ABSTRACT

Title of Dissertation: QUANTUM ROUTING FOR ARCHITECTURE-RESPECTING CIRCUIT TRANSFORMATIONS

Eddie Schoute
Doctor of Philosophy, 2021

Dissertation Directed by: Professor Andrew M. Childs
Department of Computer Science

The connectivity between qubits is one of the many design aspects that go into building a quantum computer. Better connectivity makes it easier to perform arbitrary interacting operations in quantum algorithms, but it may also come with additional noise and may be costly to manufacture. Therefore, many proposals for scalable quantum computer architectures sacrifice connectivity to obtain better modularity and suppress noise. This poses a challenge to running quantum algorithms because simulating missing connectivity can come with significant overhead.

A natural stepping stone is permuting qubits on the architecture, a task we call *quantum routing*. We first give a rigorous analysis for the special case of *classical routing* using SWAP gates. Then we present a time-independent Hamiltonian protocol that reverses a chain of qubits asymptotically 3 times faster than classical routing. Using this protocol, we exhibit the first separation between classical and quantum routing time. This leads us to lower bound unitary quantum routing to be inversely proportional to the vertex expansion of the architecture graph in a gate model and inversely proportional to the edge expansion in a Hamiltonian evolution model. We rule out a superpolynomial separation between classical and quantum routing for architectures with poor expansion properties.

We then show how to use routing to transform quantum circuits such that their interactions respect the architecture constraints while attempting to minimize the depth overhead. We benchmark the performance of our circuit transformations on grid and modular architectures.

Finally, we give a circuit transformation for fault-tolerant quantum computation in the surface code. We use a construction for parallel long-range operations in constant logical time that allows us to avoid the need for routing altogether. Our benchmarks show improved performance over our previous circuit transformations using classical routing.

QUANTUM ROUTING FOR ARCHITECTURE-RESPECTING CIRCUIT TRANSFORMATIONS

by

Eddie Schoute

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2021

Advisory Committee:
Professor Andrew M. Childs, Chair/Advisor
Professor Mohammad Hafezi, Dean's Representative
Professor Alexey V. Gorshkov
Professor Michael Hicks
Professor Yi-Kai Liu

Acknowledgements

The supervision of Andrew Childs really helped lift the research in this dissertation to greater heights. It is thanks in part to Andrew that many of the more applied results have a well-founded theoretical basis. (What came first is, by now, a chicken-and-egg problem.) Besides being a great researcher, I have found Andrew's management to be very supportive and I am grateful for having been able to learn from it.

I would like to thank the members of my dissertation committee for their service, many of whom I've interacted with during my studies. Half of the chapters in this dissertation I co-authored with Alexey Gorshkov. Somewhere in the middle of Alexey coming up with a protocol or some physical intuition and me trying to make sense of things did we find our results. And I would like to thank Yi-Kai Liu for mentoring me during my first two years at QuICS and getting me started in quantum information research.

The students and postdocs at QuICS were invaluable, be it through our shared study of quantum problems or board games. In particular, I would like to thank Aniruddha Bapat for working with me on so many of my routing papers. His insights helped push these projects forward on multiple occasions. In addition, it was really fun to work with Andrew Guo and Minh Tran. By the time of my defense, our interests seem to have become more entangled.

I would also like to thank the wonderful people I met during my internships. Ali Javadi-Abhari, thank you for the short but sweet internship at IBM, I was able to learn

a lot on topics that I didn't know much about yet. Vadym Kliuchnikov, thank you for your guidance during my Microsoft internship on fault-tolerant quantum computation using the surface code. Also involved in this project were Alexander Vaschillo, Dmitry Vasilevsky, and Michael Beverland, who took the time to explain some of the more fundamental concepts of surface codes to me.

Finally, a thank you to my direct family and grandparents.

Table of contents

Chapter 1

Introduction

Quantum algorithms promise to speed-up many useful problems in computer science, physics, and chemistry. To achieve the feats promised by quantum algorithms, many academic and industry players are currently rushing to build a scalable quantum computer [Aru+19; Jur+21; Pin+21; Bom+21] with many qubits, low enough error rates and good connectivity.

In addition to experimental developments, we find that running a quantum algorithm on a quantum computer requires many levels of translation, or compilation. At the lowest level, quantum computers can only manipulate the physical system by its natural dynamics, which is not generally described by a circuit model. A first step is to describe algorithms in a convenient, high-level circuit description. The circuit is then broken down into some finite universal gate set, describing elementary operations on a theoretical quantum computer. Since any universal gate set may be translated into another universal gate set with only polylogarithmic overhead [Kit97], it is easy to translate the chosen gate set into one that is more convenient for the quantum computer at hand.

But even an elementary gate set can still require seemingly unphyisical operations. Feynman [Fey82], in his famous keynote proposing quantum computing, postulated

"Can physics be simulated by a universal computer? I would like to have the elements of this computer *locally interconnected*." In the keynote, he required an efficient simulation of physics at some point in space-time to depend only on systems in a local region. A circuit described in an elementary gate set may still perform highly non-local operations, thus we should not expect that this can be efficiently implemented on a quantum computer.

Contemporary proposals for scalable quantum computers indeed constrain which qubits can interact and employ modularity, building a larger quantum system by connecting many smaller quantum computers. For example, superconducting qubits are fabricated on a plane and coupled through microwave resonators [Ste+11]. Prominent examples form a grid-like structure of qubits in their connectivity [Aru+19; IBM21]. One way to scale up superconducting quantum computers further is by constructing shielded modules of smaller quantum computers and connecting them through superconducting transmission lines [Bre+16]. Each module can be connected to other modules arbitrarily.

Ion traps are another paradigm for constructing a quantum computer and allow arbitrary interactions but are limited in scaling to about 100 qubits [MK13]. By interconnecting multiple ion trap modules, we can construct a scalable quantum computer. For example, in a quantum charge-coupled device (QCCD), ions are physically moved between traps using junctions [KMW02]. This also results in a grid-like connectivity between modules [Mur+20; Web+20]. An alternative is to entangle designated communication qubits in modules using photons sent through a reconfigurable optical switch [MK13; Mon+14], which allows us to couple any two arbitrary modules.

Translating elementary circuits to respect interaction constraints can cause significant overhead. With $L \times L$ grid connectivity, a circuit of depth $\Omega(L)$ is required to unitarily implement the longest-range operations. More generally, the overhead scales

with the diameter of the graph representing the architecture connectivity.

More precisely, we consider an abstract representation of the interaction constraints in the form of a connected simple graph, which we call the *architecture graph*[1], $G_{\mathcal{A}}$. The qubits of the quantum computer are modeled as the vertices of $G_{\mathcal{A}}$, which we denote by $V(G_{\mathcal{A}})$. We will, therefore, interchangeably use the terms vertices and qubits where it is clear from the context what we mean. We also let $n := |V(G_{\mathcal{A}})|$. In this dissertation, we only consider two-qubit interactions, so the set of qubit pairs that are allowed to interact define the edges of $G_{\mathcal{A}}$, which we denote by $E(G_{\mathcal{A}})$.

Let us define an *architecture-respecting evolution* of our system to be, at all times $t \geq 0$, described by a time-dependent 2-local Hamiltonian $H(t)$ respecting interactions constraints. We say that $H(t)$ respects the interactions of the architecture if it can be written as a sum

$$H(t) = \sum_{e \in E(G_{\mathcal{A}})} H_e(t) \tag{1.1}$$

of bounded-norm Hermitian operators $H_e(t)$ supported only on qubits associated with the allowed interaction e. Such Hamiltonians are universal for quantum computation since we can implement arbitrary single-qubit gates and arbitrary two-qubit gates between qubits that are allowed to interact.

In the circuit model, a circuit in a gate set of at most two-qubit interactions respects the interaction constraints if there is a corresponding two-local Hamiltonian evolution (given by the matrix logarithm) that is an architecture-respecting evolution. This is the case if, for all its two-qubit gates acting on $q_1, q_2 \in V(G_{\mathcal{A}})$, we have that $(q_1, q_2) \in E(G_{\mathcal{A}})$. We call such circuits *architecture-respecting circuits* and consider them a special case of architecture-respecting evolutions.

We fix a time scale by making the normalization of the interactions explicit. We allow arbitrarily fast single-qubit operations in our model, a common assump-

[1] In the electronic version of this dissertation, some definitions and references are highlighted, and references hyperlink to the definition. In all versions, there is also an index to look up terms, symbols, and abbreviations.

tion [VHC02; Ben+02] that is well-motivated by the practical ease of implementing single-qubit operations. Now, up to single-qubit unitaries, we can write $H_e(t)$ in *canonical form* [Ben+02]

$$K_e(t) \coloneqq \sum_{j \in \{x,y,z\}} \mu_j \sigma_j \otimes \sigma_j, \tag{1.2}$$

where σ_x, σ_y, and σ_z are the Pauli matrices and $\mu_x \geq \mu_y \geq |\mu_z| \geq 0$. Then we impose a *normalization condition* that the spectral norm $\|K_e(t)\| \leq 1$ for all $e \in E(G_{\mathcal{A}})$ at all times. In this setting, a SWAP can be implemented by, e.g., the normalized canonical Hamiltonian

$$\frac{1}{3}\left(\sigma_x \otimes \sigma_x + \sigma_y \otimes \sigma_y + \sigma_z \otimes \sigma_z\right) \tag{1.3}$$

and local operations in time $3\pi/4$. This is the minimum time for a SWAP [VHC02].

The primary motivation of this dissertation is to find efficient *architecture-respecting circuit transformations*, which transform any given circuit $\mathcal{C}$ in a gate set of single and two-qubit gates into an equivalent architecture-respecting evolution up to an initial mapping of circuit qubits to $V(G_{\mathcal{A}})$.

We give a non-comprehensive overview of the relevant background of this dissertation, and then give an overview of the contributions and chapters at the end of this chapter.

1.1 Quantum routing

Operations not respecting the connectivity can be seen as a form of long-range quantum communication. Fast protocols for transferring quantum information are likely to play an important role in scalable quantum computers [DiV00], distributed quantum computation [Kim08], and are useful in formulating architecture-respecting circuit transformations. Unfortunately, there are limits on how fast a unitary evolution of a

(non-relativistic) quantum system can transfer quantum information. In systems with with nearest-neighbor interactions, quantum information cannot travel faster than some constant velocity [LR72]. This holds even when we include sufficiently weak long-range interactions in higher-dimensional systems [Fos+15].

Let us consider the more general task of implementing arbitrary permutations of qubits, which we call *(quantum) routing*. In architecture-respecting circuit transformations, we can use routing to turn any non-local gate into a local gate by first routing the involved qubits to be adjacent. We define quantum routing as follows:

Definition 1.1 (Quantum routing)**.** Given a permutation of qubits $\pi\colon V(G_{\mathcal{A}}) \to V(G_{\mathcal{A}})$, find an architecture-respecting evolution that implements the mapping

$$|\psi_1\rangle|\psi_2\rangle \dots |\psi_n\rangle \mapsto |\psi_{\pi^{-1}(1)}\rangle \dots |\psi_{\pi^{-1}(n)}\rangle \tag{1.4}$$

for any pure states $|\psi_i\rangle$, $i \in [n]$, and $[n] \coloneqq \{1, \dots, n\}$.

