H|\leq \left\lfloor \sqrt{k/2} \right\rfloor}} \Pi^A_{q(G)} \mathcal{S}_1^{\otimes n^2} \sum_{\substack{F\in\mathcal{F} \\ \mathcal{L}'(F)=c(G)\setminus H \\ y\in D^F \text{s.t. } \mathcal{L}'(F)=c(G)\setminus H}} \beta^A_{F,y} \left|y\right\rangle_F \left|\perp\right\rangle_{[B]\setminus F} \right\|^2. \tag{3.13}$$

Since $|c(G)| \leq \min(k, n)$ we can upper bound (3.13) by

$$\min(k,n)^{\sqrt{2k}} \cdot \max_{\substack{H\subseteq c(G) \\ \text{s.t. } |H|\leq \left\lfloor \sqrt{k/2} \right\rfloor}} \left\| \Pi^A_{q(G)} \mathcal{S}_1^{\otimes n^2} \sum_{\substack{F\in\mathcal{F} \\ y\in D^F \\ \text{s.t.}\mathcal{L}'(F)=c(G)\setminus H}} \beta^A_{F,y} \left|y\right\rangle_F \left|\perp\right\rangle_{[B]\setminus F} \right\|^2. \tag{3.14}$$

We fix the set H achieving the maximum value in Equation (3.14), which fixes the value of $\mathcal{L}'(F) = c(G) \setminus H$. This fixes the set $G^c_{\mathcal{L}'(F)}$ of elements in G that are in light columns of F (equivalently, not in H) which, since $F \in \mathcal{F}$, contains at least $k/4$ elements of G. Let G' be a fixed subset of $k/4$ of the elements of $G^c_{\mathcal{L}'(F)}$. By

construction we have $c(G') \subseteq \mathcal{L}'(F)$. By only requiring that the outputs in G' are correct, we therefore can upper bound $\left\|\Pi_k \Pi_{\text{rigid}} \mathcal{S} \left|\phi_t\right\rangle\right\|^2$ by the maximum value of

$$4 \min(k,n)^{\sqrt{2k}} \left\| \Pi^A_{q(G')} \mathcal{S}_1^{\otimes n^2} \sum_{\substack{F \subseteq [B] \\ c(G') \subseteq \mathcal{L}'(F) \\ y \in D^F}} \beta'_{F,y} \left|y\right\rangle_F \left|\perp\right\rangle_{[B]\setminus F} \right\|^2 \tag{3.15}$$

over all $G' \subseteq [C]$ with $|G'| = k/4$ and $\beta'_{F,y}$ with $\sum_{F,y} |\beta'_{F,y}|^2 = 1$.

For each $j \in c(G')$, let k_j be the number of elements of G' in column j. Our overall strategy is to consider the $j \in c(G')$ one by one, and show that the total amplitude on states where these k_j outputs are correct conditioned on the success for previous values of j is of the form $d^{-\delta k_j}$ for some fixed constant $\delta > 0$. These are k_j outputs of the matrix-vector product Ay^j where y^j is the j-th column of B and the fact that $c(G') \subseteq \mathcal{L}'(F)$ implies that F has made at most $\beta\gamma n$ queries to y^j. This is very similar to the situation with the matrix-vector problem from Lemma 3.22. In analogy with the Lemma 3.22, we define U^j to be the set of k_j rows containing outputs of G' in column j.

Applying Lemma 3.24 with $c = 1$, for each $j \in c(G')$ there is a collection $V^j_1, \ldots, V^j_{\ell_j}$ of $\ell_j = \lceil \gamma n / k_j \rceil$ k_j-subsets of $[n]$ such that the $k_j \times k_j$ sub-matrix $A_{U^j V^j_i}$ has full rank.

Using the ideas of Lemma 3.22 we could bucket the possible basis states into one bucket for each large subset of the set associated with the tuple $(V^j_{i_j})_{j \in q(G')}$ using Lemmas 3.23 and 3.24 and bound each bucket separately. However, unlike its use in the proof of Lemma 3.22, the value of many of the k_j can be very small, as low as 1, in which case the upper bounds using Lemmas 3.23 and 3.24 would yield a probability bound larger than 1.

Instead, we need a stronger argument that depends on the amplitudes $\beta'_{F,y}$ in Equation (3.15). The large subsets of the sets associated with tuples $(V^j_{i_j})_{j \in q(G')}$ yield candidate buckets but there are too many of them to be used. However, we

will see in the following lemma that a relatively small collection of them can capture all but a constant fraction of the total amplitude given by the $\beta'_{F,y}$. We will then see, in Corollary 3.39, how this can be applied inductively with the portion of the total amplitude that is left over to yield a good upper bound on the total probability of producing the output values in $q(G')$, which is what we need to prove. (In the terminology of Section 3.3, Lemma 3.38 describes an admissible bucket t-reduction scheme for $q(G')$, deriving some of its implications in parallel with its construction. On the other hand, Corollary 3.39 describes how that yields the overall bound; this is essentially a combination of the ideas of Lemmas 3.16 and 3.18.)

Lemma 3.38. *Let $G' \subseteq [C]$ with $|G'| = k/4$ and $\mathcal{F}'$ be a set of $F \subseteq [B]$ such that $c(G') \subseteq \mathcal{L}'(F)$. Suppose further that $\sum_{F\in\mathcal{F}', y\in D^F} |\delta_{F,y}|^2 = 1$ for some $\delta_{F,y}$. Define $\alpha = 4\beta$. Then there is a $\mathcal{F}'' \subseteq \mathcal{F}'$ and coefficients $\delta'_{F,y}$ such that $\sum_{F\in\mathcal{F}'', y\in D^F} |\delta'_{F,y}|^2 = 1$ and*

$$\left\|\Pi^A_{q(G')}\mathbb{S}_1^{\otimes n^2}\sum_{\substack{F\in\mathcal{F}'\\ y\in D^F}}\delta_{F,y}\,|y\rangle_F\,|\perp\rangle_{[B]\setminus F}\right\|^2 \leq \frac{2^{1+H_2(\alpha)\,k/2}}{d^{(1-\alpha)\,k/4}} + \frac{1}{2}\left\|\Pi^A_{q(G')}\mathbb{S}_1^{\otimes n^2}\sum_{\substack{F\in\mathcal{F}''\\ y\in D^F}}\delta'_{F,y}\,|y\rangle_F\,|\perp\rangle_{[B]\setminus F}\right\|^2. \tag{3.16}$$

Proof. We first recall the definitions in our discussion preceding the lemma statement. For each $j \in c(G')$, define U^j to be the set of row indices of G' in column j and let $k_j = |U_j|$. Define $\ell_j = \lceil \gamma n/k_j \rceil$, apply Lemma 3.24 for each j, and let $V_1^j, \ldots, V_{\ell_j}^j$ be the collection of disjoint subsets of $[n]$ of size k_j found for each j such that each $k_j \times k_j$ sub-matrix $A_{U^j V_i^j}$ has full rank.

For each $F \in \mathcal{F}'$ and $i \in c(G')$, define F^j to be the set of row indices of elements of F in column j; since $c(G') \subseteq \mathcal{L}'(F)$, we have $|F^j| \leq \beta\gamma n$. For each $i \in [\ell_j]$ define

$$m_i^j = \sum_{F\in\mathcal{F}',\ y\in D^F} |\delta_{F,y}|^2 \cdot |F^j \cap V_i^j|.$$

Since $\sum_{F,y} |\delta_{F,y}|^2 = 1$, m_i^j can be viewed as the expected size of the overlap between the recorded queries in the j-th column of the matrix B and each V_i^j. Since for each

j, the sets V_i^j are disjoint and $|F^j| \le \beta\gamma n$ we have $\sum_{i\in[\ell_j]} m_i^j \le \beta\gamma n$. Therefore, for each j, we have some index $i_j \in [\ell_j]$ such that $m_{i_j}^j \le \beta\gamma n/\ell_j \le \beta k_j$.

Since $\sum_{j\in c(G')} k_j = |G'| = k/4$, the expected total overlap between the recorded queries in the columns of G' and the chosen sets $V_{i_j}^j$ for those columns is $\sum_j m_{i_j}^j \le \sum_j \beta k_j = \beta k/4$. Define $\mathcal{F}''$ to be the set of $F \in \mathcal{F}'$ such that $\sum_j |F^j \cap V_{i_j}^j| \ge \alpha k/4 = \beta k$. By Markov's inequality we have

$$\sum_{F\in\mathcal{F}'',\ y\in D^F} |\delta_{F,y}|^2 \le \frac{\sum_j m_{i_j}^j}{\beta k} \le 1/4. \tag{3.17}$$

We split our analysis for $\mathcal{F}'$ into two parts due to sets F in $\mathcal{F}''$ and $\mathcal{F}' \setminus \mathcal{F}''$, respectively.

We begin with $F \in \mathcal{F}''$. Write $\kappa = \sum_{F\in\mathcal{F}'',\ y\in D^F} |\delta_{F,y}|^2 \le 1/4$. For $F \in \mathcal{F}''$, define $\delta'_{F,y} = \frac{1}{\sqrt{\kappa}}\delta_{F,y}$. Then $\sum_{F\in\mathcal{F}'',y\in D^F} |\delta'_{F,y}|^2 = 1$ and

$$\begin{aligned}\left\|\Pi^A_{q(G')}\mathcal{S}_1^{\otimes n^2} \sum_{\substack{F\in\mathcal{F}''\\ y\in D^F}} \delta_{F,y} |y\rangle_F |\perp\rangle_{[B]\setminus F}\right\|^2 &= \kappa \left\|\Pi^A_{q(G')}\mathcal{S}_1^{\otimes n^2} \sum_{\substack{F\in\mathcal{F}''\\ y\in D^F}} \delta'_{F,y} |y\rangle_F |\perp\rangle_{[B]\setminus F}\right\|^2 \\ &\le \frac{1}{4}\left\|\Pi^A_{q(G')}\mathcal{S}_1^{\otimes n^2} \sum_{\substack{F\in\mathcal{F}''\\ y\in D^F}} \delta'_{F,y} |y\rangle_F |\perp\rangle_{[B]\setminus F}\right\|^2. \end{aligned} \tag{3.18}$$

We now consider $\mathcal{F}' \setminus \mathcal{F}''$. By definition, for $F \in \mathcal{F}' \setminus \mathcal{F}''$, we have $\sum_j |F^j \cap V_{i_j}^j| < \alpha k/4$. By definition we have $\sum_j |V_{i_j}^j| = \sum_j k_j = k/4$ so F must miss more than $(1-\alpha)k/4$ elements of the set $V = \bigcup_j (V_{i_j}^j \times \{\, j \,\})$ of size $k/4$. For each subset V' of V of size $k/4 - \lfloor \alpha k/4 \rfloor$ we define a bucket $\mathcal{B}_{V'}$ that contains sets F that must miss the elements of V' and assign each $F \in \mathcal{F}' \setminus \mathcal{F}''$ to a unique bucket in an arbitrary fixed way. There are at most $2^{H_2(\alpha)k/4}$ such buckets. Then

$$\begin{aligned}&\left\|\Pi^A_{q(G')}\mathcal{S}_1^{\otimes n^2} \sum_{\substack{F\in\mathcal{F}'\setminus\mathcal{F}''\\ y\in D^F}} \delta_{F,y} |y\rangle_F |\perp\rangle_{[B]\setminus F}\right\|^2 \\ &\le \left(\sum_{\substack{V'\subseteq V\\ |V'|=k/4-\lfloor\alpha k/4\rfloor}} \left\|\Pi^A_{q(G')}\mathcal{S}_1^{\otimes n^2} \sum_{\substack{F\in\mathcal{B}_{V'}\\ y\in D^F}} \delta_{F,y} |y\rangle_F |\perp\rangle_{[B]\setminus F}\right\|\right)^2\end{aligned}$$

$$\leq 2^{H_2(\alpha)\,k/2} \cdot \sum_{\substack{V' \subseteq V \\ |V'|=k/4-\lfloor \alpha k/4 \rfloor}} \left\| \Pi^A_{q(G')} \mathcal{S}_1^{\otimes n^2} \sum_{\substack{F \in \mathcal{B}_{V'} \\ y \in D^F}} \delta_{F,y} \left|y\right\rangle_F \left|\perp\right\rangle_{[B]\setminus F} \right\|^2$$

$$= 2^{H_2(\alpha)\,k/2} \cdot \sum_{\substack{V' \subseteq V \\ |V'|=k/4-\lfloor \alpha k/4 \rfloor}} \left\| \Pi^A_{q(G')} \mathcal{S}_1^{\otimes n^2} \left|\perp\right\rangle_{V'} \sum_{\substack{F \in \mathcal{B}_{V'} \\ y \in D^F}} \delta_{F,y} \left|y\right\rangle_F \left|\perp\right\rangle_{[B]\setminus (F \cup V')} \right\|^2 \qquad (3.19)$$

where we first used the triangle inequality followed by Jensen's inequality.

Now, applying the $\mathcal{S}_1^{\otimes n^2}$ operator in (3.19) will convert the $\left|\perp\right\rangle_{V'}$ to a uniform superposition of all $\left|y'\right\rangle_{V'}$ for all $y' \in D^{V'}$ and convert $\sum_{\substack{F \in \mathcal{B}_{V'} \\ y \in D^F}} \delta_{F,y} \left|y\right\rangle_F \left|\perp\right\rangle_{[B]\setminus (F \cup V')}$ to some superposition of $\left|y''\right\rangle \in D^{[B]\setminus V'}$ with amplitudes some $\delta_{V',y''}$ such that $\sum_{y''} |\delta_{V',y''}|^2 = \sum_{F \in \mathcal{B}_{V'}, y \in D^F} |\delta_{F,y}|^2$. Therefore, we can rewrite (3.19) as

$$2^{H_2(\alpha)\,k/2} \cdot \sum_{\substack{V' \subseteq V \\ |V'|=k/4-\lfloor \alpha k/4 \rfloor}} \left\| \Pi^A_{q(G')} \left[\sum_{y' \in D^{V'}} \frac{1}{\sqrt{d^{|V'|}}} \left|y'\right\rangle_{V'} \right] \otimes \sum_{y'' \in D^{[n]\setminus V'}} \delta_{V',y} \left|y\right\rangle_{[B]\setminus V'} \right\|^2 . \qquad (3.20)$$

We now consider the application of $\Pi^A_{q(G')}$. Let $V'_j \subseteq V^j_{i_j}$ be the set of row indices in column j of $V' \subseteq [B]$ and consider the corresponding set of columns in A. Since $A_{U^j V^j_{i_j}}$ has full rank, there is a subset $U^j_0 \subseteq U^j$ with $|U^j_0| = |V'_j|$ so that $A_{U^j_0 V'_j}$ also has full rank. Now define $G'_0 \subseteq G'$ to be $\bigcup_{j \in c(G)} (U_j \times \{\, j \,\})$ which has size $|V'|$.

For each j, the outputs in $U_j \times \{\, j \,\} \subset [C]$ can be expressed as the matrix-vector product $A_{U^j_0 V'_j} y^j_{V'_j} + M'$ for some $|V'_j| \times |V'_j|$ matrix M' defined by the product of the $U^j_0 \times ([n] \setminus V'_j)$ submatrix of the fixed matrix A and $y^j_{[n]\setminus V'_j}$. Since $A_{U^j_0 V'_j}$ is full rank, for each value of M' given by $y^j_{[n]\setminus V'_j}$, there is precisely one value of $y^j_{V'_j}$ that will yield the output values $q(U_j \times \{\, j \,\})$. Therefore, putting the properties for the columns of $c(G')$ together, there is precisely one value $y' \in D^{V'}$ that will yield the output values $q(G'_0)$.

(In the terminology of Section 3.3, this says that each of the $2^{H_2(\alpha)k/4}$ buckets $\mathcal{B}_{V'}$ corresponds to a c-admissible bucket for $q(G')$ with $c = d^{(1-\alpha)/4}$. Equation (3.17) means that the squared amplitude of the projection on the set $\mathcal{F}''$ corresponding to recording query basis states not associated with these buckets has total squared

amplitude at most $1/4$ and hence total amplitude at most $1/2$. Thus, this construction produces a c-admissible bucket t-reduction scheme of size $\ell = 2^{H_2(\alpha)k/4}$.)

It follows that, (3.20) is at most

$$
\begin{aligned}
&2^{H_2(\alpha)\,k/2} \cdot \sum_{\substack{V' \subseteq V \\ |V'|=k/4-\lfloor \alpha k/4 \rfloor}} \left\| \Pi^A_{q(G'_0)} \left[\sum_{y' \in D^{V'}} \frac{1}{\sqrt{d^{|V'|}}} \left| y' \right\rangle_{V'} \right] \otimes \sum_{y'' \in D^{[n]\setminus V'}} \delta_{V',y} \left| y \right\rangle_{[B]\setminus V'} \right\|^2 \\
&= 2^{H_2(\alpha)\,k/2} \cdot \sum_{\substack{V' \subseteq V \\ |V'|=k/4-\lfloor \alpha k/4 \rfloor}} \left\| \frac{1}{\sqrt{d^{|V'|}}} \sum_{y'' \in D^{[n]\setminus V'}} \delta_{V',y} \left| y \right\rangle_{[B]\setminus V'} \right\|^2 \\
&= 2^{H_2(\alpha)\,k/2} \cdot \sum_{\substack{V' \subseteq V \\ |V'|=k/4-\lfloor \alpha k/4 \rfloor}} \frac{1}{d^{|V'|}} \sum_{y'' \in D^{[n]\setminus V'}} |\delta_{V',y}|^2 \\
&= 2^{H_2(\alpha)\,k/2} \cdot \sum_{\substack{V' \subseteq V \\ |V'|=k/4-\lfloor \alpha k/4 \rfloor}} \frac{1}{d^{|V'|}} \sum_{F \in \mathcal{B}_{V'}, y \in D^F} |\delta_{F,y}|^2 \\
&= 2^{H_2(\alpha)\,k/2} \cdot \frac{1}{d^{|V'|}} \sum_{F \in \mathcal{F}' \setminus \mathcal{F}'', y \in D^F} |\delta_{F,y}|^2 \\
&\leq 2^{H_2(\alpha)\,k/2} / d^{(1-\alpha)\,k/4}
\end{aligned}
\tag{3.21}
$$

where the last equality follows since the buckets $\mathcal{B}_{V'}$ partition $\mathcal{F}' \setminus \mathcal{F}''$.

We now combine the contributions from $\mathcal{F}''$ and $\mathcal{F}' \setminus \mathcal{F}''$. Applying Jensen's inequality together with the bounds in (3.18) and (3.21) we obtain that

$$
\begin{aligned}
&\left\| \Pi^A_{q(G')} \mathbb{S}_1^{\otimes n^2} \sum_{\substack{F \in \mathcal{F}' \\ y \in D^F}} \delta_{F,y} \left| y \right\rangle_F \left| \perp \right\rangle_{[B]\setminus F} \right\|^2 \\
&\leq 2 \left[\left\| \Pi^A_{q(G')} \mathbb{S}_1^{\otimes n^2} \sum_{\substack{F \in \mathcal{F}' \setminus \mathcal{F}'' \\ y \in D^F}} \delta_{F,y} \left| y \right\rangle_F \left| \perp \right\rangle_{[B]\setminus F} \right\|^2 + \left\| \Pi^A_{q(G')} \mathbb{S}_1^{\otimes n^2} \sum_{\substack{F \in \mathcal{F}'' \\ y \in D^F}} \delta_{F,y} \left| y \right\rangle_F \left| \perp \right\rangle_{[B]\setminus F} \right\|^2 \right] \\
&\leq \frac{2^{1+H_2(\alpha)\,k/2}}{d^{(1-\alpha)\,k/4}} + \frac{1}{2} \left\| \Pi^A_{q(G')} \mathbb{S}_1^{\otimes n^2} \sum_{\substack{F \in \mathcal{F}'' \\ y \in D^F}} \delta'_{F,y} \left| y \right\rangle_F \left| \perp \right\rangle_{[B]\setminus F} \right\|^2
\end{aligned}
$$

as required. □

Corollary 3.39. *Let $G' \subseteq [C]$ with $|G'| = k/4$, $\mathcal{F}'$ be a set of $F \subseteq [B]$ such that $c(G') \subseteq \mathcal{L}'(F)$, and $\sum_{F\in\mathcal{F}', y\in D^F} |\delta_{F,y}|^2 = 1$ for some $\delta_{F,y}$. Then*

$$\left\|\Pi^A_{q(G')}\mathcal{S}_1^{\otimes n^2} \sum_{\substack{F\in\mathcal{F}'\\ y\in D^F}} \delta_{F,y} |y\rangle_F |\perp\rangle_{[B]\setminus F}\right\|^2 \leq 2^{2+H_2(4\beta)\,k/2}/d^{(1-4\beta)\,k/4}.$$

Proof. Let M be the maximum value of

$$\left\|\Pi^A_{q(G')}\mathcal{S}_1^{\otimes n^2} \sum_{F\in\mathcal{F}',\ y\in D^F} \delta_{F,y} |y\rangle_F |\perp\rangle_{[B]\setminus F}\right\|^2$$

over all choices of $\mathcal{F}'$ and $\delta_{F,y}$ with the required properties. This corollary follows from Lemma 3.38 by observing that the right-hand term in Equation (3.16) multiplied by $1/2$ is also upper bounded by M and hence $M \leq 2^{1+H_2(4\beta)\,k/2}/d^{(1-4\beta)\,k/4} + M/2$. □

Finally, plugging the bound from Corollary 3.39 into (3.15), we obtain that the probability that A and B are both $(\gamma n, \gamma n)$-rigid and $\mathcal{C}$ produces k correct output values for $C = AB$, $\left\|\Pi_k \Pi_{\text{rigid}} \mathcal{S} |\phi_t\rangle\right\|^2$, is at most

$$16\ \min(k,n)^{\sqrt{2k}} \left(\frac{4^{H_2(4\beta)}}{d^{(1-4\beta)}}\right)^{k/4}$$

as desired. □

3.5.2 Related time-space tradeoffs

Now we use Theorem 3.36 to prove some related quantum linear algebra lower bounds. Constructions of matching upper bounds can be found in Section 3.7.

Corollary 3.40. *Let $\mathbb{F}$ be a field and $D \subseteq \mathbb{F}$ with $d = |D|$. If $\mathcal{C}$ is a quantum circuit that computes the function $f : D^{n^2} \to \mathbb{F}^{n^2}$ where $f(A) = A^2$ on all upper triangular inputs in time T and space S with success probability at least $1/T$, then T must be $\Omega(n^3\sqrt{\log d\ /S})$.*

Proof. Let $A, B \in D^{n^2}$ and construct the $3n \times 3n$ matrix:

$$M = \begin{bmatrix} 0 & A & 0 \\ 0 & 0 & B \\ 0 & 0 & 0 \end{bmatrix}$$

Since the top right $n \times n$ sub-matrix of M^2 is equal to the product AB, we get a reduction from matrix multiplication and can apply Theorem 3.36 to derive the lower bound. □

3.6 Quantum tradeoffs for Boolean matrix operations

In this section we focus on Boolean matrix operations, which use (AND, OR) inner product of vectors rather than the usual $(+, \times)$ inner product. We denote this Boolean inner product of vectors u and v by $u \bullet v$ and extend this notation to Boolean matrix-vector product and Boolean matrix multiplication. For $u, v \in \{0,1\}^n$, $u \bullet v = 1$ if and only if the subsets of $[n]$ encoded by u and v intersect, so the problems of computing Boolean matrix multiplication and Boolean matrix-vector product can be seen as computing many correlated copies of the set disjointness problem.

3.6.1 Tradeoffs for Boolean matrix multiplication

Unlike what we have shown for algebraic problems, as noted in (Klauck et al., 2007), quantum algorithms for Boolean matrix multiplication have better time-space tradeoff properties than their classical counterparts.

Proposition 3.41. *For any $c > 0$, there are quantum circuits computing $n \times n$ Boolean matrix multiplication $A \bullet B$ with error at most n^{-c} using space $O(\log n)$ and a number of queries T that is $O(n^{2.5} \log n)$.*

Proof. Fix $c > 0$. Each of the n^2 entries in the product is a disjointness function of length n that can be computed with error at most n^{-c-2} and space $O(\log n)$ using Grover's algorithm in time $O(\sqrt{n} \log n)$ for error at most n^{-c} overall. □

This is in contrast to the following result of Abrahamson which shows that classical algorithms as fast as this quantum algorithm require space $\tilde{\Omega}(n^{0.5})$ rather than $O(\log n)$.

Proposition 3.42 ((Abrahamson, 1990)). *There is a probability distribution on input matrices and constants $0 < c_1 < c_2$ under which the best classical algorithms (branching programs) for Boolean matrix multiplication $A \bullet B$ using time T and space S require $T \cdot S$ that is* $\begin{cases} \Theta(n^{3.5}) & \text{for } T \leq c_1 n^{2.5} \\ \Theta(n^3) & \text{for } T \geq c_2 n^{2.5}. \end{cases}$

For quantum circuits, Klauck, Špalek, and de Wolf (Klauck et al., 2007) proved the following time-space tradeoff lower bound which proves that the quantum algorithm in Proposition 3.41 is nearly optimal when the space S is $O(\log n)$.

Proposition 3.43 ((Klauck et al., 2007)). *Any bounded error quantum circuit that computes the $n \times n$ Boolean matrix multiplication $A \bullet B$ with T queries and space S requires T to be $\Omega(n^{2.5}/S^{0.5})$.*

A key difference between the methods used in Abrahamson's bounds and our results for linear algebra versus those in this proof is that we require that the set of output values produced in each part of the computation is fixed independent of the input. (See our discussion of such output-oblivious computation in Section 3.2.1.) Such an assumption was essential for the quantum time-space lower bounds in (Klauck et al., 2007; Ambainis et al., 2009), although the bound for multiple disjoint collision pairs in (Hamoudi and Magniez, 2021) and our results in Sections 3.4 and 3.5 apply to quantum query algorithms without such a restriction on output production. Fixing the output values produced in each part of the computation allows one to go beyond using a single hard distribution on inputs, and instead choose hard distributions for each part of the computation depending on the target outputs. To give a sense of how this works we sketch the lower bound method of (Klauck et al., 2007) for Boolean matrix multiplication, which relies on a strong direct product lemma for the function OR_n^k (i.e. k independent copies of the OR function each on inputs of size n):

Proposition 3.44 (Strong Direct Product Theorem for OR_n^k (Klauck et al., 2007))**.** *There are positive constants ε and γ such that the following hold:*

(a) Any randomized algorithm making at most εkn queries has success probability at most $2^{-\gamma k}$ in computing OR_n^k.

(b) Any quantum algorithm making at most $\varepsilon k\sqrt{n}$ queries has success probability at most $2^{-\gamma k}$ in computing OR_n^k.

Proof sketch for Proposition 3.43. For any integer $k \leq n/2$, the function $OR^k_{\lfloor n/k \rfloor}$ can be embedded in any set $E \subseteq [n] \times [n]$ of k outputs of the $n \times n$ Boolean matrix product $A \bullet B$ as follows: Begin by dividing $[n]$ into k blocks $b_1, \ldots, b_k$ each of size $\lfloor n/k \rfloor$ (together with at most $k-1$ other elements) and associate each $(i,j) \in E$, with a distinct index $\ell = \ell(i,j) \in [k]$. For each $(i,j) \in E$, for $\ell = \ell(i,j)$ set every entry in A_{i,b_ℓ} to 1 and set the vector of inputs in $B_{b_\ell,j}$ to the ℓ-th block of the input to $OR^k_{\lfloor n/k \rfloor}$. Set all other bits in A and B to 0. It is easy to see that the k outputs indexed by E will be the outputs for k disjoint OR functions on $\lfloor n/k \rfloor$ bits.

Without loss of generality one can assume that the space bound S is at most αn for some small constant $\alpha > 0$ since the number of queries must be $\Omega(n^2)$ in the worst case[7]. Choose $k = cS$ for some suitably large constant c that depends on the constant γ in Proposition 3.44. Begin by slicing the circuit into layers of $\varepsilon\sqrt{kn}$ queries each. There are $\Theta(T/\sqrt{kn})$ such layers. By Proposition 3.44 and the embedding, any circuit of depth $\varepsilon\sqrt{kn} = \varepsilon k\sqrt{n/k}$ queries can produce k correct output values with probability only $2^{-\gamma k}$ for some $\gamma > 0$. This is the same depth as each of the layers but each layer also gets an S qubit input-dependent state to begin. By Proposition 3.5, the probability that the resulting layer can produce k correct output values is at most $2^S 2^{-\gamma k}$ which is at most 2^{-S} if the constant c used in defining k is sufficiently large.

[7]Note that this is not completely obvious since quantum algorithms for some problems may have a sublinear number of queries.

Therefore, the total number of correct output values that can be produced with probability larger than 2^{-S} must be $O(T/\sqrt{kn}) \cdot k$ which is $O(T\sqrt{S/n})$. On the other hand this number of outputs produced must be at least n^2. It follows that T must be $\Omega(n^{2.5}/\sqrt{S})$. □

Our improved lower bound

Theorem 3.45. *Any quantum circuit computing $n \times n$ Boolean matrix multiplication $A \bullet B$ with T queries and space S and success probability more than 2^{-S} must have T that is $\Omega(n^{2.5}/S^{1/4})$.*

Though the form of our lower bound may seem somewhat unusual, both the exponent of n and that of S are optimal: The algorithm of Proposition 3.41 shows that exponent of n is optimal since there is only a gap of $O(\log^{5/4} n)$ for space $\Theta(\log n)$. In our quantum query model, at the other end of the scale, an algorithm with space $3n^2$ can query and completely remember both matrices in $2n^2$ time and $2n^2$ space, after which a single global unitary transformation will produce the n^2 bits of output needed in the remaining n^2 qubits of working memory; hence the exponent of $1/4$ on S cannot be reduced.

Theorem 3.45 follows from the following key lemma which improves on the corresponding bound in (Klauck et al., 2007) by a factor of $\Theta(k^{1/4})$.

Lemma 3.46. *There are constants $\varepsilon, \gamma > 0$ such that the following holds. Let $k < n^2/100$ be an integer. For any quantum circuit $\mathcal{C}$ with at most $\varepsilon k^{3/4} n^{1/2}$ queries to x, the probability that $\mathcal{C}$ produces k correct output values of $n \times n$ Boolean matrix multiplication $A \bullet B$ is at most $2^{-\gamma k}$.*

We first see how this lemma suffices for the theorem:

Proof of Theorem 3.45 via Lemma 3.46. Since there are n^2 outputs, it seems that $T \geq n^2$ queries are required, but that isn't quite obvious. Nonetheless, we can, for

example, derive a $T = \Omega(n^2)$ lower bound by applying Lemma 3.46 with $k = n^2/101$ which shows that a circuit with at most some βn^2 queries can only achieve exponentially small success probability for producing a small fraction of the output. Therefore without loss of generality we can assume that $\sqrt{S} < \alpha n$ for some arbitrarily small constant $\alpha > 0$. Let ε and γ be the constants from Lemma 3.46. Let $c = 2/\gamma$ and define $k = cS$. Therefore for $\alpha \leq 1/(10\sqrt{c})$ we obtain that $5\sqrt{k} = 5\sqrt{cS} < n/2$. By Lemma 3.46, since $k < n^2/100$, any quantum query algorithm with at most $\varepsilon k^{3/4} n^{1/2}$ queries has success probability at most $2^{-\gamma k} = 2^{-2S}$ of producing k correct output values.

We prove the contrapositive of the theorem statement: Suppose that $T \leq \varepsilon n^{2.5}/(cS)^{1/4} = \varepsilon n^{2.5}/k^{1/4}$. When we divide $\mathcal{C}$ into layers with $\varepsilon k^{3/4} n^{1/2}$ quantum queries each, there are at most n^2/k layers. Since there are a total of n^2 outputs, there must be some layer i during which at least k outputs are produced. Let E be the set of the first k outputs produced in layer i. By the argument above since the space is at most S, by Proposition 3.5 the probability that these k outputs are correct given the S qubits of input-dependent initial state at the beginning of layer i is at most 2^S times larger than that of a circuit without them and the same number of queries, which is at most $2^S \cdot 2^{-2S} = 2^{-S}$ which is what we needed to show. □

The main idea behind the proof of this key lemma is an improved method for embedding the direct product of OR functions into outputs of the Boolean matrix multiplication problem; this uses the following definition of an L-coloring of subsets of $[n] \times [n]$.

Definition 3.47. For $E \subseteq [n] \times [n]$ an *L-coloring* of E is a map $\chi : E \to [L]$ such that

- within each color class either all rows are distinct or all columns are distinct, and

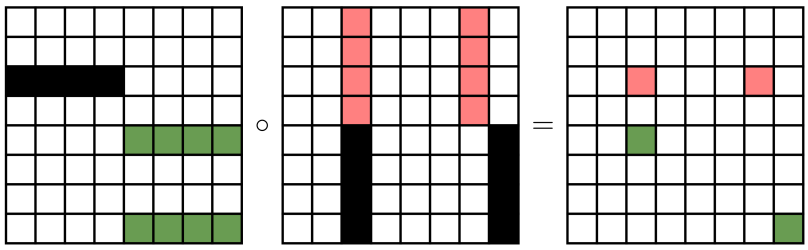

Figure 3.2: An example of a valid 3-coloring (as in Definition 3.47), where the pink and green squares on the right matrix correspond to the colored outputs. For the left two matrices, the black squares are fixed to the input 1 while the white square are fixed to the input 0. The pink and green squares in the left two matrices encode an input to OR_4^4 whose outputs are the colored entries of the right matrix.

- for each color ℓ there is a rectangle given by sets $R_\ell \subseteq [n]$ of rows and $C_\ell \subseteq [n]$ of columns such that the set of points of color ℓ is precisely $E \cap (R_\ell \times C_\ell)$.

(Note that the rectangles $R_\ell \times C_\ell$ may overlap, but their overlap must not contain any points in E, see Figure 3.2.)

We say that a rectangle $R \times C \in [n] \times [n]$ is *colorable* iff $E \cap (R \times C)$ either has all its elements in different rows or all its elements in different columns.

The motivation for this definition is given by the following lemma.

Lemma 3.48. *Let $E \subseteq [n] \times [n]$ with $|E| = k$ and L be an integer with $L \leq n/2$. If E has an L-coloring then $OR^k_{\lfloor n/L \rfloor}$ is a sub-function of the function that produces the k outputs of $A \bullet B$ indexed by E for $n \times n$ Boolean matrices A and B.*

Proof. Write $E = \dot{\bigcup}_{\ell=1}^{L} E_\ell$ where E_ℓ is the set of (i, j) in E in color class ℓ. We now divide $[n]$ into L disjoint blocks $b_1, \ldots, b_L$ of at least $\lfloor n/L \rfloor \geq 2$ elements each. Given the coloring and division into blocks, we define a partial assignment to the matrices A and B as follows:

- If color class ℓ consists of points that do not share a column, for each $(i, j) \in E_\ell$, we set all entries of A_{i,b_ℓ} to 1 and leave all entries of $B_{b_\ell,j}$ unset.

- If color class ℓ consists of points that do not share a row, for each $(i,j) \in E_\ell$, we set all entries of $B_{b_\ell,j}$ to 1 and leave all the entries of A_{i,b_ℓ} unset.
- All entries of A and B that are not defined by the above two cases are set to 0.

In particular, this means that if E_ℓ does not contain any element of the form $(i,\cdot)$ then the submatrix A_{i,b_ℓ} is all 0 and if E_ℓ does not contain any element of the form $(\cdot,j)$ then the submatrix $B_{b_\ell,j}$ is all 0.

It remains to show that the outputs in E of this matrix product are k disjoint ORs on at least $\lfloor n/L \rfloor$ bits each.

Observe that if the color of (i,j) is ℓ, there cannot be another color $\ell' \neq \ell$ and $i' \neq i,\ j' \neq j$ such that $(i,j'),(i',j) \in E$ both have color ℓ', as this would violate the rectangle condition for color ℓ'. This implies that either all entries of $A_{i,b_{\ell'}}$ are 0 or all entries of $B_{b_{\ell'},j}$ are 0 for all $\ell' \neq \ell$. Therefore, assuming that (i,j) is colored ℓ, the (i,j) entry of the product must equal $A_{i,b_\ell} \bullet B_{b_\ell,j}$.

If color class E_ℓ consists of points that do not share a column then the output for each $(i,j) \in E_\ell$ is the OR of the $\geq \lfloor n/L \rfloor$ unrestricted input bits of $B_{b_\ell,j}$; the inputs for different (i,j) are disjoint since no two points of E_ℓ share a column. The analogous property holds for each color class E_ℓ whose points do not share rows. In that case, each output $(i,j) \in E_\ell$ is the OR of $\geq \lfloor n/L \rfloor$ unrestricted input bits of A_{i,b_ℓ} and input bits of A_{i,b_ℓ} are disjoint from each other. Finally, the disjointness of the inputs to the OR functions associated with different color classes is inherited from the disjointness of $b_1,\ldots,b_L$, and the lemma follows since $|E| = k$. □

The lower bound of (Klauck et al., 2007) in Proposition 3.43 embedded $OR^k_{\lfloor n/k \rfloor}$ into any set E of k outputs of $A \bullet B$. Their argument corresponds to the trivial k-coloring that assigns each element of E to its own color class.

Definition 3.49. For integer $k > 0$ define $L_\alpha(k)$ to be the minimum number of colors L such that for all subsets $E \subseteq [n] \times [n]$ with $|E| \leq k$, there is an L-coloring of a subset $E' \subseteq E$ with $|E'| \geq \alpha|E|$.

Lemma 3.50. *There are constants $c, c' > 0$ such that the following holds. Let $\alpha > 0$ and k be an integer such that $L_\alpha(k) \leq n/2$. For any quantum circuit $\mathcal{C}$ with at most $ckn^{1/2}/L_\alpha(k)^{1/2}$ queries to x, the probability that $\mathcal{C}$ produces k correct output values of $n \times n$ Boolean matrix product $A \bullet B$ is at most $2^{-c'\alpha k}$.*

Proof. Let E be any fixed set of k output positions in $A \bullet B$. We show that for each fixed value of E the probability that $\mathcal{C}$ can correctly guess the output values at these indices is exponentially small in k. Let $L \leq L_\alpha(k)$ be such that there is an L-coloring of a subset $E' \subseteq E$ with $|E'| \geq \alpha|E|$. By Lemma 3.48, $OR^{\lceil \alpha k \rceil}_{\lfloor n/L \rfloor}$ is a sub-function of the $\lceil \alpha k \rceil$ outputs indexed by the set E'. Since $L \leq n/2$, $\lfloor n/L \rfloor \geq 2n/(3L)$ and $\sqrt{\lfloor n/L \rfloor} \geq 4\sqrt{n/L}/5$. Choose $c = 4\varepsilon\alpha/5$ and $c' = \gamma$ for ε and γ given in Proposition 3.44. By that proposition, the probability that $\mathcal{C}$ produces the values of these k outputs correctly is at most the probability that $\mathcal{C}$ produces the $\lceil \alpha k \rceil$ outputs in E' correctly which is $2^{-\gamma\lceil \alpha k \rceil} \leq 2^{-c'\alpha k}$. □

Then Lemma 3.46 is an immediate corollary of Lemma 3.50 and the following bound on $L_{1/2}(k)$.

Lemma 3.51 (Coloring Lemma[8])**.** *$L_{1/2}(k) \leq 2\sqrt{6k} < 5\sqrt{k}$.*

Proof. Without loss of generality, E is contained in a grid with side lengths at least $n > 2\sqrt{6k}$, as otherwise we could just use a single color for each row (or column). For a given subset $A \subseteq [n]$ or rows or columns, we use $\overline{A}$ to denote $[n] \setminus A$.

Our strategy is as follows: for some constant c to be determined we show that either

1. there is a row containing at least $c\sqrt{k}$ points of E, or

[8] In a preliminary version of the paper this chapter is based on, there was an error in this lemma, which claimed to show that $L_1(k) \leq 2\sqrt{6k}$. We thank the anonymous reviewers for asking the question that led to us find and address this error.

2. there is a rectangle $R \times C$ such that there are at least $c\sqrt{k}$ points in the rectangle, all of which can be colored with a single color. Moreover, in this case, we show that $|(\overline{R} \times C) \cap E| \leq |(R \times C) \cap E|$.

We now argue why the above two conditions are enough to prove that $L_{1/2}(k) \leq \frac{2}{c}\sqrt{k}$.

If we colored a single row or column, then we can inductively color the remaining points of $E' \subseteq E$ outside that row/column with no issue. However, if we colored the points in $R \times C$, inductively coloring the remaining points could cause an issue because of the rectangle requirement for colors. To address this, we discard the points of $(\overline{R} \times C) \cap E$ and proceed inductively on $E' := E \cap ([n] \times \overline{C})$. At the end of the procedure, since we always color at least the number of points we discard, we will have discarded at most $k/2$ points, as desired.

It remains to show that this such a coloring would always use at most $\frac{2}{c}\sqrt{k}$ colors. We prove this using induction. Indeed, applying induction to color at least $1/2$ of the remaining $k' \leq k - c\sqrt{k}$ elements of E' in $[n] \times \overline{C}$ will require at most $\frac{2}{c}\sqrt{k'} = \frac{2}{c}\sqrt{k-s} \leq \frac{2}{c}\sqrt{k}(1 - \frac{2s}{k}) \leq \frac{2}{c}\sqrt{k} - 1$ colors. It follows that at most $\frac{2}{c}\sqrt{k}$ colors are needed to color at least $1/2$ the points in E, as required.

We now show that we can execute this strategy with the constant $c = 1/\sqrt{6}$, which will prove the lemma. That is, we show how to find either a row containing at least $\sqrt{k/6}$ points of E or a colorable rectangle $R \times C$ with at least $\sqrt{k/6}$ points of E such that $|E \cap (\overline{R} \times C)| \leq |E \cap (R \times C)|$.

For any column j we write E^j for the set of i such that $(i, j) \in E$. Build $R \times C$ in the following way:[9]

First, observe that at the end of the procedure (and indeed at the end of every iteration) the rectangle $R \times C$ contains exactly one element of E in every row, every

[9]In Algorithm 1, instead of the constant $3/4$ in Algorithm 1, we could have chosen any $(1-\gamma)$ instead. In this case, we would achieve a bound for $L_{1-2\gamma}(k) \leq 2\sqrt{\frac{1-\gamma}{\gamma(1-2\gamma)}k}$. For simplicity, we have chosen $\gamma = 1/4$, which is quite close to optimal and has a larger value of $\alpha = 1 - 2\gamma$.

Algorithm 1 Finding a colorable rectangle with many points.

Initialize $R \leftarrow \varnothing$; $C \leftarrow \varnothing$; $D \leftarrow \varnothing$
while *there is a* j *such that* $|E^j \setminus (R \cup D)| \geq \frac{3}{4}|E^j|$ **do**
  $C \leftarrow C \cup \{j\}$
  $D \leftarrow D \cup (R \cap E^j)$
  $R \leftarrow (R \setminus E^j) \cup (E^j \setminus D)$
return $R \times C$

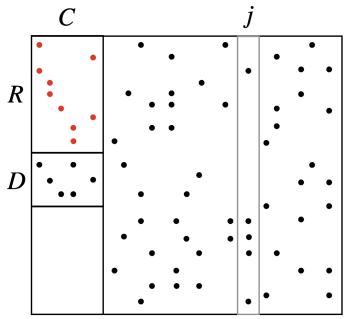

Figure 3.3: Visualization of a single iteration of Algorithm 1.

row of $D \times C$ contains at least two elements of E, and there are no elements of E in $\overline{(R \cup D)} \times C$ – see Figure 3.3 for a visualization of these observations.