We will mostly consider minimizing the circuit depth and evolution time of quantum routing. Let us focus on the depth for now. One type of system where minimizing the depth of routing is useful, is on noisy intermediate-scale quantum (NISQ) computers [Pre18]. Successful execution of algorithms on NISQ computers depends on the infidelity introduced by noise in the computer and the operations. The depth of a circuit bounds the infidelity that can be introduced by noisy operations so can function as a suitable proxy for the infidelity. This already has immediate applications in, e.g., demonstrations of quantum volume [Cro+19]. Optimizing for the fidelity directly is more difficult. This optimization depends on the underlying hardware and varying environmental conditions throughout the day [Mur+19].

For most of this dissertation, we will focus on unitary routing, that is, routing without *local operations and classical communication* (LOCC). Adding ancilla resources for LOCC requires more resources of the quantum computer, and intermediate measure-

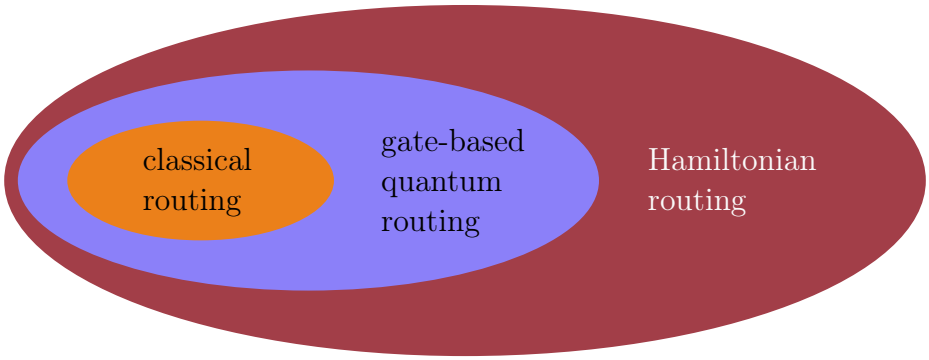


Figure 1.1: The containment of routing models that are considered in this work with regards to their routing time. Most prior work only considers using SWAP gates for routing, which we call classical routing. We attempt to see what additional power using genuinely quantum operations gives us in performing routing. We consider increasingly more powerful unitary quantum computation models: allowing architecture-respecting circuits in gate-based quantum routing, and normalized architecture-respecting evolution in Hamiltonian routing.

ments can be challenging for modern-day quantum computers [IBM21; Aru+19]. We introduce and compare models of unitary routing of increasing strength: (i) *Classical routing* only uses SWAP gates for routing. We use the term "classical" because SWAP gates cannot entangle a separable state. It is a special case of (ii) *gate-based quantum routing*, which considers unitary architecture-respecting circuits to implement routing. And (iii) *Hamiltonian (quantum) routing* considers unitary architecture-respecting evolutions for routing. We are mostly interested in minimizing the depth of routing in classical and gate-based quantum routing models, and minimizing the time for routing in the Hamiltonian routing model. By normalizing the time for a SWAP gate to 1, any gate-based quantum routing protocol of depth k also gives a Hamiltonian routing protocol in time at most k [VHC02]. This gives us a hierarchy of routing models, depicted in Figure 1.1.

1.1.1 Classical routing

Classical routing is perhaps the most natural form of routing and ubiquitous in the literature of architecture-respecting circuit transformations. A simple greedy algorithm

for classical routing performs SWAPs along a shortest path. Since SWAPs (necessarily) move information in two directions, a SWAP may move some qubits away from their destination. To work around this, early algorithms follow the greedy algorithm and move qubits (or pairs of qubits) to their destination one at a time [Met+06; LSJ15]. This does not use the potential parallelism of operations, so later algorithms assign multiple disjoint exclusive regions, such as rectangles on a grid, where only one qubit is allowed to be moved [Wil+16; Mur+19]. Modern quantum programming software greedily performs randomized local optimization [ANI+21], performing parallel SWAPs along edges that monotonically reduce the total distance of all qubits to their destinations.

More structured approaches to classical routing are based on *sorting networks* [Knu98]. A sorting network is a fixed classical circuit of *comparators*, which order two inputs, that can sort any sequence of comparable inputs. We can perform routing by preparing a labelling that, when sorted, implements the permutation. The odd-even sorting network then gives a particularly simple algorithm on path architecture graphs [KMS07], resulting in a SWAP circuit of depth at most n that can implement any permutation of qubits. Beals et al. [Bea+13] later showed how sorting networks for certain families of graphs can be used to give classical routing algorithms of the same depth[2], such as 2D grid graphs in depth $O(\sqrt{n})$ and hypercubes in depth $O(\log n)$ [AKS83].

Unfortunately, we are not aware of significant results in the more general setting of gate-based quantum routing.

[2]Beals et al. [Bea+13] show a stronger result: How to perform routing of superpositions of permutations, i.e., instead of a classical permutation $\pi\colon V(G_{\mathcal{A}}) \to V(G_{\mathcal{A}})$, we are given an input state $\sum_\pi |\pi\rangle$ which is a superposition of permutations π. One possible application is in quantum parallel RAM for implementing quantum oracle lookups. A key component of their construction is a quantum comparator, a reversible comparator. A quantum comparator requires one ancilla and a more complicated construction than a SWAP gate. If the given permutation is classical however, the comparator straightforwardly simplifies to a classically controlled SWAP gate.

1.1.2 Hamiltonian routing

A task related to Hamiltonian routing that has received considerable attention is the *state transfer* of a qubit across many ancilla in a spin chain with nearest-neighbor interactions (see, e.g., the overview [Bos07]). More precisely, an initial state $|\psi\rangle$ is transferred from site 1 to site n in a chain of n qubits initialized in $|\Psi(0)\rangle = |\psi\rangle|0\rangle^{n-1}$. We consider an evolution of the state $|\Psi(t)\rangle = \exp(-iHt)|\Psi(0)\rangle$ for time $t \geq 0$ by the Hamiltonian

$$H = \sum_{k=1}^{n} J_k^x \sigma_x^k \sigma_x^{k+1} + J_k^y \sigma_y^k \sigma_y^{k+1} + J_k^z \sigma_z^k \sigma_z^{k+1} + \sum_{l=1}^{n} h_l \sigma_z^l, \tag{1.5}$$

where σ_x^k, σ_y^k, σ_z^k are the Pauli matrices acting on qubit k, and $J_k^x, J_k^y, J_k^z, h_l \in \mathbb{R}$ are the coupling and magnetic field strengths. We now wish to maximize the fidelity $F(t)$ of the output state at site n with the input $|\psi\rangle$, i.e.

$$F(t) = \langle\psi| \operatorname{Tr}_{[n-1]}(|\Psi(t)\rangle\langle\Psi(t)|)|\psi\rangle, \tag{1.6}$$

where we trace out qubits $[n-1]$. Bose [Bos03] first showed that if $J_k^x = J_k^y = J_k^z = J$ and $h_k = h$ for all k, then a numerical optimization can find a $t \propto n/J$ that maximizes $F(t)$ and has high fidelity. This show that an—albeit noisy—state transfer occurs with uniform interactions and magnetic field.

Christandl et al. [Chr+05] subsequently showed a perfect state transfer protocol where perfect fidelity $F(t) = 1$ is achieved with engineered interactions. By setting $J_k^z = h_k = 0$ for all k and $J_k^x = J_k^y = 2\sqrt{k(n-k)}/n$, a normalized perfect state transfer over chains of arbitrary length is achieved at time $t = n\pi/4$. Later protocols remove the restriction that ancilla sites 2 through n are initialized in the $|0\rangle$ state [FPK08; Yao+11], but still perturb the state stored in the ancilla.

We can consider the harder task of *state reversal* (or state mirroring) [Alb+04]

where there are no ancilla qubits in a spin chain. Specifically, we define state reversal as implementing the unitary

$$\mathrm{R} := \prod_{k=1}^{\lfloor \frac{n}{2} \rfloor} \mathrm{SWAP}_{k,n+1-k} \tag{1.7}$$

up to a *global phase* which is independent of the state. State reversal is a form of quantum routing restricted to the state reversal permutation and the path architecture graph. It can be seen to "mirror" the state on a 1D system of n qubits around its center and as the simultaneous state transfer of n qubits. The first state reversal protocol [Alb+04] sets $J_k^x = J_k^y = \sqrt{(k-1)(n-k)}/n$ to implement state reversal at time $t = n\pi/2$.

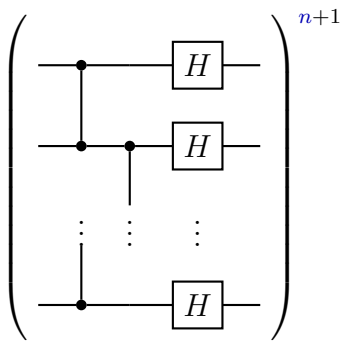
However, the protocol in [Alb+04] and other early results [KS05; Shi+05] introduce a relative phase dependent on the number of excitations M, i.e., qubits in the state $|1\rangle$. These relative phases require non-local operations to correct. For example, the protocol of [Alb+04] introduces a relative phase of $(-1)^{M(M-1)/2}$, which maps an initial state

$$|+\rangle|0\rangle^{n-2}|b\rangle \mapsto |b\rangle|0\rangle^{n-2}(Z^b|-\rangle), \tag{1.8}$$

for bit $b \in \{0,1\}$. Correcting the significant phase error Z^b requires a non-local controlled-Z (CZ) gate on qubits 1 and n. By Lieb-Robinson bounds [LR72], we know that this must take time at least linear in n in the Hamiltonian routing model.

Later state reversal protocols removed the relative phase but are time-dependent [Rau05;

FT06; KD15]. For example, Raussendorf [Rau05] shows that



$$= \mathrm{R}, \tag{1.9}$$

where we have n input qubits in an arbitrary state, and $n+1$ iterations of CZ gates then Hadamard gates. The state reversal protocols in [Rau05; FT06] were shown to be useful in implement translation-invariant universal quantum computation by, for example, modelling a quantum cellular automaton [Rau05; Ste+19; DW18; Rau+19].

Interestingly, this gives an example of the difference between time in the Hamiltonian routing model and depth in the gate-based routing model. In the gate-based model of routing, this would give a depth $2(n+1)$ circuit for implementing the state reversal permutation by absorbing the Hadamard in preceding CZ gates. (This is not the lowest-depth circuit that we know of since classical routing can implement R in depth n.) However, (1.9) corresponds to a time-dependent Hamiltonian of $n+1$ iterations of commuting ZZ interactions up to local rotations. Each such iteration takes time $\pi/4$ to implement [VHC02], giving a protocol for state reversal in time $(n+1)\pi/4$ in the Hamiltonian routing model.

1.2 Architecture-respecting circuit transformations

We now turn to related work in architecture-respecting circuit transformations. Some algorithms are well-structured and amenable to an architecture-respecting circuit transformation by hand. For example, on path architecture graphs, we know of architecture-respecting circuits for the quantum Fourier transform in depth $\Theta(n)$ [Mas07] and Shor's

algorithm in depth $O(n^2)$ [Kut06]. And on 2D grid architecture graphs, we can implement quantum adders in depth $\Theta(\sqrt{n})$ [CM11; CM12]. In the near-term, especially, hand-optimized architecture-respecting circuits may remain important [CBC21].

It is much more difficult to find architecture-respecting circuit transformations that minimize size or depth for general circuits. In fact, deciding the equivalence of a given circuit to the identity circuit is complete for the class QMA [JWB03], which contains NP. Given the complexity of the task, there are proposals for circuit transformations with runtime exponential in n based on SAT solvers [SWD11; LWD15; Ven+18] and hence can only be used for small instances. Here, we can even optimize for the fidelity directly using calibration data [Mur+19]. Recent results include the commercial t|ket⟩ compiler [Siv+20], the use of commutation rules for additional flexibility [Ito+19; Ito+20], repeated circuit transformation to reach a better fixed-point [LDX19], and the use of simulated annealing for finding suitable mappings of qubits to the architecture graph [ZLF20].

Evaluating architecture-respecting circuit transformations is a complex task and no single standard has emerged yet. Many papers use RevLib [Wil+08], a library of reversible circuits of up to 20 qubits, a size where exponential time methods still work. This is then supplemented [ZPW18; Sir+19] by larger-scale algorithms synthesized in quantum programming languages such as Quipper [Gre+13] and Scaffold [Jav+12]. Tan and Cong [TC21b] also proposed a benchmark of circuits with a known optimal solution given by a hidden intial qubit mapping.

1.3 Fault-tolerant quantum computation

For scalable quantum computing, we will need a fault-tolerant encoding of quantum information to suppress noise. The surface code [Kit03; BK98] is the implementation of *fault-tolerant quantum computation* (FTQC) that we will consider in this dissertation.

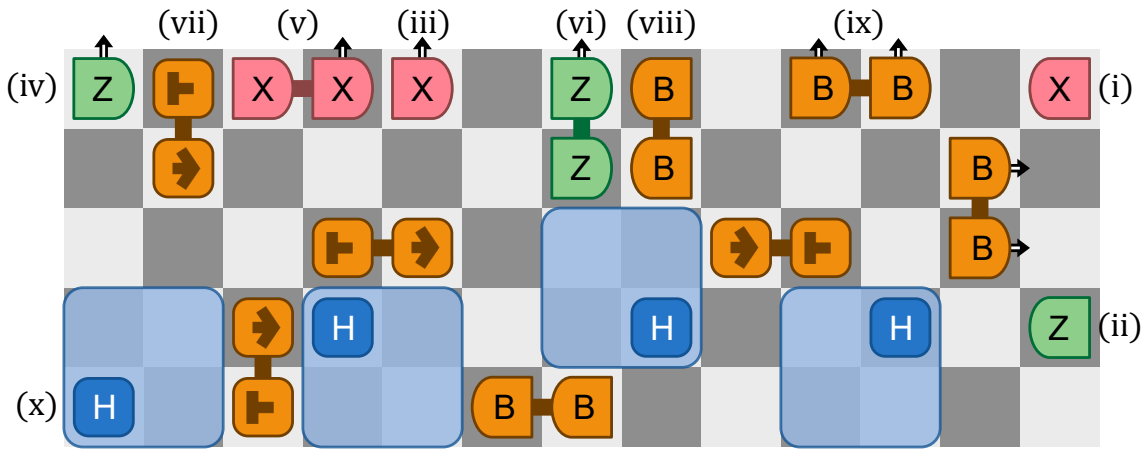


Figure 1.2: Logical qubits (light and dark gray patches) encoded in the surface code form a 2D grid. The logical operations can be applied on any lattice translations of those shown. Their times in units of surface code logical time steps are as follows. *0 Logical time steps:* Single-qubit preparation in the X basis (i), and the Z basis (ii). Single-qubit measurement in the X basis (iii), and the Z basis (iv) take 0 steps. *1 Logical time step:* Two-qubit measurement of XX (v) and ZZ (vi). A move of a logical qubit from one patch to an unused patch (vii). Two-qubit preparation (viii) and destructive measurement (ix) in the Bell basis. *3 Logical time steps:* A Hadamard gate, which uses three ancilla patches (x).

The surface code can be implemented using operations local in a 2D grid, a natural setting for quantum computing implementation such as superconducting [Fow+12; Cha+20a] and Majorana [Kar+17] qubits. By tiling the plane with surface code patches, a 2D grid of logical qubits is formed where the set of logical operations is local. The logical operations are different from the models that we have considered so far, see Figure 1.2.

The final operations that complete our gate set with the surface code are T gates. The T gates are not natural operations on the surface code, but can be implemented fault-tolerantly by consuming specialized resource states, called magic states [Kni04]. These magic states can be produced using a highly-optimized process called magic state distillation [BK05], which we assume occurs independently of our computation. We assume that logical magic states are available at the boundary of the grid.

The goals of architecture-respecting circuit transformations in the FTQC setting

are slightly different. Assuming quantum computers are large enough to perform quantum algorithms fault-tolerantly, we would like to minimize the resources we use and for how long we use them. More specifically, we would like to minimize the *physical space-time cost*, which is the product of the number of physical qubits and the time required to run the algorithm. But to avoid implementation details, we instead minimize the more abstract *logical space-time cost*, which is the number of logical qubits multiplied by the number of logical time steps of the architecture-respecting circuit expressed in logical surface code operations. The logical and physical space-time costs are expected to be 1-to-1 and monotonically related (see Appendix B.2), such that minimizing the former should minimize the latter.

Previous architecture-respecting circuit transformations techniques straightforwardly apply to the surface code at a logical level [LSJ15; Lao+18]. However, LOCC is natural in FTQC and comes with additional opportunities. For example, we can use LOCC to perform long-range CNOT gates in constant depth [LO17]. Long-range CNOT gates allow architecture-respecting circuit transformations to avoid remapping qubits at low overhead [Jav+17].

1.4 Contribution

We now give an overview of the contributions presented in this dissertation.

1.4.1 Quantum routing

We first give a rigorous analysis of minimal-depth and minimal-size classical routing in Chapter 2. We show an equivalence between minimal-depth classical routing and Routing via Matchings [ACG94], giving us access to a wealth of results. We obtain protocols with depth 2 for complete graphs [ACG94], $(d+2)n^{1/d}$ for d-dimensional grids [ACG94], and $3n/2 + O(\log n)$ for tree graphs [Zha99] (see Table 2.1 for more

details). We also give a novel efficient algorithm for minimal-depth classical routing on generalized hierarchical products [Bar+09] of graphs. Our algorithm matches earlier bounds in all known special cases, such as of Cartesian products of graphs (including grid graphs of arbitrary dimension), and is tight up to constants on our model of modular quantum architectures.

Similarly, we show an equivalence between minimizing the circuit size in classical routing and Token Swapping [Yam+14]. Let us define a k-*approximation algorithm*, for $k \geq 1$, as a time-efficient algorithm that finds a solution to some problem with value at most $k \times \mathrm{OPT}$, where $\mathrm{OPT} \in \mathbb{R}$ is the optimal solution value. By the equivalence, we obtain a 4-approximation algorithm on general architecture graphs [Mil+16].

As we have seen, it is possible in the Hamiltonian routing model to route the state reversal permutation induced by R (1.7) on the path architecture graph more quickly than is possible using classical routing, whose circuit depth is lower bounded by $n-1$ on the path by a simple diameter lower bound. The time-dependent protocol by Raussendorf [Rau05] and our novel time-independent protocol (Chapter 3) for state reversal are asymptotically 3 times faster. We also show that this is nearly optimal, up to a factor of at most $1.502(1+1/n)$, by showing a novel lower bound based on the asymptotic entanglement capacity [Ben+03], which is related to the small incremental entangling (SIE) theorem [Bra07; AMV13]. SIE shows that the rate at which any interacting Hamiltonian H acting on subsystems A and B, with finite ancilla spaces, can increase the entropy is at most

$$\alpha \|H\| \log(\min(\dim A, \dim B)), \tag{1.10}$$

for the SIE constant $0 < \alpha \leq 4$ [Aud14]. Routing can create entanglement across a bipartition, e.g. by routing halves of Bell pairs, therefore giving a lower bound on the evolution time by SIE.

We empirically compare state reversal protocols on their robustness, showing that the specialized state reversal protocols have reduced error scaling from static disorder compared to classical routing. Noise from static disorder can be caused by imperfect fabrication. The time-independent protocol also only requires engineered couplings and no dynamical control, so we expect it to be experimentally feasible on near-term systems such as superconducting qubits [Kja+20]. The absence of dynamical control could remove another source of noise.

Using fast state reversal as a primitive, we then construct a Hamiltonian routing algorithm in Chapter 4 that is asymptotically strictly faster than classical routing in the worst-case by a constant factor when $G_{\mathcal{A}}$ is a path graph. This is, to our knowledge, the first separation between classical routing and quantum routing. We also show that, in the average case over uniformly random permutations, this algorithm asymptotically performs routing in 2/3 the time of classical routing. These results imply improved Hamiltonian routing on generalized hierarchical products involving path graphs by our routing algorithm in Chapter 2.

Our separation between quantum and classical routing leads us to question how much more powerful gate-based or Hamiltonian quantum routing can be than classical routing. By bounding the entanglement generation across a bipartition [Mar+16], we obtain lower bounds on routing in terms of the expansion properties of $G_{\mathcal{A}}$ in Chapter 5. We first obtain the lower bound $2/c(G_{\mathcal{A}}) - 1$ on the depth of gate-based quantum routing, where $c(G_{\mathcal{A}})$ is the vertex expansion of $G_{\mathcal{A}}$. Secondly, we use SIE to prove a (weaker) lower bound of $2/(\alpha \cdot h(G_{\mathcal{A}}))$ on the evolution time of Hamiltonian routing, where $h(G_{\mathcal{A}})$ is the edge expansion of $G_{\mathcal{A}}$. On the converse side, we give an example of a superconstant, $\Omega(\sqrt{n})$, quantum-classical routing separation in a strengthened Hamiltonian routing model with 1 ancilla per qubit and fast SWAPs between the qubit and its ancilla.

We relate the lower bounds on our models of quantum routing to classical routing

by giving a new classical routing algorithm on connected simple graphs that generalizes an algorithm for regular graphs [ACG94] and is dependent on the spectral gap of the Laplacian of $G_{\mathcal{A}}$. This allows us to rule out superpolynomial quantum-classical routing separations under certain conditions. In particular, this excludes common architecture graphs with poor expansion properties, i.e., small spectral gap, such as grid graphs.