Our first simple claim lets us bound the number of points in $\overline{R} \times C$.

Claim 3.52. *$|E \cap (\overline{R} \times C)| \leq |E \cap (R \times C)|$, and $|D| \leq |R|/2$.*

Proof of Claim. The claim is true initially. Suppose that it is true at the beginning of an iteration. When we add j to C on Algorithm 1, we have $|E^j \setminus (R \cup D)| \geq 3|E^j|/4$, and therefore have $|R \cap E^j| \leq |E^j|/4$.

Algorithm 1 therefore adds at most $|E^j|/4$ row indices to D. Since each element of $R \times C$ contained exactly one element of E at the end of the previous iteration, each row added to D by Algorithm 1 has exactly two points of E in the columns of C and there are no points of E in $\overline{(R \cup D)} \times C$, the iteration adds at most $2|E^j|/4 = |E^j|/2$ points of E to $\overline{R} \times C$.

On the other hand, Algorithm 1 adds at least $3|E^j|/4$ elements of E^j to R and only removes the at most $|E^j|/4$ elements of $R \cap E^j$, so R grows by at least $|E^j|/2$

rows in total. Since each row of $R \times C$ has exactly one point in the columns of C, at least $|E^j|/2$ points of E get added to $R \times C$.

Counting rows, we have added at most $|E^j|/4$ rows to D and at least $|E^j|/2$ rows to R, which maintains that $|D| \leq |R|/2$.

Counting points, the increase in size of $E \cap (\overline{R} \times C)$ is at most $|E^j|/2$ which lower bounds the net gain for $E \cap (R \times C)$. This maintains $|E \cap (\overline{R} \times C)| \leq |E \cap (R \times C)|$ as required. □

We let s be the larger of $|R|$ and the maximal number of points in E of any row. For convenience, write $Z = R \cup D$.

When Algorithm 1 finishes, for every column $j \in \overline{C}$, fewer than $3/4$ of its points are in rows of $\overline{Z}$ and hence more than $1/4$ of its points are in rows of Z. So we must have that

$$|E \cap (Z \times \overline{C})| > |E \cap (\overline{Z} \times \overline{C})|/3.$$

As $\overline{Z} \times C$ has no points of E and each row has at most s points of E, the total number of points is

$$\begin{aligned} k &= |E \cap (Z \times [n])| + |E \cap (\overline{Z} \times [n])| \\ &= |E \cap (Z \times [n])| + |E \cap (\overline{Z} \times \overline{C})| \\ &\leq |E \cap (Z \times [n])| + 3|E \cap (Z \times \overline{C})| \\ &\leq 4|Z|\, s \leq 4 \cdot (3|R|/2)\, s \leq 6s^2. \end{aligned}$$

Therefore $s \geq \sqrt{k/6}$. □

Lemma 3.46 is an immediate corollary of Lemmas 3.50 and 3.51 which completes the proof of Theorem 3.45.

We also obtain a general classical lower bound from these arguments. We start by showing a classical analogue of Lemma 3.50.

Lemma 3.53. *Let $\varepsilon, \gamma > 0$ be the constants from Proposition 3.44. Let k be an integer such that $L(k) \leq n/2$. Any randomized algorithm with at most $(2\varepsilon/3)kn/L(k)$ queries to x can only produce k correct output values of $n \times n$ Boolean matrix product $A \bullet B$ with probability at most $2^{-\gamma k}$.*

Proof. Let E be any fixed set of k output indices in $A \bullet B$. Let $L \leq L(k)$ be the smallest number such that E can be colored with L colors. By Lemma 3.48 we know that $OR^k_{\lfloor n/L \rfloor}$ is a sub-function of the outputs indexed by E. Thus, by Proposition 3.44 any randomized algorithm making at most $\varepsilon k \lfloor n/L \rfloor \geq (2\varepsilon/3)kn/L(k)$ queries can compute these outputs with probability at most $2^{-\gamma k}$. □

Theorem 3.54. *Any output-oblivious classical query algorithm computing $n \times n$ Boolean matrix-multiplication with T queries and space S with success probability more than 2^{-S} must have T that is $\Omega(n^3/\sqrt{S})$.*

Proof. Since there are n^2 outputs, which is a trivial time lower bound for sequential algorithms, we can assume that $\sqrt{S}$ is at most αn for some arbitrarily small constant $\alpha > 0$. Let $c = 2/\gamma$ for γ given by Proposition 3.44 and let $k = cS$. Our assumption with $\alpha < 1/(10\sqrt{c})$ implies, by Lemma 3.51 that $L(k) < 5\sqrt{k} = 5\sqrt{cS} < n/2$. The main difference in parameters from the quantum case is that we need to apply Lemma 3.53 instead of Lemma 3.50 to say that classical output-oblivious branching programs of width 2^S have success probability at most $2^{-\gamma k} = 2^{-2S}$ of computing k correct output values of $A \bullet B$. There are at most 2^S nodes at a layer boundary and hence the probability that a layer of height $(2\varepsilon/3)kn/L(k)$ correctly produces k output values is at most 2^{-S}. Rewriting using $L(k) < 5\sqrt{k}$, we obtain that a layer of height $(2\varepsilon/15)\sqrt{k}\, n$ correctly produces outputs with probability at most 2^{-S}. Since there are n^2 outputs, for any circuit of depth T at most $(2\varepsilon/15)n^3/\sqrt{k}$ must have some layer of depth $2\varepsilon/15)\sqrt{k}\, n$ during which at most k outputs are produced and each output value must be correct for the algorithm to be correct, so the overall success probability is at most 2^{-S}. □

This achieves the goal suggested by Klauck, Špalek, and de Wolf (Klauck et al., 2007) who ventured that the likely tight tradeoff for classical computation of Boolean matrix multiplication is $T^2S = \Omega(n^6)$. Note that our quantitative bound asymptotically dominates the bounds of Abrahamson Proposition 3.42 for all values of S; it always is at least as large (up to a constant factor) and the only regimes where our quantitative bound does not strictly dominate that of Abrahamson are when S is $\Theta(1)$ and when S is $\Theta(n)$. Of course, Abrahamson's lower bounds are for the branching program model which allows for the timing of each output bit to depend on the input. (The classical lower bound of (Klauck et al., 2007) for output-oblivious query algorithms is exactly the same as that of Abrahamson for space $O(\sqrt{n})$.) Abrahamson's bound on the number of queries becomes the trivial $\Theta(n^2)$ when $S = \Theta(n^{3/2})$ which is tight for the distribution used in Abrahamson's paper, whereas the lower bound of Theorem 3.54 remains non-trivial so long as S is $o(n^2)$. In fact, just as with our quantum lower bound in Theorem 3.45, the exponents of n and S in Theorem 3.54 are optimal for a circuit model that allows arbitrary gates between queries since that would allow the circuit to simulate a decision tree of height $2n^2$ that reads and remembers the entire input and produces all of the outputs at its leaves; our lower bounds also apply to such a model. See Figure 3.4 for a comparison of our lower bounds with those of prior work for both classical and quantum computation.

Using the same proof idea as in Corollary 3.40, the bounds in Theorems 3.45 and 3.54 immediately imply lower bounds for Boolean matrix squaring.

Corollary 3.55. *Any quantum circuit computing $n \times n$ Boolean matrix squaring on all inputs with T queries, space S, and success probability more than 2^{-S} must have T that is $\Omega(n^{2.5}/S^{1/4})$. Any such output-oblivious classical query algorithm must have T that is $\Omega(n^3/S^{1/2})$.*

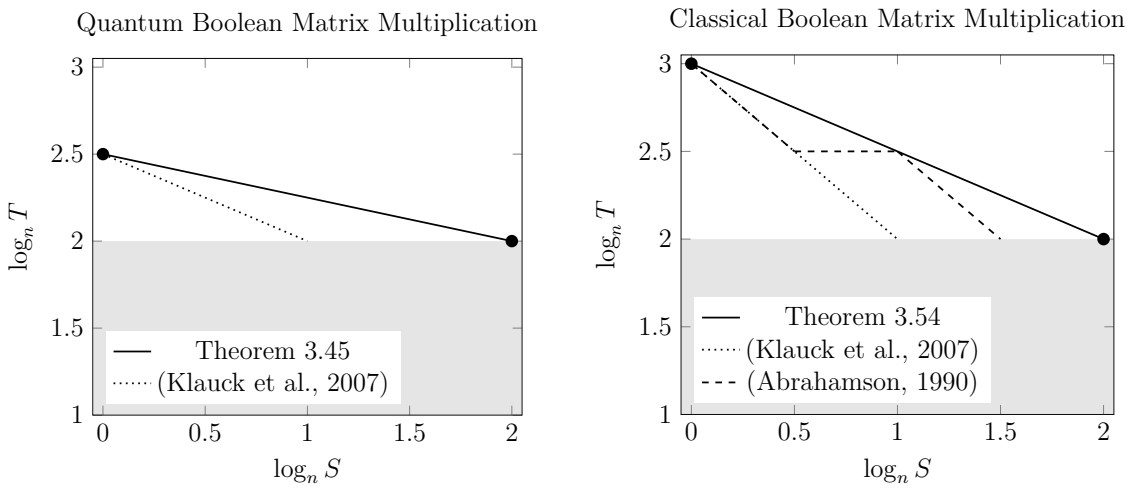

Figure 3.4: Comparison of our lower bounds for Boolean matrix multiplication with those of prior work for both quantum and classical computation. The shaded region comes from the fact that the time must always be $\Omega(n^2)$. The endpoints mark choices of parameters where the upper and lower bounds match.

3.6.2 Boolean matrix-vector product

Finally, we discuss the problem of quantum computation of Boolean matrix-vector product and the closely-associated problem of systems of linear inequalities. Here, rather than producing quantitative improvements which seem unlikely, we focus on a qualitative improvement in existing results.

Though (Abrahamson, 1990) does not contain an explicit theorem statement on time-space tradeoffs for Boolean matrix-vector products that is the analog of the linear algebra bound in (Abrahamson, 1991) or our Theorem 3.21, (Abrahamson, 1990) contains the claim that analogous results do indeed hold for this problem using the same ideas. (The lower bound would be a factor n smaller than the lower bound for linear algebra.)

For quantum circuits, Klauck, Špalek, and de Wolf (Klauck et al., 2007) prove the following results for computing Boolean matrix-vector products. (They also prove a similar result for the case of output-oblivious classical query algorithms, though that does not apply to unconstrained branching programs.)

Proposition 3.56 (Theorem 23 in (Klauck et al., 2007)). *For every S in $o(n/\log n)$, there is an $n \times n$ Boolean matrix $A^{(S)}$ such that every bounded-error quantum circuit with space at most S that computes Boolean matrix-vector product $A^{(S)} \bullet x$ in T queries requires that T is $\Omega(\sqrt{n^3/S}) = \Omega(n^{1.5}/S^{0.5})$.*

This result is weaker than a standard time-space tradeoff since the function involved is not independent of the circuits that might compute it. In particular, (Klauck et al., 2007) does not find a single function that is hard for all space bounds, as the matrix $A^{(S)}$ that they use changes depending on the value of S. Because (Klauck et al., 2007) does not express this dependence in the statement of their results, we provide a detailed discussion of their arguments to make the need for that dependence clear. We will also need their definitions in our results.

For $S = o(n/\log n)$, the matrix $A^{(S)}$ is produced via the probabilistic method using the following distribution: Choose k to be a sufficiently large constant multiple of S. This distribution chooses matrices $A \subseteq \{0,1\}^{n \times n}$ by selecting a uniformly random subset of $n/(2k)$ positions in each row to set to 1, with the remainder of the entries in each row being 0. They show that with positive probability over the choice of A, for all sets $I \subseteq [n]$ of size k, at least $k/2$ of the rows of A_I contain at least $n/(6k)$ 1's that are unique in their column of A_I; that is, those columns are 0 in all of the $k-1$ other rows of A_I. $A^{(S)}$ is then some fixed matrix for which this property is true.

More precisely, when we fix a row $j \in I$ and the $n/(2k)$ columns where it is 1, the expected number of the $(k-1)n/(2k) < n/2$ 1's among the rows in $I \setminus \{j\}$ that land in those $n/(2k)$ columns is less than $n/(4k)$. By a Hoeffding bound, the number of those 1's is at most $n/(3k)$ except with probability exponentially small in n/k, which is $n^{-\omega(1)}$ since $k = O(S) = o(n/\log n)$. Hence, except with probability $n^{-\omega(1)}$, a row $j \in I$ is *good for* I in that at least $n/(2k) - n/(3k) = n/(6k)$ of the 1's in row j are unique in their respective columns in A_I. For a fixed I, the probability that there is no $J \subseteq I$ of size $k/2$ all of whose rows are good for I is less than the probability that there are $k/2$ rows of I that are not good for I. This happens with

probability at most $n^{-\omega(k)}$ since are at most $\binom{k}{k/2}$ such subsets of rows of size $k/2$, each of which is not good for I with probability $n^{-\omega(k)}$ (and the probabilities are negatively associated). Since there are only $\binom{n}{k}$ choices of I, the total probability that A does not have desired properties is only $n^{-\omega(k)}$.

The proof of Proposition 3.56 follows from the usual time-space lower bound methodology and the following lemma:

Lemma 3.57. *There is an $\alpha > 0$ such that for every quantum circuit $\mathcal{C}$ that makes at most $\alpha\sqrt{kn}$ queries to $x \in \{0,1\}^n$, the probability that $\mathcal{C}$ produces at least k correct output values of $A^{(S)} \bullet x$ is at most $2^{-\Omega(k)}$.*

Proof. Let $I \subseteq [n]$ be the set of indices of the first k outputs of $A^{(S)} \bullet x$ produced by $\mathcal{C}$. Let $J \subseteq I$ be the set of size $k/2$ rows that are good for I guaranteed by the properties of $A^{(S)}$. We show that the probability that $\mathcal{C}$ produces all outputs even for the rows in J is exponentially small in k: For each row $j \in J$ there is a set C_j of $n/(6k)$ columns of $A^{(S)}_I$ where the unique 1 is in row j. Consider the restriction to input vectors $x \in \{0,1\}^n$ that are 0 outside of $\bigcup_{j\in J} C_j$. Then the outputs for $j \in J$ are a direct product of $k/2$ OR functions of size $n/(6k)$ on the bits of $\bigcup_{j\in J} C_j$. By a strong direct product theorem for OR (Theorem 14 of (Klauck et al., 2007)), for ε a sufficiently small constant, any circuit of height at most $\varepsilon(k/2)\sqrt{n/(6k)} = \varepsilon\sqrt{kn/24}$ is correct with probability at most $2^{-\gamma k}$ for some constant $\gamma > 0$. □

On the algorithmic side, we have the following:

Proposition 3.58. *For every $c > 0$ and every Boolean matrix $A \in \{0,1\}^{m\times n}$ there is a quantum circuit using space $O(\log n)$ and time $O(mn^{1/2} \log m)$ that computes Boolean matrix-vector product $A \bullet x$ with error at most m^{-c}. More precisely, the algorithm runs in time $O(|A|_{1/2} \log m)$ where $|A|_{1/2} = \sum_{i=1}^{m} \sqrt{|A_i|_1}$.*

Proof. For each row in turn, run Grover's algorithm to compute the OR of the bits indexed by the 1's of A_i, the i-th row of A with probability of error at most m^{-c-1} per row for a total error of at most m^{-c}. □

We note that for the fixed matrix $A^{(S)}$, each row has $\Theta(n/S)$ 1's so $|A^{(S)}|_{1/2} = \Theta(n^{3/2}/S^{1/2})$. This is an odd situation in that the matrix $A^{(S)}$ designed to require large time for space S algorithms can be solved in nearly the same time bound by space $O(\log n)$ algorithms.

Systems of linear inequalities The same space-dependent matrix $A^{(S)}$ in Proposition 3.56 was also used in (Ambainis et al., 2009) for systems of inequalities.

Proposition 3.59 (Theorem 11 in (Ambainis et al., 2009))**.** *Let $\vec{b}$ be the length n all-b vector. For every S in $\min(O(n/b), o(n/\log n))$ there exists an $n \times n$ Boolean matrix $A^{(S)}$ such that every bounded error quantum circuit with space at most S that decides the system $A^{(S)}x \geq \vec{b}$ of n inequalities requires that T is $\Omega(\sqrt{bn^3/S})$.*

Similar to (Klauck et al., 2007) this matrix is used so that any quantum circuit that computes $A^{(S)}x \geq \vec{b}$ can be broken down into slices that solve independent instances of the b-threshold function.

Our results

Using Proposition 3.56, we can obtain a time-space tradeoff lower bound for quantum computation of Boolean matrix-vector product that has an only slightly weaker lower bound in terms of the matrix dimensions but, unlike the previous bound, defines a fixed computational problem whose definition is independent of the space bound allowed.

Theorem 3.60. *There is a fixed $m \times n$ Boolean matrix A with $m \leq n \log_2 n$ such that for every S that is $o(n/\log n)$ every bounded-error quantum circuit with space at most S that computes Boolean matrix-vector product $A \bullet x$ in T queries requires that T is $\Omega(\sqrt{n^3/S})$.*

Proof. The matrix A consists of a stacked version of the matrices $A_{(S_i)}$ from Proposition 3.56 for each choice of $S_i = 2^i \log_2 n$ and $0 \leq i \leq \log_2 n - 2\log_2\log_2 n - \omega(1)$.

Any quantum circuit computing $A \bullet x$ using space S must compute $A^{(S_i)} \bullet x$ for some S_i where $S_i \leq S$ is within factor of 2 of S. It is easy to see that the construction of $A_{(S)}$ for Proposition 3.56 is flexible in terms of the constant factor by which k exceeds S and hence computing matrix $A^{(S_i)} \bullet x$ also requires time T that is $\Omega(\sqrt{n^3/S})$ as required. □

Systems of linear inequalities This same matrix A can be substituted into Proposition 3.59 to obtain a time-space tradeoff for systems of inequalities.

Corollary 3.61. *Let $\vec{b}$ be the length n all-b vector. There is a fixed $m \times n$ Boolean matrix A with $m \leq n \log_2 n$ such that for every S in $\min(O(n/b), o(n/\log n))$ every bounded error quantum circuit with space at most S that decides the system $Ax \geq \vec{b}$ requires T that is $\Omega(\sqrt{bn^3/S})$.*

3.7 Deterministic query algorithms

Here we review the matching time-space space tradeoffs that match our quantum and classical lower bounds. Most of these results were mentioned in (Abrahamson, 1991) but are more fully sketched here. In the following, for simplicity, we describe versions of several of these algorithms over finite fields rather than finite subsets of size d over arbitrary fields. For the more general case, the output values are sums of products of input values and may take more bits to represent; because of this the $\log p$ in our bounds below can be replaced by $O(\max \log d, \log n)$.

The first gives classical algorithms for matrix-vector products matching Theorem 3.21.

Proposition 3.62. *Let A be any $n \times n$ matrix over a finite field $\mathbb{F}_p$. For any $S \in [\log_2 n, n \log_2 p]$ there is a deterministic classical query algorithm computing the matrix vector product $f(x) = Ax$ for all inputs $x \in \mathbb{F}_p$ that uses space S and only $O(n^2 \log p \ /S)$ queries to the input.*

Proof. Let $s = S/\log_2 p$. The query algorithm (which has the matrix A encoded in it) reads one entry of the input x at a time and maintains a block of s different partial sums (using $s \log_2 p$ space). This algorithm produces S outputs every n queries and thus produces all outputs with $n^2/s = n^2 \log_2 p \ /S$ queries. □

Note that in the special case of computing the Discrete Fourier Transform (DFT) (Corollary 3.26), this deterministic query bound can be made explicit using standard operations:

Proposition 3.63 ((Savage and Swamy, 1978))**.** *There is a deterministic classical algorithm computing the Discrete Fourier Transform (DFT) $DFT_n(x) = Wx$ using space $S \geq \log_2 n$ and time $O(n^2/S + n \log S)$.*

Proof. Assume without loss of generality that S and n are powers of 2 and we have $O(S)$ space. This follows by evaluating the graph of the fast Fourier transform (FFT) algorithm for computing the DFT as shown in Figure 3.5. In a single pass over the

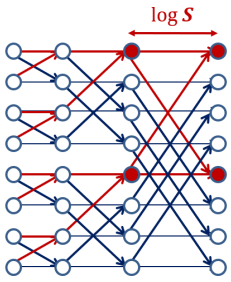

Figure 3.5: The FFT graph with the space-efficient evaluations on one pass highlighted.

input x in $O(n + S \log S)$ time the algorithm can compute the values of S of the outputs using space $O(S)$ as follows: while maintaining $\log_2(n/S) \leq S$ entries for the depth-first evaluation of each subproblem at depth $\log_2 S$ and uses space $2S$ to iterate through the top $\log_2 S$ levels which are evaluated together in a size S FFT computation. This pass is repeated for each of the n/S such blocks in turn. □

The following deterministic algorithms for convolution match Corollary 3.28.

Proposition 3.64. *For any $S \in [\log_2 n, n \log_2 p]$ there is a deterministic classical query algorithm that computes the convolution $f(u, v) = u * v$ where $u, v \in \mathbb{F}_p^n$ that uses S space and only $O(n^2 \log p \ /S)$ queries.*

Proof. Let $s = S/(2 \log_2 p)$. The indices of u, v and $w = u * v$ are reduced modulo n. The query algorithm computes outputs $w_i, \ldots w_{i+s}$ of the convolution as follows: Initialize $w_i, \ldots w_{i+s}$ to the value zero. First query and record the values of $v_{i-1}, \ldots v_{i+s-1}$. Then query values of u one at a time in increasing order $(u_1, u_2, \ldots u_n)$. After reading u_j, for each $k \in \{i, \ldots, i+s\}$, add $u_j \cdot v_{k-j}$ to the value of w_k. Then forget the value of v_{i+s-j} and query the value of v_{i-j-1}, remembering this value. After all of u has been queried, we have that $w_k = \sum_{j \in [n]} u_j v_{k-j}$ which is the correct value for these outputs. Repeating this procedure n/s times gives the convolution of u and v using only S space and $2n$ queries per iteration. Since there are n/s iterations, we get $O(n^2 \log p \ /S)$ queries. □

The algorithms below show that our matrix-inversion lower bound for upper-triangular matrices in Corollary 3.34 cannot be improved for large space bounds, even for deterministic query algorithms. This is open for small space bounds.

Proposition 3.65. *For any $S \in [n \log_2 p, n^2 \log_2 p]$ there is a deterministic classical query algorithm computing the inverse $f(A) = A^{-1}$ where $A \in \mathbb{F}_p^{n \times n}$ is a unit upper triangular matrix that uses S space and only $O(n^4 \log p \ /S)$ queries.*

Proof. Let $s = S/(2n \log p)$. We will produce columns $j_1, \ldots j_s$ of A^{-1} as follows: Let e_j be the column vector with entry 1 at index j and 0 everywhere else. We use back substitution to solve the systems $Ax_1 = e_{j_1}, \ldots, Ax_s = e_{j_s}$ by querying each entry of A exactly once. In particular, the i-th entry of x_k is $1 - \sum_{\ell \in [n-i]} A_{i,n-\ell+1} x_{n-\ell+1}$ when $i = k$ and $-\sum_{\ell \in [n-i]} A_{i,n-\ell+1} x_{n-\ell+1}$ otherwise. We start by computing the n-th entry of each x_k and work backward toward the first entry. We record each entry of each x_k

as is it computed for use in the subsequent computational steps. Note that the i-th entries of all the x_k only require making queries to the i-th row of A and so all the x_k can be computed with only $O(n^2)$ queries. Finally, each x_k is output as the j_k-th column of A^{-1}. This procedure uses $O(n^2)$ queries and at most S space to produce s columns of the output. Thus the procedure must be repeated $n/s = 2n^2 \log p \ /S$ times to produce all n columns of output. This gives a total query complexity of $O(n^4 \log p \ /S)$. □

The following gives deterministic algorithms for our matrix-multiplication, Boolean matrix-multiplication (Theorems 3.36 and 3.45) and squaring lower bounds (Corollaries 3.40 and 3.55).

Proposition 3.66. *There are deterministic query algorithms for $n \times n$ Matrix Multiplication over $\mathbb{F}_p$ using space S that make $O(n^3\sqrt{\log p}/\sqrt{S})$ queries. Further, $O(n^3/\sqrt{S})$ queries suffice for deterministic algorithms using space S to compute $n \times n$ Boolean Matrix Multiplication.*

Proof. Let $s = S/(3 \log p)$. We partition each input matrix A and B into $\sqrt{s} \times \sqrt{s}$ blocks A_{ij} and B_{ij} for $i, j \in [\ell]$ where $\ell = n/\sqrt{s}$. We compute the $\sqrt{s} \times \sqrt{s}$ blocks C_{ij} of the product as follows: Initialize the block C_{ij} to 0. For $k = 1$ to ℓ, query all entries of A_{ik} and B_{kj} and add their product $A_{ik}B_{kj}$ to C_{ij}. The 3 matrices A_{ik}, B_{kj}, and C_{ij} together require space S since each entry can be expressed using $\log p$ bits. The total number of queries to compute C_{ij} is $n\sqrt{s}$ and there are $\ell^2 = n^2/s$ blocks to compute for a total of $n^3/\sqrt{s} = O(n^3\sqrt{\log p}/\sqrt{S})$ queries as claimed.

The query algorithm for Boolean Matrix Multiplication is analogous with $s = S/3$ and entry-wise $\vee$ instead of addition. □

Finally, we see that the matrix triple-product and cubing lower bounds in Corollaries 3.32 and 3.33 have matching deterministic query algorithms.

Proposition 3.67. *For any $S \in [\log_2 n, n^2 \log_2 p]$ there is a deterministic classical query algorithm computing the Matrix Triple Product $f(A, B, C) = ABC$ where $A, B, C \in \mathbb{F}_p^{n \times n}$ that uses S space and only $O(n^4 \log p \ /S)$ queries.*

Proof. Let $s = S/(4 \log p)$. We view the product ABC as $(AB)C$ and use the same strategy as in Proposition 3.66 to compute partial products of (AB) and then ABC. We partition the input, partial product, and output matrices into blocks $A_{ij}, B_{ij}, C_{ij}, (AB)_{ij}$, and $(ABC)_{ij}$ for $i, j \in [\ell]$ where $\ell = n/\sqrt{s}$. To compute $(AB)_{ij}$ we initialize the values in the block to zero. Then, for each $k \in [\ell]$, we query each A_{ik} and B_{kj} and then perform the multiplication of these submatrices, adding the result into $(AB)_{ij}$. After iterating over all k, we have computed the value of $(AB)_{ij}$. Now to compute $(ABC)_{ij}$ we start by initializing the values in $(ABC)_{ij}$ to zero. For each $k \in [\ell]$, we first compute $(AB)_{ik}$ as a subroutine and then query C_{kj} and add the partial product $(AB)_{ik}C_{kj}$ into $(ABC)_{ij}$. After iterating over all k, we have computed the block $(ABC)_{ij}$. This query algorithm stores at most 4 different $\sqrt{s} \times \sqrt{s}$ blocks at any time step. It requires $\sqrt{s}n$ queries to compute each $(AB)_{ij}$ and needs to compute $n/\sqrt{s}$ such blocks for each $(ABC)_{ij}$. Adding the $\sqrt{s}$ queries to C needed to compute $(ABC)_{ij}$ gives $n\sqrt{s}(1 + n/\sqrt{s})$ total queries to compute each block $(ABC)_{ij}$. Since there are n^2/s such blocks, we get $O(n^4/s)$ or $O(n^4 \log p \ /S)$ queries. □

Chapter 4: Cumulative memory lower bounds for randomized and quantum computation

4.1 Introduction

Understanding time-space tradeoffs in terms of the time versus the maximum space used during an algorithm appropriate when machines must allocate said space throughout the computation. However, recent technologies like AWS Lambda (Baird et al., 2021) suggest that in the contexts that include high performance and cloud computation, space can be allocated to a program only as it is needed. When using such services, analyzing the average memory used per step leads to a more accurate picture than measuring the maximum space used.

Cumulative memory (CM), the sum over time of the space used per step of an algorithm, is an alternative notion of time-space complexity that is more fair to algorithms with rare spikes in memory. Cumulative memory complexity was introduced by Alwen and Serbinenko (Alwen and Serbinenko, 2015) who devised it as a way to analyze time-space tradeoffs for "memory hard functions" like password hashes. Since then, lower and upper bounds on the cumulative memory of problems in structured computational models using the irreversible pebble game have been extensively studied, beginning with the work of (Alwen and Serbinenko, 2015; Alwen and Blocki, 2016; Ren and Devadas, 2016; Alwen et al., 2017b, 2016, 2017a). Structured models via pebble games are natural in the context of the random oracle assumptions that are common in cryptography. By carefully interweaving their memory-intensive steps, authors of these papers devise algorithms for cracking passwords that compute many hashes in parallel using only slightly more space than is necessary to compute a single hash. While such algorithms can use parallelism to amortize costs and circumvent proven single instance TS complexity lower bounds, their cumulative memory always scales linearly with the number of computed hashes. Strong results for cumulative

memory have also been shown for the black-white pebble game and have been used to derive related bounds for resolution proof systems (Alwen et al., 2017c).

The ideas used for these structured models yield provable separations between cumulative memory and time-space product complexity in pebbling and random oracle models. The key question that we consider is whether the same applies to general models of computation without cryptographic or black-box assumptions: Are existing time-space tradeoff lower bounds too pessimistic for a world where cumulative memory is more representative of a computation's cost?

Our Results

The main answer we provide to this question is negative for both classical and quantum computation: We give *generic methods* that convert existing paradigms for obtaining time-space tradeoff lower bounds involving worst-case space to new lower bounds that replace the time-space product by cumulative space, immediately yielding a host of new lower bounds on cumulative memory complexity. With these methods, we show how to extend virtually all known proofs for time-space tradeoffs to equivalent lower bounds on cumulative memory complexity, implying that there cannot be cumulative memory savings for these problems. Our results, like those of existing time-space tradeoffs, apply in models in which arbitrary sequential computations may be performed between queries to a read-only input. Our lower bounds also apply to randomized and quantum algorithms that are allowed to make errors.

Classical computation We first focus on lower bound paradigms that apply to computations of multi-output functions $f : D^n \to R^m$. Borodin and Cook (Borodin and Cook, 1982) introduced a method for proving time-space tradeoff lower bounds for such functions that takes a property such as the following: for some $K = K(R, n)$, constant γ, and distribution μ on D^n:

(*) For any partial assignment τ of $k \leq \gamma m$ output values over R and any restriction (i.e., partial assignment) π of $h = h(k, n)$ coordinates on D^n,

$$\Pr_{x \sim \mu}[f(x) \text{ is consistent with } \tau \mid x \text{ is consistent with } \pi] \leq K^{-k}$$

and derives a lower bound of the following form:

Proposition 4.1 ((Borodin and Cook, 1982))**.** *Assume that Property (*) holds for $f : D^n \to R^m$ with $\gamma > 0$ constant. Then, T $(S + \log_2 T)$ is $\Omega(m\ h(S/\log_2 K, n) \log K)$.*

In particular, since $S \geq \log_2 n$ is essentially always required, if we have the typical case that $h(k, n) = k^{\Delta}\ h_1(n)$ for some function $h_1(n)$ and constant Δ then this says that $T \cdot S^{1-\Delta}$ is $\Omega(m\ h_1(n)\ \log^{1-\Delta} K)$ or, equivalently, that $\max(S, \log n)$ is $\Omega([(m\ h_1(n)/T]^{1/(1-\Delta)} \log K)$. As a simplified example of our new general paradigm, we prove the following analog for cumulative complexity:

Theorem 4.2. *Suppose that Property (*) holds for $f : D^n \to R^m$ with $h(k, n) = k^{\Delta} h_1(n)$ and $\gamma > 0$ constant. If $T \log_2 T$ is $o(m\ h_1(n) \log K)$ then any algorithm computing f requires cumulative memory $\Omega\left(\left[(m\ h_1(n))^{1/(1-\Delta)} \log K\right] / T^{\Delta/(1-\Delta)}\right)$.*

We note that this bound corresponds exactly to the bound on the product of time and space from the Borodin-Cook method. The full version of our general theorem for randomized computation (Theorem 4.23) is inspired by an extension made by Abrahamson (Abrahamson, 1991) to the Borodin-Cook paradigm that expands it coverage it to average case complexity.

Quantum computation We develop an extension of our general approach that applies to quantum computation as well. In this case Property (*) and its extensions that we use for our more general theorem must be replaced by statements about quantum circuits with few queries. We first generalize the quantum time-space tradeoff for sorting proven in (Klauck et al., 2007), which requires that the time order in

Problem	TS Bound	Source	CM Bound
Ranking, Sorting	$\Omega(n^2/\log n)$	(Borodin and Cook, 1982)	Theorem 4.13
Unique Elements	$\Omega(n^2)$	(Beame, 1991)	Theorem 4.29
Matrix-Vector Product	$\Omega(n^2 \log d)$	(Abrahamson, 1991)	Theorem 4.31
Matrix Multiplication	$\Omega((n^6 \log d)/T)$	(Abrahamson, 1991)	Theorem 4.34
Q Sorting	$\Omega(n^3/T)$	(Klauck et al., 2007)	Theorem 4.19
Q Mat-Vec Product	$\Omega(n^2 \log d)$	Theorem 3.21	Corollary 4.38
Q Matrix Mult	$\Omega((n^6 \log d)/T)$	Theorem 3.36	Corollary 4.40
Q k Disjoint Collisions	$\Omega(k^3 n/T^2)$	(Hamoudi and Magniez, 2021)	Theorem 4.37
Q Bool Matrix Mult	$\Omega(n^{10}/T^3)$	Theorem 3.45	Corollary 4.41

Table 4.1: All cumulative memory bounds match the time-space product lower bound when considering RAM computation or quantum circuits. For the linear algebra problems, we assume inputs come from a fixed subset D of a field with $d = |D|$. Results above the double line are for classical computation while those below are for quantum computation.

which output values are produced must correspond to the sorted order, to a matching cumulative memory complexity bound of $\Omega(n^3/T)$ that works for any fixed time-ordering of output production, yielding a more general lower bound. For example, an algorithm may be able to determine the median output long before it determines the other outputs. We then show how an analog of our classical general theorem can be applied to extend to paradigms for quantum time-space tradeoffs to cumulative memory complexity bounds for other problems.

A summary of our results for both classical and quantum complexity is given in Table 4.1, including an extension of our work in Chapter 3 to cumulative memory lower bounds.

Prior work Alwen and Serbinenko (Alwen and Serbinenko, 2015) introduced parallel cumulative (memory) complexity as a metric for analyzing the space footprint required

to compute *memory hard functions (MHFs)*, which are functions designed to require large space to compute. Most MHFs are constructed using hashgraphs (Dwork et al., 2005) of DAGs whose output is a fixed length string and their proofs of security are based on pebbling arguments similar to those in Chapter 2 on these DAGs while assuming access to truly random hash functions for their complexity bounds (Alwen and Serbinenko, 2015; Boneh et al., 2016; Ren and Devadas, 2016; Alwen et al., 2017a,b; Blocki and Zhou, 2017). (See Section 4.3 for their use in separating CM and TS complexity.) Recent constructions do not require random hash functions; however, they still rely on cryptographic assumptions (Chen and Tessaro, 2019; Ameri et al., 2022).

Our methods At the highest level, we employ part of the same paradigms previously used for time-space tradeoff lower bounds that we demonstrated in Chapter 3. Namely breaking up the computations into blocks of time and analyzing properties of the branching programs or quantum circuits based on what happens at the boundaries between time blocks. However, for cumulative memory complexity, those boundaries cannot be at fixed locations in time and their selection needs to depend on the space used in these time steps.

Further, in many cases, the time-space tradeoff lower bound needs to set the lengths of those time blocks in a way that depends on the specific space bound. When extending the ideas to bound cumulative memory usage, there is no single space bound that can be used throughout the computation; this sets up a tricky interplay between the choices of boundaries between time blocks and the lengths of the time blocks. Because the space usage within a block may grow and shrink radically, even with optimal selection of block boundaries, the contribution of each time block to the overall cumulative memory may be significantly lower than the time-space product lower bound one would obtain for the individual block.

We show how to bound any loss in going from time-space tradeoff lower bounds to cumulative memory lower bounds in a way that depends solely on the bound on

the lengths of blocks as a function h_0 of the target space bound (cf. Lemma 4.22). For many classes of bounding functions we are able to bound the loss by a constant factor, and we are able to show that it is always at most an $O(\log n)$ factor loss. If this bounding function h_0 is non-constant, we also need to bound the optimum way for the algorithm to allocate its space budget for producing the required outputs throughout its computation. This optimization again depends on the bounding function h_0. This involves minimizing a convex function based on h_0 subject to a mix of convex and concave constraints, which is not generally tractable. However, assuming that h_0 is nicely behaved, we are able to apply specialized convexity arguments (cf. Lemma 4.25) which let us derive strong lower bounds on cumulative memory complexity.

Road map We give the overall definitions in Section 4.2, including a review of the standard definitions of the work space used by quantum circuits. Section 4.3 gives a random oracle separation between the time-space product and cumulative memory. Section 4.4 is stand-alone section containing a simpler explicit cumulative memory lower bound for classical sorting algorithms that do not rely on our general theorems. In Section 4.5, we give our lower bound for quantum sorting algorithms which gives a taste of the issues involved for our general theorems. In Section 4.6, we give the general theorems that let us convert the Borodin-Cook-Abrahamson paradigm for multi-output functions to cumulative memory lower bounds for classical randomized algorithms; that section also contains the corresponding theorems for quantum lower bounds. Section 4.7 applies our general theorems from Section 4.6 to lower bound the cumulative memory complexity for some concrete problems.

4.2 Preliminaries

Cumulative memory is an abstract notion of time-space complexity that can be applied to any model of computation with a natural notion of space. Here we will use branching programs and quantum circuits as concrete models, although our results

generalize to any reasonable model of computation.

Branching programs Branching programs with input $\{x_1, \ldots, x_n\} \in D^n$ are known as D-way branching programs and are defined using a rooted DAG in which each non-sink vertex is labeled with an $i \in [n]$ and has $|D|$ outgoing edges that correspond to possible values of x_i. Each edge is optionally labeled by some number of output statements expressed as pairs (j, o_j) where $j \in [m]$ is an output index and $o_j \in R$ (if outputs are to be ordered) or simply $o_j \in R$ (if outputs are to be unordered). Evaluation starts at the root v_0 and follows the appropriate labels of the respective x_i. We consider branching programs $\mathcal{P}$ that contain $T + 1$ layers where the outgoing edges from nodes in each layer t are all in layer $t + 1$. We impose no restriction on the query pattern of the branching program or when it can produce parts of the output. Such a branching program P has the following complexity measures: The *time* of the branching program is $T(\mathcal{P}) = T$. The *space* of the branching program is $S(\mathcal{P}) = \max_t \log_2 \|L_t\|$ where L_t is the set of nodes in layer t. Observe that in the absence of any limit on its space, a branching program could equally well be a decision tree; hence the minimum time for branching programs to compute a function f is its *decision tree complexity*. The *time-space* (product) used by the branching program is $TS(\mathcal{P}) = T(\mathcal{P})S(\mathcal{P})$. The *cumulative memory* used by the branching program is $CM(\mathcal{P}) = \sum_t \log_2 |L_t|$.

Branching programs are very general and simultaneously model time and space for sequential computation. In particular, they model time and space for random-access off-line multi-tape Turing machines and random-access machines (RAMs) when time is unit-cost, space is log-cost, and the input and output are read-only and write-only respectively[1]. Branching programs are much more flexible than these models since

[1]In prior work, branching program space has often been defined to be the logarithm of the total number of nodes (e.g., (Borodin and Cook, 1982; Abrahamson, 1991)) rather than the logarithm of the width (maximum number of nodes per layer), though the latter has been used (e.g., (Chandra et al., 1983)). The natural conversion from an arbitrary space-bounded machine to a branching

they can make arbitrary changes to their storage in a single step.

Quantum Circuits We also consider quantum circuits $\mathcal{C}$ with classical read-only inputs $X = x_1, \ldots, x_n$. We use the same model of quantum circuits presented in Section 3.2.1. As a quick refresher, this is the standard quantum query model where arbitrary input-independent unitaries can be applied between quantum queries to the input. The quantum circuit is broken down into layers composed of a query to the input, an input-independent unitary, and an optional measurement on any subset of the qubits. The 'space' of a layer is the number of qubits sent from that layer to the next one.

4.3 A gap between time-space product and cumulative memory

Here we discuss how the irreversible pebble game can be used to prove a separation between time-space product and cumulative memory complexity for computation.

Irreversible pebbling separation

We would like to remind the reader of the irreversible pebbling game introduced in Chapter 2.

Definition 2.2. The irreversible pebble game is a one player game on a DAG $G = (V, E)$ where the goal is to place a pebble on exactly the nodes $T \subseteq V$ called *targets* with out-degree zero. A pebbling (strategy) is a list of subsets of V. $\mathcal{P} = [\mathcal{P}_0, \ldots, \mathcal{P}_\tau]$ where $\mathcal{P}_0 = \emptyset$ and $\mathcal{P}_\tau = T$. A strategy is valid as long as

program produces one that is not leveled (i.e., nodes are not segregated by time step). After leveling the branching program, the space of the original machine becomes the logarithm of the width (cf. (Pippenger, 1979)). The width-based definition is also the only natural one by which to measure cumulative memory complexity and, in any case, the two definitions differ by at most the additive $\log_2 T$ we used for the Borodin-Cook bound, with lower bounds on width implying lower bounds on size.

- $|\mathcal{P}_i \triangle \mathcal{P}_{i+1}| = 1$, and
- If $v \in \mathcal{P}_{i+1} \setminus \mathcal{P}_i$, then parents$(v) \subseteq \mathcal{P}_i$.

Definition 2.3. The number of steps $T(\mathcal{P})$ in a pebbling strategy $\mathcal{P} = [\mathcal{P}_0, \ldots, \mathcal{P}_\tau]$ is τ.

Definition 2.4. The number of pebbles $S(\mathcal{P})$ in a pebbling strategy $\mathcal{P} = [\mathcal{P}_0, \ldots, \mathcal{P}_\tau]$ is $\max_{i \in [\tau]} |\mathcal{P}_i|$.

Similarly, we can define the cumulative memory cost of a pebbling strategy.

Definition 4.3. The cumulative memory $CM(\mathcal{P})$ of pebbling strategy $\mathcal{P} = [\mathcal{P}_0, \ldots, \mathcal{P}_\tau]$ is $\sum_{t \in [\tau]} |\mathcal{P}_t|$.

We will be considering instances of the black pebble game where T contains exactly the unique node with out-degree zero. Intuitively, the black pebble game corresponds to strategies for evaluating straight line programs, where a pebble indicates that a particular value has been computed and is currently stored in memory. The number of pebbles used when pebbling a graph is analogous to the space used by that computation. We will construct a simple DAG where the time-space complexity is larger than the cumulative memory complexity.