1.4.2 Architecture-respecting circuit transformations

We show how to apply quantum routing in architecture-respecting circuit transformations to convert nonlocal operations to local ones quickly. In some cases, this can bound the overhead of the circuit transformation. For example, when we consider sets of disjoint two-qubit gates acting on qubits corresponding to $q_1, q_2 \in V(G_{\mathcal{A}})$ of sufficiently small size, then we can perform routing of each such pair to an edge in a maximum matching M of $G_{\mathcal{A}}$. If $|M| = \Omega(n)$, then the overhead of an architecture-respecting circuit transformation can easily be upper bounded by the routing cost times the number of *layers* in the input circuit, which are sets of simultaneous gates in the circuit.

In Chapter 2, we introduce heuristic algorithms that decide which permutation of the qubits to implement so that many gates can be performed without too much overhead of routing. By iteratively executing local gates, finding a new mapping of qubits and routing, we construct efficient architecture-respecting circuit transformations.

We implement these circuit transformations and test them empirically on a set of benchmarks on grid and modular graphs. We see that the weighted depth of the architecture-respecting circuits produced by our algorithms with depth-minimized classical routing improves on the state-of-the-art Qiskit architecture-respecting circuit transformation [Abr+19]. Similarly, the weighted size of the architecture-respecting circuits produced by our algorithms with size-minimized classical routing improve on

Qiskit's results. However, we see that using depth-minimized classical routing results in circuits of large weighted size and using size-minimized classical routing results in circuits of large weighted depth. This indicates that optimizing for depth or size are two disjoint objectives that need to be carefully chosen for the quantum hardware.

1.4.3 Surface code circuit transformation

Finally, we show how architecture-respecting circuit transformations can be adapted to FTQC in the lattice surgery surface code. In Chapter 6, we give a detailed space-time cost analysis of operations and show that parallel long-range CNOT operations at the ends of vertex-disjoint paths can be performed in 2 logical time steps. We then construct circuits to apply parallel long-range CNOTs at the ends of edge-disjoint paths within 4 logical time steps.

Applying long-range CNOT gates via edge-disjoint paths always allows us to perform at least as many long-range operations as via vertex-disjoint paths. Moreover, no $2^{O(\log^{1-\epsilon} n)}$-approximation algorithm, for constant $\epsilon > 0$, can exist for finding maximum sets of vertex-disjoint paths on grid graphs [CKN18], unless NP $\subseteq$ RTIME($n^{\text{poly} \log n}$). But there exist $O(\log n)$ [AR95] and $O(1)$-approximation algorithms [KT95] for finding maximum sets of edge-disjoint paths on grids. We similarly are able to apply T gates and other rotations requiring magic states by finding large vertex-disjoint or edge-disjoint sets of paths to the boundary of the surface code grid through a unit MAX FLOW problem, which can be solved efficiently [FF56].

We use these constructions to propose the EDPC (Edge-Disjoint Paths Compilation) architecture-respecting circuit transformation that only performs long-range gates and does not route qubits. We implement EDPC and benchmark its space-time cost against an architecture-respecting circuit transformation from Chapter 2 using classical routing. EDPC results in significantly better performance on instances of 64 qubits and higher where classical routing is inhibited by the diameter of the architecture

graph.

Chapter 2

Classical routing for architecture-respecting circuit transformations

This chapter is based on

[CSU19] Andrew M. Childs, Eddie Schoute, and Cem M. Unsal. "Circuit Transformations for Quantum Architectures". In: *14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019)*. Ed. by Wim van Dam and Laura Mančinska. Vol. 135. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 3:1–3:24. ISBN: 978-3-95977-112-2. DOI: 10.4230/LIPIcs.TQC.2019.3

2.1 Introduction

Several previous works use exhaustive approaches for architecture-respecting circuit transformations with classical routing that take time exponential in the number of

qubits (and hence can only be used for small instances). For example, Saeedi, Wille, and Drechsler [SWD11] use SAT solvers to decompose circuits so they can be run on the path architecture; [LWD15] finds an optimal circuit transformation on nearest-neighbor architectures by formulating the problem as a pseudo-boolean optimization; Venturelli et al. [Ven+18] use temporal planners to schedule gates; and [Mur+19] uses satisfiability modulo theory solvers to find mappings of the circuit with high success probability using calibration data. Other work has instead proposed minimizing the distance between all qubits in groups of gates on specific architectures [SSP14; WLD14; PS16], but this is also NP-hard in general.

A common subproblem in architecture-respecting circuit transformations is optimizing over mappings of qubits in the input circuit $\mathcal{C}$ to the architecture graph. We call this task *qubit placement* (or qubit mapping). By choosing a suitable initial placement of qubits, we can reduce overhead of the architecture-respecting circuit. An exhaustive search of all n qubit placement takes time $O(n!)$ but can work well for small numbers of locations [Wil+16], or can be done selectively using A^* heuristic search [ZPW18; ZW19] or local search [MFM08; ANI+21]. Heuristic proposals include local search [MFM08; ANI+21] and iterated architecture-respecting circuit transformations forwards and then backwards [LDX19]. We can also view (optimal) qubit placement as a subgraph isomorphism problem [Sir+19]. If an isomorphism does not exist, we can consider smaller and smaller subcircuits of $\mathcal{C}$.

To make architecture-respecting circuit transformations tractable for large instances, we can consider the circuit not as a whole, but break it into sets of disjoint gates, which we call layers. Between a layer of gates, we route the qubits. We can exhaustively search for a good solution to qubit placement [Hir+09; TC20; Jur+21] or use heuristic routines, e.g., by prioritizing gates with many dependents [Met+06]. Recent results include the commercial t|ket⟩ compiler [Siv+20], the use of commutation rules for additional flexibility [Ito+19; Ito+20], and the use of simulated annealing for qubit

placement [ZLF20].

Contribution In this chapter, we construct architecture-respecting circuit transformations that attempt to minimize the circuit depth or size overhead and have worst-case time complexity polynomial in the sizes of the circuit and architecture graph. We model the connectivity of the underlying hardware as a simple graph where vertices represent the qubits and edges represent places where a two-qubit gate can be performed.

As a simple and fast approach, we propose the *greedy swap circuit transformation* (Section 2.2.2). It inserts SWAPs on edges chosen to minimize the total distance between qubits involved in two-qubit gates until some gate(s) can be executed.

We then propose building architecture-respecting circuit transformations (Section 2.2.2) by combining algorithms for two basic subproblems: quantum routing (addressed by *permuters*, for which we provide theoretical performance guarantees) and qubit placement (addressed by *mappers*). For the latter, we specify a variety of heuristic strategies (Section 2.4) to find suitable placement of qubits from the input circuit, attempting to optimize for circuit size or depth. We implement these algorithms in software, which is publicly available under a free software license [SUC19].

Consider now the problem of routing qubits on a given architecture graph. A sorting network sorts any fixed-length sequence of integers with a circuit of comparators, which compare two inputs and output them in nondecreasing ordering. While sorting networks can be used to route qubits [Bea+13], they achieve a more general task, and the cost of routing can sometimes be lower with other methods. Specifically, we suggest Routing via Matchings [ACG94] (introduced in Section 2.3.1) as a more suitable framework for routing qubits in parallel. Deciding whether there exists a depth-k circuit for Routing via Matchings is NP-complete in general for $k > 2$ [BR17], but optimal or near-optimal protocols are known for specific graph

families [ACG94; Zha99]. In some cases it is possible to implement any permutation asymptotically more efficiently than a general sorting network (see Table 2.1). On complete graphs, for example, any permutation can be implemented in a depth-2 circuit of transpositions [ACG94], whereas an optimal sorting network has depth $\Theta(\log n)$ [AKS83].

While it is common to consider only the worst-case routing performance, we also wish to route efficiently in practice. To improve practical performance, we generalize to *partial permutations* (permutations only defined on some subdomain) so that we can also route subsets of qubits efficiently. The destinations of the remaining qubits are unconstrained. In Section 2.3.1, we present routing algorithms for the path graph, the complete graph, and the generalized hierarchical product of graphs [Bar+09], which includes the Cartesian product of graphs and modular architectures as special cases [Mon+14]. Graphs obtained as hierarchical products have many good properties for quantum architectures [Bap+18]. We establish an upper bound on the routing number of a hierarchical product (Theorem 2.4) that matches prior work for total permutations on the Cartesian product of graphs [ACG94] and depends on easily computable properties of the input partial permutation.

We also propose using Token Swapping [Yam+14] for minimizing the total number of SWAPs, which is relevant when optimizing for total circuit size (Section 2.3.2). We generalize this problem to partial permutations and obtain a 4-approximation algorithm (Theorem 2.10).

Finally, we evaluate our circuit transformations on large quantum circuits (Section 2.5) and compare their performance with the circuit transformation included in the Qiskit software (Section 2.2.2) [Abr+19]. We find that the relative performance varies significantly with the circuit type and architecture. When minimizing circuit size, the greedy swap circuit transformation is one of the best, though some improvement may be gained using some of our specialized circuit transformations.

Graph family	Worst-case circuit depth	
	Sorting (comparators)	Routing nr. (transpositions)
path (P_n)	n [Knu98]	n [ACG94]
complete (K_n)	$\Theta(\log n)$ [AKS83]	2 [ACG94]
tree	$O(\min(\Delta, \log(n/D))\, n)$ [BRS19]	$3n/2 + O(\log n)$ [Zha99]
$\Pi_{\boldsymbol{v}}(G_1, G_2)$	not known	$\left\lceil \frac{\lvert V(G_2)\rvert}{\mathrm{ham}(\boldsymbol{v})} \right\rceil (\mathrm{rt}(G_1) + \mathrm{rt}(G_2)) + \mathrm{rt}(G_2)$
$\Pi_{\mathbf{1}}(G_1, G_2)$	not known	$2\,\mathrm{rt}(G_1) + \mathrm{rt}(G_2)$ [ACG94]
$\left(\bigtimes_{i=1}^{r} P_{n_i}\right)$	$n_1 + 2\sum_{i=2}^{r} n_i + o(\cdot)$ [Kun87]	$n_1 + 2\sum_{i=2}^{r} n_i$ [ACG94]
$\Pi_{\boldsymbol{e}_1}(K_{n_1}, K_{n_2})$	not known	$3n_2 + 2$

Table 2.1: Performance bounds for sorting networks versus routing via matchings (the routing number, $\mathrm{rt}(G)$; see (2.10)) where $|V(G)| = n$ for the graph G of interest. The tree has maximum degree Δ and diameter D. We give a new bound for the routing number of the generalized hierarchical product (see Definition 2.2) between graphs G_1 and G_2, where $\boldsymbol{v} \in \{0,1\}^{|V(G_2)|}$; $\mathrm{ham}(\boldsymbol{v})$ is the Hamming weight of $\boldsymbol{v}$, $\mathbf{1} \coloneqq [1 \ldots 1]$, and $\boldsymbol{e}_1 \coloneqq [1\,0 \ldots 0]$. We also list special cases of the generalized hierarchical product: the Cartesian product of graphs $G_1 \times G_2 = \Pi_{\mathbf{1}}(G_1, G_2)$, the r-dimensional grid of paths of lengths $n_i \in \mathbb{N}$ for $i \in [r]$, and the modular graph $\Pi_{\boldsymbol{e}_1}(K_{n_1}, K_{n_2})$. See also [BRS19] for a short overview of known lower and upper bounds for sorting networks restricted to common topologies.

For depth, some of our specialized circuit transformations do best on random circuits on grid architectures, whereas Qiskit's circuit transformation does well on modular architectures. For quantum signal processing circuits [LC17] we find that the depth is best minimized by our greedy swap circuit transformation.

2.2 Constructing Circuit Transformations

Program transformations are algorithms that modify computer programs while retaining functionality [NNH99]. In a similar vein, we define a *circuit transformation* as an algorithm that modifies an input quantum circuit to produce an output quantum circuit with the same functionality. We let Q denote the set of qubits of the input circuit. A circuit transformation is *architecture-respecting* if it produces injective initial and final mappings of the form $\hat{p}\colon Q \to V(G_{\mathcal{A}})$ with an architecture-respecting

circuit. In this chapter, we only consider circuit transformations that are architecture-respecting, and we refer to them simply as circuit transformations. We propose a construction for a general circuit transformation that may use the properties of the underlying architecture by relying on a specialized subroutine for classical routing called a permuter (Section 2.3), and a subroutine determining where to place qubits, called a *mapper* (Section 2.4).

To be able to transform a circuit, we must have $|Q| \leq |V(G_{\mathcal{A}})|$, and the output circuit must contain a qubit for every vertex in the architecture. Throughout the circuit transformation, we keep track of the injective current placement of qubits $\hat{p}\colon Q \to V(G_{\mathcal{A}})$. The initial and final values of $\hat{p}$ are also the initial and final mappings, respectively, of qubits to the architecture. A gate is *executed* by appending it to the output circuit. Two-qubit gates with qubits $q_1, q_2 \in Q$ can only be executed when $(\hat{p}(q_1), \hat{p}(q_2)) \in E(G_{\mathcal{A}})$. By adding SWAP gates to the output circuit, we can change $\hat{p}$ and thereby unitarily transform quantum circuits for execution on an architecture.

2.2.1 Definitions

We define some terminology that will be used throughout the dissertation.

Partial Functions and Partial Permutations

For sets X and Y, a *partial function* $f\colon X \rightharpoonup Y$ is a mapping from $\mathrm{dom}(f) \subseteq X$ to $\mathrm{image}(f) \coloneqq \{f(x) \mid x \in \mathrm{dom}(f)\} \subseteq Y$. However, $f(x)$ is undefined for $x \in X \setminus \mathrm{dom}(f)$. We consider such elements x *unmapped*. For $x \in \mathrm{dom}(f)$, we write $x \mapsto f(x)$ and say that x is *mapped to* $f(x)$. We can then define any partial function f as a set of mappings, $f \coloneqq \{x \mapsto y \mid x \in X, y \in Y\}$, where all preimages must be distinct (i.e., if $x \mapsto y \in f$ and $x' \mapsto y' \in f$ with $y \neq y'$, then $x \neq x'$). A *total function* $\hat{f}$ is a partial function where $\mathrm{dom}\left(\hat{f}\right) = X$ and is denoted $\hat{f}\colon X \to Y$. By the term "function" we will mean a total function.

A partial function f is *injective* iff $\forall x, x' \in \mathrm{dom}(f)$ with $x \neq x', f(x) \neq f(x')$. A function $\hat{f}\colon X \to Y$ is *surjective* iff $\forall y \in Y, \exists x \in X : f(x) = y$. A *bijective partial function* f is a partial function that is injective and is denoted $f\colon X \rightharpoonup Y$ (note that such an f is necessarily surjective on its image). A *bijective function* $\hat{f}$ is both injective and surjective and is denoted by $\hat{f}\colon X \leftrightarrow Y$. For any bijective (partial) function f there exists an inverse function $f^{-1}\colon \mathrm{image}(f) \to \mathrm{dom}(f)$.

A partial permutation π is any bijective partial function with the same domain and codomain, i.e., $\pi\colon X \rightharpoonup X$. Similarly, a total permutation is any $\sigma\colon X \leftrightarrow X$. By "permutation" we mean a total permutation.

We also define some notions specifically useful for this chapter. An *unmapped vertex* is a vertex in $V(G_{\mathcal{A}}) \setminus \mathrm{dom}(\pi)$, for a graph G and $\pi\colon V(G) \rightharpoonup V(G)$. We define the union of partial functions $f\colon X \rightharpoonup Y$ and $g\colon X \rightharpoonup Y$ when $\mathrm{dom}(f) \cap \mathrm{dom}(g) = \emptyset$ as

$$(f \cup g)(x) := \begin{cases} f(x) & \text{if } x \in \mathrm{dom}(f)\,, \\ g(x) & \text{if } x \in \mathrm{dom}(g)\,. \end{cases} \tag{2.1}$$

Furthermore, $(f \cup g)$ is a bijective partial function iff f and g are bijective partial functions and $\mathrm{image}(f) \cap \mathrm{image}(g) = \emptyset$. A *completion* of $\pi\colon X \rightharpoonup X$ is a $\hat{\pi}\colon X \leftrightarrow X = (\pi \cup \sigma)$ for some $\sigma\colon X \rightharpoonup X$, where $\mathrm{dom}(\sigma) = X \setminus \mathrm{dom}(\pi)$ and $\mathrm{image}(\sigma) = X \setminus \mathrm{image}(\pi)$.

Directed Acyclic Graph Representation of a Circuit

A quantum circuit can be viewed as a *directed acyclic graph* (DAG), where vertices represent gates and directed edges represent qubit dependencies. We define the first layer of the DAG, L, to be the set of all vertices without predecessors. By removing L and taking the first layer of the resulting DAG, we can define the second layer, and so on.

The size of a circuit is the number of gates it contains (i.e., the number of vertices in the DAG); the depth of a circuit is the number of layers. It is natural to minimize

either the depth (Section 2.4.1), corresponding to the execution time when gates can be applied in parallel, or the the size (Section 2.4.2), corresponding to the total number of operations that must be performed. We are mainly interested in two-qubit gates and their qubits. Thus we define $\mathrm{tg}\colon V_D \to Q \times Q$, where V_D is the set of DAG vertices corresponding to two-qubit gates, that outputs the pair of qubits acted on by a given gate. For simplicity, we denote $\mathrm{tg}(L) \coloneqq \{\mathrm{tg}(g) \mid g \in L, g \text{ is a two-qubit gate}\}$.

2.2.2 Architecture-Respecting Circuit Transformations

We now describe some specific architecture-respecting circuit transformations. We first describe two basic circuit transformations, one provided by the Qiskit software (Section 2.2.2) and another that uses a simple greedy approach (Section 2.2.2). Then, in Section 2.2.2 we specify a family of circuit transformations that builds on specialized procedures for qubit placement and routing.

Qiskit Circuit Transformation

The open-source quantum computing software framework Qiskit [Abr+19] contains a circuit transformation[1] that we build upon in one of our approaches (Section 2.4.2). We specify this transformation here and compare it with our other approaches to circuit transformations in Section 2.5.

We initialize $\hat{p}$ arbitrarily. Fix a number of trials, $k \in \mathbb{N}$, for each layer. We do the following in trial $i \in [k]$ where $[k] \coloneqq \{1, \ldots, k\}$: For all $v, u \in V(G_{\mathcal{A}})$, sample a symmetric weight

$$d_i(v, u) = (1 + \mathcal{N}(0, 1/N))\, d(v, u)^2 \tag{2.2}$$

independently for $(v, u) \in V(G_{\mathcal{A}}) \times V(G_{\mathcal{A}})$, where $\mathcal{N}(\mu, \sigma)$ represents a sample from the normal distribution with mean $\mu \in \mathbb{R}$ and standard deviation $\sigma \geq 0$, and $d(\cdot, \cdot)\colon V(G_{\mathcal{A}}) \times V(G_{\mathcal{A}}) \to \mathbb{N}$ is the shortest distance function on the architecture

[1] We base our description on `qiskit.mapper.swap_mapper` from Qiskit version 0.6.1.

graph. We define an objective function as the sum of gate distances,

$$S := \sum_{(q_1,q_2)\in \mathrm{tg}(L)} d_i(\hat{p}(q_1), \hat{p}(q_2))\,. \tag{2.3}$$

We now try to SWAP pairs of qubits to decrease S. Specifically, we construct a set of SWAPs by iterating over all edges $e \in E(G_{\mathcal{A}})$ and greedily adding the corresponding SWAP if it decreases S and neither endpoint of e is already involved in some SWAP. We execute the set of SWAPs and update S. We then iterate this process until either $S = |\mathrm{tg}(L)|$; or there is no SWAP that decreases S; or we reach the upper bound of $2|V(G_{\mathcal{A}})|$ iterations.

Now, if $S = |\mathrm{tg}(L)|$ then the algorithm has successfully found a sequence of SWAPs and all gates in L can be executed. The result of trial i is then set to this sequence of SWAPs. Otherwise, trial i is a failure. If there is at least one successful trial out of k trials, we execute the SWAPs of a successful trial with the fewest SWAPs and then execute all gates in L.

If no trial was successful, we apply the same routine for finding SWAPs that minimize S, but taking only a single gate $(q_1, q_2) \in \mathrm{tg}(L)$ at a time. Note that this results in a sequence of SWAPs along the shortest path between $\hat{p}(q_1)$ and $\hat{p}(q_2)$. After each such step we execute the selected gate. We repeat this until all gates in $\mathrm{tg}(L)$ have been executed and also execute all single-qubit gates in L. Finally, we remove the vertices in L from the input circuit DAG and iterate this process until all gates in the input circuit are executed.

Greedy Swap Circuit Transformation

We also describe a simple greedy approach to circuit transformations. Similar to the Qiskit circuit transformation described above, we prioritize SWAPs that maximally reduce the total distance between the qubits $\mathrm{tg}(L)$, but now using the simpler objective

function

$$R \coloneqq \sum_{(q_1,q_2)\in \mathrm{tg}(L)} d(\hat{p}(q_1), \hat{p}(q_2))\,. \tag{2.4}$$

Note that this is different from (2.3), where a randomized distance d_i is used.

We construct an initial $\hat{p}$ as follows. Let us consider the first layer L' of the circuit consisting of only two-qubit gates (i.e., single-qubit gates are ignored), initialize $p' \colon Q \rightharpoonup V(G_{\mathcal{A}})$ as undefined everywhere, and set $U \leftarrow \emptyset \subseteq V(G_{\mathcal{A}})$. We iteratively construct

$$p' \leftarrow p' + \{q_1 \mapsto v_1, q_2 \mapsto v_2 \mid (q_1, q_2) \in L', (v_1, v_2) \in M\}\,, \tag{2.5}$$

where $M \subseteq E(G)$ is a maximum matching of G, remove (q_1, q_2) from L', set $U \leftarrow U \cup \{v_1, v_2\}$, and recompute M on the subgraph of G with the vertices $V(G_{\mathcal{A}}) \setminus U$.[2] The remaining qubits $Q \setminus \mathrm{dom}(p')$ are arbitrarily mapped to the available vertices $V(G_{\mathcal{A}}) \setminus \mathrm{image}(p')$ to obtain $\hat{p}$.