Proposition 4.4. *There is a family of DAGs $\{\mathcal{G}_i\}_{i \in \mathbb{N}}$ such that graph $\mathcal{G}_n$ requires $\Omega(n^{1/3})$ pebbles and $\Omega(n)$ steps to pebble but $\mathcal{G}_n$ can be pebbled with cumulative memory $\Omega(n)$.*

This gives an $\Omega(n^{1/3})$ separation between the time-space product and cumulative memory for pebbling.

Proof. We construct G_n, as shown in Figure 4.1, to contain an $n^{1/3} \times n^{1/3}$ square lattice whose node of out-degree zero now has an out-going edge to the head of a chain containing n nodes. Pebbling the $n^{1/3} \times n^{1/3}$ lattice requires pebbling a pyramid of

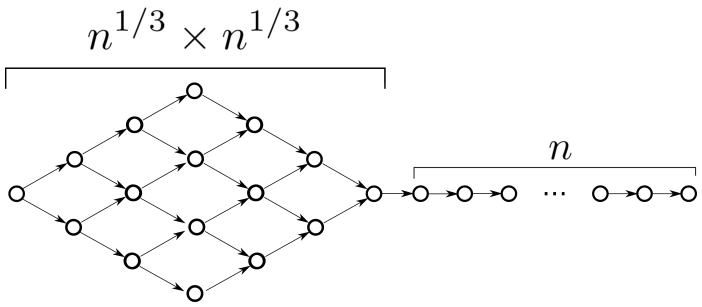

Figure 4.1: A DAG defined by parameter n. It is formed by joining an $n^{1/3} \times n^{1/3}$ lattice to a chain of length n.

height $n^{1/3}$, so Theorem 5 of (Cook, 1973) tells us that $\Omega(n^{1/3})$ pebbles are necessary to place a pebble at the end of the lattice. Since a pebble must be placed on each node in the chain of length n, pebbling this graph takes at least n steps.

Now we will show how this graph can be pebbled with less cumulative memory. Both the lattice and the chain can each be pebbled with cumulative memory that is $O(n)$. The lattice can be pebbled by placing $n^{1/3}$ pebbles along the top diagonal and then repeatedly moving (i.e. adding a new pebble then removing the old pebble) these pebbles along their downward edges until they are all on the bottom diagonal. Then all pebbles other than the one at the end of the lattice are removed. This uses $n^{1/3} + 1$ pebbles and acts on each node at most twice for a total of $O(n^{2/3})$ steps. This strategy pebbles the lattice with a cumulative memory complexity that is only $O(n)$. For the chain we can simply move one pebble from the leftmost node to the rightmost node in $2n + 1$ steps. Since this process only requires two pebbles, the cumulative memory is also $O(n)$. □

Random oracle separation

There is a group of closely related theorems from cryptography that let us instantiate pebbling graphs with the help of a random oracle (Dwork et al., 2005; Dziembowski et al., 2011; Karvelas and Kiayias, 2014). Here we walk through the ideas behind these proofs to show that the graph $\mathcal{G}_n$ in Proposition 4.4 leads to

separation between time-space and cumulative memory complexities in the random oracle model. The concrete problem we will be considering is related to labeling nodes of the pebbling graph.

Definition 4.5. Let $\mathcal{G} = (V, E)$ be a DAG with maximum in-degree two and target v_t. Fix c to be some large constant. Let $\mathcal{H} : \{0,1\}^{(2c+1)\lceil \log_2 |V| \rceil} \rightarrow \{0,1\}^{c\lceil \log_2 |V| \rceil}$ be a random function. We assign each $v_i \in V$ a label $L(v_i)$ as follows:

- If v_i has in-degree zero, then $L(v_i) = \mathcal{H}(0^{c\lceil \log_2 |V| \rceil}, 0^{c\lceil \log_2 |V| \rceil}, i)$.
- If v_i has exactly one parent v_j, then $L(v_i) = \mathcal{H}(L(v_j), L(v_j), i)$.
- If v_i has two parents v_j, v_k where $j < k$, then $L(v_i) = \mathcal{H}(L(v_j), L(v_k), i)$.

The *hash-graph* problem $H_{\mathcal{G}}^{\mathcal{H}}$ is the task of computing the label of v_t.

We will use the notation $\mathcal{A}^{\mathcal{H}}$ to denote an algorithm $\mathcal{A}$ that has query access to the random oracle (function) $\mathcal{H}$. We start by proving a weaker version of a result in (Dwork et al., 2005).

Definition 4.6 ((Dwork et al., 2005))**.** Let $\mathcal{G} = (V, E)$ be a DAG and $\mathcal{A}^{\mathcal{H}}$ be an algorithm that solves the hash-graph problem $H_{\mathcal{G}}^{\mathcal{H}}$. Then the *ex post facto* pebbling of $\mathcal{A}^{\mathcal{H}}$ is defined as follows:

- Making the call $\mathcal{H}(0^{c\lceil \log_2 |V| \rceil}, 0^{c\lceil \log_2 |V| \rceil}, i)$ when v_i has in-degree zero corresponds to placing a pebble on v_i.
- Making the call $\mathcal{H}(L(v_j), L(v_j), i)$ when v_i's only parent is v_j corresponds to placing a pebble on v_i.
- Making the call $\mathcal{H}(L(v_j), L(v_k), i)$ when v_j and v_k are the parents of v_i and $j < k$ corresponds to placing a pebble on v_i.

- A pebble is removed as soon as it is no longer needed. This happens when either the children of that node are never pebbled after this point or when the node is pebbled again before any of its children are pebbled.

Analyzing ex post facto pebbling is key to the arguments in (Dwork et al., 2005; Dziembowski et al., 2011; Karvelas and Kiayias, 2014) and lets us lower bound the space required to compute a hash-graph.

Proposition 4.7 ((Dwork et al., 2005))**.** *Consider an algorithm $\mathcal{A}^{\mathcal{H}}$ that operates for a certain number of steps with $s \cdot c \lceil \log_2 |V| \rceil$ bits of memory. Then with probability at least $1 - 1/|V|^c$ (over the choice of $\mathcal{H}$) the maximum number of pebbles placed by the ex post facto pebbling of $\mathcal{A}^{\mathcal{H}}$ is bounded above by s.*

This lets us show that a hash-graph problem requires at least as much time and space as pebbling its underlying graph, as we show below in an argument similar to ones in (Dziembowski et al., 2011; Karvelas and Kiayias, 2014).

Proposition 4.8. *Let $G = (V, E)$ be a DAG that requires s pebbles and τ steps to pebble. Then any algorithm $\mathcal{A}^{\mathcal{H}}$ that solves the hash-graph problem $H_{\mathcal{G}}^{\mathcal{H}}$ must use space S that is larger than $(s-1) \cdot c \lceil \log_2 |V| \rceil$ and time T that is at least τ or its success probability (over the randomness of $\mathcal{H}$) is at most $2T/|V|^{c-1}$.*

Proof. For an algorithm $\mathcal{A}^{\mathcal{H}}$ with space bound $S < (s-1) \cdot c \lceil \log_2 |V| \rceil$ to solve $H_{\mathcal{G}}^{\mathcal{H}}$, one of the following events must happen:

1. $\mathcal{A}^{\mathcal{H}}$ solves $H_{\mathcal{G}}^{\mathcal{H}}$ without placing a pebble on the target during the ex post facto pebbling.
2. $\mathcal{A}^{\mathcal{H}}$ places s pebbles during the ex post facto pebbling.
3. $\mathcal{A}^{\mathcal{H}}$ places a pebble during the ex post facto pebbling that would not be valid according to the black pebble game.

Since $\mathcal{H}$ is a random oracle, (1) happens with probability at most $1/|V|^c$. By Proposition 4.7 (2) happens with probability at most $1/|V|^c$. Since guessing a label that has not been pebbled is possible with probability at most $1/|V|^c$, We know that (3) happens with probability at most $T/|V|^{c-1}$ via a union bound over the queries of $\mathcal{A}^{\mathcal{H}}$ and the nodes of $\mathcal{G}$. Thus by a union bound, the probability $\mathcal{A}^{\mathcal{H}}$ can produce the correct output is at most $(T \cdot |V| + 2)/|V|^c$ which in turn is at most $2T/|V|^{c-1}$.

Now consider an algorithm $\mathcal{A}^{\mathcal{H}}$ with time bound $T < \tau$. Since $\mathcal{G}$ cannot be pebbled in this number of steps, one of the following events must happen for $\mathcal{A}^{\mathcal{H}}$ to produce the correct output:

1. $\mathcal{A}^{\mathcal{H}}$ solves $H_{\mathcal{G}}^{\mathcal{H}}$ without placing a pebble on the target during the ex post facto pebbling.

2. $\mathcal{A}^{\mathcal{H}}$ places a pebble during the ex post facto pebbling that would not be valid according to the black pebble game.

Both of the events are the same as in the space-bounded case, and therefore a union bound give $\mathcal{A}^{\mathcal{H}}$ a success probability of at most $2T/|V|^{c-1}$. □

Any algorithm that solves $H_G^{\mathcal{H}}$ must obey the space and the time bounds imposed by pebbling that graph or spend time that is $\Omega(|V|^{c-1})$.[2] Note that a pebbling of a graph G directly corresponds to a strategy for computing $H_G^{\mathcal{H}}$ with the same space, time, and cumulative memory bounds as the pebbling. By using the graph G_n from Proposition 4.4, this gives us a separation between time-space product complexity and cumulative memory in the random oracle model.

Corollary 4.9. *Relative to a random oracle $\mathcal{H}$, there is a problem with an $\Omega(n^{1/3})$ separation between its time-space product complexity and its cumulative memory complexity.*

[2]Since c is an arbitrary constant, this time bound can be made arbitrarily large.

While this will be hard to prove, we believe that replacing the random oracle with a suitable hash function gives a problem where there is an asymptotic gap between these complexity measures.

Conjecture 4.10. *Instantiating the family of DAGs from Proposition 4.4 as hash-graph problems with some concrete hash-function gives a problem with an unconditional asymptotic gap between its sequential time-space product and cumulative memory complexities.*

4.4 Cumulative memory complexity of classical sorting algorithms

For a natural number N, the standard version of *sorting* is a function $Sort_{n,N} : [N]^n \to [N]^n$ that on input $x \in [N]^n$ produces an output $y \in [N]^n$ in non-decreasing order where y is a permutation of x; that is, there is some permutation π such that $y_i = x_{\pi(i)}$ for all $i \in [n]$. A related problem is the *ranking* problem $Rank_{n,N} : [N]^n \to [n]^n$ which on input $x \in [N]^n$ produces a permutation π represented as the vector $(\pi(1), \ldots, \pi(n))$ such that $Sort_{n,N}(x) = (x_{\pi(1)}, \ldots, x_{\pi(n)})$ and whenever $x_i = x_j$ for $i < j$ we have $\pi(i) < \pi(j)$.

Proposition 4.11 ((Borodin and Cook, 1982))**.** *(a) If there is a $[nN]$-way branching program P computing $Sort_{n,nN}$ then there is a $[N]$-way branching program P' computing $Rank_{n,N}$ with $T(P') \leq T(P)$, $S(P') \leq S(P)$, and $CM(P') \leq CM(P)$.*

(b) If there is a $[N]$-way branching program P'' computing $Rank_{n,N}$ then there is a $[N]$-way branching program P''' computing $Sort_{n,N}$ with $T(P''') \leq 2T(P'')$, $S(P''') \leq S(P'') + \log_2 N$, and $CM(P''') \leq 2CM(P'') + T(P''') \log_2 N$.

Proof. For part (a), the program P' is exactly P except that when P queries $x_i \in [Nn]$, P' reads $x'_i \in [N]$ and branches on value $x_i = (x'_i, i)$ and when P outputs

$(i, y_i) = (i, x_{\pi(i)})$ on an edge for $x_{\pi(i)} = (x'_{\pi(i)}, \pi(i))$, P' outputs $(i, \pi(i))$. For part (b), the program P''' is exactly P'' except that whenever P'' outputs $(i, \pi(i))$ on an edge, P''' queries $x_{\pi(i)}$ and outputs $(i, x_{\pi(i)})$. One layer becomes two layers and the number of nodes per layer of P''' is at most N times that of P''. □

Following (Borodin and Cook, 1982), we focus on inputs where the x_i are distinct. In this case, the tie-breaking we enforced in defining $Rank_{n,N}$ when there are equal elements is irrelevant.

Proposition 4.12 ((Borodin and Cook, 1982))**.** *There is an $\alpha > 0$ such that the following holds. Let n be sufficiently large and μ be the uniform distribution over lists of n distinct integers from $[n^2]$. Then for any branching program B of height $h \leq \alpha n$ and for all integers $k \leq 2\alpha n$, the probability for $x \sim \mu$ that B produces at least k correct output values of $Rank_{n,n^2}$ on input x is at most $2^{-k/\lceil \log_2 n \rceil}$.*

Theorem 4.13. *Let P be a branching program computing $Sort_{n,n^3}$ with probability at least $n^{-O(1)}$ and $T = T(P)$. Then T is $\Omega(n^2/\log^2 n)$ or $CM(P)$ is $\Omega(n^2/\log n)$. Further, any random access machine computing $Sort_{n,n^3}$ with $n^{-O(1)}$ probability requires cumulative memory of $\Omega(n^2/\log n)$ bits.*

Proof. We prove the same bounds for branching programs P computing $Rank_{n,n^2}$ which, by Proposition 4.11, implies the bounds for computing $Sort_{n,n^3}$.

For simplicity, we first assume that P is deterministic and always produces the correct output. Let α be the constant and μ be the probability distribution on $[n^2]^n$ from Proposition 4.12, and let $H = \left\lfloor \frac{\alpha}{2} n \right\rfloor$. We partition P into $\ell = \lceil T/H \rceil$ intervals $\{I_1, \ldots, I_\ell\}$, each of length H except for the first which may be shorter than the rest. Let $t_1 = 0$, $t_{\ell+1} = T$, and for $i \in [2, \ell]$, t_i be the time-step in I_i with the fewest number of nodes. We define $S_i = \log_2(\|L_{t_i}\|)$ where L_j is the set of nodes of P in layer j. The i-th time block B_i will contain all layers from t_i to t_{i+1}. We observe:

$$CM(P) \geq \sum_{i=2}^{\ell} S_i \, H = H \sum_{i=1}^{\ell} S_i \tag{4.1}$$

since $S_1 = 0$ and our choice of t_i guarantees that for $i \in [2, \ell]$, P can be decomposed into disjoint blocks of H layers that each have at least 2^{S_i} nodes per layer. Define $k_i = \lceil \lceil \log_2 n \rceil (S_i + \log_2(2T)) \rceil$, which will be our target number of outputs for block B_i. By our choice of B_i we know its length is at most αn while starting at a layer with 2^{S_i} nodes. So, by Proposition 4.12, combined with a union bound, the probability for $x \sim \mu$ that B_i produces at least k_i correct output values of $Rank_{n,n^2}$ on input $x \sim \mu$ is at most $1/(2T)$. Thus the probability over μ that at least one block B_i produces at least k_i correct output values is at most $1/2$ and the probability that the total number of outputs produced is at most $\sum_{i=1}^{\ell}(k_i - 1)$ is at least $1/2$. Since P must always produce n correct outputs, we must have:

$$\sum_{i=1}^{\ell}(k_i - 1) \geq n.$$

Inserting the definition of k_i we get:

$$\sum_{i=1}^{\ell}(\lceil \log_2 n \rceil (S_i + \log_2(2T))) \geq n.$$

Using Equation (4.1) to express this in terms of $CM(P)$ gives us:

$$CM(P)/H + \ell \log_2(2T) \geq \frac{n}{\lceil \log_2 n \rceil}$$

or

$$CM(P) + T \log_2(2T) \geq \frac{n \left\lfloor \frac{\alpha}{2} n \right\rfloor}{\lceil \log_2 n \rceil} \geq \frac{\alpha n^2}{3 \log_2 n}.$$

Thus at least one of $T \log_2(2T)$ or $CM(P)$ is at least $\alpha n^2/(6 \log_2 n)$, as required, since $\log T$ is $O(\log n)$ wlog. The bound for random-access machines comes from observing that such a machine requires at least one memory cell of $\Omega(\log T)$ bits at every time step.

To prove the bound for algorithms with success probability n^{-c}, we multiply $\log_2(2T)$ in the above argument by $(c + 1)$. Since any sorting algorithm must have $T \geq n$, on randomly chosen inputs the probability that it produces at least $\sum_{i=1}^{\ell}(k_i - 1)$ correct outputs becomes at most $\frac{1}{2n^c} < \frac{1}{n^c}$ and hence the above bounds (reduced by

the constant factor $c+1$) apply to deterministic algorithms with success probability $1/n^c$ for inputs from the uniform distribution over lists of n distinct integers from $[n^2]$. By Yao's lemma this implies the same lower bound for randomized algorithms with success probability at most n^{-c}. □

Theorem 4.13 applies to cumulative working memory of any algorithm that produces its sorted output in a write-only output vector and can compute those values in arbitrary time order. If the algorithm is constrained to produce its sorted output in the natural time order then, following (Beame, 1991), one can obtain a slightly stronger bound.

Theorem 4.14. *Any branching program P computing the outputs of $Sort_{n,n}$ in order in time T and probability at least $4/5$ requires T to be $\Omega(n^2/\log n)$ or $CM(P)$ to be $\Omega(n^2)$. Further, any random access machine computing $Sort_{n,n}$ in order with probability at least $4/5$ requires cumulative memory $\Omega(n^2)$.*

Proof Sketch. Any such algorithm can easily determine all the elements of the input that occur uniquely, and the lower bounds follow from the bounds on Unique Elements that we prove in Section 4.7. □

4.5 Quantum cumulative memory complexity of sorting

As an illustrative example, we first show that the quantum cumulative memory complexity of sorting is $\Omega(n^3/T)$, matching the TS complexity bounds given in (Klauck et al., 2007; Hamoudi and Magniez, 2021). This involves the quantum circuit model which, as we have noted, produces each output position at a predetermined input-independent layer. We restrict our attention to circuits that output all elements in the input in some fixed rank order. While our proof is inspired by the time-space lower bound of (Klauck et al., 2007), it can be easily adapted to follow the proof in (Hamoudi and Magniez, 2021) instead. We start by constructing a probabilistic reduction from the k-threshold problem to sorting.

Definition 4.15. In the *k-threshold problem* we receive an input $X = x_1, \ldots, x_n$ where $x_i \in \{0, 1\}$. We want to accept iff there are at least k distinct values for i where $x_i = 1$.

Proposition 4.16 (Theorem 13 in (Klauck et al., 2007))**.** *For every $\gamma > 0$ there is an $\alpha > 0$ such that any quantum k-threshold circuit with at most $T \leq \alpha\sqrt{kn}$ queries and with perfect soundness must have completeness $\sigma \leq e^{-\gamma k}$ on inputs with Hamming weight k.*

Lemma 4.17. *Let $\gamma > 0$. Let n be sufficiently large and $\mathcal{C}(X)$ be a quantum circuit with input $X = x_1, \ldots, x_n$. There is a $\beta < 1$ depending only on γ such that for all $k \leq \beta^2 n$ and $R \subseteq \{n/2 + 1, \ldots, n\}$ where $\|R\| = k$, if $\mathcal{C}(X)$ makes at most $\beta\sqrt{kn}$ queries, then the probability that $\mathcal{C}(X)$ can correctly output all k pairs (x_i, r_j) where $r_j \in R$ and x_i is the r_j-th smallest element of X is at most $e^{(1-\gamma)k-1}$. If R is a contiguous set of integers, then the probability is at most $e^{-\gamma k}$.*

A version of this lemma was first proved in (Klauck et al., 2007) with the additional assumption that the set of output ranks R is a contiguous set of integers; this was sufficient to show that any quantum circuit that produces its sorted output in sorted time order requires that T^2S is $\Omega(n^3)$. The authors stated that their proof can be generalized to any fixed rank ordering, but the generalization is not obvious. We generalize their lemma to non-contiguous R, which is sufficient to obtain an $\Omega(n^3/T)$ lower bound on the cumulative complexity of sorting independent of the time order in which the sorted output is produced.

Proof of Lemma 4.17. Choose α as the constant for γ in Proposition 4.16 and let $\beta = \sqrt{2}\alpha/6$. Let $\mathcal{C}$ be a circuit with at most $\beta\sqrt{kn}$ layers that outputs the k correct pairs (x_i, r_j) with probability p. Let $R = \{r_1, \ldots r_k\}$ where $r_1 < r_2 < \ldots < r_k$. We describe our construction of a circuit $\mathcal{C}'(X)$ solving the k-threshold problem on inputs $X = x_1, \ldots, x_{n/2}$ with exactly k ones in terms of a function $f : [n/2] \to R$. Given f, we re-interpret the input as follows: we replace each x_i with $x'_i = f(i)x_i$, add k

dummy values of 0, and add one dummy value of j for each $j \in \{n/2+1, \ldots, n\} \setminus R$. Doing this gives us an input $X' = x'_1, \ldots, x'_n$ that has $n/2$ zeroes.

If we assume that f is 1-1 on the k ones of X, then the image of the ones of X will be R and there will be precisely one element of X' for each $j \in \{n/2+1, \ldots, n\}$. Therefore the element of rank $j > n/2$ in X' will have value j, and hence the rank $r_1, \ldots, r_k$ elements of X' will be the images of precisely those elements of X with $x_i = 1$.

To obtain perfect soundness, we cannot rely on the output of $\mathcal{C}(X')$ and must be able to check that each of the output ranks was truly mapped to by a distinct one of X. For each element x_i of X we simply append its index i as $\log_2 n$ low order bits to its image x'_i and append an all-zero bit-vector of length $\log_2 n$ to each dummy value to obtain input X''. Doing so will not change the ranks of the elements in X', but will allow recovery of the k indices that should be the ones in X. In particular, circuit $\mathcal{C}'(X)$ will run $\mathcal{C}(X'')$ and then for each output x''_j with low order bits i, $\mathcal{C}'(X)$ will query x_i, accepting if and only if all of those $x_i = 1$. More precisely, since the mapping from each x_i to the corresponding x''_i is only a function of f, x_i, and i, as long as $\mathcal{C}'(X)$ has an explicit representation of f, it can simulate each query of $\mathcal{C}(X'')$ with two oracle queries to X. Since $\mathcal{C}'$ has at most

$$2\beta\sqrt{kn} + k \leq 3\beta\sqrt{kn} \leq \alpha\sqrt{kn/2}$$

layers, by Proposition 4.16, it can only accept with probability $\leq e^{-\gamma k}$ on inputs with k ones.

We now observe that for each fixed X with exactly k ones, for a randomly chosen function $f : [n/2] \to R$, the probability that f is 1-1 on the ones of X' is exactly $k!/k^k \geq e^{1-k}$. Therefore $\mathcal{C}'(X)$ will give the indices of the k ones in X with probability[3] at least $p \cdot e^{1-k}$. However, this probability must be at most $e^{-\gamma k}$, so we

[3]Note that though this is exponentially small in k it is still sufficiently large compared to the completeness required in the lower bound for the k-threshold problem.

can conclude that $p \leq e^{(1-\gamma)k-1}$. In the event that R is a contiguous set of integers, observe that any choice for the function f will make X'' have the ones of X become ranks $r_1, \ldots, r_k$. So the probability of finding the ones is at least $p \leq e^{-\gamma k}$. □

By setting k and γ appropriately, Lemma 4.17 gives a useful upper bound on the number of fixed ranks successfully output by any $\beta\sqrt{Sn}$ query quantum circuit that has access to S qubits of input dependent initial state. To handle input-dependent initial state, we will need to use the following proposition from our Chapter 3.

Proposition 3.5 ((Aaronson, 2005))**.** *Let $\mathcal{C}$ be a quantum circuit, ρ be an S-qubit (possibly mixed) state, and π_{mix} be the S-qubit maximally mixed state. If $\mathcal{C}$ starting in initial state ρ produces some output z with probability p, then $\mathcal{C}$ starting in state π_{mix} will produce z with probability q which is at least $p/2^S$.*

This allows us to bound the overall progress made by any short quantum circuit.

Lemma 4.18. *There is a constant $\beta > 0$ such that, for any fixed set of $S \leq \beta^2 n$ ranks that are greater than $n/2$, the probability that any quantum circuit $\mathcal{C}$ with at most $\beta\sqrt{Sn}$ queries and S qubits of input-dependent initial state correctly produces the outputs for these S ranks is at most $1/e$.*

Proof. Choose β as the constant when γ is $1 + \ln(4)$ in Lemma 4.17. Applying Proposition 3.5 to the bound in Lemma 4.17 gives us that a quantum circuit with S qubits of input-dependent state can produce a fixed set of $k \leq \beta^2 n$ outputs larger than median with a probability at most $2^{2S}e^{(1-\gamma)k-1}$. Since $\gamma = 1 + \ln(4)$ setting $k = S$ gives that this probability is $\leq 1/e$. □

Theorem 4.19. *When n is sufficiently large, any quantum circuit $\mathcal{C}$ for sorting a list of length n with success probability at least $1/e$ and at most T layers that produces its sorted outputs in any fixed time order requires cumulative memory that is $\Omega(n^3/T)$.*

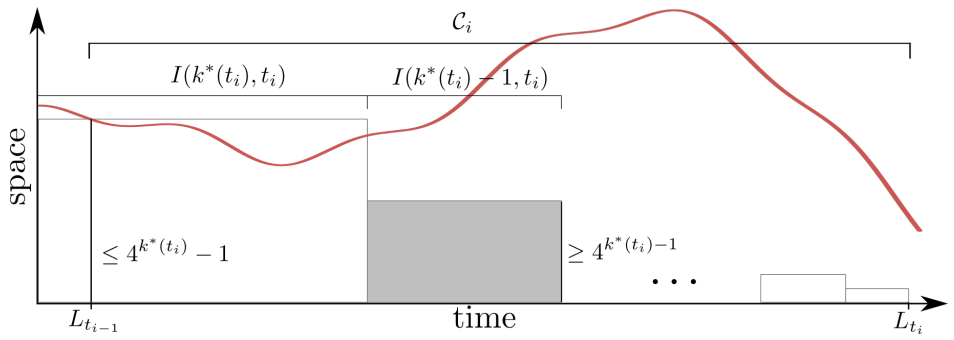

Figure 4.2: How we define the block $\mathcal{C}_i$ that ends at layer L_{t_i}. The red line is a plot of $\mathcal{C}$'s space over time. The gray layers are the ones used to lower bound the cumulative memory complexity of $\mathcal{C}_i$, as each of these layers uses at least $4^{k^*(t_i)-1}$ qubits and the length of this interval is $\frac{\beta}{2}2^{k^*(t_i)-1}\sqrt{n}$.

Proof. We partition $\mathcal{C}$ into blocks with large cumulative memory that can only produce a small number of outputs. We achieve this by starting at last unpartitioned layer and finding a suitably low space layer before it so that we can apply Lemma 4.18 to upper bound the number of correct outputs that can be produced in that block with a success probability of at least $1/e$. Let β be the constant from Lemma 4.18 and $k^*(t)$ be the least non-negative integer value of k such that the interval:

$$I(k,t) = \left[t - \frac{\beta}{2}(2^{k+1}-1)\sqrt{n}, t - \frac{\beta}{2}(2^k-1)\sqrt{n}\right]$$

contains some t' such that $S_{t'} \leq 4^k - 1$. We recursively define our blocks as follows. Let ℓ be the number of blocks generated by this method. The final block $\mathcal{C}_\ell$ starts with the first layer $t_{\ell-1} \in I(k^*(T), T)$ where $S_{t_{\ell-1}} \leq 4^{k^*(T)} - 1$ and ends with layer $t_\ell = T$. Let t_i be the first layer of block $\mathcal{C}_{i+1}$. Then the block $\mathcal{C}_i$ starts with the first layer $t_{i-1} \in I(k^*(t_i), t_i)$ where $S_{t_{i-1}} \leq 4^{k^*(t_i)} - 1$ and ends with t_i. See Figure 4.2 for an illustration of our partitioning. Since $S_0 = 0$ we know that $k^*(t) \leq \log(T)$. Likewise since $S_t > 0$ when $t > 0$, for all $t > \frac{\beta}{2}\sqrt{n}$ we know that $0 < k^*(t) \leq \log(T)$.

Block $\mathcal{C}_i$ starts with less than $4^{k^*(t_i)}$ qubits of initial state and has length at most $\beta 2^{k^*(t_i)}\sqrt{n}$; so by Lemma 4.18, if $4^{k^*(t_i)} \leq \beta^2 n$, the block $\mathcal{C}_i$ can output at most $4^{k^*(t_i)}$ inputs with failure probability at most $1/e$. Additionally $\mathcal{C}_i$ has at least

$\frac{\beta}{2}2^{k^*(t_i)-1}\sqrt{n}$ layers so

$$\sum_{i=1}^{\ell} \frac{\beta}{4} 2^{k^*(t_i)}\sqrt{n} \leq T \tag{4.2}$$

and each of these layers has at least $4^{k^*(t_i)-1}$ qubits[4], so the cumulative memory of $\mathcal{C}_i$ is at least $\frac{\beta}{2}2^{3k^*(t_i)-3}\sqrt{n}$ so

$$CM(\mathcal{C}) \geq \sum_{i=1}^{\ell} \frac{\beta}{2} 2^{3k^*(t_i)-3}\sqrt{n}. \tag{4.3}$$

We now have two possibilities: If we have some i such that $4^{k^*(t_i)} > \beta^2 n$, the cumulative memory of $\mathcal{C}_i$ alone is at least $\beta^4 n^2/16$ which is $\Omega(n^2)$ and hence $\mathcal{C}$ has cumulatively memory $\Omega(n^3/T)$ since $T \geq n$. Otherwise, since we require that the algorithm is correct with probability at least $1/e$, each block $\mathcal{C}_i$ can produce at most $4^{k^*(t_i)}$ outputs. Since our circuit must output all $n/2$ elements larger than the median, we know $\sum_{i=1}^{\ell} 4^{k^*(t_i)} \geq n/2$. For convenience we define $w_i = 2^{k^*(t_i)}$ which allows us to express the constraints as

$$CM(\mathcal{C}) \geq \frac{\beta}{16}\sqrt{n}\sum_{i=1}^{\ell} w_i^3 \text{ and } \frac{\beta}{4}\sqrt{n}\sum_{i=1}^{\ell} w_i \leq T \text{ and } \sum_{i=1}^{\ell} w_i^2 \geq n/2. \tag{4.4}$$

Minimizing $\sum_{i=1}^{\ell} w_i^3$ is a non-convex optimization problem and can instead be solved using

$$\text{Minimize} \quad \sum_{i=1}^{\ell} x_i^3 \text{ subject to } \sum_{i=1}^{\ell} x_i^2 \geq \xi \text{ and } \sum_{i=1}^{\ell} x_i \leq \xi \text{ and } \forall i, x_i \geq 0, \tag{4.5}$$

for $x_i = \frac{8T}{\beta n^{3/2}} w_i$ and $\xi = \frac{32T^2}{\beta^2 n^2}$. Lemma G.3 in the Appendix shows that for non-negative x_i with $\sum x_i \leq \sum x_i^2$, we have $\sum x_i^2 \leq \sum x_i^3$. Thus $\sum x_i^3 \geq \xi$ and applying the variable substitution gives us: $\sum_{i=1}^{\ell} w_i^3 \geq \frac{\beta n^{5/2}}{16T}$. Plugging this into Equation (4.4) gives us the bound: $CM(\mathcal{C}) \geq \frac{\beta^2 n^3}{256T}$ and hence the cumulative memory of $\mathcal{C}$ is $\Omega(n^3/T)$. □

[4]This may not hold for $\mathcal{C}_1$ with length less than $\frac{\beta}{2}\sqrt{N}$, but Lemma 4.17 gives us that this number of layers is insufficient to find a fixed rank input with probability at least $1/e$. Thus we can omit such a block from our analysis.

In Section G.0 we also show how we can change the length of the blocks to generalize the above proof to arbitrary success probabilities.

When n is sufficiently large, any quantum circuit $\mathcal{C}$ for sorting a list of length n with failure probability at most δ and at most T layers that produces its sorted outputs in any fixed time order requires cumulative memory that is $\Omega((1-\delta)n^3/T)$.

4.6 General methods for proving cumulative memory lower bounds

Our method involves adapting techniques previously used to prove tradeoff lower bounds on worst-case time and worst-case space. We show that the same properties that yield lower bounds on the product of time and space in the worst case can also be used to produce nearly identical lower bounds on cumulative memory. To do so, we first revisit the standard approach to such time-space tradeoff lower bounds.

The standard method for time-space tradeoff lower bounds for multi-output functions

Consider a multi-output function f on D^n where the output $f(x)$ is either unordered (the output is simply a set of elements from R) or ordered (the output is a vector of elements from R). Then $|f(x)|$ is either the size of the set or the length of the vector of elements. The standard method for obtaining ordinary time-space tradeoff lower bounds for multi-output functions on D-way branching programs is the following:

The part that depends on f: Choose a suitable probability distribution μ on D^n, often simply the uniform distribution on D^n and then:

(A) Prove that $\Pr_{x\sim\mu}[|f(x)| \geq m] \geq \alpha$.

(B) Prove that for all $k \leq m'$ and any branching program B of height $\leq h'(k,n)$,

the probability for $x \sim \mu$ that B produces at least k correct output values of f on input x is at most $C \cdot K^{-k}$ for some m', h', $K = K(R, n)$, and constant C independent of n.

Observe that under any distribution μ, a branching program with ordered outputs that makes no queries can produce k outputs that are all correct with probability at least $|R|^{-k}$, so the bound in (B) shows that, roughly, up to the difference between K and $|R|$ there is not much gained by using a branching program of height h.

The generic completion: In the following outline we omit integer rounding for readability.

- Let $S' = S + \log_2 T$ and suppose that

$$S' \leq m' \log_2 K - \log_2(2C/\alpha). \tag{4.6}$$

- Let $k = [S' + \log_2(2C/\alpha)]/\log_2 K$, which is at most m' by hypothesis on S', and define $h(S', n) = h'(k, n)$.

- Divide time T into $\ell = T/h$ blocks of length $h = h(S', n)$.

- The original branching program can be split into at most $T \cdot 2^S = 2^{S'}$ sub-branching programs of height $\leq h$, each beginning at a boundary node between layers. By Property (B) and a union bound, for $x \sim \mu$ the probability that at least one of these $\leq 2^{S'}$ sub-branching programs of height at most h produces k correct outputs on input x is at most $2^{S'} \cdot C \cdot K^{-k} \leq \alpha/2$ by our choice of k.

- Under distribution μ, by (A), with probability at least α, an input $x \sim \mu$ has some block of time where at least $m/\ell = m \cdot h(S', n)/T$ outputs of f must be produced on input x.

- If $m \cdot h(S', n)/T \leq k$, this can occur for at most an $\alpha/2$ fraction of inputs under μ. Therefore we have $m \cdot h(S', n)/T > k = [S' + \log_2(2C/\alpha)]/\log_2 K$ and hence since $h(S', n) \geq h(S, n)$, combining with Equation (4.6), we have

$$T \cdot (S + \log_2 T) = T \cdot S' \geq \min\left(m\ h(S, n),\ m'\ n'\right)\ \log_2 K - \log_2(C/\alpha) \cdot T$$

  where $n' \leq n$ is the decision tree complexity of f and hence a lower bound on T.

Remark 4.1. Though it will not impact our argument, for many instances of the above outline, the proof of Property (B) is shown for a decision tree of the same height by proving an analog for the conditional probability along each path in the decision tree separately; this will apply to the tree as a whole since the paths are followed by disjoint inputs, so Property (B) follows from the alternative property below:

(B') For any partial assignment τ of $k \leq m'$ output values over R and any restriction (i.e., partial assignment) π of $h'(k, n)$ coordinates within D^n,

$$\Pr_{x \sim \mu}[f(x) \text{ is consistent with } \tau \mid x \text{ is consistent with } \pi] \leq C \cdot K^{-k}.$$

Observe that Property (B') is only a slightly more general version of Property (*) from the introduction where $C = 1$, m' is arbitrary, and h' is used instead of h.

Remark 4.2. The above method still gives lower bounds for many multi-output functions $g : D^N \to R^M$ that have individual output values that are easy to compute or large portions of the input space on which they are easy to compute. The bounds follow by applying the method to some subfunction f of g given by $f(x) = \Pi_O(g(x, \pi))$ where π is a partial assignment to the input coordinates and Π_O is a projection onto a subset O of output coordinates. In the subsequent discussions we ignore this issue, but the idea can be applied to all of our lower bound methods.

A general extension to cumulative memory bounds

To give a feel for the basic ideas of the method, we first show this for a simple case. Observe that, other than the separate bound on time, the lower bound on

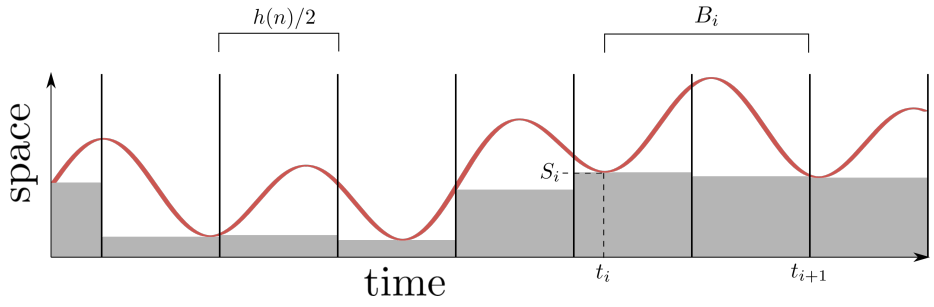

Figure 4.3: Our generic method for choosing blocks when $h(k,n) = h(n)$. The area marked in gray corresponds to the cumulative memory lower bound we obtain.

cumulative memory usage we prove in this case is asymptotically identical to the bound achieved for the product of time and worst-case space using the standard outline.

Theorem 4.20. *Let $c > 0$. Suppose that properties (A) and (B) apply for $h'(k,n) = h(n)$, $m' = m$, and $\alpha = C = 1$. If $T \log_2 T \leq \frac{m\ h(n)\ \log_2 K}{6(c+1)}$ then the cumulative memory used in computing $f : D^n \to R^m$ in time T with success probability at least T^{-c} is at least $\frac{1}{6}\, m\ h(n) \log_2 K$.*

Proof. Fix a deterministic branching program P of length T computing f. Rather than choosing fixed blocks of height $h = h(n)$, layers of nodes at a fixed distance from each other, and a fixed target of k outputs per block, we choose the block boundaries depending on the properties of P and the target k depending on the property of the boundary layer chosen.

Let $H = \lfloor h(n)/2 \rfloor$. We break P into $\ell = \lceil T/H \rceil$ time segments of length H working backwards from step T so that the first segment may be shorter than the rest. We let $t_1 = 0$ and for $1 < i \leq \ell$ we let $t_i = \arg\min\{\ |L_t| \ :\ T - (\ell - i + 1) \cdot H \leq t < T - (\ell - i) \cdot H\ \}$ be the time step with the fewest nodes among all time steps $t \in [T - (\ell - i + 1) \cdot H, T - (\ell - i) \cdot H)$.

The i-th time block of P will be between times t_i and t_{i+1}. Observe that by construction $|t_{i+1} - t_i| \leq h(n)$, so each block has length at most $h(n)$. This

construction is shown in Figure 4.3. Set $S_i = \log_2 |L_{t_i}|$ so that L_{t_i} has at 2^{S_i} nodes. By definition of each t_i, the cumulative memory used by P,

$$CM(P) \geq \sum_{i=1}^{\ell} S_i \cdot H. \tag{4.7}$$

(Note that since $S_1 = 0$, it does not matter that the first segment is shorter than the rest[5].)

We now define the target k_i for the number of output values produced in each time block to be the smallest integer such that $K^{-k_i} \leq 2^{-S_i}/T^{c+1}$. That is,

$$k_i = \lceil (S_i + (c+1)\log_2 T)/\log_2 K \rceil.$$

For $x \sim \mu$, for each $i \in [\ell]$ and each sub-branching program B rooted at some node in L_{t_i} and extending until time t_{i+1}, by our choice of k_i and Property (B), if $k_i \leq m$, the probability that B produces at least k_i correct outputs on input x is at most $2^{-S_i}/T^{c+1}$. Therefore, by a union bound, for $x \sim \mu$ the probability that P produces at least k_i correct outputs in the i-th time block on input x is at most $|L_{t_i}| \cdot 2^{-S_i}/T^{c+1} = 1/T^{c+1}$. Therefore, if each $k_i \leq m$, the probability for $x \sim \mu$ that there is some i such that P produces at least k_i correct outputs on input x during the i-th block is at most $\ell/T^{c+1} < T^c$. Therefore, if each $k_i \leq m$, the probability for $x \sim \mu$ that P produces at most $\sum_{i=1}^{\ell}(k_i - 1)$ correct outputs in total on input x is $> 1 - 1/T^c$.

If each $k_i \leq m$, since P must produce m correct outputs on $x \in D^n$ with probability at least $1/T^c$, we must have $\sum_{i=1}^{\ell}(k_i - 1) \geq m$. On the other hand, if some $k_i > m$ we have the same bound. Using our definition of k_i we have $\sum_{i=1}^{\ell}[(S_i + (c+1)\log_2 T)]/\log_2 K] \geq m$ or $\sum_{i=1}^{\ell}(S_i + (c+1)\log_2 T) \geq m \cdot \log_2 K$. Plugging in the bound (4.7) on the cumulative memory and the value of ℓ, it implies that $CM(P)/H + (c+1)\lceil T/H \rceil \cdot \log_2 T \geq m \cdot \log_2 K$ or that $CM(P) + (c+1)T\log_2 T \geq$

[5]This simplifies some calculations and is the prime reason for starting the time segment boundaries at T rather than at 0.

$\frac{1}{3}\ m \cdot h(n) \cdot \log_2 K$, where the 3 on the right rather than a 2 allows us to remove the ceiling. Therefore either

$$T \log_2 T > \frac{m \cdot h(n) \cdot \log_2 K}{6(c+1)} \text{ or } CM(P) \geq \frac{1}{6}\ m\ h(n)\ \log_2 K. \qquad \square$$

In the general version of our theorem there are a number of additional complications, most especially because the branching program height limit $h(k, n)$ in Property (B) *can depend on* k, the target for the number of outputs produced. This forces the lengths of the blocks and the space used at the boundaries between blocks to depend on each other in a quite delicate way. In order to discuss the impact of that dependence and state our general theorem, we need the following definition.

Definition 4.21. Given a non-decreasing function $p : \mathbb{R} \to \mathbb{R}$ with $p(1) = 1$, we define $p^{-1} : \mathbb{R} \to \mathbb{R} \cup \{\infty\}$ by $p^{-1}(k) = \min\{j \mid p(j) \geq k\}$. We also define the *loss*, $\mathcal{L}_p$, of p by

$$\mathcal{L}_p(n) = \min_{1 \leq k \leq p(n)} \frac{\sum_{j=1}^{k} p^{-1}(j)}{k \cdot p^{-1}(k)}.$$

Intuitively, $\mathcal{L}_p$ characterizes the smallest possible ratio between how much cumulative memory we can prove a block uses versus the time-space product complexity[6] of that block. This ratio depends on a function p — which is h_0 in our use case — indicating how the length of a block should scale with its initial space. Choices of p that make $\mathcal{L}_p$ large indicate that our techniques are able to recover most of the time-space product cost as cumulative memory costs while choices of p that make $\mathcal{L}_p$ small indicate a regime where our analysis may not be tight. Nevertheless, with some mild assumptions on p, we are able to give strong bounds on $\mathcal{L}_p$, showing that the loss is never too large.