In every iteration, we construct a set of disjoint gates to execute. We first execute as many gates from L as possible given $\hat{p}$, and we remove these gates from the input circuit. Second, let $E_i \subseteq E(G_{\mathcal{A}})$, for $i \in [2]$, be the set of edges where executing a SWAP would decrease R by i, excluding edges which already had a vertex involved in a gate this iteration. We then greedily execute gates from E_2 first and E_1 second, updating both E_is as we go. If we were not able to execute a gate from L and no SWAPs were executed, then, as a fallback, we deterministically pick a two-qubit gate $(q_1, q_2) \in \mathrm{tg}(L)$ and SWAP along the first edge on the shortest path between $\hat{p}(q_1)$ and $\hat{p}(q_2)$. We update $\hat{p}$ according to the inserted SWAPs, update L, and finally update R. This process is repeated until the input circuit is empty.

The fallback routine ensures that this circuit transformation always produces an output circuit. The value R strictly decreases in every iteration until a gate

[2] This is equivalent to runnning the greedy depth mapper (Section 2.4.1) on the input circuit with only two-qubit gates, an arbitrary $\hat{p}$, and free permutations of qubits. In other words, the greedy depth mapper will pick a placement of qubits on the architecture unconstrained by routing, since this is the initial placement.

can be executed unless the fallback routine is performed, in which case R stays the same. On repeated calls to the fallback routine, the same two-qubit gate is picked deterministically until it is executed. This happens within $\mathrm{diam}(G) + 1$ iterations, where the diameter of a graph G is defined as

$$\mathrm{diam}(G) \coloneqq \max_{u,v \in V(G)} d(u, v). \tag{2.6}$$

By induction we see that the whole circuit will be executed.

Let us analyze the time complexity of this circuit transformation. We ignore the initial placement since it is insignificant for large circuits. A gate from L is executed in at most $\mathrm{diam}(G)$ iterations, where $\mathrm{diam}(G)$ is the diameter of G. In every iteration, $O(|E(G_{\mathcal{A}})|)$ edges are checked to determine gates that can be executed and SWAPs that will decrease R. Therefore, the total time complexity is $O(|C||E(G_{\mathcal{A}})|\,\mathrm{diam}(G))$, where $|C|$ denotes the size of circuit C. There is a tighter bound in terms of output circuit C' since every iteration creates a layer in the transformed circuit, the complexity is $O(\mathrm{depth}(C')|E(G_{\mathcal{A}})|)$, where $\mathrm{depth}(C')$ denotes the circuit depth of C'.

Constructing Architecture-Respecting Circuit Transformations

We now present our construction for a general circuit transformation and make some definitions more precise. Let a *permuter* (see also Section 2.3) be a subroutine that, given $\pi\colon V(G_{\mathcal{A}}) \rightharpoonup V(G_{\mathcal{A}})$, outputs a sequence of transpositions that implements π while respecting the architecture constraints. Let a mapper (see also Section 2.4) be a qubit placement subroutine that, given $\hat{p}$, a permuter, and a quantum circuit, computes a new placement of qubits, $p\colon Q \rightharpoonup V(G_{\mathcal{A}})$, such that some gates of the input circuit can be executed.

Initialize $\hat{p}$ in the same way as the greedy swap circuit transformation. We repeat the following steps until the entire circuit has been transformed:

1. Use the given mapper to find a placement, $p\colon Q \rightharpoonup V(G_{\mathcal{A}})$, for the remaining input circuit;

2. Let "$\circ$" denote partial function composition, i.e., given $g\colon X \rightharpoonup Y$ and $f\colon Y \rightharpoonup Z$,

$$(f \circ g)(x) \coloneqq f(g(x))\,, \quad \text{for } x \in \mathrm{dom}(g) \text{ and } g(x) \in \mathrm{dom}(f)\,. \tag{2.7}$$

   We use the permuter to find transpositions implementing $p \circ \hat{p}^{-1}\colon V(G_{\mathcal{A}}) \rightharpoonup V(G_{\mathcal{A}})$ and replace the transpositions with SWAP gates to construct a permutation circuit to execute. We also update $\hat{p}$ to reflect the new placement of qubits after running the permutation circuit.

3. Execute all gates in L that can be executed in accordance with $\hat{p}$, remove these gates from the input circuit, and recompute L.

We note that the permuter used by the circuit transformation can be different from the one used by the mapper. This can, for example, be useful if the permuter is randomized and can be run multiple times in an attempt to obtain a better result. The number of such trials can be set much higher for the circuit transformation since only the permutation circuit for $p \circ \hat{p}^{-1}$ needs to be computed in every iteration. We make use of this flexibility in our implementation (Section 2.5).

Let us analyze the time complexity of this circuit transformation. We again ignore the time complexity of computing the initial placement. Let t_m be an upper bound on the time complexity of the mapper, and let t_p be an upper bound on the time complexity of the permuter. Computing $p \circ \hat{p}^{-1}$ takes time $O(|V(G_{\mathcal{A}})|)$. The number of transpositions produced by the permuter is at most t_p, so executing the associated SWAPs takes time $O(t_p)$. Only one gate from L may be executed every iteration so we upper bound the number of iterations by $|C|$. We find a time complexity of $O(|C|\,(t_m + |V(G_{\mathcal{A}})| + t_p))$. Clearly, if $t_p, t_m \in \mathrm{poly}(|C|, |V(G_{\mathcal{A}})|)$ then our circuit transformation is also poly-time as desired.

2.3 Partial Permutations via Transpositions

In this section we provide routing algorithms for implementing partial permutations via transpositions constrained to edges of a graph. We call such algorithms *permuters*. The Routing via Matchings and Token Swapping problems capture exactly our optimization goals of implementing a permutation of qubits on a quantum architecture while minimizing the circuit depth and size, respectively.

2.3.1 Partial Routing Via Matchings

The framework of Routing via Matchings captures how to permute qubits on a graph using a circuit of the smallest possible depth [ACG94]. We first define a generalization of Routing via Matchings that allows for partial permutations and then provide permuters for implementing partial permutations for some architectures of interest.

Definition 2.1 (*Partial Routing via Matchings*)**.** Partial Routing via Matchings is the following optimization problem. Given a simple graph G and partial permutation $\pi\colon V(G) \rightharpoonup V(G)$, the objective is to find the smallest $k \in \mathbb{N}$ such that there exist matchings $M_1, \ldots, M_k \subseteq E(E)$ on G, where each matching induces a permutation as a product of disjoint transpositions

$$\pi_{M_i} = \prod_{(v,u)\in M_i} (v\ u)\,, \tag{2.8}$$

such that

$$\hat{\pi} = \prod_{i=1}^{k} \pi_{M_i} \tag{2.9}$$

is a completion of π.

Routing via Matchings is the special case of Partial Routing via Matchings where π is constrained to be a (total) permutation. The *partial routing number* of

$\pi\colon V(G) \rightharpoonup V(G)$ on G is $\mathrm{rt}(G,\pi) \coloneqq k$, where k obtains the minimum in Definition 2.1. The *routing number* [ACG94] is the special case of the partial routing number where π is total. In this chapter, we simply refer to the partial routing number as the routing number. The routing number of G is defined as

$$\mathrm{rt}(G) \coloneqq \max_{\sigma \in \mathrm{Sym}(V(G))} \mathrm{rt}(G,\sigma)\,, \tag{2.10}$$

where we maximize over all permutations $\sigma\colon V(G) \leftrightarrow V(G)$ (here $\mathrm{Sym}(V(G))$ denotes the group of such permutations). Note that we only optimize over permutations, since for any $\pi\colon V(G) \rightharpoonup V(G)$,

$$\mathrm{rt}(G,\pi) = \min_{\hat{\pi}} \mathrm{rt}(G,\hat{\pi})\,, \tag{2.11}$$

where we minimize over all completions $\hat{\pi}$ of π.

The correspondence with minimal-depth classical routing is immediate by replacing transpositions with SWAP gates.

An alternate way to interpret (PARTIAL) ROUTING VIA MATCHINGS is to assign *tokens* to all $v \in \mathrm{dom}(\pi)$ and destinations $\pi(v)$ for the tokens. A token can only by moved through an exchange of tokens between adjacent vertices. The goal is to move all tokens to their destinations in as few matchings (specifying exchange locations) as possible. If a vertex does not hold a token at the time of an exchange with a neighbor, as can be the case in PARTIAL ROUTING VIA MATCHINGS, then after the exchange the neighbor will not hold a token.

Complete Graph

We give a simple construction of a permuter for the n-vertex *complete graph*, K_n. Given $\pi\colon V(K_n) \rightharpoonup V(K_n)$, do the following. If

$$|\mathrm{dom}(\pi) \cup \mathrm{image}(\pi)| = 2|\mathrm{dom}(\pi)|, \tag{2.12}$$

all mappings are disjoint, so we return

$$\{(v, \pi(v)) \mid v \in \mathrm{dom}(\pi)\} \tag{2.13}$$

as a single matching that implements π. Otherwise, we construct an arbitrary completion $\hat{\pi}$ of π and run the standard algorithm for Routing via Matchings for complete graphs on $\hat{\pi}$ [ACG94]. This trivially achieves the same $\mathrm{rt}(K_n) \leq 2$ bound for all π, but will obtain $\mathrm{rt}(K_n, \pi) = 1$ for all π with disjoint domain and image.

The time complexity of the Routing via Matchings algorithm for K_n is $O(n)$ [ACG94]. The other operations described above also take time $O(n)$, so we get a time complexity of $O(n)$ for the complete graph permuter.

Path Graph

We construct a permuter for the n-vertex *path graph*, P_n, by first giving a completion and then using the standard complete permuter for paths [ACG94]. Different completions achieve different routing numbers. We give a heuristic for constructing a completion that seems to result in a low routing number in practice.

We are given $\pi\colon V(P_n) \rightharpoonup V(P_n)$ and construct a completion $\hat{\pi}$ of π as follows: Let $V(P_n) \cong [n]$, ordered from one end of the path to the other (picking ends arbitrarily).

Iterate through $i \in V(P_n)$ in ascending order, setting

$$\hat{\pi}(i) = \begin{cases} \pi(i) & \text{if } i \in \mathrm{dom}(\pi), \\ \min\left(V(P_n) \setminus \mathrm{image}(\hat{\pi})\right) & \text{otherwise.} \end{cases} \tag{2.14}$$

It can easily be seen that $\hat{\pi}$ is a completion of π. We have $\mathrm{rt}(P_n, \pi) \leq \mathrm{rt}(P_n, \hat{\pi}) \leq n$ by the standard path routing algorithm [ACG94]. It remains open whether a tighter bound can be proven as a function of some parameters of π.