Lemma 4.22. *The following hold for every non-decreasing function* $p : \mathbb{R} \to \mathbb{R}$ *with* $p(1) = 1$*:*

[6]Technically the product of time and initial space, which is what matters for determining the amount of produced output.

(a) $1/p(n) \le \mathcal{L}_p(n) \le 1$.

(b) If p *is a polynomial function* $p(s) = s^{1/c}$ *then* $\mathcal{L}_p(n) > 1/2^{c+1}$.

(c) For any $c > 1$, $\mathcal{L}_p(n) \ge \min_{1 \le s \le n} \dfrac{p(s) - p(s/c)}{cp(s)}$.

(d) We say that p *is* nice *if it is differentiable and there is an integer* $c > 1$ *such that for all* x, $p'(cx) \ge p'(x)/c$. *If* p *is nice then* $\mathcal{L}_p(n)$ *is* $\Omega(1/\log_2 n)$. *This is tight for* p *with* $p(s) = 1 + \log_2 s$.

We prove these technical statements in Section G.0. Here is our full general theorem.

Theorem 4.23. *Let* $c > 0$. *Suppose that function* f *defined on* D^n *has properties (A) and (B) with* α *that is* $1/n^{O(1)}$ *and* m' *that is* $\omega(\log_2 n)$. *For* $s > 0$, *define* $h(s, n)$ *to be* $h'(k, n)$ *for* $k = s/\log_2 K$. *Suppose that* $h(s, n) = h_0(s)\, h_1(n)$ *with* $h_0(1) = 1$ *and* h_0 *is constant or a differentiable function such that* $s/h_0(s)$ *is increasing and concave. Define* $S^* = S^*(T, n)$ *by*

$$\frac{S^*}{h_0(S^*)} = \frac{m\ h_1(n)\ \log_2 K}{6T}.$$

(a) Either

$$T \log_2(2CT^{c+1}/\alpha) > \frac{1}{6}\, m\ h_1(n)\ \log_2 K,$$

which implies that T *is* $\Omega(\frac{m\ h_1(n)\ \log K}{\log n})$, *or the cumulative memory used by a randomized branching program in computing* f *in time* T *with error* $\varepsilon \le \alpha(1 - 1/(2T^c))$ *is at least*

$$\frac{1}{6}\, \mathcal{L}_{h_0}(n \log_2 |D|) \cdot \min\left(m\ h(S^*(T, n), n),\ 3m'\ h'(m'/2, n)\right) \cdot \log_2 K.$$

(b) Further any randomized random-access machine computing f *in time* T *with error* $\varepsilon \le \alpha(1 - 1/(2T^c))$ *requires cumulative memory*

$$\Omega\left(\mathcal{L}_{h_0}(n \log_2 |D|) \cdot \min\left(m\ h(S^*(T, n), n),\ m'\ h'(m'/2, n)\right) \cdot \log_2 K\right).$$

Before we give the proof of the theorem, we note that by Lemma 4.22, in the case that h_0 is constant or $h_0(s) = s^{\Delta}$ for some constant $\Delta > 0$, which together account for all existing applications we are aware of, the function $\mathcal{L}_{h_0}$ is lower bounded by a constant. In the latter case, h_0 is differentiable, has $h_0(s) = 1$, and the function $s/h_0(s) = s^{1-\Delta}$ is increasing and concave so it satisfies the conditions of our theorem. By using $\alpha = 1$, $m' = m$, and $C = 1$ with h from Property (*) in place of h' in Property (B'), Theorem 4.23 yields Theorem 4.2.

More generally, the value S^* in the statement of this theorem is at least a constant factor times the value of S used in the generic time-space tradeoff lower bound methodology. Therefore, for example, the cumulative memory lower bound derived for random-access machines via Theorem 4.23 is close to the lower bound on the product of time and worst-case space given by standard methods.

Proof of Theorem 4.23. We prove both (a) and (b) directly for branching programs, which can model random-access machines, and will describe the small variation that occurs in the case that the branching program in question comes from a random-access machine. To prove these properties for randomized branching programs, by Yao's Lemma (Yao, 1977) it suffices to prove the properties for deterministic branching programs that have error at most ε under distribution μ. Fix a (deterministic) branching program P of length T computing f with error at most ε under distribution μ. Without loss of generality, P has maximum space usage at most $S^{max} = n \log_2 |D|$ space since there are at most $|D^n|$ inputs.

Let $H = \lfloor h_1(n)/2 \rfloor$. We break P into $\ell = \lceil T/H \rceil$ time segments of length H working backwards from step T so that the first segment may be shorter than the rest. We then choose a sequence of *candidates* for the time steps in which to begin new blocks, as follows: We let $\tau_1 = 0$ and for $1 < i \leq \ell$ we let

$$\tau_i = \arg\min\{\ |L_t|\ :\ T - (\ell - i + 1) \cdot H \leq t < T - (\ell - i) \cdot H\ \}$$

be the time step with the fewest nodes among all time steps $t \in [T - (\ell - i + 1) \cdot H, T - (\ell - i) \cdot H)$. Set $\sigma_i = \log_2 |L_{\tau_i}|$ so that L_{τ_i} has at 2^{σ_i} nodes. This segment contributes at least $\sigma_i \cdot H$ to the cumulative memory bound of P.

To choose the beginning t_{i^*} of the last time block[7]. We find the smallest k such that $h_0(\sigma_{\ell-k+1}) < k$. Such a k must exist since h_0 is a non-decreasing non-negative function, $h_0(1) = 1$ and $\sigma_1 = 0 < 1$. We now observe that the length of the last block is at most $k \cdot H$ which by choice of k is less than $h(\sigma_{\ell-k+1}, n)$ and hence we have satisfied the requirements for Property (B) to apply at each starting node of the last time block.

By our choice of each τ_i, the cumulative memory used in the last k segments is at least $\sum_{j=1}^{k} \sigma_{\ell+1-j} \cdot H$. Further, since k was chosen as smallest with the above property, we know that for every $j \in [k-1]$ we have $h_0(\sigma_{\ell-j+1}) \geq j$ Hence we have $\sigma_{\ell-j+1} \geq h_0^{-1}(j)$, and we get a cumulative memory bound for the last k segments of at least

$$(\sigma_{\ell-k+1} + \sum_{j=1}^{k-1} h_0^{-1}(j)) \cdot H. \tag{4.8}$$

Claim 4.24. $\sigma_{\ell-k+1} + \sum_{j=1}^{k-1} h_0^{-1}(j) \geq \mathcal{L}_{h_0}(S^{max}) \cdot \sigma_{\ell-k+1} \cdot k$.

Proof of Claim. Observe that it suffices to prove the claim when we replace $\sigma_{\ell-k+1}$, which appears on both sides, by a larger quantity. In particular, we show how to prove the claim with $h_0^{-1}(k)$ instead, which is larger since $h_0(\sigma_{\ell-k+1}) < k$. But this follows immediately since by definition $\mathcal{L}_{h_0}(S^{max}) \leq \frac{\sum_{j=1}^{k} h_0^{-1}(j)}{k \cdot h_0^{-1}(k)}$, which is equivalent to what we want to prove. □

Write $S_{i^*} = \sigma_{\ell-k+1}$. By the claim, the cumulative memory contribution associated with the last block beginning at t_{i^*} is at least $\mathcal{L}_{h_0}(S^{max}) \cdot S_{i^*} \cdot h_0(S_{i^*})H$.

[7]Since we are working backwards from the end of the branching program, and we do not know how many segments are included in each block, we don't actually know this index until things stop with $t_1 = 0$

We repeat this in turn to find the time step for the beginning of the next block from the end, t_{i^*-1}. One small difference now is that there is a last partial segment of height at most H from the beginning of segment containing t_{i^*} to layer t_{i^*}. However, this only adds at most $h_1(n)/2$ to the length of the segment which still remains well within the height bound of $h(S_{i^*-1}, n) = h_0(S_{i^*-1})h_1(n)$ for Property (B) to apply.

Repeating this back to the beginning of the branching program we obtain a decomposition of the branching program into some number i^* of blocks, the i-th block beginning at time step t_i with 2^{S_i} nodes, height between $h_0(S_i)H$ and $h_0(S_i)H + H \leq 2h_0(S_i)H$, and with an associated cumulative memory contribution in the i-th block of $\geq \mathcal{L}_{h_0}(S^{max}) \cdot S_i \cdot h_0(S_i)H$. (This is correct even for the partial block starting at time $t_1 = 0$ since $S_1 = 0$.) Since we know that $i^* \leq \ell$, for convenience, we also define $S_i = 0$ for $i^* + 1 \leq i \leq \ell$. Then, by definition

$$CM(P) \geq \mathcal{L}_{h_0}(S^{max}) \cdot \left(\sum_{i=1}^{i^*} S_i \cdot h_0(S_i) \right) \cdot H = \mathcal{L}_{h_0}(S^{max}) \cdot \left(\sum_{i=1}^{\ell} S_i \cdot h_0(S_i) \right) \tag{4.9}$$

$$\text{and} \qquad \sum_{i=1}^{\ell} h_0(S_i) \leq T/H. \tag{4.10}$$

As in the previous argument for the simple case, for $i \leq i^*$, we define the target k_i for the number of output values produced in each time block to be the smallest integer such that $C \cdot K^{-k_i} \leq 2^{-S_i}\alpha/(2T^{c+1})$. That is, $k_i = \lceil (S_i + \log_2(2CT^{c+1}/\alpha))/\log_2 K \rceil$.

If $k_i > m'$ for some i, then $S_i \geq m' \cdot \log_2 K - \log_2(2CT^{c+1}/\alpha) \geq (m' \log_2 K)/2$ since m' is $\omega(\log n)$ and $1/\alpha$ and T are $n^{O(1)}$. Therefore $h_0(S_i) \geq h'(m'/2, n)$ and hence

$$CM(P) \geq \frac{1}{2} \mathcal{L}_{h_0}(S^{max}) \cdot m' \cdot h'(m'/2, n) \cdot \log_2 K$$

Suppose instead that $k_i \leq m'$ for all $i \leq i^*$. Then, for $x \sim \mu$, for each $i \in [i^*]$ and each sub-branching program B rooted at some node in L_{t_i} and extending until time t_{i+1}, by our choice of k_i and Property (B), the probability that B produces at least k_i correct outputs on input x is at most $\alpha \cdot 2^{-S_i}/(2T^{c+1})$. Therefore, by a union

bound, for $x \sim \mu$ the probability that P produces at least k_i correct outputs in the i-th time block on input x is at most

$$|L_{t_i}| \cdot \alpha \cdot 2^{-S_i}/(2T^{c+1}) = \alpha/(2T^{c+1})$$

and hence the probability for $x \sim \mu$ that there is some i such that P produces at least k_i correct outputs on input x during the i-th block is at most $\ell \cdot \alpha/(2T^{c+1}) < \alpha/(2T^c)$. Therefore, the probability for $x \sim \mu$ that P produces at most $\sum_{i=1}^{\ell}(k_i - 1)$ correct outputs in total on input x is $> 1 - \alpha/(2T^c)$.

Since, by Property (A) and the maximum error it allows, P must produce at least m correct outputs with probability at least $\alpha - \varepsilon \geq \alpha - \alpha(1 - 1/(2T^c)) = \alpha/(2T^c)$ for $x \sim \mu$, we must have $\sum_{i=1}^{i^*}(k_i - 1) \geq m$. Using our definition of k_i we obtain

$$\sum_{i=1}^{i^*}(S_i + \log_2(2CT^{c+1}/\alpha)) \geq m \log_2 K.$$

This is the one place in the proof where there is a distinction between an arbitrary branching program and one that comes from a random access machine.

We first start with the case of arbitrary branching programs: Note that $i^* \leq \ell = \lceil T/H \rceil = \lceil T/\lfloor h_1(n)/2 \rfloor \rceil$. Suppose that $T \log_2(2CT^{c+1}/\alpha) \leq \frac{1}{6}\, m \cdot h_1(n) \cdot \log_2 K$. Then, even with rounding, we obtain $\sum_{i=1}^{i^*} S_i \geq \frac{1}{2}\, m \log_2 K$.

Unlike an arbitrary branching program that may do non-trivial computation with sub-logarithmic S_i, a random-access machine with even one register requires at least $\log_2 n$ bits of memory (just to index the input for example) and hence $S_i + \log_2(2CT^{c+1}/\alpha)$ will be $O(S_i)$, since T is at most polynomial in n without loss of generality and $1/\alpha$ is at most polynomial in n by assumption. Therefore we obtain that $\sum_{i=1}^{i^*} S_i$ is $\Omega(m \log_2 K)$ without the assumption on T.

In the remainder we continue the argument for the case of arbitrary branching programs and track the constants involved. The same argument obviously applies for programs coming from random-access machines with slightly different constants that

we will not track. In particular, since $S_i = 0$ for $i > i^*$ we have

$$\sum_{i=1}^{\ell} S_i \geq \frac{1}{2}\, m \cdot \log_2 K. \tag{4.11}$$

From this point we need to do something different from the argument in the simple case because the lower bound on the total cumulative memory contribution is given by Equation (4.9) and is not simply $\sum_{i=1}^{\ell} S_i \cdot H$. Instead, we combine Equation (4.11) and Equation (4.10) using the following technical lemma that we prove in Section G.0.

Lemma 4.25. *Let $p : \mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0}$ be a differentiable function such that $q(x) = x/p(x)$ is a concave increasing function of x. For $x_1, x_2, \ldots \in \mathbb{R}^{\geq 0}$, if $\sum_i x_i \geq K$ and $\sum_i p(x_i) \leq L$ then $\sum_i x_i p(x_i) \geq q^{-1}(K/L) \cdot L$.*

In our application of Lemma 4.25, $p = h_0$, $K = \frac{1}{2}\, m \cdot \log_2 K$, and $L = T/H$. Let S^* be the solution to $\frac{S^*}{h_0(S^*)} = K/L = \frac{m \cdot H \cdot \log_2 K}{2T} \geq \frac{m \cdot h_1(n) \log_2 K}{6T}$. Then Lemma 4.25 implies that $\sum_{i=1}^{\ell} S_i \cdot h_0(S_i) \geq S^* \cdot T/H = \frac{1}{2}, m \cdot h_0(S^*) \cdot \log_2 K.$ and hence

$$CM(P) \geq \mathcal{L}_{h_0}(S^{max}) \cdot \frac{1}{2}\, m \cdot h_0(S^*) \cdot H \cdot \log_2 K \geq \frac{1}{6}\, \mathcal{L}_{h_0}(S^{max}) \cdot m \cdot h(S^*, n) \cdot \log_2 K$$

since $H = \lfloor h_1(n)/2 \rfloor$ and $h(S^*, n) = h_0(S^*) \cdot h_1(n)$. □

In the special case that $h_0(s) = s^{\Delta}$ (and indeed for any nice function h_0), there is an alternative variant of the above in which one breaks up time into exponentially growing segments starting with time step T. We used that alternative approach in Section 4.5.

Remark 4.3. If we restrict our attention to $o(m' \log K)$-space-bounded computation, then each $k_i \leq m'$ and the cumulative memory bound for a branching program in Theorem 4.23 becomes $\frac{1}{6}\, \mathcal{L}_{h_0}(n \log_2 |D|) \cdot m \cdot h(S^*(T, n), n) \cdot \log_2 K$. And the bound for RAM cumulative memory becomes $\Omega\left(\mathcal{L}_{h_0}(n \log_2 |D|) \cdot m \cdot h(S^*(T, n), n) \cdot \log_2 K\right)$.

Generic method for quantum time-space tradeoffs

Quantum circuit time-space lower bounds have the same general structure as their classical branching program counterparts. They require a lemma similar to (B) that gives an exponentially small probability of producing k outputs with a small number of queries.

Lemma 4.26 (Quantum generic property)**.** *There are constants C and K such that, for all k where $\log_K n \leq k \leq m'$ and any quantum circuit $\mathcal{C}$ with at most $h'(k, n)$ layers, there exists a distribution μ such that when $x \sim \mu$, the probability that $\mathcal{C}$ produces at least k correct output values of $f(x)$ is at most $C \cdot K^{-k}$.*

Note that, unlike branching programs, quantum circuits must always have at least $\log_K n$ qubits in order to query their inputs. Hence, the quantum generic property only needs to apply when algorithms are expected to produce at least $\log_K n$ outputs.

Such lemmas have historically been proving using direct product theorems (Klauck et al., 2007; Ambainis et al., 2009) or the recording query technique (Hamoudi and Magniez, 2021). Quantum time-space tradeoffs use the same blocking strategy as branching programs; however, they cannot use union bounds to account for input dependent state at the start of a block. Instead, Proposition 3.5 lets us apply Lemma 4.26 to blocks in the middle of a quantum circuit.

The 2^{2S} factor in Proposition 3.5 means that a quantum time-space or cumulative memory lower bound will be half of what you would expect from a classical bound with the same parameters. Since a quantum circuit must have $\log_2 n$ qubits to make a query, we know that the space between layers is always at least $\log_2 n$. Therefore the generic time-space tradeoff for quantum circuits is

$$T \cdot S \text{ is } \Omega\left(\min\{m\ h'(S, n), m'\ Q(f)\} \cdot \log_2 K\right)$$

where $Q(f)$ is the bounded-error quantum query complexity of f.

Generic method for quantum cumulative complexity bounds

Our generic argument can just as easily be applied to quantum lower bounds for problems where we have an instance of Lemma 4.26 using Proposition 3.5 to bound the number of outputs produced even with initial input-dependent state. Since quantum circuits require at least $\log_2 n$ qubits to hold the query index, the bounds derived are like those from Theorem 4.23(b).

Corollary 4.27. *Let $c > 0$. Suppose that function $f : D^n \to R^m$ satisfies generic Lemma 4.26 with m' that is $\omega(\log_2 n)$. For $s > 0$, let $h(s, n) = h'(s/\log_2 K, n)$. Let $h(s, n) = h_0(s)h_1(n)$ where $h_0(1) = 1$ and h_0 is constant or a differentiable function where $s/h_0(s)$ is increasing and concave. Let S^* be defined by:*

$$\frac{S^*}{h_0(S^*)} = \frac{m\ h_1(n)\ \log_2 K}{6T}$$

Then the cumulative memory used by a quantum circuit that computes f in time T with error $\varepsilon \leq (1 - 1/(2T^c))$ is at least

$$\frac{1}{6}\,\mathcal{L}_{h_0}(n \log_2 |D|) \cdot \min\{m\ h(S^*, n),\ 3m'\ h'(m'/2, n)\} \cdot \log_2 K.$$

Additionally if the quantum circuit uses $o(m' \log K)$ qubits, then the cumulative memory bound instead is $\frac{1}{6}\,\mathcal{L}_{h_0}(n \log_2 |D|) \cdot m \cdot h(S^*, n) \cdot \log_2 K$.

4.7 Applications of our general theorems to classical and quantum computation

Theorems 4.20 and 4.23 are powerful tools that can convert most existing time-space lower bounds into asymptotically equivalent lower bounds on the required cumulative memory. We give a few examples to indicate how our general theorems can be used.

4.7.1 Classical applications of the generic method

Unique elements Define $Unique_{n,N} : [N]^n \to \mathcal{P}([N])$ by

$$Unique_{n,N}(x) = \{\, x_i \mid x_j \neq x_i \text{ for all } j \neq i \,\}.$$

Proposition 4.28 (Lemmas 2 and 3 in (Beame, 1991))**.** *For the uniform distribution μ on $[N]^n$ with $N \geq n$,*

(A) $\Pr_{x\sim\mu}[\|Unique_{n,N}(x)\| \geq n/(2e)] \geq 1/(2e-1)$

(B') For any partial assignment τ of $k \leq n/4$ output values over $[N]$ and any restriction π of $n/4$ coordinates in $[n]^n$, $\Pr_{x\sim\mu}[Unique_{n,N}(x)$ *is consistent with* $\tau \mid x$ *is consistent with* $\pi] \leq e^{-k/2}$.

The above lemma is sufficient to prove that TS is $\Omega(n^2)$ for the unique elements problem, and can be easily extended to a cumulative complexity bound using Theorem 4.23.

Theorem 4.29. *For $n \geq N$, any branching program computing $Unique_{n,N}$ in time T and probability at least $4/5$ requires T to be $\Omega(n^2/\log n)$ or $CM(P)$ to be $\Omega(n^2)$. Further, any random access machine computing $Unique_{n,N}$ with probability at least $4/5$ requires cumulative memory $\Omega(n^2)$*

Proof. By Proposition 4.28, $Unique_{n,N}$ satisfies conditions (A) and (B) of Section 4.6 with $h'(k,n) = n/4$, $m' = n/4$, $m = n/(2e)$, $C = 1$, $K = 1/(2\ln N)$ and $\alpha = 1/(2e-1) \geq 0.2254$. Since $h'(k,n)$ is independent of k, the function h_0 is the constant function 1 and $h_1(n) = n/4$ so $\mathcal{L}_{h_0} \equiv 1$. We then apply Theorem 4.23 to obtain the claimed lower bounds. □

The above theorem is tight for $N = n$ using the algorithm in (Beame, 1991).

Linear Algebra We consider linear algebra over some finite field $\mathbb{F}$. Let D be a subset of $\mathbb{F}$ with d elements.

Definition 3.2. An $m \times n$ matrix is (g,h,c)-*rigid* iff every $k \times w$ submatrix where $k \leq g$ and $w \geq n-h$ has rank at least ck. We call $(g,h,1)$-rigid matrices (g,h)-rigid.

Matrix rigidity is a robust notion of rank and is an important property for proving time-space and cumulative complexity lower bounds for linear algebra. Fortunately, Abrahamson proved that there are always rigid square matrices.

Proposition 3.4. *[Lemma 4.3 in (Abrahamson, 1991)] There is a constant $\gamma \in (0, \frac{1}{2})$ such that at least a $1 - d^{-1}(2/3)^{\gamma n}$ fraction of the matrices over $D^{n \times n}$ with $|D| = d$ are $(\gamma n, \gamma n)$-rigid.*

Abrahamson shows in (Abrahamson, 1991) that for any constant $c \in (0, \frac{1}{2})$ and $m \times n$ matrix A that is (cm, cn, c)-rigid, any D-way branching program that computes the function $f(x) = Ax$ with expected time $\overline{T} \geq n$ and expected space[8] $\overline{S}$ has $\overline{TS} = \Omega(nm \log d)$ where $d = |D|$. We restate the key property used in that proof.

Proposition 4.30 (Theorem 4.6 in (Abrahamson, 1991))**.** *Let $c \in (0, \frac{1}{2}]$, A be any $m \times n$ matrix that is (g, h, c)-rigid and f be the function $f(x) = Ax$ over $\mathbb{F}$. Let μ be the uniform distribution on D^n for $D \subseteq \mathbb{F}$ with $|D| = d$. For any restriction π of h coordinates to values in D and any partial assignment τ of $k \leq g$ output coordinates over $\mathbb{F}^m$,*

$$\Pr_{x \sim \mu}[(f(x)||\tau) \mid (x||\pi)] \leq d^{-ck}$$

Proof. By permuting the rows and columns of A, we can assume that π restricts the first h coordinates of x and τ restricts the first $k \leq g$ output coordinates. We can view our system as:

$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

Where A is fixed by our problem instance and both x_1 and y_1 are fixed by π and τ respectively. We want to find the number of $x_2 \in D^{n-h}$ that can satisfy the equation $A_{1,2}x_2 = y_1 - A_{1,1}x_1$. Since A is a (g, h, c)-rigid matrix and $A_{1,2}$ is a $k \times (n - h)$ sub matrix of A, we know that $A_{1,2}$ has rank at least $\lceil ck \rceil$. The set of x_2 that agree with π

[8] (Abrahamson, 1991) defines expected space as the expected value of the $\log_2$ of the largest number of a branching program node that is visited during a computation under best case node numbering.

and τ have dimension at most $n - h - \lceil ck \rceil$, so there are at most $d^{n-h-\lceil ck \rceil}$ choices for x_2. Since there are d^{n-h} choices for x_2 that are consistent with π, if μ is the uniform distribution over D^n:

$$\Pr_{x \sim \mu}[(f(x)||\tau)|(x||\pi)] \leq \frac{d^{n-h-\lfloor ck \rfloor}}{d^{n-h}} \leq d^{-ck}$$

□

Theorem 4.31. *Let $c \in (0, \frac{1}{2}]$. Let A be an $m \times n$ matrix over D, with $|D| = d$ that is $(g(m), h(n), c)$-rigid. Then, for any D-way branching program P computing $f(x) = Ax$ in T steps with probability at least $n^{-O(1)}$, either T is $\Omega(g(m)h(n)\log_n d)$ or $CM(P)$ is $\Omega(g(m)h(n)\log d)$. Further, computing f on a random access machine requires cumulative memory $\Omega(g(m)h(n)\log d)$ unconditionally.*

Proof. We invoke Theorem 4.20 using Proposition 4.30 to obtain Property (B') with $K = d^c$ and $C = 1$. Property (A) is trivial since $|f(x)| = m$. □

By Proposition 3.4 we know that for some constant γ, a random matrix has a good chance of being $(\gamma m, \gamma n)$-rigid. This means that computing $f(x) = Ax$ for a random matrix A in time at most T is likely to require either the cumulative memory or $T \log T$ to be $\Omega(mn \log d)$. Since Yesha (Yesha, 1984) proved that the $n \times n$ DFT matrix is $(n/4, n/4, 1/2)$-rigid, the DFT is a concrete example where the cumulative memory or $T \log T$ is $\Omega(n^2 \log d)$; other examples include generalized Fourier transform matrices over finite fields (Beame et al., 2001, Lemma 28).

Corollary 4.32. *If A is an $n \times n$ generalized Fourier transform matrix over field $\mathbb{F}$ with characteristic relatively prime to n then any random-access machine computing $f(x) = Ax$ for $x \in D^n$ where $D \subseteq \mathbb{F}$ has $\|D\| = d$ with probability at least $n^{-O(1)}$ requires cumulative memory that is $\Omega(n^2 \log d)$.*

It is easy to see that our lower bound is asymptotically optimal in these cases.

Proposition 4.33 (Theorem 7.1 in (Abrahamson, 1991)). *Let $f : D^{2n^2} \to \mathbb{F}^{n^2}$ for $D \subseteq \mathbb{F}$ and $d = |D|$ be the matrix multiplication function, γ be the constant from Proposition 3.4, and μ be the uniform distribution over $(\gamma m, \gamma n)$-rigid matrices. Choose any integers h and k such that $2(h/\gamma n)^2 \leq k$. If $\gamma n \geq 1$ then for any D-way branching program B of height $\leq h$ the probability that B produces at least k correct output values of f is at most $d^{2-\gamma k/4}$.*

Theorem 4.34. *Multiplying two random matrices in D^{n^2} with $D \subseteq \mathbb{F}$ and $d = |D|$ with probability at least $n^{-O(1)}$ requires time T that is $\Omega((n^3\sqrt{\log d})/\log n)$ or cumulative memory $\Omega((n^6 \log d)/T)$. On random access machines, the cumulative memory bound is unconditional.*

Proof. Proposition 4.33 lets us apply Theorem 4.23 with $m = n^2$, $h'(k, n) = \gamma n\sqrt{k/2}$, $C = d^2$, $\alpha = 1$, and $K = d^{\gamma/4}$. This gives us that $h(s, n) = n\sqrt{2\gamma s/\log_2 d}$, so $h_0(s) = \sqrt{s}$. Then we get that $\sqrt{S^*} = \frac{mn\sqrt{2\gamma/\log_2 d}\cdot\log_2 K}{6T}$ and hence

$$S^* \text{ is } \Omega\left(\frac{n^6 \log d}{T^2}\right).$$

Therefore, we get that either

$$T \text{ is } \Omega\left(\frac{n^3 \log^{1/2} d}{\log n}\right)$$

or, since the loss function for h_0 is a constant, the cumulative memory is

$$\Omega\left(\min\left((n^6 \log d)/T, n^5 \log^{1/2} d\right)\right).$$

Since the decision tree complexity of matrix multiplication is $\Omega(n^2)$, this expression is $\Omega((n^6 \log d)/T)$. For random access machines, the same cumulative memory bound applies without the condition on T. □

Matrix problems We can extend our classical results on Boolean matrix multiplication to get a matching lower bound on the cumulative memory complexity. As a reminder, we used the following key lemma to obtain our time-space tradeoff:

Lemma 3.53. *Let $\varepsilon, \gamma > 0$ be the constants from Proposition 3.44. Let k be an integer such that $L(k) \leq n/2$. Any randomized algorithm with at most $(2\varepsilon/3)kn/L(k)$ queries to x can only produce k correct output values of $n \times n$ Boolean matrix product $A \bullet B$ with probability at most $2^{-\gamma k}$.*

By Lemma 3.51 $L(k) \leq 5\sqrt{k}$. We can use this to obtain the following bound on the cumulative memory complexity.

Corollary 4.35. *Any classical circuit (or other sequential model in which each output value is produced at a fixed time step) computing $n \times n$ Boolean matrix-multiplication or Boolean matrix squaring with T queries and space S with success probability more than $1/(2T)$ must have cumulative memory that is $\Omega(n^6/T)$.*

Proof. Using Lemma 3.53 we can apply Theorem 4.23 with $m(n) = m'(n) = n^2$, $h'(k, n) = (2\varepsilon/15)\sqrt{k}n$ and $K = 2^{\gamma/2}$. This gives us that $h(s, n) = (2\varepsilon/15)\sqrt{2k/\gamma}n$ and

$$S^* = \Omega\left(\frac{n^6}{T^2}\right)$$

Hence, either T is $\Omega(n^3/\log n)$ or, as $\mathcal{L}_{h_0}$ is lower bounded by a constant, the cumulative memory complexity for multiplication is

$$\Omega(\min(n^6/T, n^4)) = \Omega(n^6/T)$$

As T must be $\Omega(n^2)$ to have a success probability of at least $1/(2T)$.

The bound for squaring follows from the same reduction presented in Corollary 3.40. □

4.7.2 Quantum applications of the generic method

Disjoint Collision Pairs Finding In (Hamoudi and Magniez, 2021) the authors considered the problem of finding k disjoint collisions in a random function $f : [m] \to [n]$, and were able to prove a time-space tradeoff that T^3S is $\Omega(k^3n)$ for circuits that

solve the problem with success probability 2/3. Specifically, they consider circuits that must output triples $(x_{j_{2i}}, x_{j_{2i+1}}, y_{j_i})$ where $f(x_{j_{2i}}) = f(x_{j_{2i+1}}) = y_{j_i}$. To obtain this result, they prove the following theorem using the recording query technique:

Proposition 4.36 (Theorem 4.6 in (Hamoudi and Magniez, 2021))**.** *For all $1 \leq k \leq n/8$ and any quantum circuit $\mathcal{C}$ with at most t quantum queries to a random function $f : [m] \to [n]$, the probability that $\mathcal{C}$ produces at least k disjoint collisions in f is at most $O(t^3/(k^2 n))^{k/2} + 2^{-k}$.*

The above theorem can be extended to a lemma matching Lemma 4.26 by choosing a sufficiently small constant δ and setting $T = \delta\, k^{2/3} n^{1/3}$ to obtain a probability of at most 2^{1-k}. This is sufficient to obtain a matching lower bound on the cumulative memory complexity using Corollary 4.27.

Theorem 4.37. *Finding $\omega(\log_2 n) \leq k \leq n/8$ disjoint collisions in a random function $f : [m] \to [n]$ with probability at least 2/3 requires cumulative memory $\Omega(k^3 n/T^2)$.*

Proof. Our discussion based on Proposition 4.36 lets us apply Corollary 4.27 with $m = m' = k, h'(k, n) = \delta k^{2/3} n^{1/3}$, and $C = K = 2$. Thus we have $h(s, n) = h'(s, n)$ and h_0 is a differentiable function where $s/h_0(s)$ is an increasing and concave function. With these parameters, we have:

$$S^* \text{ is } \Omega\left(\frac{k^3 n}{T^3}\right)$$

By Corollary 4.27 with the observation that the loss is constant we get that the quantum cumulative memory is:

$$\Omega\left(\min\left(\frac{k^3 n}{T^2}, k^{5/3} n^{1/3}\right)\right).$$

By Proposition 4.36 we know that any quantum circuit with at most $T' = \alpha k^{2/3} n^{1/3}$ layers can produce k disjoint collisions with probability at most 2^{1-k}. Thus we know that $T > T'$ and our cumulative memory bound becomes $\Omega(k^3 n/T^2)$. □

Quantum cumulative memory lower bounds for matrix problems Our results in Chapter 3 can also be converted to give matching cumulative memory lower bounds. Here are the bounds we obtain from the results in Section 3.4.

We start by reminding the reader of the key lemma used to prove our quantum time-space lower bound for matrix vector products.

Lemma 3.22. *Let A be any (k, h, c)-rigid $m \times n$ matrix over a finite field $\mathbb{F}$ and let $f_A : D^n \to \mathbb{F}^m$ for $D \subseteq \mathbb{F}$ be defined by $f_A(x) = Ax$. Then for $\alpha > 0$ and for input x sampled uniformly from D^n and any quantum circuit $\mathcal{C}$ with at most αh queries to x, the probability that $\mathcal{C}$ produces k correct output values of $f_A(x)$ is at most $\lceil h/(ck) \rceil^2 \, (4^{H_2(\alpha)}/|D|^{1-\alpha})^{ck}$.*

This lemma can be used to obtain a matching lower bound on cumulative memory complexity.

Corollary 4.38. *Let $\gamma > 0$ and $c \in (0, 1/2]$ be fixed. If A is a $(\gamma n, \gamma n, c)$-rigid $n \times n$ matrix over a field $\mathbb{F}$ then any quantum circuit using time T and space S that computes the function $f : D^n \to \mathbb{F}^n$ for $D \subseteq \mathbb{F}$ with $d = |D|$ given by $f(x) = Ax$ with success probability larger than $1/T$ requires cumulative memory that is $\Omega(n^2 \log d)$.*

Proof. We can apply Corollary 4.27 where $C = \lceil 1/c \rceil$, $m'(n) = \gamma n$, $m(n) = n$, $h'(k, n) = \alpha\gamma n$, and $K = d^{c/6}$. Thus we have $h(s, n) = \alpha\gamma n$ and h_0 is a constant function. Since h_0 is a constant function, $\mathcal{L}_{h_0}$ is lower bounded by a constant and $h(S^*, n)$ is not sensitive to S^*. This allows us to directly conclude that the cumulative memory complexity is at least $\Omega(n^2 \log d)$. □

Directly applying this in place of Theorem 3.45 gives us matching cumulative (CM) memory lower bounds for Corollary 3.26 through Corollary 3.35.

Corollary 4.39. *Let $\mathbb{F}$ be a field and $D \subseteq \mathbb{F}$ such that $d = |D|$. Any quantum circuit with inputs over D that computes the DFT or vector convolution requires CM that is*

$\Omega(n^2 \log d)$. *Any quantum circuit that computes the product of three matrices, matrix cubing, or matrix inversion requires CM that is* $\Omega(n^4 \log d)$. *Any quantum circuit that solves* $n \times n$ *systems of linear equations requires CM that is* $\Omega(n^3 \log d)$. *Additionally, any quantum circuit that multiplies two* n *bit binary numbers requires CM that is* $\Omega(n^2/\log^2 n)$.

Next are the results from Section 3.5. The key lemma used to prove our lower bound in this section was the following:

Lemma 3.37. *Let* $\gamma \in (0, 1/2)$ *and* $f : D^{n^2} \times D^{n^2} \to \mathbb{F}^{n^2}$ *for* $D \subseteq \mathbb{F}$ *with* $|D| = d$ *be defined by* $f(A, B) = AB$. *Then for any constant* $\beta > 0$ *and quantum circuit* $\mathcal{C}$ *with at most* $h = \beta\gamma n\sqrt{k/2}$ *queries to input matrices* A, B *sampled uniformly from* D^{n^2}, *the probability that* A *and* B *are* $(\gamma n, \gamma n)$*-rigid and* $\mathcal{C}$ *produces* k *correct output values of* $f(A, B)$ *is at most* $16 \min(k, n)^{\sqrt{2k}}(4^{H_2(4\beta)}/d^{1-4\beta})^{k/4}$

Again we can use this lemma to get a matching lower bound on the cumulative memory complexity.

Corollary 4.40. *Let* $\mathbb{F}$ *be a field and* $D \subseteq \mathbb{F}$ *with* $d = |D|$. *If* $\mathcal{C}$ *is a quantum circuit that computes the function* $f : D^{2n^2} \to \mathbb{F}^{n^2}$ *for sufficiently large* n *given by* $f(A, B) = AB$ *or the function* $g : D^{n^2} \to \mathbb{F}^{n^2}$ *given by* $f(A) = A^2$ *with success probability at least* $1/T$, *then* $\mathcal{C}$ *must have cumulative memory complexity* $\Omega(n^6 \log(d)\ /T)$.

Proof. When $\beta \leq 0.0429$ we have $1 - 4\beta - H_2(4\beta) > 1/6$ so the bound in Lemma 3.37 is at most $16 \min(k, n)\sqrt{k/2}d^{-k/24}$. Additionally, since we only need to consider k that is $\Omega(\log n)$, we can assume that k is sufficiently large that $16 \min(k, n)\sqrt{k/2}d^{-k/24} \leq 16d^{-k/48}$. Thus apply Corollary 4.27 with $C(n) = 16, m'(n) = m(n) = n^2, h'(k, n) = \beta\gamma n\sqrt{k/2}$, and $K(n) = d^{1/48}$. Thus we have $h(s, n) = \beta\gamma n\sqrt{24s/\log_2 d}$ and so h_0 is a differentiable function where $s/h_0(s)$ is an increasing and concave function. With these parameters, we have:

$$S^* \text{ is } \Omega\left(\frac{n^6 \log d}{T^2}\right)$$

By Corollary 4.27 with the observation that the loss is constant we get that the quantum cumulative memory is:

$$\Omega\left(\min\left(n^6/T, n^4\right) \cdot \log d\right).$$

Since any quantum algorithm with $o(n^2)$ queries cannot solve this problem with probability $1/T$, this is always $\Omega(n^6 \log d\ /T)$. The bound for computing g follows from the same reduction □

Finally, we can get quantum cumulative memory lower bounds for our results on Boolean matrix problems in Section 3.6. Our results in this section used the following key lemma:

Lemma 3.46. *There are constants $\varepsilon, \gamma > 0$ such that the following holds. Let $k < n^2/100$ be an integer. For any quantum circuit $\mathcal{C}$ with at most $\varepsilon k^{3/4} n^{1/2}$ queries to x, the probability that $\mathcal{C}$ produces k correct output values of $n \times n$ Boolean matrix multiplication $A \bullet B$ is at most $2^{-\gamma k}$.*

We use the above to obtain the following bound on the cumulative memory complexity of Boolean matrix multiplication.

Corollary 4.41. *Any quantum circuit computing $n \times n$ Boolean matrix multiplication $A \bullet B$ or Boolean matrix squaring $A \bullet A$ with T queries, space S, and success probability more than $1/(2T)$ must have cumulative memory that is $\Omega(n^{10}/T^3)$*

Proof. Using Lemma 3.46, we can apply Corollary 4.27 with $C = 1$, $m(n) = n^2$, $m'(n) = n^2/8$, $h'(k, n) = \varepsilon k^{3/4}\sqrt{n}$, $K(n) = 2^{\gamma}$. This gives us that $h(s, n) = (s/\gamma)^{3/4}\sqrt{n}$ and

$$S^* \text{ is } \Omega\left(\frac{n^{10}}{T^4}\right)$$

And as the loss function is lower bounded by a constant, the cumulative memory complexity for multiplication is

$$\Omega(\min(n^{10}/T^3, n^4)) = \Omega(n^{10}/T^3)$$

as T must be $\Omega(n^2)$ for any algorithm with success probability at least $1/(2T)$.

The bound for squaring follows from the same reduction presented in Corollary 3.40. $\square$

Part II

Energy and the thermodynamics of computation

Chapter 5: Energy-efficient Brownian sampling and computation

5.1 Introduction

Energy is an increasingly important constraint on our capacity for performing computation. A study in 2022 found that the global computational power is increasing at 68% per year while our energy production capacity is only growing at 8% per year (Luccioni et al., 2023). There has been a growing international interest in building increasingly large language models — a computational endeavor with massive energetic costs. The model GPT-3 required an estimated 1,287 MWh of electricity to train, resulting in an estimated carbon footprint of around 502 tons of CO_2 (Luccioni et al., 2023). These models are not only expensive to train, but also expensive to use. Chairman of Alphabet John Hennessy said in an interview that interactions with a LLM likely cost 10 times as much as a regular keyword search, although this overhead is expected to go down with the help of fine-tuning models (Dastin and Nellis, 2023). Benchmarking on Meta's 65 billion parameter open source LLM model LLaMA 65B (Touvron et al., 2023) on the Alpaca dataset (Taori et al., 2023) using MIT's SuperCloud high-performance computing system (Reuther et al., 2018) shows that the model requires approximately 3 to 4 Joules per generated token (Samsi et al., 2023). As computation becomes an increasingly large share of global energy expenditure, efforts to improve the efficiency of computational devices will yield larger savings. In addition to strain on the world's energy infrastructure, it is vitally important to optimize energetic costs in settings like high performance and mobile computing. The intense energetic costs of computation often result in large amounts of heat dissipation, making cooling an expensive engineering consideration and cost for data centers. Improving the energy efficiency of mobile devices increases the time needed between recharging and reduces the strain on batteries before they need to be replaced.

Perhaps the best known insights regarding energetic costs of computation

come from the works of Landauer and Bennett (Landauer, 1961; Bennett, 1973). Landauer observed that, in accordance with the second law of thermodynamics, any computational device which erases information must also dissipate heat. Bennett then observed that such thermodynamic 'Landauer costs' can be avoided by making computation logically reversible. However, doing so comes with an unavoidable cost in the time and space of computation (Frank and Ammer, 2017). The ideas of Landauer and Bennett have inspired a multitude of follow-up works in computer science that investigate logical reversibility in computation (Li and Vitanyi, 1996; Demaine et al., 2016). However, recent developments in the emerging field of stochastic thermodynamics indicate that logical reversibility is not the be-all and end-all of energetic costs in computation.

Stochastic thermodynamics paints a slightly different picture. The field is interested in optimizing a quantity known as *entropy production*, corresponding to overall change in entropy associated with completing a process (Esposito and Van den Broeck, 2011; Sagawa, 2014; Strasberg et al., 2015; Wolpert, 2019; Wolpert et al., 2024). In principle, a process can recuperate some Landauer costs by running it in reverse and reabsorbing all the dissipated heat. This is only possible when the process has zero entropy production (i.e., the entropy of the universe did not change), making it 'thermodynamically reversible'. Moreover, as we show in Section 6.4, a process that is logically reversible might still result in positive entropy production depending on the details of its implementation. The thermodynamic reversibility of a computation is harder to define than logical reversibility, as it not only depends on the details of the computation, but also the choice of input distribution and the underlying physics of the device. In the near term, these kinds of optimizations are unlikely to be the most efficient route to improving energetic costs of computation. Higher order energetic inefficiencies in computational devices dominate theoretical thermodynamic considerations. Nevertheless, after such engineering inefficiencies are addressed, understanding and optimizing the stochastic thermodynamics of computation is essential to further reducing energetic costs.