Constructing the completion takes time $O(n)$. The total complexity for running the path permuter is $O(n^2)$, where the time complexity of the Routing via Matchings algorithm [ACG94] dominates the construction of $\hat{\pi}$.

Hierarchical Product

The *generalized hierarchical product* (henceforth *hierarchical product*) of graphs [Bar+09] produces various subgraphs of the Cartesian product of graphs that include natural models of quantum computer architectures [Bap+18].

Definition 2.2 (Hierarchical product [Bar+09])**.** For $j \in \{1, 2\}$, let G_j be a graph with n_j vertices and $n_j \times n_j$ boolean adjacency matrix A_j, the *hierarchical product* $\Pi_{\boldsymbol{v}}(G_1, G_2)$, for $\boldsymbol{v} \in \{0, 1\}^{n_2}$, has vertex set $V(G_1) \times V(G_2)$ and adjacency matrix

$$A_1 \otimes \mathrm{diag}(\boldsymbol{v}) + \mathbb{1}_1 \otimes A_2 \,,$$

where $\mathbb{1}_1$ is the identity matrix in the vector space of A_1 and $\mathrm{diag}(\boldsymbol{v})$ is the $n_2 \times n_2$ diagonal matrix with the entries of $\boldsymbol{v}$ on the diagonal.

Intuitively, the hierarchical product $\Pi_{\boldsymbol{v}}(G_1, G_2)$ consists of n_1 copies of G_2, where the jth vertices in all copies of G_2 are connected by a copy of G_1 if $\boldsymbol{v}_j = 1$. We restrict ourselves to connected simple graphs, so A_1 and A_2 are symmetric 0–1 matrices and

$\boldsymbol{v}$ is nonzero. An example of the hierarchical product of two path graphs is

$$\Pi_{[1\ 0\ 1]}(P_2, P_3) = \Pi_{[1\ 0\ 1]}\left(\begin{array}{c}2\\ \\ 1\end{array}, 1 - 2 - 3\right) = \begin{array}{ccc}2,1 & - & 2,2 & - & 2,3\\ | & & & & |\\ 1,1 & - & 1,2 & - & 1,3\end{array} \tag{2.15}$$

The Cartesian product is $\Pi_{\mathbf{1}}$, where $\mathbf{1} \coloneqq [1 \ldots 1]$ (see Line 9). Furthermore, $\Pi_{\boldsymbol{e}_1}$ is the standard hierarchical product, and $\Pi_{\boldsymbol{e}_i}$ is the rooted product of graphs, rooted at the ith vertex of G_2.

We define the *induced subgraph* $G[U]$ of any graph G for vertex set $U \subseteq V(G)$ as

$$G[U] \coloneqq (U, E(G) \cap (U \times U)) \ . \tag{2.16}$$

Now, let $G = \Pi_{\boldsymbol{v}}(G_1, G_2)$ and denote the vertices of G by $v = (v_1, v_2) \in V_1 \times V_2 = V$. We define

$$\mathcal{G}_i \coloneqq G\left[\{i\} \times V_2\right] \ , \tag{2.17}$$

for $i \in V(G_1)$. Note that each $\mathcal{G}_i$ is isomorphic to G_2, so the permuter for G_2 can be used for $\mathcal{G}_i$. We also define the *communicator vertices* of $\mathcal{G}_i$ as the vertices

$$\{i\} \times \{j \in V(G_2) \mid \boldsymbol{v}_j = 1\} \subseteq V(\mathcal{G}_i) \, , \tag{2.18}$$

and index them in ascending order (for some ordering of $V(G)$). Note that the jth communicator vertex (of any $\mathcal{G}_i$) also belongs to $G[V(G_1) \times \{j\}]$, which is isomorphic to G_1.

A useful metric is

$$\deg(\pi) \coloneqq \max\left(\max_{i \in V(G_1)} A_i, \max_{i \in V(G_1)} B_i\right) , \tag{2.19}$$

where

$$A_i = |\{v \in \mathrm{dom}(\pi) \cap V(\mathcal{G}_i) \mid \pi(v) \notin V(\mathcal{G}_i)\}|, \tag{2.20}$$

$$B_i = |\{v \in \mathrm{dom}(\pi) \setminus V(\mathcal{G}_i) \mid \pi(v) \in V(\mathcal{G}_i)\}|, \tag{2.21}$$

which represents the maximum number of vertices that need to leave or enter any $\mathcal{G}_i$ to implement π.

In every iteration of the routing algorithm, we route a set $R = \{v^{(i)} \in \mathcal{V}_i \mid i \in V_1\}$ such that all $\pi(v)_1$ are distinct, for $v \in R$ and $\pi(v) = (\pi(v)_1, \pi(v)_2) \in V$. Undefined values are always considered distinct. We call such R a set of *representative vertices*, and we view $v^{(i)}$ as the representative vertex of V_i.

Lemma 2.3. *For a graph $G = \Pi_{\boldsymbol{v}}(G_1, G_2)$, $\pi\colon V(G) \rightharpoonup V(G)$, let $d \coloneqq \mathrm{deg}(\pi)$. We can find distinct sets of representative vertices R_i, for $i \in [d]$, such that*

$$\{v \in \mathrm{dom}(\pi) \mid v_1 \neq \pi(v)_1\} \subseteq \bigcup_{i \in [d]} R_i\,.$$

Proof. Let $H = (U, V, E)$ be a bipartite multi-graph, with $U = V \coloneqq [|V(G_1)|]$ the left and right vertex sets, and the edge multi-set

$$E = \{(v_1, \pi(v)_1) \mid v \in \mathrm{dom}(\pi)\}\,. \tag{2.22}$$

Each vertex $k \in U$ belongs to at most d edges (k, l), for $l \in V$ and $k \neq l$, and each vertex $l' \in V$ belongs to at most d edges (k', l'), for $k' \in U$ and $k' \neq l'$. However, for any $k \in U$ there could be as many as $|V(G_2)|$ edges (k, k). For all $k \in U$ we remove as many $(k, k) \in E$ as necessary to ensure that the maximum degree of any vertex in H is d.

We make H d-regular by repeating the following: If $\nexists k \in U$ with $\mathrm{deg}(k) < d$ we

```
   input   : π : V(G) ⇀ V(G); permuters on G_1 and G_2
1  Let R_i, for i ∈ [deg(π)], be given by Lemma 2.3
2  for i = 1, ..., ⌈deg(π) / ham(v)⌉ :
3      foreach j ∈ V(G_1) :
4          on G_j, for all k ∈ [ham(v)], route the (unique) vertex
             v ∈ R_{(i−1)·ham(v)+k} ∩ V(G_j) to the k-th communicator vertex of G_j
             // For R_ℓ with ℓ > deg(π), do nothing
5      foreach communicator vertex (v_1, v_2) of G_1 :      // All copies of G_1
6          on G' = G[V(G_1) × {v_2}], route each v ∈ V(G') ∩ dom(π) to
             (π(v)_1, v_2) ∈ V(G')
7  foreach i ∈ V(G_1) :
8      route all v ∈ dom(π) ∩ V(G_i) to π(v) within G_i
9  return the transpositions that implement this routing
```

Algorithm 2.3.1: Partial Routing via Matchings on the hierarchical product of graphs $G = \Pi_{\boldsymbol{v}}(G_1, G_2)$. In Lines 4 and 6, *routing* means constructing a partial permutation σ on a subgraph (G_1 or G_2), using the applicable permuter to find transpositions implementing σ, and applying those transpositions to update π and each R_i.

are done. Otherwise, such a k exists and $\exists k' \in V$ with $\deg(k') < d$, since

$$\sum_{k \in U} \deg(k) = \sum_{k' \in V} \deg(k') \,. \tag{2.23}$$

It follows that there exist vertices $u \in \mathcal{V}_k \setminus \mathrm{dom}(\pi)$ and $v \in \mathcal{V}_{k'} \setminus \mathrm{image}(\pi)$. For the purposes of this proof, we set $\pi(u) = v$, effectively adding an edge (k, k') to E.

Now we have modified π so that H is d-regular. By Hall's marriage theorem, there exists a perfect matching in H, and removing it results in a $(d-1)$-regular graph. We iterate this to find d distinct perfect matchings in H. Each edge $(k, k') \in E$ corresponds to some $v \in V(\mathcal{G}_k)$ and $u \in V(\mathcal{G}_{k'})$, with $\pi(v) = u$. Therefore, each perfect matching corresponds to a set of representative vertices, R_i. Since all perfect matchings are distinct, and all $e \in E$ are covered by some matching, the Lemma follows. □

Algorithm 2.3.1 specifies a permuter for the hierarchical product. We prove the

following performance bounds for this algorithm.

Theorem 2.4. *For a graph $G = \Pi_{\boldsymbol{v}}(G_1, G_2)$, Algorithm 2.3.1 finds a sequence of transpositions that implements $\pi\colon V(G) \rightleftharpoons V(G)$ so that*

$$\mathrm{rt}(\Pi_{\boldsymbol{v}}(G_1, G_2), \pi) \leq \left\lceil \frac{\deg(\pi)}{\mathrm{ham}(\boldsymbol{v})} \right\rceil (\mathrm{rt}(G_1) + \mathrm{rt}(G_2)) + \mathrm{rt}(G_2)\,,$$

where $\mathrm{ham}(\boldsymbol{v})$ is the Hamming weight of $\boldsymbol{v}$, i.e., the number of ones in $\boldsymbol{v}$.

Proof. In every round of routing, we route $\mathrm{ham}(\boldsymbol{v})$ sets R_i to their destination $\mathcal{G}_j$s, for $j \in V(G_1)$. In each round, we route on all copies of G_2 in parallel and then route on all copies of G_1 in parallel. After routing all R_i in at most $\lceil \deg(\pi)/\,\mathrm{ham}(\boldsymbol{v}) \rceil$ rounds, Lemma 2.3 ensures that only permutations local to each $\mathcal{G}_j$ remain. Finally, we route vertices to their destinations, as given by π, in each $\mathcal{G}_j$ independently using the permuter for G_2. □

Corollary 2.5.

$$\mathrm{rt}(\Pi_{\boldsymbol{v}}(G_1, G_2)) \leq \left\lceil \frac{|V(G_2)|}{\mathrm{ham}(\boldsymbol{v})} \right\rceil (\mathrm{rt}(G_1) + \mathrm{rt}(G_2)) + \mathrm{rt}(G_2)\,.$$

Proof. We maximize Theorem 2.4 over π by definition (2.10), and bound $\deg(\pi) \leq |V(G_2)|$ to get the result. □

As a possible optimization, we can remove some vertices from the partial permutations in the routing steps. For each removed vertex, we must ensure that the remaining steps of the routing algorithm remain valid. Specifically, let there be a $u \in \mathcal{G}_i \cap R_k$ for $i \in V(G_1)$ and $k \in [\deg(\pi)]$. If $u \in \mathrm{dom}(\pi)$ and $\pi(u) \in V(\mathcal{G}_i)$, then we remove it since it does not need to be routed outside of $\mathcal{G}_i$. Otherwise, if $u \notin \mathrm{dom}(\pi)$, we remove it unless

$$\exists v \in \{R_k \cap \mathrm{dom}(\pi) \mid \pi(v) \in \mathcal{G}_i\} \tag{2.24}$$

since an unmapped vertex is expected at the communicator vertex in the second loop of the routing round. We apply this optimization in our implementation of the permuter for modular graphs (Line 9).