Reversible computation A (deterministic) computation is considered *logically reversible* if each reachable configuration of the machine has at most one possible predecessor state. Thus knowing the current configuration of a logically reversible machine uniquely determines its history. Whenever two or more predecessor configurations of a machine lead to the same successor configuration, the step is said to be irreversible. A typical example of an irreversible operation is bit erasure where an erased bit could have had the value zero or one but in either case becomes zero.

As argued by Landauer and Bennett, computation must dissipate $k\mathbb{T}\ln 2$ heat per irreversible step (or more concretely $k\mathbb{T}\ln n$ if there are n possible predecessor configurations, where the scaling factor $k\mathbb{T}$, in units of Joules, is the Boltzmann constant times temperature in Kelvin) (Landauer, 1961; Bennett, 1973). For simplicity, in the rest of this dissertation, we will think of heat as being measured in units such that $k\mathbb{T} = 1$ as we will be discussing systems at a fixed temperature. Intuitively this heat dissipation, also known as a Landauer cost, is required to counteract a decrease in the entropy of a machine's state when it erases information. This is a necessary consequence of the second law of thermodynamics, which states that the entropy of the universe can never decrease. However, *reversible* computation, in which no step is irreversible, has no such per-step heat dissipation lower-bound. Moreover, it is possible to reprogram any deterministic computation as a sequence of logically reversible steps, provided that the computation is allowed to save a copy of its input (Bennett, 1973). While general methods of constructing reversible computations from irreversible computation come at some asymptotic cost to the space and/or time complexity of the program, the overhead can be made surprisingly small (Bennett, 1989; Lange et al., 2000; Saeedi and Markov, 2013; Aaronson et al., 2017).

Although reversible computation gives a way to sidestep Landauer costs, it is insufficient to make "zero energy" computation. The two best known models for low energy computing are Ballistic (Fredkin and Toffoli, 1982) and Brownian computation (Bennett, 1973, 1982; Bennett and Landauer, 2011). Ballistic computation is designed using odd degrees of freedom (e.g. momentum) to drive computation in the forward

direction. However, as pointed out in (Bennett, 1982), this model assumes the ability to design devices with perfect precision and isolate them from all thermal noise. On the other hand, Brownian computation takes advantage of thermal fluctuations to drive computation. Without any physical asymmetry, such as energy dissipation, to bias a computation in the desired direction, a device governed by Brownian motion will simply take a random walk over its state space. Thus, Bennett argues that simply making a computation logically reversible and not using any energy to drive it forward turns it into a 1D unbiased random walk that "does not deserve to be called a computation" (Bennett, 1973). While such a machine will eventually enter a state containing the desired output, it only spends a negligible fraction of its time in such a state. Bennett gives a simple remedy by putting a small (constant) bias on each forward step, i.e. an energetic cost per step. Our results show how to do logically reversible computation, with high probability of output, but without requiring any such energetic forward bias per step.

Another line of work has argued that *reusable* computation without energy dissipation is impossible, even for reversible computation (Norton, 2013; Strasberg et al., 2015). These works argue that a logically reversible computer running a computation that takes T steps and starts in a fixed input state will relax to a distribution over states that has $\Omega(\log T)$ bits of entropy. Thus resetting such a machine in order to run it on a new input would require dissipating $\Omega(\log T)$ heat. Our results will sidestep this issue to perform computation where the thermodynamic costs are independent of T.

Stochastic thermodynamics of computation Recent developments in the field of stochastic thermodynamics indicate that counting irreversible operations may not be the correct way to measure the energy costs of a computation. Instead, physicists (Esposito and Van den Broeck, 2011; Sagawa, 2014; Strasberg et al., 2015; Wolpert, 2019; Wolpert et al., 2024) suggest that the entropy production — or the total change of the entropy of the universe — is a more accurate method of lower bounding the

energy costs of computation. It is possible to have logically irreversible operations like bit erasure that can be performed with zero entropy production for particular input distributions. While Landauer showed that heat must be released when erasing a bit, that heat can later be reabsorbed if the bit is reassigned to a uniformly random value.

More specifically one can distinguish between the heat that must be released by a process and the net change of the entropy of the universe due to a process. These quantities are referred to in the literature as entropy flow (EF) and entropy production (EP) respectively. The entropy flow of a process is lower bounded by its Landauer cost and any excess entropy flow becomes entropy production. A process is said to be *thermodynamically reversible* if its entropy production is zero. EP is an extremely relevant thermodynamic quantity, as it can be viewed as a fundamental lower bound on the energetic costs of a process.

Computational sampling In this chapter, we find that in addition to typical function computation, our techniques are applicable to a type of computation called computational distribution sampling. While more niche in its applications than function computation, distribution sampling is an important problem. For example sampling large prime numbers is essential for creating (non-quantum) cryptographically secure RSA keys and sampling from the co-domain of hash functions is the primary mechanism behind many proof of work schemes (Rivest et al., 1978; Nakamoto, 2009). Distribution sampling can even be a computationally hard problem — in fact boson sampling is the problem behind many recent attempts at quantum supremacy (Aaronson and Arkhipov, 2011).

Prior work has defined physical processes that can be used to sample from desirable distributions $\mathcal{D}$ in a thermodynamically reversible manner (Owen et al., 2019; Cappelletti et al., 2020). In these works the distribution $\mathcal{D}$ is "hard-coded" and no computation is performed; the device is defined in terms of the target distribution's probability density function (PDF). Our constructions are defined in terms of a

computational device $\mathcal{M}$ (e.g. a Turing machine) that maps a uniformly random value in $\{0,1\}^n$ to a corresponding sample from the target distribution $\mathcal{D}$ in T steps on all inputs. With access to a description of $\mathcal{M}$, we can always efficiently construct our sampling device. However, computing the PDF of $\mathcal{D}$ from a description of $\mathcal{M}$ is highly non-trivial. If $\mathcal{D}$ represents the output of a Boolean circuit $\mathcal{M}$ on a uniformly random input then computing the PDF of $\mathcal{D}$ is a hard problem. Moreover, any construction built directly from the PDF encodes the truth table of this function and thus requires descriptional complexity that scales linearly with the size of the distribution's support. Since our constructions are based on $\mathcal{M}$, their descriptional complexity is directly proportional to that of $\mathcal{M}$ which is typically significantly smaller.

5.1.1 Main results

We design machines that only change state according to a 'Brownian motion' like random walk over their state space. One can think of these devices as being driven only by thermal fluctuations. By itself such a machine may not seem very useful — even if the unbiased random walk causes the machine to perform a useful computation — intuition suggests it will not be able to "lock in" its answer. Nonetheless, we design such machines in a way where, after giving the random walk sufficient time to explore the computational space, an observer can measure its state and will likely find that the desired computation has completed (specifically, with constant probability per computation). Whereas Bennett (Bennett, 1973) added heat dissipation to drive his reversible computation to the output state, our constructions avoid any such cost while only slightly increasing the time complexity of the underlying computation.

Turing machine function computation In Theorem 5.15 we show that any Turing machine that runs in time $\leq T$ on all inputs of size $n \in \mathbb{N}$ and has outputs of size $\leq m$ can be simulated by another device that runs in $\Theta(T^2 + n^2)$ time and produces the intended output with probability $\geq 1/4$ while only expending $O(n + m)$ energy. *Notably, the energy cost is not dependent on the running time of the original*

Turing machine. This is in contrast to Bennett's approach (Bennett, 1973) to this same task, which requires per-step heat dissipation to drive the computation forward.

Computational sampling results We also give two sampling computation results in Section 5.3 and Section 5.4. We design machines where either: (1) after sufficient time the machine has a constant probability of producing independent and identically distributed (i.i.d.) samples according to our desired probability distribution in Theorem 5.9, or (2) after sufficient time the machine will always be in a state containing samples from a distribution close to the target in Theorem 5.12. These kinds of machines are called Las Vegas and Monte Carlo, respectively, in homage to the similar categories of randomized algorithms. Given a logically reversible Turing machine that samples from the target distribution on all inputs in T steps and uses n bits of randomness, our constructions produce samples after $\Theta(T^2 + n^2)$ time. Moreover, as with the function computation result, our sampling machines are driven entirely by Brownian motion and do not require any heat dissipation during computation or resetting; *the only thermodynamic costs associated with our devices are due to observations necessary to record desired outputs, which scale linearly with the output size.*

5.1.2 Roadmap

In Section 5.2 we give a brief overview of our underlying physical assumptions and computational models. Next in Section 5.3 and Section 5.4 we present our Las Vegas and Monte Carlo constructions. In Section 5.5 we show how our sampling constructions can be extended to perform function computation. Finally, in Section 5.6 we give some concluding remarks on the results in this chapter.

5.2 Preliminaries

Brownian computation In this chapter we are primarily interested in designing abstract devices that can perform computation without being driven forward with a battery. Instead, we desire our device to perform computation entirely driven by thermal fluctuations caused by interactions with its environment (so-called Brownian computation; see the Introduction for a discussion of alternative settings.) We often take for granted that computational models like Turing machines or branching programs advance forwards in time from an initial to a final configuration. However, in Brownian computation, the particles have no intrinsic notion of which transitions correspond to performing computation in the "forward" direction. Unless energy is dissipated to bias a computation in the correct direction, it will be just as likely to transition to any valid predecessor state as any desired successor state.

Thus we can view the physical evolution of a machine as a random walk on something called its *configuration graph*.

Definition 5.1. Let $\mathcal{M}$ be some computational device with a (possibly infinite) set of possible states Σ and a transition function $f : \Sigma \to 2^{\Sigma}$. Then the *configuration graph* $\mathcal{G}_{\mathcal{M}}$ of $\mathcal{M}$ is the directed graph featuring one node for each element of Σ and the directed edge (u, v) when $v \in f(u)$.

Note that following the directed edges in $\mathcal{G}_{\mathcal{M}}$ corresponds to simulating a computation of $\mathcal{M}$. In the language of computation graphs, we can say that a machine $\mathcal{M}$ is *deterministic* if all nodes have maximum out-degree 1. Likewise, we can say a computation is *logically reversible* if all nodes have maximum in-degree 1.

Without energy to bias computation in the correct direction, one should expect a machine to evolve according to an *unbiased* random walk on an undirected version of its computation graph. However, energy can be dissipated to bias a computation towards particularly desired transitions. We can model this by assigning *free energy* to the configurations of a machine.

Definition 5.2. Let $\mathcal{M}$ be a machine with a configuration graph $\mathcal{G}_{\mathcal{M}}$ and a set of possible states Σ and transition function f. We can augment $\mathcal{M}$ with a *free energy function* $\Delta G : \Sigma \to \mathbb{R} \cup \{\infty\}$. For a free energy function ΔG, when in some state $A \in \Sigma$, the rate of transitioning to another state $B \in \Sigma$ is given by Metropolis dynamics (Metropolis et al., 1953). More specifically the time until A transitions to B is distributed according to an exponential random variable with parameter $\lambda_{A \to B}$ given by:

$$\lambda_{A \to B} = \begin{cases} \kappa e^{\Delta G(A) - \Delta G(B)} & \text{if } \Delta G(A) < \Delta G(B) \\ \kappa & \text{otherwise} \end{cases} \tag{5.1}$$

When either $B \in f(A)$ or $A \in f(B)$ and zero otherwise. When there are multiple possible transitions from state A, the time to transition is the minimum of the exponential random variables for each possible transition and the argmin of these variables dictates the next state of $\mathcal{M}$. In other words, let $\mathcal{M}$ starts in a state A and $f(A) = \{X_1, \ldots, X_k\}$. Let $t_1, \ldots, t_k$ be random variables drawn according to an exponential random variable with parameter $\lambda_{A \to X_1}, \ldots, \lambda_{A \to X_k}$. Then the time to transition to the next state is $\min(t_1, \ldots, t_k)$ and the next state will be $\operatorname{argmin}(t_1, \ldots, t_k)$. We call a machine driven by such dynamics a *Brownian computer*.

We choose to consider Metropolis dynamics in this chapter, although there are other choices such as Kawasaki dynamics (Kawasaki, 1966), as Metropolis dynamics provide a fundamental upper bound on the rate at which transitions can occur. While the time between measurements in our protocols depend on the choice of dynamics, they have no impact on the equilibrium distribution over states. Thus our results could be directly adapted to other kinetic models.

For a logically reversible computation, Metropolis dynamics mean that by dissipating constant energy per forward step of a computation that takes T steps, one can have a constant probability of ending up at the correct final state (Bennett, 1973). Logically irreversible steps require additional energy dissipation. If a transition has p predecessor states and q successor states, then $\log(p/q) + \varepsilon$ energy must be

dissipated as heat to make the transition have the same forward bias as a logically reversible step (Landauer, 1961; Bennett, 1973). This *Landauer* cost can be avoided by making a computation logically reversible (Bennett, 1973); however, all known general techniques for converting irreversible algorithms to reversible ones feature an asymptotic time or space overhead (Bennett, 1989; Lange et al., 2000; Saeedi and Markov, 2013; Aaronson et al., 2017). In fact, relative to random oracles there is a provable separation between time-space trade-offs for reversible and irreversible computation (Frank and Ammer, 2017).

Entropy flow and production The entropy production of a process is however much excess entropy the process produces relative to the change in the entropy of the system. This can be characterized in terms of the entropy flow (EF), or heat released, and the change in the Shannon entropy ($\Delta\mathcal{S}$) of the system.

$$0 \leq EP = EF + \Delta\mathcal{S} \tag{5.2}$$

In theory, this makes it possible to offset increases in entropy by absorbing heat from the environment or vice versa. However, for our purposes, such savings require complex interactions between a device and its observer that complicate our model. Thus we will use Definition 5.4 to explicitly define the thermodynamic costs of our operations without diving into the details of the underlying thermodynamics. It is worth noting that this means our constructions only represent upper bounds on thermodynamic costs, as more fine-grained analysis and complex protocols may recover some (or all) of the costs we outline in this definition.

Distribution sampling A standard way to define a distribution sampler (or random number generator) is as a probabilistic algorithm which has access to fair coin flips and output samples from distribution $\mathcal{D}$ when executed. It is well-known that

the randomized algorithm's randomness can be moved "up-front" as input to a deterministic algorithm. Therefore we define:

Definition 5.3. Let $\mathcal{D}$ be a probability distribution over finite domain D. We say a function $g : \{0,1\}^n \to D$ is a *$\mathcal{D}$-generator* if $g(x) \sim \mathcal{D}$ when x is chosen uniformly at random.

Note that for all distributions other than uniform, a $\mathcal{D}$-generator cannot be directly computed in a reversible manner since that would require g to be one-to-one. We can get around this restriction by instead reversibly computing a function $h(x) = (x, g(x))$, which is injective since it preserves the input x.

In theory one could — given a machine $\mathcal{M}$ that computes a $\mathcal{D}$-generator function h — run $\mathcal{M}$ and pre-compute the value of h on all inputs in $\{0,1\}^n$. Doing so would allow the construction of an energy and time efficient device that samples from $\mathcal{D}$ via a lookup table, albeit physically of exponential (in n) size. This is similar to how all boolean functions have an exponential sized constant depth circuit. As such we will only be concerned with the task of designing devices that are constructed from a description of $\mathcal{M}$ in time that is polynomial in both n and the size of $\mathcal{M}$.

Upper bounding energy costs To upper bound the energy used by a computation, we need to formalize the energy costs of certain actions involving a device and an operator. We assume that the energy costs of protocols we consider in this chapter can be broken down as follows:

Definition 5.4. Let $\mathcal{M}$ be some computational device and $\mathcal{G}_\mathcal{M}$ be its configuration graph. An operator interacts with the device via some protocol $\mathcal{P}$ described as a sequence of control flow statements and the following actions on $\mathcal{M}$:

1. Set_Energy($\Delta G : \Sigma \to \mathbb{R} \cup \{\infty\}$): change the energy of the states in $\mathcal{G}_\mathcal{M}$ to match this function.

2. Measure($f : \Sigma \to R$): the operator records the value of f applied to the state of $\mathcal{M}$.

3. Wait(t): the operator waits at least t time.

As justified below, we ascribe the following energy costs to each of these operations when $\mathcal{M}$ is in a distribution over states $\mathcal{D}$ and currently has the energy function ΔG:

1. $C(\text{Set\_Energy}(\Delta G' : \Sigma \to \mathbb{R})) = \sum_{x \in \Sigma} \Pr_{x' \sim \mathcal{D}}[x = x'](\Delta G'(x) - \Delta G(x))$

2. $C(\text{Measure}(f : \Sigma \to R)) = \mathcal{S}(f(\mathcal{D}))$

3. $C(\text{Wait}(t)) = 0$

The energy cost of executing a protocol is a sum of the energy costs of each of these steps.

Now we provide some thoughts on these three costs.

First $C(\text{Set\_Energy}(\cdot))$: Intuitively when changing the free energy of the states of a system we need to exert work (or energy) equal to the change in free energy of its state. This means that there is no inherent cost to changing the free energy of states that are not occupied (i.e. $\Pr_{x' \sim \mathcal{D}}[x = x'] = 0$). This cost is unavoidable due to conservation of energy and can be exactly achieved by performing the transformation quasi-statically (Esposito and Van den Broeck, 2011).

$C(\text{Measure}(\cdot))$: Measuring and recording the value of a random variable X with $\mathcal{S}$ entropy induces a Landauer energy cost of $\mathcal{S}$ when that variable is subsequently deleted. While writing the value of the random variable on an empty tape has no inherent cost (and in fact does not change the entropy of the universe, as the random variable is currently a deterministic function of the state of the device), over time the recorded value becomes decoupled from the state of $\mathcal{M}$ and after the two variables are perfectly uncorrelated, the entropy of the universe must have increased by $\mathcal{S}$.

Fine-tuning a device so that the energetic cost of erasure is exactly the Landauer cost is computationally intensive. If this is the case, then our energy upper bound for recording should be the number of recorded bits instead. This is the bound we use in our results. While we define a very general measurement operator, in practice we will restrict our constructions to only use measurements that project onto a subset of the bits. This ensures that nontrivial computation cannot hide in our measurements.

$C(\text{Wait}(t))$: While there is existing literature on the thermodynamic costs of timekeeping (Pearson et al., 2021), the energy cost of this operation can be amortized over an arbitrary number of concurrent tasks all completed by a single observer who only has to pay this cost once. Thus we will assume the existence of a global clock and say that our protocols do not need to pay for the energy costs of its timekeeping.[1] In theory the control flow that governs the behavior of the observer in a protocol could induce non-trivial energy costs;[2] however, we will only be considering very simple protocols where these energy costs are asymptotically dominated by those mentioned above.

Avoiding reset costs Often computational protocols defined in a model like this would require a special reset operation to restore the device into a specific configuration before it can be reused. This operation reduces the entropy of the system and thus requires expending work (or energy). In fact if a device performs a computation that takes T steps and all of its states have equal values of ΔG then when that device reaches its equilibrium state, $\Omega(\log T)$ work must be spent to reset it (Norton, 2013). We will be interested in designing computational devices and protocols that perform computational tasks where their energy costs scale only linearly with the input size n and independently of the time complexity T. In doing so, we will need to sidestep the

[1] In addition to amortization, our protocols do not need particularly accurate clocks. In fact, our protocols all work in the limit of each wait instruction taking infinite time. We only require clocks to upper bound the time until our systems are sufficiently well mixed.

[2] In fact in this model the observer could just perform our desired computation without interacting with the device at all!

resetting cost described in (Norton, 2013). We do this by defining our machines in a way where they can perform the intended protocols from any initial distribution over states, making it unnecessary to reset the machine before performing the next round of computation. There will be a cost writing down any output produced in the case of a sampler, and a cost for writing both an input and output in the function computation case, but these will not require any additional 'resetting' due to how our protocols are defined. We will consider three specific types of computational protocols, two of which are samplers, namely Las Vegas sampling and Monte Carlo sampling, and one being function computation. The choice of names Las Vegas and Monte Carlo for these samplers comes from the use of the terms in the context of randomized algorithms. A *Las Vegas* algorithm always produces the correct output with a small resource cost in expectation. Meanwhile a *Monte Carlo* algorithm has a fixed resource cost, but in exchange has a small failure probability.

With these definitions, it is possible to describe general strategies that convert a machine $\mathcal{M}$ into one that can perform sampling or function computation where the energy costs are independent of the time complexity of $\mathcal{M}$. We can describe this conversion for an arbitrary machine $\mathcal{M}$ in terms of its configuration graph $\mathcal{G}_{\mathcal{M}}$, giving a complete description of our construction in the language of Turing machines.

Turing machines While our constructions can be applied more generally, we will work through the details of how a Turing machine $\mathcal{M}$ can be adapted to give our devices for sampling and function computation.

Definition 5.5. An m tape *Turing machine* $\mathcal{M}$ is a tuple (S, Σ, f) where S is a set of configurations and Σ is a set of symbols for the tape. f is a set of rewrite rules of the form $(s, \sigma) \to (s', \sigma')$ or $s \to (s', d)$, where $s, s' \in S$, $\sigma, \sigma' \in \Sigma^m$ and $d \in \{L, \emptyset, R\}^m$.

In each timestep, the machine reads the values on the tape at each of its heads and its configuration and then applies some rewrite rule in f that matches this configuration. Rewrite rules of the form $(s, \sigma) \to (s', \sigma')$ change the configuration and

the values at the current locations on the tapes. Rules of the form $s \to (s', d)$ change the configuration and cause the i'th tape head to move one space to the left when $d_i = L$, one space to the right when $d_i = R$, or stay in the same place when $d_i = \emptyset$.

Readers familiar with Turing machines might be used to rewrite rules that map a head configuration and a tape symbol to a new head configuration, tape symbol, and a direction to move the head. The above definition of rewrite rules is functionally equivalent to this; however, by separating rules that write to the tape from those that move the head, it becomes easy to check that a Turing machine is logically reversible by inspecting its rewrite rules (Bennett, 1973). Our results work equally well for the more standard definition of a Turing machine, although defining the exact modifications to the transition rules requires a bit more care.

Our starting machine We will make a couple of simplifying assumptions about the machine $\mathcal{M}$ that will make the proofs easier to follow; however, they are not strictly necessary for our argument to work.

1. $\mathcal{M}$ has a read-only input tape and a write-only output tape and starts with the input (or input seed) $b \in \{0,1\}^n$ written on its input tape with blank characters on either end of the input.

2. $\mathcal{M}$ is a reversible Turing machine. This can be achieved by applying Bennett's reversible pebble game construction (Bennett, 1989) to $\mathcal{M}$.

3. $\mathcal{M}$ starts in a configuration A with the head at the far-right of the input tape (looking at the first blank). The output tape starts empty.

4. $\mathcal{M}$ produces the output in time at most T, which is known to this construction. We can get the same results for a slightly different construction if T is not known

ahead of time, but instead we have the guarantee that all paths in $\mathcal{M}$ have the same length.[3]

5.3 Las Vegas sampler

We first give our formal definition of what it means for a machine to be a Las Vegas sampler. Intuitively, we can generate the exact probability distribution at the cost of having to reject some samples as invalid.

Definition 5.6. Let $\mathcal{M}$ be a machine augmented with a free energy function ΔG where each configuration $A \in \Sigma$ can be represented as a tuple of registers (W, M, O) where W is the internal state of the machine, M are measurable metadata bits, and O is a potential output. We say that $\mathcal{M}$ is a *Las Vegas sampler* for a distribution $\mathcal{D}$ if the outputs of the following protocol starting from any initial configuration are independent samples from $\mathcal{D}$ for some $\tau > 0$:

$\mathcal{P}_{LV}$:
    $v :=$ `Measure`$((W, M, O) \to M)$
    `while`$(v = 0)$:
        `Wait`(τ)
        $v :=$ `Measure`$((W, M, O) \to M)$
    `repeat` :
        `Wait`(τ)
        `if`(`Measure`$((W, M, O) \to M) = -v)$:
            `output` (`Measure`$((W, M, O) \to O)$)
            $v := -v$

The steps before the repeat statement in the above protocol are used for initialization and only need to be completed once. Then the device can be infinitely reused to generate new samples.

[3]Moreover, a slight modification of the construction carries through even if we merely have the guarantee that $\mathcal{M}$ halts on all inputs (i.e. if we don't know T), however in that case we loose the probability bounds in Theorem 5.9. See Section 5.6 for further discussion.

For our construction, the metadata register in the above protocol are used to guarantee the independence of the samples. In particular the metadata register takes on values in $\{-1, 0, 1\}$ where transitions between values -1 and 1 require going through states with the value 0 that cause all information about the previous sample to be forgotten. After obtaining a sample where the metadata had the value -1 the next time we can obtain a sample where the metadata has the value 1 we know that the value is independent of the previous choice. For this sampler to be useful, it is important that only a few measurements are necessary to produce each new sample.

Definition 5.7. A Las Vegas sampler is (δ, T)-*efficient* if, for any distribution over states at the beginning of the "repeat" statement, the following "if" statement in the protocol is always satisfied with probability at least δ assuming that $\tau \geq T$.

If we can show that a Las-Vegas sampler is (δ, τ)-efficient then the probability that the metadata register needs to be read c times before producing a new output is at most δ^c. This gives us a strong tail bound on the number of trials before each successful sample.

5.3.1 Construction

In this section we describe how to modify a Turing machine computing a $\mathcal{D}$-generator into an efficient Las Vegas sampler for $\mathcal{D}$. Importantly, while we specify how to modify a Turing machine, our modifications could be applied to almost any reasonable computational device. Our construction is divided into three parts: the randomizer, the computation, and the output holding regions; below we describe at a high level how these parts are designed for the construction of Las Vegas samplers.

As a starting point we take a reversible Turing machine $\mathcal{M}$ computing a $\mathcal{D}$-generator using inputs over $\{0, 1\}^n$. The requirement that the Turing machine be reversible is not restrictive as any Turing machine computing this function can be converted to a reversible one with only a small asymptotic overhead in time and space

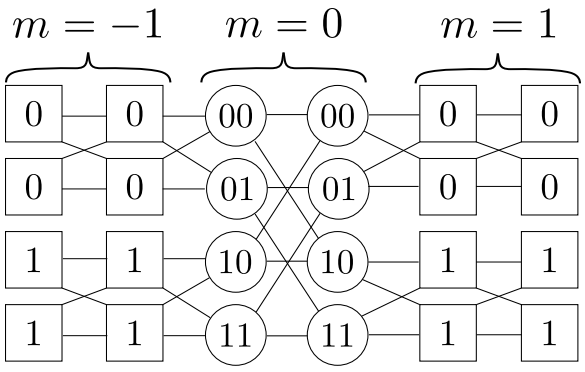

Figure 5.1: A simple example of our Las Vegas construction, $\mathcal{M}^*$, applied to a Turing machine $\mathcal{M}$ with no transition rules and a single bit of input (i.e. $n = 1$), that computes the identity function. Circles: randomizer with one random bit ($n = 1$, for input), one ancillary bit and with metadata $m = 0$. Squares: output holding regions labeled with their output bit with metadata $m = -1$ or $m = 1$. Note that for simplicity multiple distinct states of the device representing the same high level state of the machine have been condensed into a single node.

complexity (Bennett, 1989). Since the Turing machine is reversible, its configuration graph can be described as a collection of 2^n chains. We assume that $\mathcal{M}$ runs in at most T steps, by which we mean that any chain in $\mathcal{M}$'s configuration graph is of length at most T.

Our modified machine $\mathcal{M}^*$ will have a configuration graph that is layered in the sense that the nodes are partitioned such that nodes in layer L_i only have edges to layers L_{i-1} and L_{i+1}. A simple example of the configuration graph for our construction acting on the trivial machine $\mathcal{M}$ with only a single bit of input and no machine transitions can be seen in Figure 5.1, and the configuration graph for our full construction is illustrated schematically in Figure 5.2. Below we describe how our Las Vegas sampler $\mathcal{M}^*$ is constructed, assuming we start with a reversible Turing machine $\mathcal{M}$.

Adding the output holding region In order to have a constant probability of producing a new output after measurement, we force the computation's graph to have a constant fraction of its configurations containing the output. We achieve this by

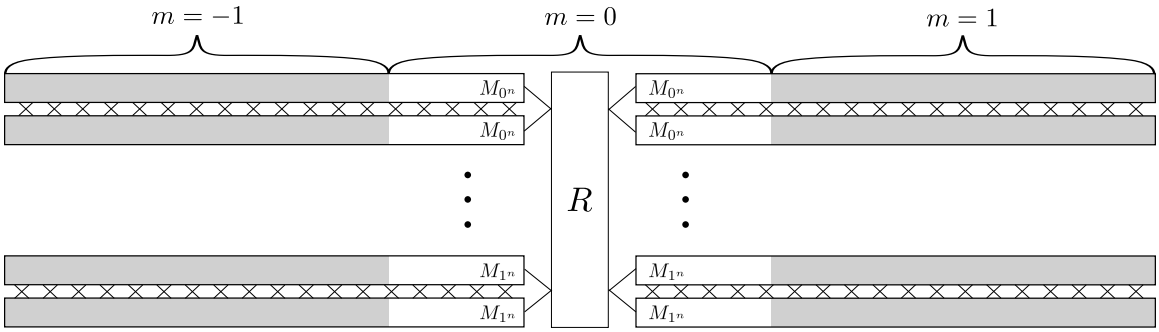

Figure 5.2: Our general Las Vegas construction $\mathcal{M}^*$ from a reversible Turing machine $\mathcal{M}$. R is the randomizer (see Figure 5.1) which generates random binary input words b, and $\mathcal{M}_b$ represents the chain of states for $\mathcal{M}$ on input b. The gray area represents the output holding states.

artificially boosting the length of $\mathcal{M}$'s computation by adding a tail of redundant configurations containing the output. Specifically, we elongate 'short' computations: we modify $\mathcal{M}$ by elongating paths so that all the chains in its configuration graph have exactly the same length. We then further extend the length of each chain C with redundant "output holding" nodes that all contain the value in the output register from the terminal configurations of C. We can implement these changes in a Turing machine as follows:

Let T_{comp} be the smallest power of two such that $T \leq T_{\text{comp}}$. Let T_{out} be the smallest power of two that is at least as large as $2T_{\text{comp}} + n + 1$. Note that T_{out} and T_{comp} are both $O(T + n)$. We will start by extending $\mathcal{M}$ to machine $\mathcal{M}_1$ by adding the output holding region. In other words, we will artificially manipulate the number of steps so that our machine takes the same number of steps for all input bitstrings and then further extend the length of the computation so that the machine is likely to observe the output when sampled.

We will do this by augmenting the machine with two new tapes designed to contain logically reversible counters. We will refer to these tapes as the computation-counting and output-counting tapes. The computation-counting tape starts with the value 0 expressed as a $\log_2 T_{\text{comp}}$ bit binary number (surrounded by _ symbols) with the least significant bit on the right and the head on this tape starts to the right of this number. The output-counting tape starts with the value 0 expressed as a $\log_2 T_{\text{out}}$ bit binary number with the head to the left of the number. For each rewrite rule in f that changes the state of $\mathcal{M}$ to some state

B, we change the target state of that rule to a new state α_B. We additionally add new rewrite rules to f for each configuration that was terminal in $\mathcal{M}$ to map that state to a new state α_B. We add the following auxiliary states and rules that only act on the computation-counting tape:

$$\begin{aligned} \alpha_{c,B} &\to (\beta_{c,B}, L) & \gamma_{c,B} &\to (\xi_{c,B}, R) \\ (\beta_{c,B}, 1) &\to (\alpha_{c,B}, 0) & (\xi_{c,B}, 0) &\to (\gamma_{c,B}, 0) \\ (\beta_{c,B}, 0) &\to (\gamma_{c,B}, 1) & (\xi_{c,B}, \_) &\to (B, \_) \\ (\beta_{c,B}, \_) &\to (\omega_c, \_) \end{aligned}$$

Adding the above rules creates a (reversible) counter in the computation-counting tape that will increment after simulating each step of $\mathcal{M}$. When $\mathcal{M}$ would have halted, the additional rules cause the machine to instead increment the counter. These new rules give us a machine where a state is terminal if and only if the state is ω_c and the computation always runs for the same number of steps regardless of the input. Now we can add more rules that transition out of ω_c and use the output-counting tape to produce our output holding region. Specifically the new rules for this act only on the output-counting tape.

$$\begin{aligned} (\omega_c, \_) &\to (\gamma_o, \_) & \gamma_o &\to (\xi_o, R) \\ (\xi_o, 0) &\to (\gamma_o, 0) & (\xi_o, \_) &\to (\chi_o, \_) \\ (\chi_o, \_) &\to (\alpha_o, \_) & (\beta_o, 1) &\to (\alpha_o, 0) \\ \alpha_o &\to (\beta_o, L) & (\beta_o, 0) &\to (\gamma_o, 1) \\ (\beta_o, \_) &\to \bot \end{aligned}$$

Where $\bot$ is a new halting configuration for the machine. We add χ_o as an additional state this counter goes through so that the number of steps between each time this counter is incremented is the same as the number of steps for the computation-counting tape. If instead of knowing the value of T when designing this machine we have the guarantee that all chains in $\mathcal{M}$ have the same length, we can instead design the counters so that we first count up to T on an initially blank computation-counting tape as we perform our computation. Then we count up to $2T + n$ on an initially blank output-counting tape. This requires that all chains of $\mathcal{M}$ have the same length to sample from the desired distribution, but removes the requirement of knowing T ahead of time.

While incrementing and decrementing length $O(\log T)$ counters on a Turing machine can take $\Theta(\log T)$ steps for some configurations, the amortized cost of these operations is only $O(1)$ each, so adding these counters does not change the asymptotic

complexity of the computation. Together, these additional states and modified transitions give us a reversible machine $\mathcal{M}_1$ where all computation paths take the same number of steps. The state-space of the machine contains sufficient output-holding states for our general Las Vegas construction so that the machine will have a constant fraction of the states containing a sample from the distribution.

The randomizer Let n be the number of bits that machine $\mathcal{M}$ starts with on its random bitstring input tape. We next design a *randomizer* that allows $\mathcal{M}_1$ to change its input in order to sample from the correct distribution. Specifically the randomizer modifies the configuration graph by adding $n+1$ layers of states each containing one node corresponding to each length $n+1$ bitstring. Edges connect these nodes such that the node at layer i representing input x is connected to the two nodes at layer $i+1$ representing inputs x and $x^{\oplus i}$.[4] This is known as a butterfly graph and has the property that any random walk over its state space that starts on the first layer and ends on the final layer will have a uniformly random assignment to the first n bits. The extraneous final bit in the bitstrings associated with the states of this graph will be used later to ensure that our configuration graph has the invariant where every node has exactly 0 or 2 neighbors to its left and to its right. Importantly, this final bit is stored in the configuration of the head of the Turing machine rather than an unexpected bit on the input tape.

Lemma 5.8. *An unbiased random walk that starts on the leftmost layer of this graph and ends on the rightmost layer will end up in a node whose work register contains $n+1$ bits of which the first n are uniformly random.*

Proof. Let p be any path through this graph that starts on the left and ends on the right. Starting from column 0, let $t_1, \ldots t_n$ be the last time-instant where the random walk enters each column $1, \ldots, n$. We note that the choice of edge taken at time

[4] If $x = x_1, \ldots x_n$, we use $x^{\oplus i}$ to denote $x^{\oplus i} = x_1, \ldots, x_{i-1}, 1 - x_i, x_{i+1}, \ldots, x_n$.

t_i determines the value of bit $i-1$ assigned to the node reached at the end of the walk. Since this is an unbiased random walk, the choice of which edge is taken during each of these time-steps is uniformly random. Thus the random walk will end with a uniformly random bitstring of length n, plus one ancillary bit. □

We can describe such a randomizer in a Turing machine as follows:

Here we will describe a set of additional configurations and transition rules that create the butterfly graph described above. These additions will actually be added to our machine in the next step — here we are merely defining the configurations and rules that implement a randomizer. We add transition rules that can be used to randomize the input before leading to the starting state of the machine (lets consider this state in $\mathcal{M}_1$ to be A). More specifically we add the following new states and transition rules that act only on the input tape:

$$
\begin{aligned}
(\alpha_r, 0) &\to (\beta_r, 0) & \qquad (\alpha_r, 0) &\to (\beta_r, 1)\\
(\alpha_r, 1) &\to (\beta_r, 0) & \qquad (\alpha_r, 1) &\to (\beta_r, 1)\\
(\alpha_r, \_) &\to (A^0, \_) & \qquad (\alpha_r, \_) &\to (A^1, \_)\\
(\overline{A^0}, \_) &\to (\beta_r, \_) & \qquad (\overline{A^1}, \_) &\to (\beta_r, \_)\\
\beta_r &\to (\alpha_r, R)
\end{aligned}
$$

Instead of starting in the configuration A with the head to the right of the input tape, these rules require a machine that starts in the configuration α_r with the head to the left of the input tape. These changes lead to a machine that is neither logically reversible nor deterministic. However, each non-deterministic step of the machine overwrites one (uniform) random bit with another, so the computation is not biased in either direction. Note that instead of transitioning to the configuration A, these rules describe 4 replacement configurations $A^0, A^1, \overline{A^0}, \overline{A^1}$. These new configurations will replace the original configuration A when we connect the randomizer to the computation in the next step. Together, these rules enable the machine to select a uniformly random input $b \in \{0,1\}^n$ on the input tape and end in one of two possible configurations representing the same input.

These states and transition rules will be added later to complete the construction.

Linking the randomizer and the computation Observe that there is one node on either side of the randomizer corresponding to each bit-string of length $n+1$, which gives us two nodes corresponding to each bit-string b of length n by ignoring the last bit. As shown in Figure 5.2, on each side of the randomizer, for each bitstring b of length n, we add two copies of the chain in $\mathcal{M}'$ corresponding to input b and connect their starting configurations to both of the nodes in the randomizer corresponding to input b. We finally add edges that link the node for step t in each of these chains to the node for step $t+1$ in the other chain. Doing this gives us a layered graph where each node has degree zero or two in each direction, so a random walk on this graph behaves like a one-dimensional random walk on the layers. We assign metadata value $M = -1$ to all the nodes we added in this process to the left of the randomizer that appear after chain link T (where we know the nodes contain samples from $\mathcal{D}$ in their output values) and metadata value $M = 1$ to the same nodes for the chains on the right of the randomizer. All other nodes have metadata value $M = 0$.

We can implement these changes in the Turing machine as follows:

Let $\overline{\mathcal{M}_1}$ be the *chiral inversion* of $\mathcal{M}_1$. Specifically for each transition rule of $\mathcal{M}_1$ that moves the heads of the Turing machine $\overline{\mathcal{M}_1}$ moves the heads of all the tapes except for the computation-counting, output-counting, and output tapes in the opposite direction. We note that $\mathcal{M}_1$ behaves the same as $\overline{\mathcal{M}_1}$ if the starting configurations of all the other tapes is inverted around the heads. We will have two copies each of $\mathcal{M}_1$ and $\overline{\mathcal{M}_1}$ denoted as $\mathcal{M}_1^0, \mathcal{M}_1^1, \overline{\mathcal{M}_1^0}$, and $\overline{\mathcal{M}_1^1}$. Each configuration B will be denoted $B^0, B^1, \overline{B^0}$ and $\overline{B^1}$ for these respective machines.

We will start constructing $\mathcal{M}^*$ by combining the states and transition rules of $\mathcal{M}_1^0, \mathcal{M}_1^1, \overline{\mathcal{M}_1^0}$ and $\overline{\mathcal{M}_1^1}$. Now to connect the states of the four machines, we will add the rules we defined above for the randomizer and change the initial configuration to start in the state α_r with the head to the left of the input tape.

$\mathcal{M}_1^0$ and $\mathcal{M}_1^1$ acting on the (forward) input and serve as the chains on the right in the configuration graph. The states of $\overline{\mathcal{M}_1^0}$ and $\overline{\mathcal{M}_1^1}$ act on the (backwards) input like the chains to the left of the randomizer in the configuration graph. All that remains is to allow the states in $\mathcal{M}_1^0$ / $\mathcal{M}_1^1$ and $\overline{\mathcal{M}_1^0}$ / $\overline{\mathcal{M}_1^1}$ to transition to one another so that all nodes in the configuration graph have 2 left and 2 right neighbors. We can achieve this by taking each transition rule from each of these

machines and making a copy of it that maps to the equivalent state in the other machine. For example if $\mathcal{M}_1$ had the transition rule $B \to (C, R)$ then we would add the four rules:

$$B^0 \to (C^1, R) \qquad B^1 \to (C^0, R)$$
$$\overline{B^0} \to (\overline{C^1}, L) \qquad \overline{B^1} \to (\overline{C^1}, L)$$

This completes our Turing machine construction of $\mathcal{M}^*$.

Specifically we can view $\mathcal{M}^*$ as a Las Vegas sampler where the tapes, head locations, and state are in the work registers and the output register is the output tape. The metadata is set to -1 (or 1) when the machine is in states corresponding to incrementing the output-counting tape in the chirally inverted (or non-inverted) machines and is otherwise set to 0. By Theorem 5.9, this makes $\mathcal{M}^*$ a 3/4-efficient Las Vegas sampler.

The sampling procedure We let $\mathcal{M}^*$ be the machine constructed above. We will now give a general strategy for an observer to measure $\mathcal{M}^*$ to get independent samples from the distribution $\mathcal{D}$. The first sample is obtained by waiting time $\Theta(T^2 + n^2)$ between measurements of the metadata register, iterating until a non-zero metadata value is observed. The observer remembers the last observed metadata bit. The observer then waits time $\Theta(T^2+r^2)$ between measurements until the metadata register is -1 times the value it had during the last recorded sample, and then records the value of the output register.

The configuration graph of our construction has diameter $O(T + n)$. Thus waiting time $\Theta(T^2+n^2)$ between samples is sufficient for there to be a $\geq 1/4$ probability of observing the needed metadata on measurement.

Theorem 5.9. *Let $\mathcal{M}$ be a reversible Turing machine that computes a $\mathcal{D}$-sampler with n input bits that runs in time at most T on all inputs. Then applying the above construction and procedure to $\mathcal{M}$ yields a Brownian computer $\mathcal{M}^*$ that when augmented with the constant energy function yields a $(1/4, \Theta(T^2 + n^2))$-efficient Las*

Vegas sampler for $\mathcal{D}$. Moreover, the energetic cost of each sample generated from $\mathcal{M}^*$ when the support of $\mathcal{D}$ is $\{0,1\}^m$ is $O(m)$ in expectation.

Proof. We note that the machine from the above construction yields a configuration graph where the nodes can be partitioned into layers such that the nodes in layer i are only neighbors of the nodes in layers $i-1$ and $i+1$. Additionally, each node in an internal layer has two edges to nodes in the layer before and after it. Thus, the machine will evolve according to an unbiased random walk over the layers of the graph, which is an unbiased walk on a line of length $O(T+n)$. Starting from any distribution over nodes on this line, a continuous time random walk will have probability density at least $1/4$ on the layers where nodes have the desired metadata value after time that is $\Theta(T^2+n^2)$. Thus, by waiting this amount of time between measurements, we get a $\geq 1/4$ probability that each measurement will give the desired metadata value. Each measurement before a sample is produced has a constant energetic costs and measuring the sample has an energetic cost of m. Thus, the overall energetic cost is $O(m)$ in expectation. □

Within the framework established in Definition 5.4, since a device needs to measure a sample from $\mathcal{D}$ to produce each output, this energy cost is asymptotically optimal. In fact, the only way for such a protocol to have an asymptotically lower energy cost is if after taking each measurement, the observer can absorb heat from the environment as the mutual information between the observed sample and the state of the machine drops. This heat could then be released later to erase recorded samples.

5.4 Monte Carlo sampler

In contrast to the Las Vegas sampler, a Monte Carlo sampler produces a new sample after each observation. However, this comes with an unavoidable price that the samples are correlated with one another.

Definition 5.10. Let $\mathcal{M}$ be a Brownian computer with a free energy function ΔG where each configuration $A \in \Sigma$ has can be represented as a tuple of registers $(W, M, (O_{-1}, O_1))$ where w is the internal state of the machine, m are measurable metadata bits, and (O_{-1}, O_1) is a pair of potential outputs. We say that $\mathcal{M}$ is a *Monte Carlo sampler* for a distribution $\mathcal{D}$ if there is a $\tau > 0$ such that for any constant $c > 0$ the outputs of the following protocol, conditioned on any choice of previous outputs from this protocol and any initial state, come from a distribution $\mathcal{D}'$ such that the total variation distance between $\mathcal{D}$ and $\mathcal{D}'$ is at most ε^c:

$\mathcal{P}_{MC}$:
    `repeat:`
        `Wait`$(c\tau)$
        `if(Measure`$((W, M, (O_{-1}, O_1)) \to M) = -1)$:
            `output (Measure`$((W, M, (O_{-1}, O_1)) \to O_{-1})$`)`
        `else:`
            `output (Measure`$((W, M, (O_{-1}, O_1)) \to O_1)$`)`

While the Monte Carlo sampler cannot guarantee the independence of its samples, it instead promises a new sample that is close to independent after each observation. Our construction achieves this by physically coupling two independent devices $\mathcal{M}_{-1}, \mathcal{M}_1$ such that when one device is performing computation, the other stores a sample from the target distribution in its output register O_{-1} or O_1. By reading a metadata value $M \in \{-1, 1\}$ we know that $\mathcal{M}_m$ is the one currently containing output.

The accuracy of a Monte Carlo sampler can be quantified as follows:

Definition 5.11. A Monte Carlo sampler is (ε, T)-*accurate* for a distribution $\mathcal{D}$ if when $\tau \geq T$, for any distribution over previous outputs of the protocol, the output of the next sample generated comes from a distribution $\mathcal{D}'$ such that the total variation distance between $\mathcal{D}$ and $\mathcal{D}'$ is at most ε^c.

As the parameter c in the protocol increases, the samples of a Monte Carlo sampler become closer to the true distribution. If the observer allows the machine to

reach its ground state before making a sample, then that sample would come from the desired distribution.

5.4.1 Construction

Again, we start with a reversible Turing machine $\mathcal{M}$ computing a $\mathcal{D}$-generator using inputs over $\{0,1\}^n$. The configuration graph of this Turing machine can be described as a collection of 2^n chains. We assume that $\mathcal{M}$ runs in at most T steps on all inputs.

Our modified machine $\mathcal{M}^*$ will have a layered configuration graph where each layer describes the state of each submachine $\mathcal{M}_{-1}, \mathcal{M}_1$. Moving from layer L_i to L_{i+1} will advance the state of $\mathcal{M}_1$ and reverse the state of $\mathcal{M}_{-1}$. Figure 5.3 gives an example of the configuration graph for our construction acting on the trivial machine $\mathcal{M}$ with a single bit of input and no machine transitions. The configuration graph for our full construction is illustrated schematically in Figure 5.4. Below we describe how our Monte Carlo sampler $\mathcal{M}^*$ is constructed from $\mathcal{M}$.

We will start my constructing machine $\mathcal{M}_1$ and then discuss how it can be "linked" to a machine $\mathcal{M}_{-1}$.

Adding the output holding region For the Monte Carlo construction to work as intended, it is vital that $\mathcal{M}_1$ has at least as many states with a computed sample as without. That way when it becomes coupled to $\mathcal{M}_{-1}$, one of the two machines will always have an output. As was the case for the Las Vegas construction, we achieve this with the help of counters. One counter will be used to ensure that all inputs yield computational paths (chains) of the same length while another ensures that there are sufficient states containing outputs at the end of each chain.

We can implement this in a Turing machine as follows:

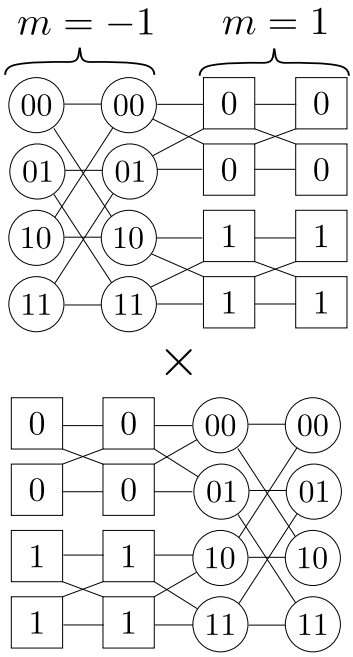

Figure 5.3: Our Monte Carlo construction applied to a general machine with no transition rules and a single bit of input. Each node in the graph is a pair of nodes in the same column of the sub-machines.

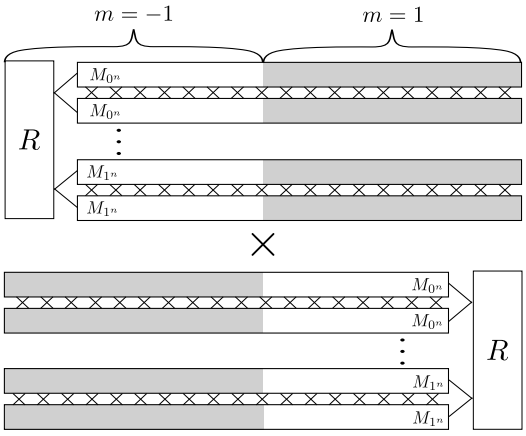

Figure 5.4: Monte Carlo: Each node in the graph is a pair of nodes from the sub-machines.

Let T_{comp} be the least power of two larger than T and T_{out} be the least power of two larger than $T_{\text{comp}} + r$. We will augment $\mathcal{M}$ with a computation counting tape of length $\log_2 T_{comp}$ and an output counting tape of length $\log_2 T_{\text{out}}$ each initialized to the all zeroes string. By introducing the same set of transitions described in the 'output holding region' part of Section 5.3, we can appropriately elongate the lengths of the chains in the general machine.

We call the machine with these additional tapes and rules $\mathcal{M}'_1$.

Adding the randomizer We will use the same randomizer construction as in Section 5.3 to create a butterfly graph over states where a random walk from the leftmost node to the rightmost node always yields a uniform distribution over the first n bits of the string. This gives the same set of rules as were described in the similar paragraph of Section 5.3, including the new initial starting configuration of the machine with tape head state α_r and the head of the machine to the left of the input tape.

Linking the randomizer and the computation Unlike in Section 5.3, we will not require additional states to the left of the randomizer. Instead, we simply need to connect the nodes at the right end of the randomizer to the chains representing the computation. Again, there is one node on the right of the randomizer corresponding to each bit-string of length $n + 1$, which gives us two nodes corresponding to each bit-string b of length n. As shown in Figure 5.4, on the right side of the randomizer, for each bitstring b of length n, we add two copies of the chain in $\mathcal{M}'_1$ corresponding to input b and connect their starting configurations to both of the nodes in the randomizer corresponding to input b. We finally add edges that link the node for step t in each of these chains to the node for step $t + 1$ in the other chain. Doing this gives us a layered graph where each node has degree zero or two in each direction, so a random walk on this graph behaves like a one-dimensional random walk on the layers.

We can implement these changes in the Turing machine as follows:

We will have two copies of $\mathcal{M}_1'$ denoted as $\mathcal{M}_1^0$, and $\mathcal{M}_1^1$. Each configuration B will be denoted B^0, and B^1 for these respective machines. We will start constructing $\mathcal{M}_1$ by combining the states and transition rules of $\mathcal{M}_1^0$, and $\mathcal{M}_1^1$. Now to connect the states of the two machines, we will add the rules we defined above for the randomizer and change the initial configuration to start in the state α_r with the head to the left of the input tape.

All that remains is to allow the states in $\mathcal{M}_1^0$ / $\mathcal{M}_1^1$ to transition to one another so that all nodes in the configuration graph have either 0 or 2 left and right neighbors. We can achieve this by taking each transition rule from each of these machines and making a copy of it that maps to the equivalent state in the other machine. For example if $\mathcal{M}_1'$ had the transition rule $B \to (C, R)$ then we would add the four rules:

$$B^0 \to (C^1, R) \qquad\qquad B^1 \to (C^0, R)$$

Unlike in the Las-Vegas construction, we will add no additional transitions interacting with head configurations $\overline{A^0}$ and $\overline{A^1}$. These states now prevent any computation from occurring to the left of the randomizer. This completes our Turing machine construction of $\mathcal{M}_1$.

Now we need to combine our submachines to get $\mathcal{M}^*$.

Combining the submachines Our final machine $\mathcal{M}^*$ will have the same number of layers as $\mathcal{M}_1$, but quadratically many states per layer. Incrementing layers corresponds to simulating forwards computation on $\mathcal{M}_1$ and backwards computation on $\mathcal{M}_{-1}$. Each layer of $\mathcal{M}^*$ corresponds to a tensor product of the configurations for submachines $\mathcal{M}_1, \mathcal{M}_{-1}$ at that layer. Since the output holding regions of submachines $\mathcal{M}_1, \mathcal{M}_{-1}$ each contain over half the layers in the overall graph, at any given layer, at least one of the two machines contains an output.

We can implement this machine as a Turing machine with the following modifications:

Let $\mathcal{M}_1 = (S_1, \Sigma, f_1)$ be the machine created above and $\mathcal{M}_{-1} = (S_{-1}, \Sigma, f_{-1})$ be a copy of that machine with a distinct set of configurations for the head.[a] We will build machine $\mathcal{M}^* = (S_1 \times S_{-1}, \Sigma, f^*)$ that combines these machines. This machine will have two copies of each tape in $\mathcal{M}_1$ that are used for running instructions from the different submachines. Additionally, the state of the head of $\mathcal{M}^*$ will now be a product of the states of heads for $\mathcal{M}_1, \mathcal{M}_{-1}$. For each pair of transition rules from f_1 and f_{-1}, f^* has a transition rule that implements the rule from f_1 forwards and the rule from f_{-1} backwards.[b] Then the initial state of $\mathcal{M}^*$ would correspond to some initial configuration of $\mathcal{M}_1$ and a terminal configuration of $\mathcal{M}_{-1}$. The generated machine M^* has a state space exactly like that of the general Monte Carlo sampler we described above.

[a] Note that this $\mathcal{M}_{-1}$ does not move in reverse, we will make it do so when adding these configurations and transitions to $\mathcal{M}^*$.

[b] For example if these were one tape Turing machines, f_1 contained transition $(A_1, \alpha) \to (B_1, \beta)$, and f_{-1} contained the transition $(C_{-1}, \gamma) \to (D_{-1}, \delta)$ then f^* would have the transition $((A_1, D_{-1}), (\alpha, \delta)) \to ((B_1, C_{-1}), (\beta, \gamma))$.

The metadata values of our construction can be defined such that when submachine $\mathcal{M}_1$ has a head configuration indicating that it is in the output holding region, the metadata value of $\mathcal{M}^*$ is 1. Otherwise, we know that submachine $\mathcal{M}_{-1}$ must be in an output holding region, and we set the metadata value to -1. Reading the metadata bit informs which submachine's output tape currently contains a sample from the desired distribution.

The sampling procedure We let $\mathcal{M}^*$ be the machine constructed above. We will now give a general strategy for an observer to measure $\mathcal{M}^*$ to get samples from the distribution $\mathcal{D}$. The observer waits $\Theta(T^2)$ then measures the metadata bit. Depending on the received value, they then measure and record the value stored in one of two output tapes.

The configuration graph of our construction has diameter $O(T + n)$. Thus waiting time $\Theta(T^2 + n^2)$ between samples is sufficient for there to be $1/2$ probability that the random walk has crossed the randomizers of both submachines.

Theorem 5.12. *Let $\mathcal{M}$ be a reversible Turing machine that computes a $\mathcal{D}$-sampler*

with n input bits that runs in time at most T on all inputs. Then applying the above construction and procedure to $\mathcal{M}$ yields a Brownian computer $\mathcal{M}^*$ that when augmented with the constant energy function yields a $(1/2, \Theta(T^2 + n^2))$-accurate Monte Carlo sampler for $\mathcal{D}$. Moreover, the energetic cost of each sample generated from $\mathcal{M}^*$ when the support of $\mathcal{D}$ is $\{0,1\}^m$ is always $O(m)$.

Proof. As with the Las Vegas construction, we note that the machine from the above construction yields a configuration graph where the nodes can be partitioned into layers such that the nodes in layer i are only neighbors of the nodes in layers $i-1$ and $i+1$. Additionally, each node in an internal layer has two edges to nodes in the layer before and after it. Thus, the machine will evolve according to an unbiased random walk over the layers of the graph, which is an unbiased walk on a line of length $O(T+n)$. Starting from any distribution over nodes on this line, a continuous time random walk will have probability at least $1/2$ of hitting both ends of the line after $O(T+n^2)$ time. Thus after c times as much time, the probability of hitting both ends of the line go up to $1 - 1/2^c$. By the construction of our randomizer, this event indicates that the next observed sample will come from distribution $\mathcal{D}$, and so the total variation distance between $\mathcal{D}$ and the true sampled distribution $\mathcal{D}'$ is at most $1/2^c$. Therefore, $\mathcal{M}^*$ is a $1/2$-accurate Las Vegas sampler for $\mathcal{D}$. The energetic costs of each sample come from measuring the metadata bits and then the output in the appropriate register, which is always $O(m)$. □

5.5 Function computation

In this section, we define Brownian computers that compute functions, a more standard form of computation than the sampling algorithms in previous sections. We describe how a Turing machine $\mathcal{M}$ (or any other reasonable computational device) computing a function $f : \{0,1\}^n \to \{0,1\}^m$ can be modified to compute f with an energy cost that only scales linearly in input length n and output length m.

Definition 5.13. Let $\mathcal{M}$ be a machine where each configuration $A \in \Sigma$ has can be represented as a tuple of registers (I, W, M, O) where I is the input on which $\mathcal{M}$ is performing computation, W is the internal state of the machine, M is a metadata register, and O is an output. We say that $\mathcal{M}$ is a *thermodynamic computer* for a function $f : \{0,1\}^n \to \{0,1\}^m$ if there is a $\tau > 0$ and a mapping from pairs $(x_1, x_2) \in \{0,1\}^{2n}$ to a free energy function $\Delta G_{x_1,x_2}$ such that, starting from any configuration, the following protocol outputs the pair $(x, f(x))$:

$\mathcal{P}_{TC}(x)$:
    `Wait`(τ)
    `while`(`Measure`$((I, W, M, O) \to M) = 0)$:
        `Wait`(τ)
    $(i, m) :=$ `Measure`$((I, W, M, O) \to (I, M))$
    `Set_Energy`$(\Delta G_{i,x})$
    `Wait`(τ)
    `while`(`Measure`$((I, W, M, O) \to M) \neq -m)$:
        `Wait`(τ)
    `output (Measure`$((I, W, M, O) \to (I, O))$`)`

Definition 5.14. A thermodynamic computer is (δ, T)-*efficient* if, for any distribution over states at the beginning of the protocol, the probability of leaving each while loop on every iteration when $\tau \geq T$ is at least δ. We say the computer is *efficient* if it is (δ, T)-efficient for some constant δ bounded away from zero.

When designing machines for sampling, we wanted our states to all have equal free energy. This was important so that — when driven by Brownian motion — the device would simulate an unbiased random walk over the state space. However for function computation, it is vital that we can change the free energies of states in a way to bias us towards the intended computational path. Ensuring that our hypothetical devices could be built, we will want to use free energy functions composed of a sum of 'local' terms. In particular, for our construction, we will only be using choices of ΔG that are a sum of terms for each bit of the state of the input tape. Additionally, we will require our device to end up in a distribution over states with at least as much

ΔG as the initial configuration. This way the total amount of free energy within the device does not decrease on each use. For our construction, we will allow ΔG to assign the value ∞ to states it cannot currently be in. However, we will also mention how it would be possible to modify the construction so that all states have finite energies in Section 5.6.

5.5.1 Construction

Our construction uses the exact same modifications to a reversible Turing machine $\mathcal{M}$ as in Section 5.3 to produce the machine $\mathcal{M}^*$ (notably the computation and output counters), except with the following change:

The randomizer now uses four symbols to denote the input to the tape: $0, 1, \hat{0}, \hat{1}$. The input tape starts with the value 0 and the randomizer is instead defined with the following set of rules to change the input on the tape:

$$
\begin{aligned}
(\alpha_r, 0) &\to (\beta_r, \hat{0}) & \qquad (\alpha_r, 0) &\to (\beta_r, \hat{1}) \\
(\alpha_r, 1) &\to (\beta_r, \hat{0}) & \qquad (\alpha_r, 1) &\to (\beta_r, \hat{1}) \\
(\alpha_r, \_) &\to (A^0, \_) & \qquad (\alpha_r, \_) &\to (A^1, \_) \\
(\overline{A^0}, \_) &\to (\beta_r, \_) & \qquad (\overline{A^1}, \_) &\to (\beta_r, \_) \\
\beta_r &\to (\alpha_r, R)
\end{aligned}
$$

These changed rules to describe the randomizer result in the chirally inverted machine (see 'linking the randomizer and the computation' in Section 5.3) operating on an input composed of $0, 1$ while the other machine operates on an input composed of $\hat{0}, \hat{1}$. This change will be useful when defining our free energy function. We then need to modify the rules of the Turing machine so that $\hat{0}, \hat{1}$ are treated as equivalent to $0, 1$ on the input tape. This can be done by modifying the rules of $\mathcal{M}_1^0$ / $\mathcal{M}_1^1$ so that they use $\hat{0}, \hat{1}$ instead of $0, 1$ for all of their rules.

The configuration graph of this machine is illustrated in Figure 5.5. While we assumed that $\mathcal{M}$ was a $\mathcal{D}$-sampler in Section 5.3, all the results would have applied equally well to a Turing machine computing some function f. For simplicity, we will assume that $\mathcal{M}$ has three tapes (i, w, o) for the input, work, and output respectively. While $\mathcal{M}^*$ has more than these three tapes, we will think of the additional tapes as

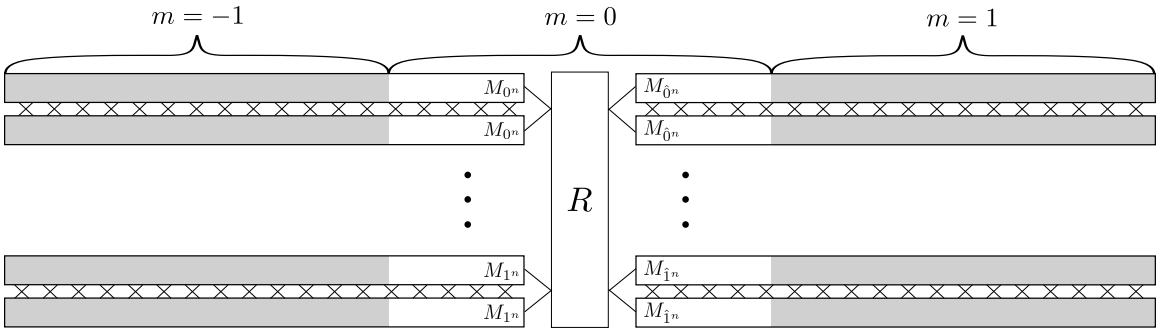

Figure 5.5: The configuration graph of our machine $\mathcal{M}^*$ for function computation.

being part of the unqueried W register, and thus we will only ever query the input, output tapes and the metadata register of $\mathcal{M}^*$.

The free energy function Now that we have specified our machine $\mathcal{M}^*$, all that remains is to pick a free energy function $\Delta G_{i,x}$. This free energy function should map the measured value of the input register $I = i_1, \ldots, i_n$ (read from left to right) and the target input $x = x_1, \ldots, x_n$ to output a free energy function ΔG such that when we later observe the machine we will have a $1/4$ chance of it being in a configuration with the desired output. Moreover, we need to pick such a ΔG such that the energy of the system does not decrease as the machine reaches equilibrium. Let $b : \{0, 1, \hat{0}, \hat{1}\} \to \{0, 1\}$ project an input tape element onto its logical value and $h : \{0, 1, \hat{0}, \hat{1}\} \to \{0, 1\}$ project an input tape element onto whether that element has a hat. By Definition 5.13, we know that i has the same value of $h(i_j)$ for all j. If $h(i_j) = 0$ then we will do this with the following free energy function, which assigns free energies that depend only on the value $v = v_1, \ldots, v_n$ stored in that states input tape (from left to right):

$$\Delta G^0_{i,x}(v) = \sum_{j \in [n]} \begin{cases} 0 & b(v_j) = b(i_j) \text{ and } h(v_j) = 0 \\ 0 & b(v_j) = b(x_j) \text{ and } h(v_j) = 1 \\ \infty & \text{otherwise} \end{cases} \tag{5.3}$$

When $h(i_j) = 1$ we instead use the following free energy function:

$$\Delta G^1_{i,x}(v) = \sum_{j \in [n]} \begin{cases} 0 & b(v_j) = b(i_j) \text{ and } h(v_j) = 1 \\ 0 & b(v_j) = b(x_{n-j+1}) \text{ and } h(v_j) = 0 \\ \infty & \text{otherwise} \end{cases} \tag{5.4}$$

When $h(i_j) = 1$ we need to invert x for it to be interpreted as the correct input. Our choice of ΔG assigns a free energy of zero to all states where the input register agrees with its current value on the same side of the randomizer and the desired input on the other side of the randomizer. Within the randomizer, the path between these inputs also has zero free energy. However, every other state in the graph has infinite energy. Thus, per Definition 5.2 our device will never take such a transition. Since there is exactly one state with zero energy per layer,[5] The system evolves according to a random walk on the layers. Per the same argument as in Theorem 5.9, after time $\Theta(T^2 + n^2)$, the device will be in a state with at least $1/4$ probability of containing the necessary metadata values to proceed through the while loops.

Theorem 5.15. *Given an arbitrary Turing machine $\mathcal{M}$ that computes a function $f : \{0,1\}^n \to \{0,1\}^m$ for some fixed n and halts in at most T time on all inputs, we can build a $(1/4, \Theta(T^2 + n^2))$-efficient thermodynamic computer that also computes f. Moreover, the energetic cost of each output generated from $\mathcal{M}^*$ is $O(n + m)$ in expectation.*

For this result, the expected energetic costs scale linearly in both n and m, since both the input and output registers need to be measured a constant number of times for this protocol. Since we leave the free energy of the machine's state the same when we change the energy landscape, this has no energetic cost.

If we prevent our protocol from making any assumptions on the initial distribution over states, measuring the full value of the input register is needed to define

[5] Technically there are two states per layer outside the randomizer due to $\mathcal{M}^0_1$ / $\mathcal{M}^1_1$ or $\overline{\mathcal{M}^0_1}$ / $\overline{\mathcal{M}^1_1}$, but we could either remove these states from the construction or (equivalently) always set the free energy of states corresponding to $\mathcal{M}^1_1$ / $\overline{\mathcal{M}^1_1}$ to infinity.

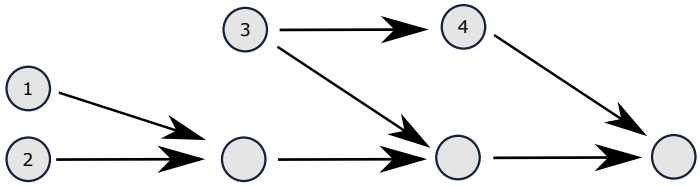

Figure 5.6: An irreversible computation. States 1, 2 are inputs while 3, 4 are false paths that do not come from an input.

the correct free energy function. However, if the free energy of states is not changed between uses, measuring the metadata bit is sufficient to learn the current value of the input register. Thus on repeated uses, it is no longer necessary to measure the full register.

5.6 Discussion

On the need for reversibility Our constructions start with the assumption that the machine $\mathcal{M}$ is a logically reversible Turing machine. Thanks to constructions like those of (Bennett, 1989), we can safely make this assumption with only small overheads to the time and space complexities of $\mathcal{M}$. It is natural to ask whether instead we can start with irreversible computation, and adjust ΔG values of states so that we obtain an unbiased random walk over the layers. Unfortunately, as Figure 5.6 shows, a random walk over the layers will hit obstacles called *false paths*: states that are not reachable from any valid input by applying the irreversible transition function f. As in general there can be exponentially many such false paths in the length of the computation, it could take an exponential time for the machine to return to the randomizer. This, in turn, could result in an exponential increase in the time the observer needs to wait before measuring the device.

Unknown bounds on running time Our Las Vegas, Monte Carlo, and thermodynamic computer constructions make the assumption that for the input machine $\mathcal{M}$, the quantity T representing the maximum number of steps for $\mathcal{M}$ to halt on any input

of length n is known to our construction. While we are able to prove upper bounds on the running time for specific algorithms, as pointed out in (Norton, 2013), this is in general impossible for an arbitrary Turing machine $\mathcal{M}$ due to the undecidability of the halting problem. There are ways to modify our constructions to work when T is unknown, but they are less than satisfactory.

If we know that $\mathcal{M}$ takes the same number of steps to run on all length n inputs, then we can modify $\mathcal{M}$ by adding a new tape that counts the number of steps during the computation. The length of the output holding region can then be programmatically determined by performing arithmetic on this tape and then having it count down to make the output-holding states. If $\mathcal{M}$ takes a different number of steps on different inputs then the probabilities of being in each output holding region may be different. This leads to a machine that gives samples from a different distribution from the one desired.

Without this assumption, it is still possible to define a sampler when different inputs require differing number of steps. We can do this my modifying $\mathcal{M}$ so that it evaluates on all inputs of length n before writing to the output tape. By doing this, we can artificially ensure that $\mathcal{M}$ takes the same number of steps on all inputs, and we can then use the above method to ensure the correct number of output-holding states. This both results in an exponential in n blowup of the time between samples and causes issues, as without a universal upper bound on T, the observer cannot know how long they need to wait before they will have a constant probability of obtaining a new sample.

Removing additional assumptions In Section 5.5 we relied on two physically unrealistic assumptions. First we assumed that we can set ΔG to ∞ for some states. In reality, it is sufficient to set the free energies of these states to a sufficiently large quantity that it is unlikely for the system to drift into such a state. In fact, if we are okay with only observing the intended output with a constant probability, we can

replace the ∞ with $O(\log n)$ in Equations (5.3) and (5.4). When doing so, the ground state of $\mathcal{M}^*$ will have constant probability mass on the intended output. However, it becomes less clear how long the observer would need to wait before each measurement.

Second we assumed that an observer can change the state of the system as an immediate response to a measurement. In reality, we should assume that some time δt passes between any actions of the observer. This could result in us increasing the ΔG of the machines state if it changes between measurement and setting the new energy function. If we set ΔG of some states to ∞ this would be a major problem, as then changing ΔG would require infinite work if the device drifts into such a state. However, if we make the energy values finite, we can see that the cost of changing ΔG must decrease as the length of the computation path T increases. This is because as the computation gets longer, it becomes less likely that in δt time the machine can change from one output holding region to a state corresponding to a different input tape value. Therefore, adding δt time between measurements will not cause the energetic costs of our device to exceed $O(n)$. Adding time between observer actions might also cause minor issues with Definitions 5.6 and 5.10 making back to back measurements of the state. These protocols can either be fixed by combining these measurements into a single operation or accepting a small rate of error that decreases with T.

Chapter 6: The computational complexity of optimizing the thermodynamics of Boolean circuits

6.1 Introduction

For a more detailed background on the general connections between thermodynamics and computation, see Section 5.1. Here, we discuss specific connections between thermodynamics and Boolean circuits. Prior works have investigated the entropy production of Boolean circuits. In (Wolpert and Kolchinsky, 2020) the authors gave the first stochastic thermodynamics model of Boolean circuits and examined how locality of gate operations induces unavoidable thermodynamic costs, even if the behavior of the overall circuit is logically reversible. This work and model is expanded on by (Yadav et al., 2024), where the authors consider more general *mismatch costs* caused by running Boolean circuits on suboptimal input distributions. These works are primarily concerned with characterizing the entropic costs of Boolean circuits with a fixed physical implementation.

We extend on the works of (Wolpert and Kolchinsky, 2020; Yadav et al., 2024) by considering what happens when the logical behavior of a Boolean circuit is fixed, but we are allowed to change details of its physical implementation. In particular, we extend their models of Boolean circuits by adding a programmable 'heat function' to each gate which characterizes thermodynamic details of that gates physical implementation. While our extensions of their model are obvious from a physics perspective, this framework naturally leads itself to many interesting computer science questions regarding the thermodynamics of Boolean circuits. In this dissertation we tackle one such question: what is the computational complexity of optimizing heat functions in a Boolean circuit? In this work we show that, in general, this task is intractable. Given a Boolean circuit as input, we show that it is PP-hard to determine the optimal heat function for gates when considering a uniform distribution of inputs

to the overall circuit. Moreover, unless $P = NP$, we show that even approximately optimizing these heat functions is impossible in polynomial time.

6.2 Preliminaries

We start with some basic facts and definitions.

Definition 6.1. Let $\mathcal{D}$ be a distribution over a set D and $x \in D$. Then we use $\Pr_{\mathcal{D}}[x]$ to denote the probability of sampling x from the distribution $\mathcal{D}$.

Definition 6.2. Let $\mathcal{D}$ be a distribution over a set D. Then $\text{supp}(\mathcal{D})$ is the set of all $x \in D$ such that $\Pr_{\mathcal{D}}[x] > 0$.

Definition 6.3. Let $\mathcal{D}$ be a distribution over a set D and $f : D \to R$ be a function. Then $f(\mathcal{D})$ is the unique distribution where for all $y \in R$:

$$\Pr_{f(\mathcal{D})}[y] = \sum_{x \in f^{-1}(y)} \Pr_{\mathcal{D}}[x].$$

If m is a mapping that takes each $x \in D$ to a distribution $\mathcal{D}_x$ over R then $m(\mathcal{D})$ is the unique distribution where for all $y \in R$:

$$\Pr_{m(\mathcal{D})}[y] = \sum_{x \in m^{-1}(y)} \Pr_{\mathcal{D}}[x] \Pr_{\mathcal{D}_x}[y]$$

Definition 6.4. Let $\mathcal{D}$ be a distribution over a set D. Then the *entropy* of $\mathcal{D}$ (denoted $\mathcal{S}(\mathcal{D})$) is given by:

$$\mathcal{S}(\mathcal{D}) = \sum_{x \in \text{supp}(\mathcal{D})} -\Pr_{\mathcal{D}}[x] \log_2(\Pr_{\mathcal{D}}[x])$$

Definition 6.5. Let $\mathcal{D}, \mathcal{D}'$ be two distributions over a set D. Then the Kullback-Leibler (KL) divergence between $\mathcal{D}$ and $\mathcal{D}'$ is given by:

$$D_{KL}(\mathcal{D}||\mathcal{D}') = \sum_{x \in \text{supp}(\mathcal{D})} \Pr_{\mathcal{D}}[x] \log \frac{\Pr_{\mathcal{D}}[x]}{\Pr_{\mathcal{D}'[x]}}$$

If there is an $x \in \text{supp}(\mathcal{D})$ such that $x \notin \text{supp}(\mathcal{D}')$ then $D_{KL}(\mathcal{D}||\mathcal{D}')$ is infinite.

Proposition 6.6 (Data processing inequality, (Cover, 2006))**.** *Let $f : D \to R$ be an arbitrary function and $\mathcal{D}, \mathcal{D}'$ be two distributions over D. If supp$(\mathcal{D}) \subseteq$ supp$(\mathcal{D}')$, then:*

$$D_{KL}(\mathcal{D}||\mathcal{D}') \geq D_{KL}(f(\mathcal{D})||f(\mathcal{D}')).$$

Definition 6.7. The *mutual information* between random variables X, Y is denoted:

$$\mathcal{I}(X;Y) = \mathcal{S}(X) + \mathcal{S}(Y) - \mathcal{S}(X,Y)$$

6.2.1 Thermodynamics of Boolean circuits

The entropy production of a computation depends both on the physical implementation of the computer and on the desired distribution over inputs. Therefore, it is natural to ask the following question: can we optimize a physical computational device to run as efficiently as possible on a target input distribution? This question falls under the study of optimal processes, or processes that achieve minimal dissipation subject to physical constraints (Hasegawa et al., 2010; Parrondo et al., 2015). A careful reader may notice that we specify minimizing the entropy flow (EF) rather than the entropy production (EP). As we saw earlier in Equation (5.2), if the logical mapping of a process is fixed, then the entropy production is always minimized by minimizing the entropy flow. We chose to express our results in this section in terms of entropy flow, as minimizing the flow of heat (EF) out of the system is ultimately how we minimize the entropy production.

Irreversibly driven computation In Section 5.2 we described how heat must be released to drive computation in the forward direction. In what follows, we will simplify our analysis by considering processes that move irreversibly forwards in time. This means that we are considering physical processes with infinite energy barriers and energy gaps between states which, as is common in the thermodynamics literature (i.e. (Wolpert and Kolchinsky, 2020)), means that the energetic costs we describe here should be viewed as lower bounds. Instead of assigning free energies to the

states of our system and having it evolve according to a random walk, we will require it to dissipate heat as it evolves in a way that is consistent with the second law of thermodynamics. The energetic costs of computation can be decomposed into two parts: (1) the energy required to counteract decreases in entropy (2) additional energy needed to drive the computation forwards. Of these costs, (1) depends on the logical and physical behavior of the computation and the distribution over inputs and will be the cost of interest in this dissertation. On the other hand, (2) represents costs that depend on the desired speed and forward drive of the computation, which can be made zero in principle via a quasi-static process or by running the computation with no forward bias (Wolpert, 2019). More formally to drive computation 'irreversibly forwards' we require that any process mapping each state $x \in D$ to a distribution over states $m(x)$ must have an associated heat function $Q : D \to (\mathbb{R} \cup \{\infty\})$ that, for any distribution $\mathcal{D}$ over D, satisfies:

$$\mathcal{S}(\mathcal{D}) - \mathcal{S}(m(\mathcal{D})) \leq \sum_{x \in \text{supp}(\mathcal{D})} \Pr_{\mathcal{D}}[x] Q(x) \tag{6.1}$$

The function $Q(x)$ represents how much heat the process must dissipate (in expectation) to perform its mapping when it receives the input x. Here we allow $Q(x)$ to be infinite only when the process can never actually be initialized to state x.

The value $\mathcal{S}(\mathcal{D}) - \mathcal{S}(m(\mathcal{D}))$ is also known as the Landauer cost (also denoted $\mathcal{L}_m(\mathcal{D})$) of computing m on input distribution $\mathcal{D}$. Again we will think of energy (heat) as being expressed in units such that Boltzmann's constant k times the fixed temperature of the system is one. Prior works have used a similar formulation of heat to express thermodynamic costs in Boolean circuits (Wolpert and Kolchinsky, 2020; Yadav et al., 2024). However, these works only require that Equation (6.1) holds for the true distribution over inputs $\mathcal{D}$. This is primarily because they are interested in quantifying thermodynamic costs for a given choice of Q rather than optimizing this over the set of functions that follow Equation (6.1). On the other hand, we are interested in how the underlying implementation of a fixed Boolean circuit can be optimized to minimize its energetic costs. For our purposes, it is important

that Equation (6.1) holds for all possible distributions over initial states; otherwise carefully initializing the distribution and running the process irreversibly forwards would decrease the entropy of the universe. When the inequality is tight this process is thermodynamically reversible and has zero entropy production for the distribution $\mathcal{D}$. This describes a setting where applying f and its inverse are equally likely, which satisfies the costs of type (1), and so any additional heat dissipation would drive the process forwards.

Boolean circuits Here we are going to discuss the computational complexity of defining optimal processes for the evaluation of Boolean circuits.

Definition 6.8. A Boolean circuit $\mathcal{C}$ on n inputs with T gates and S wires is described as an ordered set of T gates $g_t = (D_{g_t}, f^{g_t})$ for $t \in [1, T]$. Each gate specifies a set $D_{g_t} \subseteq [S]$ and a Boolean function $f^{g_t} : \{0,1\}^{D_{g_t}} \to \{0,1\}^{D_{g_t}}$. On input $x_1, \ldots x_n \in \{0,1\}^n$ the circuit starts in the state $\mathcal{C}_0 \in \{0,1\}^S$ where $\mathcal{C}_0[i] = x_i$ for $i \in [n]$ and $\mathcal{C}_0[i] = 0$ for $i \in [S] \setminus [n]$. The state of the circuit at time t is given by applying f^{g_t} to the state of the circuit at time $t-1$. More formally, if the gate operates on wires $D_{g_t} = d_1, \ldots, d_k$ then:

$$\mathcal{C}_t[i] = \begin{cases} f^{g_t}(\mathcal{C}_{t-1}[d_1], \ldots, \mathcal{C}_{t-1}[d_k]) & i \in D_{g_t} \\ \mathcal{C}_{t-1}[i] & \text{otherwise} \end{cases}$$

We say that wire $i \in [S]$ of $\mathcal{C}$ *evaluates to* $x \in \{0,1\}$ when $\mathcal{C}_T[i] = x$.

Our definition of a Boolean circuit is analogous to the standard model of quantum circuits. Gates perform local in place operations to update the values of their inputs, which are stored in wires. We assume that inputs start in the first n wires with the remaining wires initialized to 0. The final configuration of all wires is considered the output of the Boolean circuit. In (Wolpert and Kolchinsky, 2020; Yadav et al., 2024) the authors used a different model of Boolean circuits where information is stored inside gates instead of wires. In their model a circuit is represented with a

DAG where the nodes represent gates and edges show the flow of information. Once the parents of a gate have computed their output, the gate is then able to compute its associated function, storing the value in its own state. Then that gate must erase the states of its parent nodes to complete its computation. Our definition of Boolean circuits is more general than that of (Wolpert and Kolchinsky, 2020; Yadav et al., 2024). We can represent the action of a gate in their model with a gate that erases the values stored in the inputs and writes the output into an initially zeroed wire.

When each gate g_t in our circuit model performs its intended mapping on input $x \in \{0,1\}^{D_{g_t}}$, it releases $Q_{g_t}(x)$ heat into the environment. For such a circuit to be implementable, it is essential that the function Q associated with each gate obeys the second law of thermodynamics for every possible input to the gate. We allow $Q(x)$ to be infinite on some x when the gate never receives that input. This allows us to, for example, make a bit-erasure gate that dissipates not heat if it always receives the same input. When discussing the details of a single gate g_t, we will use D to refer to $\{0,1\}^{D_{g_t}}$, Q to refer to $Q_{g_t}(x)$, and f to refer to f^{g_t}.

Definition 6.9. Let $Q : D \to \mathbb{R}$ and $f : D \to D$. Then Q is a *valid heat function for computing* f if, for any distribution $\mathcal{D}$ over D, we have that:

$$\mathcal{L}_f(\mathcal{D}) \leq \sum_{x \in \text{supp}(\mathcal{D})} \Pr_{\mathcal{D}}[x] Q(x)$$

Proposition 6.10 (second law of thermodynamics)**.** *Let $f : D \to D$ denote the transformation performed by a gate in a circuit. If $Q : D \to (\mathbb{R} \cup \{\infty\})$ denotes the amount of heat released by that gate on input $x \in D$, then Q must be a valid heat function for computing f.*

Definition 6.11. The *entropy flow* (EF) of a gate that releases $Q(x)$ heat on each input $x \in D$ when the input distribution is $\mathcal{D}$ is given by:

$$EF(\mathcal{D}) = \sum_{x \in \text{supp}(\mathcal{D})} \Pr_{\mathcal{D}}[x] Q(x)$$

Following the convention of (Wolpert, 2019), we define the entropy flow as the total flow of entropy out of the system. This is in contrast to much of the stochastic thermodynamics literature, which defines it as the total flow of entropy into the system.

Definition 6.12. The *entropy production* (EP) of a gate computing a function $f : D \to D$ that releases $Q(x)$ heat on each input $x \in D$ when the input distribution is $\mathcal{D}$ is given by:

$$EP(\mathcal{D}) = \mathcal{S}(f(\mathcal{D})) - \mathcal{S}(\mathcal{D}) + \sum_{x \in \text{supp}(\mathcal{D})} \Pr_{\mathcal{D}}[x] Q(x) = EF(\mathcal{D}) - \mathcal{L}_f(\mathcal{D})$$

Corollary 6.13. *The entropy production of any valid heat function for a gate is non-negative on every input distribution.*

Mismatch costs Previous works on the thermodynamic costs of Boolean circuits have focused on *mismatch costs* as a source of entropy production. The mismatch cost of a gate g in a Boolean circuit can be defined as follows (Yadav et al., 2024):

Definition 6.14. Let g be a gate in a Boolean circuit computing a function f that releases $Q(x)$ heat on input $x \in D$. Let $\mathcal{D}$ be a distribution[1] over inputs to gate g that minimizes its entropy production $EP(\mathcal{D})$. Then the *mismatch cost* of running g on input distribution $\mathcal{D}'$ is given by:

$$MC(\mathcal{D}') = D_{KL}(\mathcal{D}' || \mathcal{D}) - D_{KL}(f(\mathcal{D}') || f(\mathcal{D}))$$

Mismatch costs were first identified in (Kolchinsky and Wolpert, 2017) as a source of entropy production caused by running a physical process on an unintended input distribution. It is possible to decompose entropy production as a sum of the mismatch costs and the entropy production on the best possible input distributions over states that lead to each output.

[1]Note that this distribution need not be unique.

Proposition 6.15 ((Kolchinsky and Wolpert, 2017; Wolpert and Kolchinsky, 2020)). *Let g be a gate in a Boolean circuit computing a function f. Let $\mathcal{D}_y$ be the distribution over inputs x to gate g where $f(x) = y$ that minimizes its entropy production $EP(\mathcal{D}_y)$. Then the entropy production of running g on input distribution $\mathcal{D}'$ is given by:*

$$EP(\mathcal{D}') = MC(\mathcal{D}') + \sum_y \Pr_{f(\mathcal{D}')}[y] EP(\mathcal{D}_y)$$

When the target distribution over inputs to a Boolean circuit is known, it is always possible to pick a heat function $Q : \{0,1\}^S \to (\mathbb{R} \cup \{\infty\})$ for each gate so that the mismatch cost is zero. But since we only allow the heat function of a gate to depend on its inputs, there can be pairs of gates g and input distributions $\mathcal{D}$ where any choice of heat function $Q : \{0,1\}^{D_{g_t}} \to (\mathbb{R} \cup \{\infty\})$ will result in a positive mismatch cost. Heat functions that are restricted to depend only on a gate's inputs must lead to optimal distributions when there is no correlation between a gate's inputs and the other wires. This leads to unavoidable mismatch costs (and in turn entropy production) for fixed circuits caused by local erasure of correlated information. We will show an example of this in more detail in Section 6.4. Such a restriction on the heat functions of our gates comes from the assumption that Boolean circuits are *modular*; the behavior of each gate can only depend on the information presented to it. In stochastic thermodynamics this is formalized with the notion of a *solitary process*, or a process acting only on a subsystem of a state while the remaining state is unchanged (Wolpert, 2019; Wolpert and Kolchinsky, 2020). By treating the gates in our circuit as solitary processes, we can see that minimizing the entropy flow of a fixed Boolean circuit depends only on minimizing the heat released by each gate on its marginal input distribution.

In (Wolpert and Kolchinsky, 2020; Yadav et al., 2024) the authors are primarily concerned with lower bounding the entropy production of Boolean circuits when the heat functions for the individual gates are not fine-tuned for the true distribution over inputs. These works are concerned with using mismatch costs to lower bound the

entropy production of circuits with suboptimal heat functions. We justify this setting by showing that it is computationally infeasible to pick optimal heat functions for gates in Boolean circuits.

6.2.2 Computational complexity

We aim to show that optimizing the thermodynamics of Boolean circuits is a *hard* problem, but what exactly do we mean by hard? Complexity theory can be used to sort problems into sets that characterize their difficulty. In this section, we will mostly be concerned with the complexity classes NP and PP, which stands for nondeterministic and probabilistic polynomial time respectively.

Definition 6.16. A nondeterministic Turing machine is a Turing machine $\mathcal{M}$ where multiple rules can be applicable from any configuration. When a configuration has multiple applicable rules, $\mathcal{M}$ applies each rule in a separate "computational path." During each time step, all computational paths advance according to the rules of the Turing machine.

Definition 6.17. NP is the set of languages $\mathcal{L}$ where there exists a polynomial time non-deterministic Turing machine with an accepting computational path when the input $x \in \mathcal{L}$.

Another way to view NP is the set of languages that can be efficiently verified. A *deterministic* Turing machine can efficiently verify that a non-deterministic machine $\mathcal{M}$ would accept an input x if it is told which path leads to $\mathcal{M}$ accepting x. Determining if every problem in NP can be efficiently solved by a randomized Turing machine is a major open problem in theoretical computer science (Cook, 1971).

Proposition 6.18 ((Cook, 1971))**.** *Let $\mathcal{L}_{SAT}$ be the set of Boolean circuits where the first wire evaluates to 1 on some input. Then deciding membership in $\mathcal{L}_{SAT}$ is a NP-complete problem.*

NP-completeness means that $\mathcal{L}_{SAT} \in NP$ and that an efficient algorithm for deciding membership in $\mathcal{L}_{SAT}$ could be used to efficiently solve any other problem in NP.

Definition 6.19 ((Gill, 1974))**.** PP is the set of languages $\mathcal{L}$ where there exists a polynomial time non-deterministic Turing machine that accepts on at least $1/2$ of all computational paths when the input $x \in \mathcal{L}$ and accepts on less than $1/2$ of all computational paths when the input $x \notin \mathcal{L}$.

The complexity class PP is very powerful. Importantly the gap between the number of accepting and rejecting paths can be arbitrarily small. In fact, PP is equivalent to another class $PostBQP$, which captures polynomial time quantum algorithms augmented with the ability to post-select on measurement outcomes (Aaronson, 2005). For any fixed k, there is a language in PP that cannot be decided by quantum circuits of size n^k even with quantum advice (Aaronson, 2006; Yirka, 2024). Another example of a problem in PP is determining how often a Boolean circuit produces a particular output on a random input.

Proposition 6.20 (implicit in (Valiant, 1979))**.** *Let $\mathcal{L}_{BC/2}$ be the set of Boolean circuits where the first wire evaluates to 1 on at least half of all inputs. Then deciding membership in $\mathcal{L}_{BC/2}$ is a PP-complete problem.*

We will prove the PP-hardness of optimizing Boolean circuits with a reduction from this problem.

6.3 Optimal heat functions for known input distributions

To prove our main results, we need a key lemma relating the valid heat function Q for a gate that minimizes its entropy flow for that gate's distribution over inputs.

Lemma 6.21. *Let $f : D \to D$ be an arbitrary Boolean function, $\mathcal{D}$ be the distribution over D where for each $x \in D$ let $p_x = \Pr_{\mathcal{D}}[x]$. This implies a distribution on the*

outputs $f(\mathcal{D})$. Let $p'_y = \Pr_{f(\mathcal{D})}[y]$. Then the unique valid heat function for a gate computing f with the least entropy flow on input distribution $\mathcal{D}$ is given by:

$$\forall x \in D, Q(x) = \begin{cases} \log_2(p'_{f(x)}/p_x) & p_x > 0 \\ \infty & p_x = 0 \end{cases}$$

Proof. For Q to be a valid heat function (Definition 6.9) for computing f we need that for all distributions $\mathcal{D}'$ where $\Pr_{\mathcal{D}'}[x] = q_x$ and $\Pr_{f(\mathcal{D}')}[y] = q'_y$:

$$\mathcal{L}_f(\mathcal{D}') = \sum_{x \in \text{supp}(\mathcal{D}')} q_x \log_2(1/q_x) - \sum_{y \in \text{supp}(f(\mathcal{D}'))} q'_y \log_2(1/q'_y) \leq \sum_{x \in \text{supp}(\mathcal{D}')} q_x Q(x) \quad (6.2)$$

The entropy flow is minimized when the above inequality is tight for the distribution $\mathcal{D}$, giving us the following additional constraint on the heat function:

$$\mathcal{L}_f(\mathcal{D}) = \sum_{x \in \text{supp}(\mathcal{D})} p_x \log_2(1/p_x) - \sum_{y \in \text{supp}(f(\mathcal{D}))} p'_y \log_2(1/p'_y) = \sum_{x \in \text{supp}(\mathcal{D})} p_x Q(x) \quad (6.3)$$

Picking $Q(x)$ as described in this lemma statement satisfies Equation (6.3). We now show that Equation (6.2) is also satisfied by this choice of Q. If $\text{supp}(\mathcal{D}') \supset \text{supp}(\mathcal{D})$ then there must be some $x \in \text{supp}(\mathcal{D}')$ such that $Q(x) = \infty$. Since $\mathcal{L}_f(\mathcal{D}')$ is always finite, such choices of $\mathcal{D}'$ trivially satisfy Equation (6.2). When $\text{supp}(\mathcal{D}') \subseteq \text{supp}(\mathcal{D})$:

$$\begin{aligned}
\mathcal{L}_f(\mathcal{D}') &= \sum_{x \in \text{supp}(\mathcal{D}')} q_x \log_2(1/q_x) - \sum_{y \in \text{supp}(f(\mathcal{D}'))} q'_y \log_2(1/q'_y) \\
&\leq D_{KL}(\mathcal{D}'||\mathcal{D}) - D_{KL}(f(\mathcal{D}')||f(\mathcal{D})) + \sum_{x \in \text{supp}(\mathcal{D}')} q_x \log_2(1/q_x) - \sum_{y \in \text{supp}(f(\mathcal{D}'))} q'_y \log_2(1/q'_y) \\
&= \sum_{x \in \text{supp}(\mathcal{D}')} q_x \log_2(1/p_x) - \sum_{y \in \text{supp}(f(\mathcal{D}'))} q'_y \log_2(1/p'_y) \\
&= \sum_{x \in \text{supp}(\mathcal{D}')} q_x \log_2(p'_{f(x)}/p_x) \\
&= \sum_{x \in \text{supp}(\mathcal{D}')} q_x Q(x)
\end{aligned}$$

where the inequality follows from Proposition 6.6.

Now that we have established that this Q works, we want to show that it is the unique choice satisfying Equations (6.2) and (6.3). Let $\hat{Q}$ be another heat function

that satisfies Equations (6.2) and (6.3). Then by Equation (6.3) we get that:

$$\sum_{x \in \text{supp}(\mathcal{D})} p_x(Q(x) - \hat{Q}(x)) = 0$$

Also when we apply Equation (6.2) with $\mathcal{D}'$ to be a Dirac Delta distribution (i.e. $q_x = 1$ and all other $q_{x'} = 0$) we get that:

$$\hat{Q}(x) \geq Q(x)$$

The only way to satisfy these constraints is when $\hat{Q} = Q$. □

Corollary 6.22. *Let $\mathcal{D}$ be the distribution over D where for each $x \in D$ let $p_x = \Pr_{\mathcal{D}}[x]$. Let $f : D \to D$ be a constant-valued function. Then the unique valid heat function for a gate computing f with the least entropy flow on input distribution $\mathcal{D}$ is given by:*

$$\forall x \in D, Q(x) = \begin{cases} \log_2(1/p_x) & p_x > 0 \\ \infty & p_x = 0 \end{cases}$$

Corollary 6.23. *Let $Q : D \to (\mathbb{R} \cup \infty)$ be the valid heat function for computing a constant-valued function $f : D \to D$ with the least entropy flow on some unknown input distribution $\mathcal{D}$. Then:*

$$\forall x \in D, \Pr_{\mathcal{D}}[x] = \begin{cases} 2^{-Q(x)} & Q(x) \in \mathbb{R} \\ 0 & Q(x) = \infty \end{cases}$$

6.4 Logically reversible operations may require entropy production

Here we present an example justifying that logically reversible computation may result in positive entropy production depending on the physical implementation. This is a well known result in the stochastic thermodynamics literature, and has already been extensively studied within the context of Boolean circuits (Wolpert and Kolchinsky, 2020). However, we believe this more formal example will be valuable to readers who are not intimately familiar with the field.

Lemma 6.24. *Let X, Y be a partition of $[S]$ and $\mathcal{C}_{t-1}[X], \mathcal{C}_{t-1}[Y] \sim \mathcal{D}_{t-1}$ be random variables for the state of wires X, Y after applying gate $t-1$ in circuit $\mathcal{C}$. Then if gate g_t computes the constant valued function $f^{g_t} : \{0,1\}^X \to \{0\}^X$, the entropy production of applying g_t as the next gate in $\mathcal{C}$ is at least $\mathcal{I}(\mathcal{C}_{t-1}[X]; \mathcal{C}_{t-1}[Y])$.*

Proof. Let $\mathcal{D}_X$ be the marginal distribution of random variable $\mathcal{C}_{t-1}[X]$ according to distribution $\mathcal{D}_{t-1}$. Then by Corollary 6.22, the optimal heat function for gate g_t is $Q(x) = \log_2(1/\Pr_{\mathcal{D}_X}[x])$. So, the entropy flow of implementing this gate is at least:

$$EF \geq \sum_{x \in \text{supp}(\mathcal{D}_X)} \Pr_{\mathcal{D}_X}[x] Q(x) = \mathcal{S}(\mathcal{C}_{t-1}[X])$$

Let $\mathcal{D}_t$ be the distribution over states of the circuit after applying g_t. Since $\mathcal{C}_{t-1}[Y]$ is unchanged by applying g_t, we know that according to Definition 6.12:

$$\begin{aligned} EP &= EF + \mathcal{S}(\mathcal{D}_t) - \mathcal{S}(\mathcal{D}_{t-1}) \\ &\geq \mathcal{S}(\mathcal{C}_{t-1}[X]) + \mathcal{S}(\mathcal{D}_t) - \mathcal{S}(\mathcal{D}_{t-1}) \\ &= \mathcal{S}(\mathcal{C}_{t-1}[X]) + \mathcal{S}(\mathcal{C}_{t-1}[Y]) - \mathcal{S}(\mathcal{C}_{t-1}[X], \mathcal{C}_{t-1}[Y]) \\ &= \mathcal{I}(\mathcal{C}_{t-1}[X]; \mathcal{C}_{t-1}[Y]) \end{aligned}$$

as desired. □

Intuitively, this bound comes from the fact that gate g_t cannot use a heat function whose values depend on non-input wires. Since g_t needs a valid heat function in accordance with Definition 6.9, it needs to dissipate enough heat to obey the second law of thermodynamics even when there are no correlations between sets of wires X and Y. Thus, when $\mathcal{C}$ forces correlations between these wires, there will be positive entropy production. This argument can be generalized past erasure gates. More generally, the entropy production of a gate is lower bounded by the drop of mutual information between the input wires to that gate and the state of the rest of the circuit (Wolpert and Kolchinsky, 2020). While the choice of f^{g_t} is not an injective function, depending on the details of $\mathcal{C}$, its action on the overall state of the circuit

might be logically reversible. For example, if g_t erased values that are copied in other wires, its action would be logically reversible. Yet Lemma 6.24 shows that g_t must have positive entropy production on some input distributions.

Remark 6.1. A logically reversible operation may have positive entropy production depending on its physical implementation and input distribution.

6.5 PP-hardness of minimizing entropy flow

Here is the language we will prove to be PP-complete.

Definition 6.25. The *Boolean circuit entropy flow minimization language* $\mathcal{L}_{EF\ \min}$ is a subset of tuples $(\mathcal{C}, g, x, v)$ containing Boolean circuit $\mathcal{C}$ with n input wires, a gate g of that circuit, an assignment x to the inputs of g, and a rational number v. A tuple is in $\mathcal{L}_{EF\ \min}$ iff when the circuit $\mathcal{C}$ receives the uniform distribution over its input wires, the valid heat function for $\mathcal{C}$ that minimizes the entropy flow releases at least $\log_2(v)$ heat on input x.

Theorem 6.26. *$\mathcal{L}_{EF\ min}$ is a PP-complete language.*

We break the proof of Theorem 6.26 into two parts: containment in PP and completeness.

Lemma 6.27. $\mathcal{L}_{EF\ min} \in PP$

Proof. Consider an input $(\mathcal{C}, g, x, v)$. Per Lemma 6.21 the entropy flow is minimized by a heat function that dissipates $\log_2(p'_{f(x)}/p_x)$ heat on input x. Thus we need to design a polynomial time nondeterministic program (Turing machine) that accepts on half of the computational paths when $v \leq p'_{f(x)}/p_x$. Rewriting this give us that we want:

$$vp_x \leq p'_{f(x)}$$

Let $p'_{\neg x \wedge f(x)} = p'_{f(x)} - p_x$ denote the probability that the gate outputs $f(x)$ when the input is not x. Then we can rewrite the above as:

$$(v-1)p_x \leq p'_{\neg x \wedge f(x)}$$

Since $v - 1$ is a rational number we can rewrite it as a/b for $a, b \in \mathbb{Z}$. Therefore, we want to reach an accepting state on at least half of the computational paths exactly for the inputs where $a \cdot p_x \leq b \cdot p'_{\neg x \wedge f(x)}$. We can do this with the following nondeterministic program: non-deterministically pick an input $y \in \{0,1\}^n$ for the Boolean circuit. Then evaluate the circuit until right before the application of gate g. Let y be the input to gate g. If $x = y$ then the program non-deterministically branches a times, rejecting on all of these branches. If $f(x) = f(y)$ then the program non-deterministically branches b times, accepting on all of these branches. Otherwise the program non-deterministically branches once, accepting on one branch while rejecting on the other. □

Lemma 6.28. *$\mathcal{L}_{EF\ min}$ is PP-hard.*

Proof. We will prove this with a Karp reduction $\mathcal{L}_{BC/2} \leq_p \mathcal{L}_{EF\ \min}$. Let $\mathcal{C}$ be an input to $\mathcal{L}_{BC/2}$ where the first wire evaluates to 1 with some unknown probability p_1. We will augment $\mathcal{C}$ with a single input erasure gate g on the first wire applied after all the gates in $\mathcal{C}$. Let this new circuit be called $\mathcal{C}'$. Then our input to $\mathcal{L}_{EF\ \min}$ is the tuple $(\mathcal{C}, g, 0, 2)$. Per Corollary 6.22, the minimum entropy flow heat function for gate g should dissipate $\log_2(1/(1-p_1))$ heat on input 0. This value is at least 1 exactly when $p_1 \geq \frac{1}{2}$. □

In fact, we can extend our result to other input distributions that can be efficiently sampled.

Corollary 6.29. *If we augment the language $\mathcal{L}_{EF\ min}$ with an input distribution $\mathcal{D}$ specified by an efficiently computable $\mathcal{D}$-generator, then the problem remains PP-complete.*

Changing the language in this way does not change the proof of Lemma 6.28 and the algorithm in Lemma 6.27 can be easily modified to first apply the $\mathcal{D}$-generator to a uniform input to obtain inputs from the desired distribution $\mathcal{D}$ for the circuit.

6.6 Hardness of approximation

In Section 6.5 we showed that computing the *optimal* heat function for gates in a Boolean circuit is hard. Our proof hinges on the difficulty of exactly determining the fraction of inputs that cause a wire to produce the output value 1. However, one might hope that this complexity can be avoided if we merely want to approximate the optimal heat function. Let $\mathcal{D}_t$ be the marginal distribution over $\{0,1\}^D_{g_t}$ induced by the uniform distribution over inputs before the application of gate g_t. Let $EF_{Q_t}[\mathcal{D}_t]$ be the entropy flow produced by the optimal heat function Q_t for gate g_t. Without direct access to $\mathcal{D}_t$, can we efficiently construct a heat function $\hat{Q}_t$ such that its entropy flow $EF_{\hat{Q}_t}[\mathcal{D}_t] \leq (1+\varepsilon)EF_{Q_t}[\mathcal{D}_t]$ for a constant $\varepsilon > 0$? An efficient algorithm for constructing such a $\hat{Q}_t$ would let us design Boolean circuits that dissipate at most $(1+\varepsilon)$ times as much heat as would be optimal, regardless of the number of component gates. Unfortunately we prove that this problem remains hard — even determining if a gate needs to dissipate *any heat* is a NP-complete problem.

Definition 6.30. $\mathcal{L}_{EF>0}$ is the subset of $\{0,1\}^*$ interpreted as tuples $(\mathcal{C}, g)$ where $\mathcal{C}$ is a Boolean circuit on n inputs and g is a gate in $\mathcal{C}$. A string is in $\mathcal{L}_{EF>0}$ iff when $\mathcal{C}$ receives a uniformly random input, any valid heat function for g must have positive entropy flow on the marginal input distribution.

Theorem 6.31. *$\mathcal{L}_{EF>0}$ is a NP-complete language.*

We again break this proof into containment in NP and hardness for NP.

Lemma 6.32. *$\mathcal{L}_{EF>0} \in NP$.*

Proof. Consider an input $(\mathcal{C}, g)$ where gate g computes the Boolean function f and let $\mathcal{D}$ be the marginal distribution over inputs to g when $\mathcal{C}$ receives a uniformly random

input. Per Definition 6.9, since f has a deterministic output for each input, there is a valid heat function for g with zero entropy flow iff $f(x) \neq f(x')$ for all pairs x, x' such that $x \neq x'$ and $\Pr_{\mathcal{D}}[x], \Pr_{\mathcal{D}}[x'] > 0$. We will construct a polynomial time nondeterministic program (Turing machine) that only accepts pairs $(\mathcal{C}, g)$ with this property. The program non-deterministically picks $x, x' \in \{0,1\}^n$ and rejects if $x = x'$. When $x \neq x'$ the program simulates $\mathcal{C}$ on these inputs until it is about to execute gate g. Let y, y' be the state of the input wires to g on inputs x, x' respectively. Then the program accepts if $f(y) = f(y')$ and otherwise rejects. □

Lemma 6.33. *$\mathcal{L}_{EF>0}$ is NP-hard.*

Proof. We will prove this with a Karp reduction $\mathcal{L}_{SAT} \leq_p \mathcal{L}_{EF>0}$. Let $\mathcal{C}$ be the input to $\mathcal{L}_{SAT}$. We will construct a new circuit $\mathcal{C}'$ on $n+1$ inputs consisting of the same sequence of gates[2] in $\mathcal{C}$ followed by a new gate g acting on the first wire of $\mathcal{C}$ as the first input and the new input wire as the second input. This gate implements the function $f(x_1, x_2) = (x_1, (1 - x_1)x_2)$. Our input to $\mathcal{L}_{EF>0}$ is then the pair $(\mathcal{C}', g)$. The only inputs $(x_1, x_2) \neq (x_1', x_2')$ where $f(x_1, x_2) = f(x_1, x_2)$ is when $x_1 = 1$. Thus $(\mathcal{C}', g) \in \mathcal{L}_{EF>0}$ exactly when the first wire of $\mathcal{C}$ evaluates to 1, implying that $\mathcal{C} \in \mathcal{L}_{SAT}$. □

Corollary 6.34. *Unless $P = NP$, there is no polynomial algorithm for constructing a heat function Q for a gate g in a Boolean circuit $\mathcal{C}$ such that when $\mathcal{C}$ is run on the uniform distribution of inputs, the entropy flow of gate g on its marginal distribution over inputs $\mathcal{D}$ is at most $(1 + \varepsilon)$ times that of the optimal heat function.*

[2]The gates are redefined to behave the same as in $\mathcal{C}$; they do not touch the wire containing the new input.

Chapter 7: Conclusions

7.1 Summary

Time, space, and energy are the three most important measures of cost for computation. Recent innovations in computer science and statistical physics have led to new ways to think about these resources when computing. In this work we have explored many results relating to these modernized notions of complexity. We have shown how to improve the time-qubit efficiency of classical subroutines in quantum algorithms, lower bound the time-space and cumulative memory efficiency of classical and quantum computation, perform useful computation driven entirely by Brownian motion, and characterize the computational complexity of optimizing Boolean circuits.

The first generations of fault-tolerant quantum computers will likely have a very limited number of logical qubits, thus it is extremely important that we understand the capabilities and limitations of *space-bounded* quantum computation. To this aim we have discussed the *spooky pebble game* — a tool for simulating logically irreversible classical subroutines in quantum algorithms with smaller overheads in terms of time and qubit count. We gave a tight characterization of the spooky pebble game played on the line graph, leading to optimal black box time-qubit savings with the technique. Next we discussed new *time-space tradeoff lower bounds* for linear algebra problems, showing that there is no quantum time-space advantage for many of these problems in a query model.

In addition to more traditional time-space tradeoffs, cumulative memory is another important notion of space and space for near term quantum computation. As long as fresh ancillary qubits are reset to $|0\rangle$ before use, the expected number of errors in a quantum circuit scales linearly with its cumulative memory complexity. This implies that quantum algorithms with lower cumulative memory complexity can be implemented using fewer physical qubits for each logical qubit. Cumulative memory

complexity is also natural in settings like high performance and cloud computing where space resources can be shared between concurrently executed tasks. We proved *unconditional* classical and quantum cumulative memory lower bounds for sorting. We also proved a general theorem that can be used to convert virtually all existing classical and quantum time-space tradeoff lower bounds to matching bounds on the cumulative memory complexity. Hence, we immediately get tight lower bounds on the cumulative memory complexity for problems where the existing time-space tradeoff bounds are known to be tight. Interpreted another way, our general theorem establishes that — if one wants to prove an unconditional separation between cumulative memory and time-space product complexity for any problem — then fundamentally new techniques would be necessary to prove the requisite time-space product lower bounds.

Finally, we reviewed thermodynamic costs in computation through the lens of stochastic thermodynamics. Entropy production is a natural metric of unavoidable thermodynamic computational costs. The entropy production of a computation depends not only on its logical behavior, but also the distribution over inputs and the physics of the underlying device, making it harder to work with than other notions of complexity. We present an abstract way to construct devices driven by Brownian motion that perform computational sampling and function computation tasks where the energetic costs are independent of the problem's time complexity. In other words, we built hypothetical devices that can perform computation where the energetic costs scale linearly with the input / output size and otherwise independently of the computation time. Additionally, we consider the thermodynamics of Boolean circuits and show that fine-tuning their energetic costs is a PP-hard problem, making it intractable in general.

7.2 Future work

Spooky pebble game We have shown asymptotically tight upper and lower bounds for the spooky pebble game when played on the line graph. Going beyond the line

graph to arbitrary DAGs presents significant challenges as the number of pebbles needed in the spooky pebble game is PSPACE-hard to approximate. Despite this, it is still possible to design efficient pebbling strategies for specific graphs like the complete binary tree. We think it would be interesting to explore strategies for the spooky pebble game on graph topologies like butterfly graphs used in computing the DFT (Tompa, 1980), and those used in the construction of data-independent memory-hard functions (Alwen and Serbinenko, 2015; Ren and Devadas, 2016; Alwen et al., 2017a; Blocki et al., 2022). In particular, it would be interesting to explore the cumulative pebbling costs of these graphs in the spooky pebble game. Finally, the spooky pebble game is but one way to use intermediate measurements to improve the time-space efficiency of quantum algorithms. In some cases other methods of using intermediate measurements — like block encodings ((Gilyén et al., 2019)) — could yield better time and space efficiency tradeoffs for quantum algorithms.

Quantum time-space tradeoffs for linear algebra problems The techniques we used to prove our lower bounds for these problems are very general and could likely be applied to give similar bounds on other problems like universal hashing. More generally the recording query method gives a much more intuitive way to think about quantum query complexity. This method can be used to give a cleaner proofs of many existing results and has potential applications for proving new quantum lower bounds on query problems like property and distribution testing.

Cumulative memory complexity In this dissertation we presented negative results — showing that virtually all existing classical and quantum time-space product lower bounds can be extended to give matching lower bounds on the (always smaller) notion of cumulative memory complexity. Thus fundamentally new techniques for time-space product lower bounds are necessary if we want to prove an unconditional asymptotic separation between time-space product and cumulative memory complexity. While proving an unconditional classical or quantum advantage for cumulative memory

complexity seems unlikely, designing concrete classical and quantum algorithms that are optimized for low cumulative memory complexity is still a promising research direction. Our general theorem only precludes the existence of cumulative memory efficient sequential algorithms for tasks where existing techniques yield a tight time-space tradeoff lower bound. Thus we suggest focusing on parallel algorithms and sequential problems without existing tight lower bounds.

In the cryptographic setting, there have been many interesting parallel classical and quantum algorithms designed to break password hashing functions using low cumulative memory. These algorithms often exhibit many steps with low space interspersed with short bursts that use high space so that the cumulative memory remains lower than the time-space product complexity (e.g. (Alwen and Blocki, 2016; Blocki et al., 2022)). Particular non-cryptographic domains where similar tricks could be used include divide and conquer, streaming, and graph problems. While there is extensive prior work on classical and quantum time-space efficient algorithms in other settings, design of such algorithms is focused on how to make the most of a fixed maximum space bound. Thus there are ample opportunities to design classical and quantum algorithms for non-cryptographic problems that can take advantage of spikes in memory to maintain low cumulative memory complexity. While seeking out new cumulative memory efficient algorithms, we also want to develop novel techniques for proving tight time-space product lower bounds on problems where we believe there may be an unconditional asymptotic cumulative memory advantage.

Thermodynamics of computation The connections between stochastic thermodynamics and computation are very recent, leaving many open avenues for further exploration. In particular, we are interested in better understanding the thermodynamics of branching programs. Historically, branching programs have been used to lower bound time-space tradeoffs for computation. If it is possible to give a reasonable definition of entropy production in branching programs, it may lead to unconditional tradeoffs between time, space, and energy for classical computation.

Appendix

Cumulative memory complexity

This section contains key lemmas needed to prove our results in Chapter 4.

Extending the lower bound to arbitrary success probabilities

It is possible to modify Theorem 4.19 for different success probabilities. When n is sufficiently large, any quantum circuit $\mathcal{C}$ for sorting a list of length n with failure probability at most δ and at most T layers that produces its sorted outputs in any fixed time order requires cumulative memory that is $\Omega((1-\delta)n^3/T)$. The key is to choose a different value for the parameter γ in Proposition 4.16 governing the completeness bound, which will change the corresponding value of α and β. If we want to deal with non-constant values of γ, it is important to understand how γ and α are related in Proposition 4.16. The following lemma is sufficient to prove Theorem 13 in (Klauck et al., 2007) (our Proposition 4.16). Although the authors of (Klauck et al., 2007) prove a more general version of this proposition, the statement below captures what is necessary in our proof. Specifically, we invoke their Lemma 12 where $\delta = 0, C = ke^{\gamma+1}$ and $D = \alpha\sqrt{kn}$.

Proposition G.1 (Special case of Lemma 12 in (Klauck et al., 2007))**.** *Let p be a degree $2\alpha\sqrt{kn}$ univariate polynomial such that:*

- $p(i) = 0$ *when* $i \in \{0, \ldots, k-1\}$
- $p(k) = \sigma$
- $p(i) \in [0,1]$ *when* $i \in \{k+1, \ldots, n\}$

Then there exists universal positive constants a and b such that for any $\gamma > 0$ where

$ke^{\gamma+1} \leq n - k$:

$$\sigma \leq a \cdot exp\left(\frac{b(2\alpha\sqrt{kn} - k)^2 + 4e^{\gamma/2+1/2}k\sqrt{n-k}(2\alpha\sqrt{n} - \sqrt{k})}{n - k(e^{\gamma+1}+1)} - k - \gamma k\right).$$

The σ in this bound gives the completeness bound on the k-threshold problem. We now prove that σ is sufficiently small when $\alpha \in \Omega(e^{-\gamma/2})$.

Lemma G.2. *Let $a, b > 0$ be the constants from Proposition G.1. When we have $\sqrt{k/n} < \alpha < \min\left(1/(16\sqrt{e^{\gamma+1}+1}), \sqrt{1/(8b)}\right)$, the completeness bound σ for the k-threshold problem with $\alpha\sqrt{kn}$ queries is less than $a \cdot e^{-\gamma k}$.*

Proof. By Proposition G.1 we have

$$\sigma \leq a \cdot \exp\left(\frac{b(2\alpha\sqrt{kn} - k)^2 + 4e^{\gamma/2+1/2}k\sqrt{n-k}(2\alpha\sqrt{n} - \sqrt{k})}{n - k(e^{\gamma+1}+1)} - k - \gamma k\right) \quad \text{(G.1)}$$

We first bound the first term in the numerator

$$\begin{aligned}
b(2\alpha\sqrt{kn} - k)^2 &= b(4\alpha^2 kn - 4\alpha k\sqrt{kn} + k^2) \\
&= kb\alpha^2(4n - 4\sqrt{kn}/\alpha + k/\alpha^2) \\
&< kb\alpha^2(4n - 3k/\alpha^2) \qquad \text{since } \sqrt{k/n} < \alpha \\
&< kb\alpha^2(4n - 4k(e^{\gamma+1}+1)) \qquad \text{as } \alpha < 1/(16\sqrt{e^{\gamma+1}+1}) \\
&= 4kb\alpha^2(n - k(e^{\gamma+1}+1)).
\end{aligned}$$

Next we bound the second term in the numerator

$$\begin{aligned}
4e^{\gamma/2+1/2}k\sqrt{n-k}(2\alpha\sqrt{n} - \sqrt{k}) &\leq 4e^{\gamma/2+1/2}k\sqrt{n}(2\alpha\sqrt{n} - \sqrt{k}) \\
&= 4e^{\gamma/2+1/2}k\alpha(2n - \sqrt{nk}/\alpha) \\
&< 4e^{\gamma/2+1/2}k\alpha(2n - k/\alpha^2) \qquad \text{as } \sqrt{k/n} < \alpha \\
&< 4e^{\gamma/2+1/2}k\alpha(2n - 2k(e^{\gamma+1}+1)) \quad \text{as } \alpha < 1/(16\sqrt{e^{\gamma+1}+1}) \\
&= 8e^{\gamma/2+1/2}k\alpha(n - k(e^{\gamma+1}+1))
\end{aligned}$$

Plugging these bounds into (G.1) we get

$$\begin{aligned}\sigma &< a \cdot \exp\left(4kb\alpha^2 + 8e^{\gamma/2+1/2}k\alpha - k - \gamma k\right) \\ &< a \cdot \exp\left(4kb\alpha^2 - k/2 - \gamma k\right) \qquad \text{as } \alpha < 1/(16\sqrt{e^{\gamma+1}+1}) \\ &< a \cdot \exp\left(-\gamma k\right) \qquad \text{as } \alpha < \sqrt{1/8b}\end{aligned}$$

□

We now describe how to substitute the bound of Lemma G.2 in place of Proposition 4.16 to yield cumulative memory lower bounds for quantum circuits with failure probability at most δ: To prove the analog of Lemma 4.18, for any S, k, and $\delta \in (0,1)$, we can reprove a more precise version of Lemma 4.17 using Lemma G.2 instead of Proposition 4.16 to bound the success probability for the k-threshold problem and choose

$$\gamma = \frac{\ln(a \cdot 2^{2S}/(1-\delta)) - 1}{k} + 1$$

to get a success probability of less than $1-\delta$ for circuits with as many as

$$\Omega\left(\left(\frac{1-\delta}{2^{2S}}\right)^{1/2k}\sqrt{kn}\right)$$

layers and S qubits of advice to produce k outputs. If we repeat the proof of Theorem 4.19 for failure probability less than δ, we can set β to a value that is $\Omega(\sqrt{1-\delta})$ to obtain a lower bound on the cumulative memory that is $\Omega((1-\delta)n^3/T)$.

Optimizations

In this section we prove general optimization lemmas that allow us to derive worst-case properties of the allocation of branching program layers into blocks.

The first special case is relevant for our analysis of quantum sorting algorithms.

Lemma G.3. *For non-negative reals $x_1, x_2, \ldots$ if $\sum_i x_i \leq \sum_i x_i^2$ then $\sum_i x_i^3 \geq \sum_i x_i^2$.*

Proof. Without loss generality we remove all x_i that are 0 or 1 since they contribute the same amount to each of $\sum_i x_i$, $\sum_i x_i^2$, and $\sum_i x_i^3$. Therefore every x_i satisfies

$0 < x_i < 1$ or it satisfies $x_i > 1$. We rename those x_i with $0 < x_i < 1$ by y_i and those x_i with $x_i > 1$ by z_j.

Then $\sum_i x_i \leq \sum_i x_i^2$ can be rewritten as $\sum_i y_i(1-y_i) \leq \sum_j z_j(z_j-1)$, and both quantities are positive. Let y^* be the largest value < 1 and z^* be the smallest value > 1. Thus:

$$\sum_i (y_i^2 - y_i^3) = \sum_i y_i^2(1-y_i) \leq \sum_i y^* y_i(1-y_i) = y^* \sum_i y_i(1-y_i) \leq y^* \sum_j z_j(z_j-1)$$
$$< z^* \sum_j z_j(z_j-1) = \sum_j z^* z_j(z_j-1) \leq \sum_j z_j^2(z_j-1) = \sum_j (z_j^3 - z_j^2).$$

Rewriting gives $\sum_i y_i^2 + \sum_j z_j^2 < \sum_i y_i^3 + \sum_j z_j^3$, or $\sum_i x_i^3 > \sum_i x_i^2$, as required. □

The following is a generalization of the above to all differentiable functions $p : \mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0}$ such that $s/p(s)$ is a concave increasing function of s.

Lemma 4.25. *Let $p : \mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0}$ be a differentiable function such that $q(x) = x/p(x)$ is a concave increasing function of x. For $x_1, x_2, \ldots \in \mathbb{R}^{\geq 0}$, if $\sum_i x_i \geq K$ and $\sum_i p(x_i) \leq L$ then $\sum_i x_i p(x_i) \geq q^{-1}(K/L) \cdot L$.*

Proof. By hypothesis, $\sum_i (x_i - Kp(x_i)/L) \geq 0$. Observe that $s - Kp(s)/L$ is an increasing function of s since $s/p(s)$ is an increasing function of s that is 0 precisely when $s = q^{-1}(K/L)$. Since all x_i with $x_i = q^{-1}(K/L)$ evaluate to 0 in the sum, we can rewrite it as

$$\sum_{x_i > q^{-1}(K/L)} (x_i - Kp(x_i)/L) \geq \sum_{x_i < q^{-1}(K/L)} (Kp(x_i)/L - x_i)\,, \tag{G.2}$$

where each of the summed terms is positive. For $x_i \neq q^{-1}(K/L)$, define

$$f(x_i) = x_i \cdot \frac{p(x_i) - q^{-1}(K/L) \cdot L/K}{x_i - Kp(x_i)/L}.$$

Observe that for $x_i = q^{-1}(K/L)$ the denominator is 0 and the numerator equals $p(x_i) - x_i \cdot L/K$ which is also 0. For $x_i > q^{-1}(K/L)$ both the numerator and denominator are positive and for $x_i < q^{-1}(K/L)$ both the numerator and denominator are negative. Hence $f(x_i)$ is non-negative for every $x_i \neq q^{-1}(K/L)$.

Claim G.4. *If q is a convex differentiable function, we can complete f to a (non-decreasing) continuous function of x with $f'(x) \geq 0$ for all x with $0 < x \neq q^{-1}(K/L)$*

Proof of Claim. Write $a = q^{-1}(K/L)$. Then since $p(x) > 0$ and $q(a) > 0$, we have

$$\begin{aligned} f(x) &= \frac{x \cdot p(x) - x \cdot a/q(a)}{x - q(a) \cdot p(x)} = \frac{x - (x/p(x)) \cdot a/q(a)}{x/p(x) - q(a)} \\ &= \frac{x - q(x) \cdot a/q(a)}{q(x) - q(a)} = \frac{1}{q(a)} \cdot \frac{q(a) \cdot x - a \cdot q(x)}{q(x) - q(a)}. \end{aligned}$$

Therefore

$$\begin{aligned} f'(x) &= \frac{1}{q(a)} \cdot \frac{(q(a) - a \cdot q'(x))(q(x) - q(a)) - (q(a) \cdot x - a \cdot q(x)) \cdot q'(x)}{(q(x) - q(a))^2} \\ &= \frac{q(x) - q(a) + (a - x) \cdot q'(x)}{(q(x) - q(a))^2}. \end{aligned}$$

Since the denominator is a square and q is increasing, to prove that $f'(x) \geq 0$ for $x \neq a$ it suffices to prove that the numerator is non-negative.

Suppose first that $x < a$, Then $a - x > 0$ and the numerator $q(x) - q(a) + (a - x) \cdot q'(x) \geq 0$ if and only if $q'(x) \geq \frac{q(a) - q(x)}{a - x}$, which is equivalent to the slope of the tangent to q at x being at least that of the chord from x to a. This is certainly true since q is a concave function.

Suppose now that $x > a$. Then $a - x < 0$ and the numerator $q(x) - q(a) + (a - x) \cdot q'(x) \geq 0$ if and only if $q'(x) \leq \frac{q(x) - q(a)}{x - a}$. Again, this is true since q is a concave function.

It remains to show that we can complete f to a continuous function by giving it a finite value at $a = q^{-1}(K/L)$. By l'Hôpital's rule, the limit of $q(a) \cdot f(x)$ as x approaches a is

$$\frac{q(a) - a \cdot q'(a)}{q'(a)}$$

if the denominator is non-zero, which it is, since q is an increasing differentiable function at a. □

We now have the tools we need. Let x^*_- be the largest $x_i < q^{-1}(K/L)$ and x^*_+ be the smallest $x_i > q^{-1}(K/L)$. Then we have $f(x^*_+) \geq f(x^*_-)$ and

$$\begin{aligned}
&\sum_{x_i > q^{-1}(K/L)} \Big(x_i \; p(x_i) - q^{-1}(K/L) \cdot L/K \cdot x_i \Big) \\
&\qquad = \sum_{x_i > q^{-1}(K/L)} f(x_i) \cdot (x_i - Kp(x_i)/L) \\
&\qquad \geq \sum_{x_i > q^{-1}(K/L)} f(x^*_+) \cdot (x_i - Kp(x_i)/L) \\
&\qquad \geq f(x^*_-) \sum_{x_i > q^{-1}(K/L)} (x_i - Kp(x_i)/L) \\
&\qquad \geq f(x^*_-) \sum_{x_i < q^{-1}(K/L)} (Kp(x_i)/L - x_i) \qquad \text{by Equation (G.2)} \\
&\qquad \geq \sum_{x_i < q^{-1}(K/L)} f(x_i) \cdot (Kp(x_i)/L - x_i) \\
&\qquad = \sum_{x_i < q^{-1}(K/L)} \Big(q^{-1}(K/L) \cdot L/K \cdot x_i - x_i \; p(x_i) \Big) .
\end{aligned}$$

Adding back the terms where $x_i = q^{-1}(K/L)$, which have value 0, and rewriting we obtain

$$\sum_i \Big(x_i \; p(x_i) - q^{-1}(K/L) \cdot L/K \cdot x_i \Big) \geq 0.$$

Therefore we have

$$\sum_i x_i \; p(x_i) \geq q^{-1}(K/L) \cdot L/K \cdot \sum_i x_i \geq q^{-1}(K/L) \cdot (L/K) \cdot K = q^{-1}(K/L) \cdot L. \quad \square$$

Bounding the loss

Lemma 4.22. *The following hold for every non-decreasing function $p : \mathbb{R} \to \mathbb{R}$ with $p(1) = 1$:*

(a) $1/p(n) \leq \mathcal{L}_p(n) \leq 1$.

(b) If p is a polynomial function $p(s) = s^{1/c}$ then $\mathcal{L}_p(n) > 1/2^{c+1}$.

(c) For any $c > 1$, $\mathcal{L}_p(n) \geq \min\limits_{1 \leq s \leq n} \dfrac{p(s) - p(s/c)}{cp(s)}$.

(d) We say that p is nice *if it is differentiable and there is an integer $c > 1$ such that for all x, $p'(cx) \geq p'(x)/c$. If p is nice then $\mathcal{L}_p(n)$ is $\Omega(1/\log_2 n)$. This is tight for p with $p(s) = 1 + \log_2 s$.*

Proof. Since p is non-decreasing, $1 \leq p^{-1}(j) \leq p^{-1}(k)$ for $1 \leq j \leq k$ and hence

$$\frac{1}{k} \leq \frac{\sum_{j=1}^{k} p^{-1}(j)}{k \cdot p^{-1}(k)} \leq 1 \tag{G.3}$$

since $p^{-1}(k)$ is included in the numerator. $\mathcal{L}_p(n)$ is the minimum over all integers $k \in [1, p(n)]$ of $\frac{\sum_{j=1}^{k} p^{-1}(j)}{k \cdot p^{-1}(k)}$ and p is non-decreasing so we have $1/p(n) \leq \mathcal{L}_p(n) \leq 1$, which proves part (a)

When $p(s) = s^{1/c}$ we have

$$\sum_{j=1}^{k} p^{-1}(j) \geq \sum_{j=\lceil (k+1)/2 \rceil}^{k} j^{\,c} > \lceil k/2 \rceil (k/2)^c \geq (k/2)^{c+1} = k \cdot p^{-1}(k)/2^{c+1}$$

so each term in the definition of $\mathcal{L}_p(n)$ is larger than $1/2^{c+1}$ which proves part (b). (More precise bounds can be shown, but we are not focused on the specific constant.)

Let $1 \leq k \leq p(n)$ be an integer. Then $1 \leq s = p^{-1}(k) \leq n$. Observe that there are at least $p(s) - p(s/c)$ integers $j \leq k$ with $p^{-1}(j) \geq s/c$. Therefore

$$\frac{\sum_{j=1}^{k} p^{-1}(j)}{k \cdot p^{-1}(k)} \geq \frac{(p(s) - p(s/c)) \cdot s/c}{k p^{-1}(k)} = \frac{p(s) - p(s/c)}{ck} = \frac{p(s) - p(s/c)}{cp(s)}. \tag{G.4}$$

The minimum over all $k \in [1, p(n)]$ is equivalent to the minimum over all $s \in [1, n]$, which proves part (c).

Now suppose that p is nice. Since p is differentiable, for any s,

$$\begin{aligned} p(cs) - p(s) &= \int_{s}^{cs} p'(y)\, dy \\ &= \int_{s/c}^{c} p'(cx) c\, dx \quad \text{by substitution } y = cx \\ &\geq \int_{s/c}^{c} p'(x)\, dx \quad \text{since } p \text{ is nice} \\ &= p(s) - p(s/c). \end{aligned}$$

Then by induction we have that for every positive integer $i \leq \log_c s$, $p(s) - p(s/c) \geq p(s/c^{i-1}) - p(s/c^i)$. Write $\ell = \lfloor \log_c s \rfloor$. Then $s/c^\ell < c$ and

$$p(s) - p(s/c^\ell) = \sum_{i=1}^{\ell} [p(s/c^{i-1}) - p(s/c^i)] \leq \ell \cdot [p(s) - p(s/c)],$$

or equivalently that $p(s) - p(s/c) \geq (p(s) - p(s/c^\ell))/\ell$ and hence

$$p(s) - p(s/c) \geq (p(s) - p(c))/\log_c s$$

since p is a non-decreasing function. Applying the lower bound from Equation (G.3) when $k = p(s) < 2p(c)$ and the lower bound from Equation (G.4) when $p(s) \geq 2p(c)$ we obtain

$$\mathcal{L}_p(n) \geq \min\left(\frac{1}{2p(c)}, \min_{1 \leq s \leq n : p(s) \geq 2p(c)} (1 - p(c)/p(s))/(c \log_c s)\right).$$

Since c is a constant, we obtain that $\mathcal{L}_p(n)$ is $\Omega(1/\log n)$.

Observe that p given by $p(s) = 1 + \log_2 s$ is nice for every constant $c > 0$ since $p'(cx) = (\ln 2)^{-1}/(cx) = p'(x)/c$. In this case we have $p^{-1}(j) = 2^{j-1}$ and $\mathcal{L}_p(n) < 2/p(n) < 2/\log_2 n$ since the largest term $p^{-1}(k)$ in each numerator is (a little) more than the sum of all smaller terms put together. Together with the lower bound, this proves part (d). □

Works Cited

Scott Aaronson. Limitations of quantum advice and one-way communication. *Theory of Computing*, 1, February 2005. ISSN 1557-2862. doi: 10.4086/toc.2005.v001a001. URL https://doi.org/10.4086/toc.2005.v001a001.

Scott Aaronson. Oracles are subtle but not malicious. In *Proceedings of the 21st Annual IEEE Conference on Computational Complexity*, CCC '06, pages 340–354. IEEE, 2006. ISBN 0769525962. doi: 10.1109/CCC.2006.32. URL https://doi.org/10.1109/CCC.2006.32.

Scott Aaronson. Introduction to quantum information science lecture notes, 2018. URL https://scottaaronson.com/qclec.pdf.

Scott Aaronson and Alex Arkhipov. The computational complexity of linear optics. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing*, STOC '11, pages 333–342. ACM, 2011. ISBN 9781450306911. doi: 10.1145/1993636.1993682. URL https://doi.org/10.1145/1993636.1993682.

Scott Aaronson, Daniel Grier, and Luke Schaeffer. The classification of reversible bit operations. In *Proceedings of the 8th Innovations in Theoretical Computer Science Conference*, volume 67 of *ITCS '17*, pages 23:1–23:34. LIPIcs, 2017. ISBN 978-3-95977-029-3. doi: 10.4230/LIPIcs.ITCS.2017.23. URL https://doi.org/10.4230/LIPIcs.ITCS.2017.23.

Farid Ablayev, Cristopher Moore, and Christopher Pollett. Quantum and stochastic branching programs of bounded width. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, ICALP '02, pages 343–354. Springer, 2002. ISBN 978-3-540-45465-6. doi: 10.1007/3-540-45465-9_30. URL https://doi.org/10.1007/3-540-45465-9_30.

Karl R. Abrahamson. A time-space tradeoff for Boolean matrix multiplication. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, volume 1 of *SFCS '90*, pages 412–419. IEEE, 1990. doi: 10.1109/FSCS.1990.89561. URL https://doi.org/10.1109/FSCS.1990.89561.

Karl R. Abrahamson. Time-space tradeoffs for algebraic problems on general sequential machines. *Journal of Computer and System Sciences*, 43(2):269–289, 1991. ISSN 1090-2724. doi: 10.1016/0022-0000(91)90014-v. URL https://doi.org/10.1016/0022-0000(91)90014-V.

Google Quantum AI and Collaborators. Quantum error correction below the surface code threshold. *Nature*, 638:920–926, December 2024. ISSN 1476-4687. doi: 10.1038/s41586-024-08449-y. URL https://doi.org/10.1038/s41586-024-08449-y.

Joël Alwen and Jeremiah Blocki. Efficiently computing data-independent memory-hard functions. In *Proceedings of Advances in Cryptology – CRYPTO 2016*, CRYPTO '16, pages 241–271. Springer, 2016. ISBN 978-3-662-53008-5. doi: 10.1007/978-3-662-53008-5_9. URL https://doi.org/10.1007/978-3-662-53008-5_9.

Joël Alwen and Vladimir Serbinenko. High parallel complexity graphs and memory-hard functions. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing*, STOC '15, pages 595–603. ACM, 2015. ISBN 9781450335362. doi: 10.1145/2746539.2746622. URL https://doi.org/10.1145/2746539.2746622.

Joël Alwen, Binyi Chen, Chethan Kamath, Vladimir Kolmogorov, Krzysztof Pietrzak, and Stefano Tessaro. On the complexity of scrypt and proofs of space in the parallel random oracle model. In *Proceedings of Advances in Cryptology – EUROCRYPT 2016*, EUROCRYPT '16, pages 358–387. Springer, 2016. doi:

10.1007/978-3-662-49896-5_13. URL https://doi.org/10.1007/978-3-662-49896-5_13.

Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Depth-robust graphs and their cumulative memory complexity. In *Proceedings of Advances in Cryptology – EUROCRYPT 2017*, EUROCRYPT '17, pages 3–32. Springer, 2017a. ISBN 978-3-319-56617-7. doi: 10.1007/978-3-319-56617-7_1. URL https://doi.org/10.1007/978-3-319-56617-7_1.

Joël Alwen, Binyi Chen, Krzysztof Pietrzak, Leonid Reyzin, and Stefano Tessaro. Scrypt is maximally memory-hard. In *Proceedings of Advances in Cryptology – EUROCRYPT 2017*, EUROCRYPT '17, pages 33–62. Springer, 2017b. doi: 10.1007/978-3-319-56617-7_2. URL https://doi.org/10.1007/978-3-319-56617-7_2.

Joël Alwen, Susanna F. de Rezende, Jakob Nordström, and Marc Vinyals. Cumulative space in black-white pebbling and resolution. In *Proceedings of the 8th Innovations in Theoretical Computer Science Conference*, volume 67 of *ITCS '17*, pages 38:1–38:21. LIPIcs, 2017c. ISBN 978-3-95977-029-3. doi: 10.4230/LIPIcs.ITCS.2017.38. URL http://doi.org/10.4230/LIPIcs.ITCS.2017.38.

Andris Ambainis. Quantum lower bounds by quantum arguments. *Journal of Computer and System Sciences*, 64(4):750–767, June 2002. ISSN 0022-0000. doi: 10.1006/jcss.2002.1826. URL https://doi.org/10.1006/jcss.2002.1826.

Andris Ambainis, Robert Špalek, and Ronald de Wolf. A new quantum lower bound method, with applications to direct product theorems and time-space tradeoffs. *Algorithmica*, 55(3):422–461, September 2009. ISSN 1432-0541. doi: 10.1007/s00453-007-9022-9. URL https://doi.org/10.1007/s00453-007-9022-9.

Mohammad Hassan Ameri, Alexander R. Block, and Jeremiah Blocki. Memory-hard puzzles in the standard model with applications to memory-hard functions and resource-bounded locally decodable codes. In *Proceedings of the 13th International Conference on Security and Cryptography for Networks*, SCN '22, pages 45–68. Springer, 2022. ISBN 978-3-031-14790-6. doi: 10.1007/978-3-031-14791-3_3. URL https://doi.org/10.1007/978-3-031-14791-3_3.

Andrew Baird, Bryant Bost, Stefano Buliani, Vyom Nagrani, Ajay Nair, Rahul Popat, and Brajendra Singh. Aws serverless multi-tier architectures with amazon api gateway and aws lambda, 2021. URL https://docs.aws.amazon.com/whitepapers/latest/serverless-multi-tier-architectures-api-gateway-lambda/welcome.html.

Ainesh Bakshi and Ewin Tang. An improved classical singular value transformation for quantum machine learning. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '24, pages 2398–2453. SIAM, 2024. doi: 10.1137/1.9781611977912.86. URL https://doi.org/10.1137/1.9781611977912.86.

Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum lower bounds by polynomials. *Journal of the ACM*, 48 (4):778–797, July 2001. ISSN 0004-5411. doi: 10.1145/502090.502097. URL https://doi.org/10.1145/502090.502097.

Paul Beame. A general sequential time-space tradeoff for finding unique elements. *SIAM Journal on Computing*, 20(2):270–277, April 1991. ISSN 1095-7111. doi: 10.1137/0220017. URL https://doi.org/10.1137/0220017.

Paul Beame and Niels Kornerup. Cumulative memory lower bounds for randomized and quantum computation. In *Proceedings of the 50th International Colloquium on Automata, Languages, and Programming*, volume 261 of *ICALP*

'23, pages 17:1–17:20. LIPIcs, 2023. ISBN 978-3-95977-278-5. doi: 10.4230/LIPIcs.ICALP.2023.17. URL https://doi.org/10.4230/LIPIcs.ICALP.2023.17.

Paul Beame and Niels Kornerup. Cumulative memory lower bounds for randomized and quantum computation. *ACM Transactions on Computation Theory*, June 2025. ISSN 1942-3454. doi: 10.1145/3728715. URL https://doi.org/10.1145/3728715.

Paul Beame, T. S. Jayram, and Michael E. Saks. Time-space tradeoffs for branching programs. *Journal of Computer and System Sciences*, 63(4):542–572, December 2001. ISSN 1090-2724. doi: 10.1006/jcss.2001.1778. URL https://doi.org/10.1006/jcss.2001.1778.

Paul Beame, Niels Kornerup, and Michael Whitmeyer. Quantum time-space tradeoffs for matrix problems. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, STOC '24, pages 596–607. ACM, 2024. ISBN 9798400703836. doi: 10.1145/3618260.3649700. URL https://doi.org/10.1145/3618260.3649700.

Charles H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17(6):525–532, November 1973. ISSN 0018-8646. doi: 10.1147/rd.176.0525. URL https://doi.org/10.1147/rd.176.0525.

Charles H. Bennett. The thermodynamics of computation - a review. *International Journal of Theoretical Physics*, 21:905–840, December 1982. doi: 10.1007/BF02084158. URL https://doi.org/10.1007/BF02084158.

Charles H. Bennett. Time/space trade-offs for reversible computation. *SIAM Journal on Computing*, 18(4):766–776, August 1989. doi: 10.1137/0218053. URL https://doi.org/10.1137/0218053.

Charles H. Bennett and Rolf Landauer. The fundamental physical limits of computation, June 2011. URL https://www.scientificamerican.com/article/the-fundamental-physical-limits-of-computation/.

Ethan Bernstein and Umesh V. Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, October 1997. ISSN 1095-7111. doi: 10.1137/S0097539796300921. URL https://doi.org/10.1137/S0097539796300921.

Jeremiah Blocki and Samson Zhou. On the depth-robustness and cumulative pebbling cost of argon2i. In *Proceedings of the Theory of Cryptography*, TCC '17, pages 445–465. Springer, 2017. ISBN 978-3-319-70500-2. doi: 10.1007/978-3-319-70500-2_15. URL https://doi.org/10.1007/978-3-319-70500-2_15.

Jeremiah Blocki, Blake Holman, and Seunghoon Lee. The parallel reversible pebbling game: Analyzing the post-quantum security of imhfs. In *Proceedings of the 20th International Theory of Cryptography Conference*, TCC '22, pages 52–79. Springer, 2022. ISBN 978-3-031-22317-4. doi: 10.1007/978-3-031-22318-1_3. URL https://doi.org/10.1007/978-3-031-22318-1_3.

Dan Boneh, Henry Corrigan-Gibbs, and Stuart Schechter. Balloon hashing: A memory-hard function providing provable protection against sequential attacks. In *Proceedings of Advances in Cryptology – ASIACRYPT 2016*, ASIACRYPT '16, pages 220–248. Springer, 2016. ISBN 978-3-662-53887-6. doi: 10.1007/978-3-662-53887-6_8. URL https://doi.org/10.1007/978-3-662-53887-6_8.

Allan Borodin and Stephen A. Cook. A time-space tradeoff for sorting on a general sequential model of computation. *SIAM Journal on Computing*, 11(2):287–297, May 1982. ISSN 1095-7111. doi: 10.1137/0211022. URL https://doi.org/10.1137/0211022.

Allan Borodin, Michael J. Fischer, David G. Kirkpatrick, Nancy A. Lynch, and Martin Tompa. A time-space tradeoff for sorting on non-oblivious machines. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, SFCS '79, pages 319–327. IEEE, 1979. doi: 10.1109/SFCS.1979.4. URL https://doi.org/10.1109/SFCS.1979.4.

Daniele Cappelletti, Andrés Ortiz-Muñoz, David F. Anderson, and Erik Winfree. Stochastic chemical reaction networks for robustly approximating arbitrary probability distributions. *Theoretical Computer Science*, 801:64–95, January 2020. doi: 10.1016/j.tcs.2019.08.013. URL https://doi.org/10.1016/j.tcs.2019.08.013.

Ashok K. Chandra, Merrick L. Furst, and Richard J. Lipton. Multi-party protocols. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, STOC '83, pages 94–99. ACM, 1983. ISBN 0897910990. doi: 10.1145/800061.808737. URL https://doi.org/10.1145/800061.808737.

Binyi Chen and Stefano Tessaro. Memory-hard functions from cryptographic primitives. In *Proceedings of Advances in Cryptology – CRYPTO 2019*, CRYPTO '19, pages 543–572. Springer, 2019. ISBN 978-3-030-26950-0. doi: 10.1007/978-3-030-26951-7_19. URL https://doi.org/10.1007/978-3-030-26951-7_19.

Nadiia Chepurko, Kenneth L. Clarkson, Lior Horesh, Honghao Lin, and David P. Woodruff. Quantum-inspired algorithms from randomized numerical linear algebra. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *PMLR '22*, pages 3879–3900. PMLR, 2022. URL https://proceedings.mlr.press/v162/chepurko22a.html.

Nai-Hui Chia, András Gilyén, Tongyang Li, Han-Hsuan Lin, Ewin Tang, and Chunhao Wang. Sampling-based sublinear low-rank matrix arithmetic framework for dequantizing quantum machine learning. In *Proceedings of the 52nd*

Annual ACM Symposium on Theory of Computing, STOC 2020, pages 387–400. ACM, 2020a. ISBN 9781450369794. doi: 10.1145/3357713.3384314. URL https://doi.org/10.1145/3357713.3384314.

Nai-Hui Chia, András Gilyén, Han-Hsuan Lin, Seth Lloyd, Ewin Tang, and Chunhao Wang. Quantum-inspired algorithms for solving low-rank linear equation systems with logarithmic dependence on the dimension. In *Proceedings of the 31st International Symposium on Algorithms and Computation*, volume 181 of *ISAAC '20*, pages 47:1–47:17. LIPIcs, 2020b. ISBN 978-3-95977-173-3. doi: 10.4230/LIPIcs.ISAAC.2020.47. URL https://doi.org/10.4230/LIPIcs.ISAAC.2020.47.

Andrew M. Childs, Robin Kothari, and Rolando D. Somma. Quantum algorithm for systems of linear equations with exponentially improved dependence on precision. *SIAM Journal on Computing*, 46(6):1920–1950, November 2017. doi: 10.1137/16M1087072. URL https://doi.org/10.1137/16M1087072.

John F. Clauser, Michael A. Horne, Abner Shimony, and Richard A. Holt. Proposed experiment to test local hidden-variable theories. *Physical Review Letters*, 23(15):880–884, October 1969. ISSN 1079-7114. doi: 10.1103/PhysRevLett.23.880. URL https://doi.org/10.1103/PhysRevLett.23.880.

Alan Cobham. The recognition problem for the set of perfect squares. In *Proceedings of the 7th Annual Symposium on Switching and Automata Theory*, SWAT '66, pages 78–87. IEEE, 1966. doi: 10.1109/swat.1966.30. URL https://doi.org/10.1109/swat.1966.30.

Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158. ACM, 1971. ISBN 9781450374644. doi: 10.1145/800157.805047. URL https://doi.org/10.1145/800157.805047.

Stephen A. Cook. An observation on time-storage trade off. In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing*, STOC '73, pages 29–33. ACM, 1973. ISBN 9781450374309. doi: 10.1145/800125.804032. URL https://doi.org/10.1145/800125.804032.

Alessandro Cosentino, Robin Kothari, and Adam Paetznick. Dequantizing read-once quantum formulas. In *Proceedings of the 8th Conference on the Theory of Quantum Computation, Communication and Cryptography*, volume 22 of *TQC '13*, pages 80–92. LIPIcs, 2013. doi: 10.4230/LIPICS.TQC.2013.80. URL https://doi.org/10.4230/LIPIcs.TQC.2013.80.

Thomas M. Cover. *Elements of Information Theory, 2nd Edition*. Wiley, 2006. ISBN 978-0-471-24195-9.

Jeffrey Dastin and Stephen Nellis. Focus: For tech giants, ai like bing and bard poses billion-dollar search problem, February 2023. URL https://reuters.com/technology/tech-giants-ai-like-bing-bard-poses-billion-dollar-search-problem-2023-02-22/.

Erik D. Demaine and Quanquan C. Liu. Inapproximability of the standard pebble game and hard to pebble graphs. In *Proceedings of Algorithms and Data Structures*, WADS '17, pages 313–324. Springer, 2017a. ISBN 978-3-319-62127-2. doi: 10.1007/978-3-319-62127-2_27. URL https://doi.org/10.1007/978-3-319-62127-2_27.

Erik D. Demaine and Quanquan C. Liu. Inapproximability of the standard pebble game and hard to pebble graphs, 2017b. URL https://doi.org/10.48550/arXiv.1707.06343.

Erik D. Demaine, Jayson Lynch, Geronimo J. Mirano, and Nirvan Tyagi. Energy-efficient algorithms. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, ITCS '16, pages 321–332. ACM, 2016. ISBN

9781450340571. doi: 10.1145/2840728.2840756. URL https://doi.org/10.1145/2840728.2840756.

David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society A*, 439(1907):553–558, December 1992. ISSN 1471-2946. doi: 10.1098/rspa.1992.0167. URL https://doi.org/10.1098/rspa.1992.0167.

David Doty, Niels Kornerup, Austin Luchsinger, Leo Orshansky, David Soloveichik, and Damien Woods. Harvesting brownian motion: Zero energy computational sampling, 2024. URL https://doi.org/10.48550/arXiv.2309.06957.

Philippe Dumas. Reversing a finite sequence, 1995. URL https://algo.inria.fr/seminars/sem94-95/pottier.pdf. Summary of a seminar talk given by Loïc Pottier.

Cynthia Dwork, Moni Naor, and Hoeteck Wee. Pebbling and proofs of work. In *Proceedings of Advances in Cryptology – CRYPTO 2005*, CRYPTO '05, pages 37–54. Springer, 2005. ISBN 978-3-540-31870-5. doi: 10.1007/11535218_3. URL https://doi.org/10.1007/11535218_3.

Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. One-time computable self-erasing functions. In *Proceedings of the 8th Theory of Cryptography Conference*, TCC '11, pages 125–143. Springer, 2011. ISBN 978-3-642-19571-6. doi: 10.1007/978-3-642-19571-6_9. URL https://doi.org/10.1007/978-3-642-19571-6_9.

Massimiliano Esposito and Christian Van den Broeck. Second law and landauer principle far from equilibrium. *Europhysics Letters*, 95(4), August 2011. ISSN 1286-4854. doi: 10.1209/0295-5075/95/40004. URL https://doi.org/10.1209/0295-5075/95/40004.

Michael P. Frank and M. Josephine Ammer. Relativized separation of reversible and irreversible space-time complexity classes, 2017. URL https://doi.org/10.48550/arXiv.1708.08480.

Edward Fredkin and Tommao Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21:219–253, April 1982. doi: 10.1007/BF01857727. URL https://doi.org/10.1007/BF01857727.

Craig Gidney. Spooky pebble games and irreversible uncomputation, August 2019. URL https://algassert.com/post/1905.

John T. Gill. Computational complexity of probabilistic turing machines. In *Proceedings of the 6th Annual ACM Symposium on Theory of Computing*, STOC '74, pages 91–95. Association for Computing Machinery, 1974. ISBN 9781450374231. doi: 10.1145/800119.803889. URL https://doi.org/10.1145/800119.803889.

András Gilyén, Yuan Su, Guang Hao Low, and Nathan Wiebe. Quantum singular value transformation and beyond: Exponential improvements for quantum matrix arithmetics. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing*, STOC '19, pages 193–204. ACM, 2019. ISBN 9781450367059. doi: 10.1145/3313276.3316366. URL https://doi.org/10.1145/3313276.3316366.

András Gilyén, Zhao Song, and Ewin Tang. An improved quantum-inspired algorithm for linear regression. *Quantum*, 6, June 2022. doi: 10.22331/q-2022-06-30-754. URL https://doi.org/10.22331/q-2022-06-30-754.

José Grimm, Loïc Pottier, and Nicole Rostaing-Schmidt. Optimal time and minimum space-time product for reversing a certain class of programs. Technical report, INRIA, 1996.

Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 212–219. ACM, 1996. ISBN 0897917855. doi: 10.1145/237814.237866. URL https://doi.org/10.1145/237814.237866.

Yassine Hamoudi and Frédéric Magniez. Quantum time-space tradeoff for finding multiple collision pairs. In *Proceedings of the 16th Conference on Theory of Quantum Computation, Communication and Cryptography*, volume 197 of *TQC '21*, pages 1:1–1:21. LIPIcs, 2021. ISBN 978-3-95977-198-6. doi: 10.4230/LIPIcs.TQC.2021.1. URL https://doi.org/10.4230/LIPIcs.TQC.2021.1.

Yassine Hamoudi, Qipeng Liu, and Makrand Sinha. The nisq complexity of collision finding. In *Proceedings of Advances in Cryptology – EUROCRYPT 2024*, EUROCRYPT '24, pages 3–32. Springer, 2024. ISBN 978-3-031-58737-5. doi: 10.1007/978-3-031-58737-5_1. URL https://doi.org/10.1007/978-3-031-58737-5_1.

Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103(15), October 2009. ISSN 1079-7114. doi: 10.1103/physrevlett.103.150502. URL https://doi.org/10.1103/physrevlett.103.150502.

Hiroshi Hasegawa, Junichi Ishikawa, Kazuma Takara, and Dean J. Driebe. Generalization of the second law for a nonequilibrium initial state. *Physics Letters A*, 374(8):1001–1004, February 2010. ISSN 0375-9601. doi: 10.1016/j.physleta.2009.12.042. URL https://doi.org/10.1016/j.physleta.2009.12.042.

John Hopcroft, Wolfgang Paul, and Leslie Valiant. On time versus space. *Journal of the ACM*, 24(2):332–337, April 1977. ISSN 0004-5411. doi: 10.1145/322003.322015. URL https://doi.org/10.1145/322003.322015.

Joseph F. JáJá and Janos Simon. Space efficient algorithms for some graph theoretical problems. *Acta Informatica*, 17:411–423, October 1982. doi: 10.1007/BF00264160. URL https://doi.org/10.1007/BF00264160.

Nikolaos P. Karvelas and Aggelos Kiayias. Efficient proofs of secure erasure. In *Proceedings of Security and Cryptography for Networks*, SCN '14, pages 520–537. Springer, 2014. ISBN 978-3-319-10879-7. doi: 10.1007/978-3-319-10879-7_30. URL https://doi.org/10.1007/978-3-319-10879-7_30.

Kyozi Kawasaki. Diffusion constants near the critical point for time-dependent ising models. i. *Physical Review*, 145(1):224–230, May 1966. ISSN 0031-899X. doi: 10.1103/PhysRev.145.224. URL https://doi.org/10.1103/PhysRev.145.224.

Aleksei Y. Kitaev. Quantum computations: Algorithms and error correction. *Russian Mathematical Surveys*, 52(6):1191–1249, December 1997. doi: 10.1070/RM1997v052n06ABEH002155. URL https://doi.org/10.1070/RM1997v052n06ABEH002155.

Hartmut Klauck, Robert Špalek, and Ronald de Wolf. Quantum and classical strong direct product theorems and optimal time-space tradeoffs. *SIAM Journal on Computing*, 36(5):1472–1493, February 2007. ISSN 1095-7111. doi: 10.1137/05063235x. URL https://doi.org/10.1137/05063235X.

Artemy Kolchinsky and David H Wolpert. Dependence of dissipation on the initial distribution over states. *Journal of Statistical Mechanics: Theory and Experiment*, 2017(8), August 2017. ISSN 1742-5468. doi: 10.1088/1742-5468/aa7ee1. URL https://doi.org/10.1088/1742-5468/aa7ee1.

Balagopal Komarath, Jayalal Sarma, and Saurabh Sawlani. Reversible pebble game on trees. In *Proceedings of Computing and Combinatorics*, COCOON '15, pages 83–94. Springer, 2015. ISBN 978-3-319-21398-9. doi: 10.1007/978-3-319-21398-9_7. URL https://doi.org/10.1007/978-3-319-21398-9_7.

Niels Kornerup, Jonathan Sadun, and David Soloveichik. Tight bounds on the spooky pebble game: Recycling qubits with measurements. *Quantum*, 9, February 2025. ISSN 2521-327X. doi: 10.22331/q-2025-02-18-1636. URL https://doi.org/10.22331/q-2025-02-18-1636.

Richard Král'ovič. Time and space complexity of reversible pebbling. In *Proceedings of the 28th Conference on Current Trends in Theory and Practice of Informatics*, SOFSEM '01, pages 292–303. Springer, 2001. ISBN 978-3-540-45627-8. doi: 10.1007/3-540-45627-9_26. URL https://doi.org/10.1007/3-540-45627-9_26.

R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3):183–191, July 1961. ISSN 0018-8646. doi: 10.1147/rd.53.0183. URL https://doi.org/10.1147/rd.53.0183.

Klaus-Jörn Lange, Pierre McKenzie, and Alain Tapp. Reversible space equals deterministic space. *Journal of Computer and System Sciences*, 60(2):354–367, April 2000. ISSN 0022-0000. doi: 10.1006/jcss.1999.1672. URL https://doi.org/10.1006/jcss.1999.1672.

Thomas Lengauer and Robert E. Tarjan. Asymptotically tight bounds on time-space trade-offs in a pebble game. *Journal of the ACM*, 29(4):1087–1130, October 1982. ISSN 0004-5411. doi: 10.1145/322344.322354. URL https://doi.org/10.1145/322344.322354.

Thomas Lengauer and Robert Endre Tarjan. Upper and lower bounds on time-space tradeoffs. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*, STOC '79, pages 262–277. ACM, 1979. ISBN 9781450374385. doi: 10.1145/800135.804420. URL https://doi.org/10.1145/800135.804420.

Robert Y. Levine and Alan T. Sherman. A note on bennett's time-space tradeoff for reversible computation. *SIAM Journal on Computing*, 19(4):

673–677, August 1990. ISSN 1095-7111. doi: 10.1137/0219046. URL https://doi.org/10.1137/0219046.

Ming Li and Paul Vitanyi. Reversibility and adiabatic computation: Trading time and space for energy. *Proceedings of the Royal Society A*, 452(1947): 769–789, April 1996. ISSN 13645021. doi: 10.1098/rspa.1996.0039. URL https://doi.org/10.1098/rspa.1996.0039.

Ming Li, John Tromp, and Paul Vitányi. Reversible simulation of irreversible computation. *Physica D: Nonlinear Phenomena*, 120(1-2):168–176, September 1998. ISSN 1872-8022. doi: 10.1016/s0167-2789(98)00052-9. URL https://doi.org/10.1016/s0167-2789(98)00052-9.

Qipeng Liu and Mark Zhandry. On finding quantum multi-collisions. In *Proceedings of Advances in Cryptology – EUROCRYPT 2019*, EUROCRYPT '19, pages 189–218. Springer, 2019. ISBN 978-3-030-17658-7. doi: 10.1007/978-3-030-17659-4_7. URL https://doi.org/10.1007/978-3-030-17659-4_7.

Guang Hao Low and Isaac L. Chuang. Hamiltonian simulation by qubitization. *Quantum*, 3, July 2019. doi: 10.22331/q-2019-07-12-163. URL https://doi.org/10.22331/q-2019-07-12-163.

Alexandra Sasha Luccioni, Sylvain Viguier, and Anne-Laure Ligozat. Estimating the carbon footprint of bloom, a 176b parameter language model. *The Journal of Machine Learning Research*, 24(1), January 2023. ISSN 1532-4435.

Alessandro Luongo, Antonio Michele Miti, Varun Narasimhachar, and Adithya Sireesh. Measurement-based uncomputation of quantum circuits for modular arithmetic, 2024. URL https://doi.org/10.48550/arXiv.2407.20167.

Yishay Mansour, Noam Nisan, and Prasoon Tiwari. The computational complexity of universal hashing. *Theoretical Computer Science*, 107(1):121–133,

January 1993. ISSN 1879-2294. doi: 10.1016/0304-3975(93)90257-T. URL https://doi.org/10.1016/0304-3975(93)90257-T.

Daniel D. McCracken and William S. Dorn. *Numerical Methods and FORTRAN Programming: With Applications in Engineering and Science*. Wiley, 1964. ISBN 9780471582854.

Ralph C. Merkle. A digital signature based on a conventional encryption function. In *Proceedings of Advances in Cryptology – CRYPTO 1987*, CRYPTO '87, pages 369–378. Springer, 1988. ISBN 978-3-540-48184-3. doi: 10.1007/3-540-48184-2_32. URL https://doi.org/10.1007/3-540-48184-2_32.

Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, June 1953. ISSN 0021-9606. doi: 10.1063/1.1699114. URL https://doi.org/10.1063/1.1699114.

Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, March 2009. URL https://bitcoin.org/bitcoin.pdf.

Lee A. Newberg. Memory-efficient dynamic programming backtrace and pairwise local sequence alignment. *Bioinformatics*, 24(16):1772–1778, August 2008. ISSN 1367-4811. doi: 10.1093/bioinformatics/btn308. URL https://doi.org/10.1093/bioinformatics/btn308.

Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010. ISBN 9781107002173. doi: 10.1017/CBO9780511976667. URL https://doi.org/10.1017/CBO9780511976667.

John D Norton. Brownian computation is thermodynamically irreversible. *Foundations of Physics*, 43:1384–1410, October 2013.

Mark Oskin, Frederic T. Chong, and Isaac L. Chuang. A practical architecture for reliable quantum computers. *Computer*, 35(1):79–87, January 2002. ISSN 1460-2067. doi: 10.1109/2.976922. URL https://doi.org/10.1109/2.976922.

Thomas E. Ouldridge and David H. Wolpert. Thermodynamics of deterministic finite automata operating locally and periodically. *New Journal of Physics*, 25 (12), December 2023. ISSN 1367-2630. doi: 10.1088/1367-2630/ad1070. URL https://doi.org/10.1088/1367-2630/ad1070.

Jeremy A. Owen, Artemy Kolchinsky, and David H. Wolpert. Number of hidden states needed to physically implement a given conditional distribution. *New Journal of Physics*, 21(1), January 2019. ISSN 1367-2630. doi: 10.1088/1367-2630/aaf81d. URL https://doi.org/10.1088/1367-2630/aaf81d.

Anouk Paradis, Benjamin Bichsel, Samuel Steffen, and Martin Vechev. Unqomp: Synthesizing uncomputation in quantum circuits. In *Proceedings of the 42nd International Conference on Programming Language Design and Implementation*, PLDI '21, pages 222–236. ACM, 2021. ISBN 9781450383912. doi: 10.1145/3453483.3454040. URL https://doi.org/10.1145/3453483.3454040.

Juan M. R. Parrondo, Jordan M. Horowitz, and Takahiro Sagawa. Thermodynamics of information. *Nature Physics*, 11(2):131–139, February 2015. ISSN 1745-2481. doi: 10.1038/nphys3230. URL https://doi.org/10.1038/nphys3230.

Wolfgang J. Paul and Robert Endre Tarjan. Time-space trade-offs in a pebble game. *Acta Informatica*, 10(2):111–115, June 1978. ISSN 1432-0525. doi: 10.1007/BF00289150. URL https://doi.org/10.1007/BF00289150.

Wolfgang J. Paul, Robert Endre Tarjan, and James R. Celoni. Space bounds for a game on graphs. In *Proceedings of the 8th Annual ACM Symposium on Theory of*

Computing, STOC '76, pages 149–160. ACM, 1976. ISBN 9781450374149. doi: 10.1145/800113.803643. URL https://doi.org/10.1145/800113.803643.

Anna N. Pearson, Yelena Guryanova, Paul Erker, Edward A. Laird, Andrew D. Briggs, Marcus Huber, and Natalia Ares. Measuring the thermodynamic cost of timekeeping. *Physical Review X*, 11, May 2021. ISSN 2160-3308. doi: 10.1103/PhysRevX.11.021029. URL https://doi.org/10.1103/PhysRevX.11.021029.

Simon Perdrix and Philippe Jorrand. Classically-controlled quantum computation. *Electronic Notes in Theoretical Computer Science*, 135(3):119–128, March 2006. ISSN 1571-0661. doi: 10.1016/j.entcs.2005.09.026. URL https://doi.org/10.1016/j.entcs.2005.09.026.

Nicholas Pippenger. A time-space trade-off. *Journal of the ACM*, 25(3): 509–515, July 1978. ISSN 0004-5411. doi: 10.1145/322077.322091. URL https://doi.org/10.1145/322077.322091.

Nicholas Pippenger. On simultaneous resource bounds. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, SFCS '79, pages 307–311. IEEE, 1979. doi: 10.1109/SFCS.1979.29. URL https://doi.org/10.1109/SFCS.1979.29.

Arend-Jan Quist and Alfons Laarman. Optimizing quantum space using spooky pebble games. In *Proceedings of the 15th International Conference on Reversible Computation*, RC '23, pages 134–149. Springer, 2023. ISBN 978-3-031-38100-3. doi: 10.1007/978-3-031-38100-3_10. URL https://doi.org/10.1007/978-3-031-38100-3_10.

Arend-Jan Quist and Alfons Laarman. Trade-offs between classical and quantum space using spooky pebbling, 2024. URL https://doi.org/10.48550/arXiv.2401.10579.

Ling Ren and Srinivas Devadas. Proof of space from stacked expanders. In *Proceedings of the 14th International Conference on Theory of Cryptography*, volume 9985 of *TCC '16*, pages 262–285. Springer, 2016. ISBN 9783662536407. doi: 10.1007/978-3-662-53641-4_11. URL https://doi.org/10.1007/978-3-662-53641-4_11.

Albert Reuther, Jeremy Kepner, Chansup Byun, Siddharth Samsi, William Arcand, David Bestor, Bill Bergeron, Vijay Gadepally, Michael Houle, Matthew Hubbell, Michael Jones, Anna Klein, Lauren Milechin, Julia Mullen, Andrew Prout, Antonio Rosa, Charles Yee, and Peter Michaleas. Interactive supercomputing on 40,000 cores for machine learning and data analysis. In *Proceedings of the 2018 IEEE High Performance Extreme Computing Conference*, HPEC '18, pages 1–6. IEEE, 2018. doi: 10.1109/HPEC.2018.8547629. URL https://doi.org/10.1109/HPEC.2018.8547629.

Eleanor Rieffel and Wolfgang Polak. *Quantum Computing: A Gentle Introduction*. The MIT Press, 2011. ISBN 9780262015066.

Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978. ISSN 0001-0782. doi: 10.1145/359340.359342. URL http://doi.org/10.1145/359340.359342.

Ansis Rosmanis. Tight bounds for inverting permutations via compressed oracle arguments, 2022. URL https://doi.org/10.48550/arXiv.2103.08975.

Mehdi Saeedi and Igor L. Markov. Synthesis and optimization of reversible circuits—a survey. *ACM Computing Surveys*, 45(2), March 2013. ISSN 1557-7341. doi: 10.1145/2431211.2431220. URL http://doi.org/10.1145/2431211.2431220.

Takahiro Sagawa. Thermodynamic and logical reversibilities revisited. *Journal of Statistical Mechanics: Theory and Experiment*, 2014(3), March 2014. ISSN 1742-5468. doi: 10.1088/1742-5468/2014/03/P03025. URL https://doi.org/10.1088/1742-5468/2014/03/P03025.

Siddharth Samsi, Dan Zhao, Joseph McDonald, Baolin Li, Adam Michaleas, Michael Jones, William Bergeron, Jeremy Kepner, Devesh Tiwari, and Vijay Gadepally. From words to watts: Benchmarking the energy costs of large language model inference. In *Proceedings of the 2023 IEEE High Performance Extreme Computing Conference*, HPEC '23, pages 1–9, December 2023. doi: 10.1109/HPEC58863.2023.10363447. URL https://doi.org/10.1109/HPEC58863.2023.10363447.

John E. Savage and Sowmitri Swamy. Space-time trade-offs on the fft algorithm. *IEEE Transactions on Information Theory*, 24(5):563–568, September 1978. ISSN 1557-9654. doi: 10.1109/TIT.1978.1055938. URL https://doi.org/10.1109/TIT.1978.1055938.

Ravi Sethi. Complete register allocation problems. In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing*, STOC '73, pages 182–195. ACM, 1973. ISBN 9781450374309. doi: 10.1145/800125.804049. URL https://doi.org/10.1145/800125.804049.

Arman Shehabi, Sarah J. Smith, Alex Hubbard, Alex Newkirk, Nuoa Lei, Md Abu Bakar Siddik, Billie Holecek, Jonathan Koomey, Eric Masanet, and Dale Sartor. 2024 united states data center energy usage report. Technical report, Lawrence Berkeley National Laboratory, 2024.

Alexander A. Sherstov. Strong direct product theorems for quantum communication and query complexity. *SIAM Journal on Computing*, 41(5):1122–1165, September 2012. ISSN 1095-7111. doi: 10.1137/110842661. URL https://doi.org/10.1137/110842661.

Yaoyun Shi. Both toffoli and controlled-not need little help to do universal quantum computing. *Quantum Information and Computation*, 3(1):84–92, January 2003. ISSN 1533-7146. doi: 10.26421/QIC3.1-7. URL https://doi.org/10.26421/QIC3.1-7.

Daniel Simon. On the power of quantum computation. *SIAM Journal on Computing*, 26(5):1474–1483, October 1997. doi: 10.1137/S0097539796298637. URL https://doi.org/10.1137/S0097539796298637.

Robert Spalek. The multiplicative quantum adversary. In *Proceedings of the 23rd Annual IEEE Conference on Computational Complexity*, CCC '08, pages 237–248. IEEE, 2008. doi: 10.1109/CCC.2008.9. URL https://doi.org/10.1109/CCC.2008.9.

Robert Spalek and Mario Szegedy. All quantum adversary methods are equivalent. *Theory of Computing*, 2, 2006. ISSN 1557-2862. doi: 10.4086/TOC.2006.V002A001. URL https://doi.org/10.4086/toc.2006.v002a001.

Philipp Strasberg, Javier Cerrillo, Gernot Schaller, and Tobias Brandes. Thermodynamics of stochastic turing machines. *Physical Review E*, 92(4), October 2015. ISSN 2470-0053. doi: 10.1103/PhysRevE.92.042104. URL https://doi.org/10.1103/PhysRevE.92.042104.

Ewin Tang. A quantum-inspired classical algorithm for recommendation systems. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing*, STOC '19, pages 217–228. ACM, 2019. doi: 10.1145/3313276.3316310. URL https://doi.org/10.1145/3313276.3316310.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Martin Tompa. Time-space tradeoffs for computing functions, using connectivity properties of their circuits. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, pages 196–204. ACM, 1978. ISBN 9781450374378. doi: 10.1145/800133.804348. URL https://doi.org/10.1145/800133.804348.

Martin Tompa. Time-space tradeoffs for computing functions, using connectivity properties of their circuits. *Journal of Computer and System Sciences*, 20(2): 118–132, 1980. ISSN 0022-0000. doi: 10.1016/0022-0000(80)90056-2. URL https://doi.org/10.1016/0022-0000(80)90056-2.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023. URL https://doi.org/10.48550/arXiv.2302.13971.

Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979. ISSN 1095-7111. doi: 10.1137/0208032. URL https://doi.org/10.1137/0208032.

Raymond Wheeler and Richard Hughey. Optimizing reduced-space sequence analysis. *Bioinformatics*, 16(12):1082–1090, 2000. ISSN 1367-4811. doi: 10.1093/bioinformatics/16.12.1082. URL https://doi.org/10.1093/bioinformatics/16.12.1082.

David H Wolpert. The stochastic thermodynamics of computation. *Journal of Physics A: Mathematical and Theoretical*, 52(19), 2019. ISSN 1751-8121. doi: 10.1088/1751-8121/ab0850. URL https://doi.org/10.1088/1751-8121/ab0850.

David H Wolpert and Artemy Kolchinsky. Thermodynamics of computing with circuits. *New Journal of Physics*, 22(6), June 2020. ISSN 1367-2630. doi: 10.1088/1367-2630/ab82b8. URL https://doi.org/10.1088/1367-2630/ab82b8.

David H. Wolpert, Jan Korbel, Christopher W. Lynn, Farita Tasnim, Joshua A. Grochow, Gülce Kardeş, James B. Aimone, Vijay Balasubramanian, Eric De Giuli, David Doty, Nahuel Freitas, Matteo Marsili, Thomas E. Ouldridge, Andréa W. Richa, Paul Riechers, Édgar Roldán, Brenda Rubenstein, Zoltan Toroczkai, and Joseph Paradiso. Is stochastic thermodynamics the key to understanding the energy costs of computation? *Proceedings of the National Academy of Sciences*, 121(45), 2024. ISSN 1091-6490. doi: 10.1073/pnas.2321112121. URL https://doi.org/10.1073/pnas.2321112121.

Abhishek Yadav, Francesco Caravelli, and David Wolpert. Mismatch cost of computing: From circuits to algorithms, 2024. URL https://doi.org/10.48550/arXiv.2411.16088.

Andrew C. Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 222–227. IEEE, 1977. doi: 10.1109/sfcs.1977.24. URL https://doi.org/10.1109/SFCS.1977.24.

Yaacov Yesha. Time-space trade-offs for matrix multiplication and the discrete fourier transform on any general sequential random-access computer. *Journal of Computer and System Sciences*, 29(2):183–197, 1984. ISSN 0022-0000. doi: 10.1016/0022-0000(84)90029-1. URL https://doi.org/10.1016/0022-0000(84)90029-1.

Justin Yirka. Even quantum advice is unlikely to solve pp, 2024. URL https://doi.org/10.48550/arXiv.2403.09994.

Mark Zhandry. How to record quantum queries, and applications to quantum indifferentiability. In *Proceedings of Advances in Cryptology – CRYPTO 2019*, CRYPTO '19, pages 239–268. Springer, 2019. ISBN 978-3-030-26951-7.