Next, we analyze the time complexity of Algorithm 2.3.1. Let t_1 and t_2 upper bound the time complexity of algorithms for Partial Routing via Matchings on G_1 and G_2, respectively. We first find $\deg(\pi)$ distinct sets of representative vertices by Lemma 2.3. The time to find one set of representative vertices is dominated by the time to find the maximum bipartite matching, $O(n_1^{2.5})$ [HK73]. Then, for $\lceil \deg(\pi)/\operatorname{ham}(\boldsymbol{v}) \rceil$ iterations, we route on all copies of G_2 and then G_1 in parallel. Overall, we get a time complexity of

$$O\left(\deg(\pi) \cdot n_1^{2.5} + \left\lceil \frac{\deg(\pi)}{\operatorname{ham}(\boldsymbol{v})} \right\rceil (\operatorname{ham}(\boldsymbol{v}) t_1 + n_1 t_2) + n_1 t_2\right). \tag{2.25}$$

We show a lower bound on the routing number of hierarchical products of graphs and prove that it is tight, up to constant factors.

Theorem 2.6. *For a graph $G = \Pi_{\boldsymbol{v}}(G_1, G_2)$ and any $\pi \colon V(G) \rightharpoonup V(G)$,*

$$2\left\lceil \frac{\deg(\pi)}{\operatorname{ham}(\boldsymbol{v})} \right\rceil - 1 \leq \operatorname{rt}(\Pi_{\boldsymbol{v}}(G_1, G_2), \pi)\,.$$

Proof. Let us consider the token-based formulation of Partial Routing via Matchings. At most $\deg(\pi)$ tokens need to be moved out of any $\mathcal{G}_i$, for $i \in V(G_1)$. Every matching can move at most $\operatorname{ham}(\boldsymbol{v})$ tokens out of their original $\mathcal{G}_i$. Once moved out, a new set of tokens must be moved onto the $\operatorname{ham}(\boldsymbol{v})$ communicator vertices. Therefore, it takes at least $2\lceil \deg(\pi)/\operatorname{ham}(\boldsymbol{v}) \rceil - 1$ matchings to move $\deg(\pi)$ tokens out of any $\mathcal{G}_i$. □

We now show that Theorem 2.6 is tight up to constant factors by considering a specific permutation on the path graph P_{2n}, with $n \in \mathbb{N}^+$. We have $P_{2n} \cong \Pi_{\boldsymbol{e}_1}(P_2, P_n)$

by a relabeling of vertices. We define $\pi' \colon V(P_{2n}) \leftrightarrow V(P_{2n})$ as the product of transpositions

$$\pi' \coloneqq \prod_{i=0}^{n-1} (i\ (n+i))\ . \tag{2.26}$$

Then,

$$2\left\lceil \frac{\deg(\pi')}{\mathrm{ham}(\boldsymbol{e}_1)} \right\rceil - 1 = 2n - 1 \leq \mathrm{rt}(\Pi_{\boldsymbol{e}_1}(P_2, P_n), \pi') = \mathrm{rt}(P_{2n}, \pi') \leq \mathrm{rt}(P_{2n}) \leq 2n\,, \tag{2.27}$$

where we used Section 2.3.1 for the last inequality, and $\boldsymbol{e}_i \in \{0,1\}^n$ is the ith standard basis vector. This also matches the tightest known (diameter) lower bound for $\mathrm{rt}(P_{2n})$.

Modular Graphs

Large-scale quantum computation may benefit from a modular design, with many interconnected subunits [MK13; Mon+14; Bre+16]. As a simple model of a modular quantum processor consisting of n_1 modules with n_2 qubits each, we define the *modular graph*,

$$\mathrm{Mod}(n_1, n_2) \coloneqq \Pi_{\boldsymbol{e}_1}(K_{n_1}, K_{n_2}). \tag{2.28}$$

In this architecture, any two qubits in the same module can be directly coupled, and any two modules can be coupled through their unique communicator qubits. With one minor modification to Theorem 2.4, we get the following bounds on the routing number of the modular graph.

Corollary 2.7. *For $n_1, n_2 \in \mathbb{N}$, $G = \mathrm{Mod}(n_1, n_2)$ and $\pi \colon V(G) \rightharpoonup V(G)$, we have*

$$2\deg(\pi) - 1 \leq \mathrm{rt}(\mathrm{Mod}(n_1, n_2), \pi) \leq 3\deg(\pi) + 2\,.$$

Proof. Directly applying Theorem 2.4 gives

$$\mathrm{rt}(\mathrm{Mod}(n_1, n_2), \pi) \leq 4\deg(\pi) + 2\,. \tag{2.29}$$

However, only one token needs to be routed to the communicator vertex in every round of Algorithm 2.3.1 and this satisfies (2.12). Therefore, we can route with one set of parallel transpositions, saving us one matching every round.

To show the lower bound, we apply Theorem 2.6 with $\mathrm{ham}(\boldsymbol{e}_1) = 1$. □

We evaluate the time complexity of this permuter using (2.25). Recall from Section 2.3.1 that the time complexity of the permuter is $O(n)$. Thus we have $t_1 = O(n_1)$ and $t_2 = O(n_2)$, giving an overall time complexity of $O(dn_1^{2.5} + n_1 n_2)$, where we noted that $t_2 = O(1)$ while doing the $\deg(\pi)$ rounds of routing.

Cartesian Product

The Cartesian product of graphs is a special case of the hierarchical product, namely $\pi_{\mathbf{1}}$ for $\mathbf{1} \coloneqq [1 \dots 1]$. We refer to a copy of G_1 in $G_1 \times G_2$ (i.e., $G[V(G_1) \times \{v_2\}]$ for some $v_2 \in V(G_2)$) as a *row* of $G_1 \times G_2$, and, vice versa, to a copy of G_2 as a *column*. Also, let $n_1 \coloneqq |V(G_1)|$ and $n_2 \coloneqq |V(G_2)|$. Theorem 2.4 allows us to reprove an upper bound on the routing number of a Cartesian product of graphs [ACG94].

Corollary 2.8. *For any graphs G_1 and G_2,*

$$\mathrm{rt}(G_1 \times G_2) = \mathrm{rt}(\Pi_{\mathbf{1}}(G_1, G_2)) \leq \mathrm{rt}(G_1) + 2\,\mathrm{rt}(G_2)\,.$$

Proof. Since $\mathrm{ham}(\mathbf{1}) = n_2$, we get the result by Corollary 2.5. □

Lemma 2.3 does not specify the order in which systems of distinct representatives are picked, but this order matters in practice. Since $\mathrm{ham}(\boldsymbol{v}) = n_2$, we can pick n_2 distinct sets of representative vertices without incurring another round of routing (in Algorithm 2.3.1). We propose a heuristic for picking these $|V(G_2)|$ sets that seems to produce low-depth implementations of partial permutations in practice (Algorithm 2.3.2).

Algorithm 2.3.2 uses a modification of Lemma 2.3 to choose representative vertices. The proof of Lemma 2.3 can be straightforwardly extended by not initially removing edges of the form (k, k) and adding edges until an n_2-regular bipartite multi-graph, B, is constructed. Thus, by Hall's marriage theorem, there exist n_2 distinct perfect matchings in B, enough for all the rows. We choose a perfect matching of minimum weight for each row with respect to a heuristic cost function $c\colon \mathrm{dom}(\pi) \times V(G_2) \to \mathbb{N}$, with the rows processed in a random order.

We add additional edges to B to allow for more options to minimize the weight. We construct a bipartite multi-graph B' that contains B, disregarding some duplicated edges. Edge duplication does not change the minimum-weight perfect matching. Instead of adding an edge for unmapped vertices as in Lemma 2.3, we add edges to all possible destination columns for each column with an unmapped vertex.

Let $\sigma\colon V(G_1) \times V(G_2) \rightharpoonup V(G_1) \times V(G_2)$ be the partial permutation defined on Line 1 of Algorithm 2.3.2. The cost function depends on the current value of σ and is defined as

$$c(v, i) \coloneqq \mathrm{rt}(G_2, \pi_1) + \mathrm{rt}(G_2, \{i \mapsto \pi(v)_2\}) - \mathrm{rt}(G_2, \pi_2) - \mathrm{rt}(G_2, \{v_2 \mapsto \pi(v)_2\})\,, \tag{2.30}$$

where

$$\pi_1\colon u \mapsto \begin{cases} \sigma(v_1, u)_2 & \text{if } (v_1, u) \in \mathrm{dom}(\sigma)\,, \\ i & \text{if } u = v_2 \end{cases} \tag{2.31}$$

is the partial permutation routing v to row i within its column, and $\pi_2\colon u \mapsto \sigma(v_1, u)_2$ is the current partial permutation already planned for column v_1. For simplicity, we assume the routing time along rows is the same in both cases, so it cancels out. To compute an upper bound on the routing number in (2.30) we use the given permuter for G_2.

To implement routing on the Cartesian product of graphs, we route σ obtained

```
   input   : π : V(G) ⇀ V(G)
1  σ ← empty set of V(G) ⇀ V(G)
2  r ← |V(G_2)|                                          // #remaining rows
3  foreach row i ∈_R V(G_2) :
4      E ← {(v_1, π(v)_1, c(v, i)) | v ∈ dom(π) \ dom(σ)}
       // Add edges for unmapped vertices
5      H = (U, V, E), with U = V := [n_1]
6      foreach u ∈ U with deg_E(u) < r :
7          foreach v ∈ V with deg_E(v) < r :
8              Add (u, v, ϵ) to E(H)
9      Find a minimum-weight perfect matching E_match in H
10     V_match ← the set of vertices associated with E_match
11     σ ← σ + {v ↦ (v_1, i) | v ∈ V_match}                  // Recall (2.1)
12     r ← r − 1
13 return σ
```

Algorithm 2.3.2: Heuristically choosing distinct sets of representative vertices for the Cartesian product of graphs $G = G_1 \times G_2$, for graphs G_1 and G_2. We modify Lemma 2.3 to pick n_2 minimum-weight perfect matchings, with respect to the heuristic cost function c ((2.30)). The notation $\in_R$ indicates that we select elements uniformly at random without replacement. The edges of the weighted undirected bipartite multi-graph H are specified as a multi-set of triples from $V(G) \times V(G) \times \mathbb{R}$. We pick $\epsilon > 0$ so that zero-cost edges for mapped vertices are favored over edges for unmapped vertices.

from Algorithm 2.3.2 within each column independently, and proceed with Line 5 of Algorithm 2.3.1.

Finally, we analyze the time complexity of the permuter for Cartesian products of graphs. Assume the time complexity of computing $\mathrm{rt}(G_1, \sigma)$ and $\mathrm{rt}(G_2, \sigma')$ is upper bounded by t_1 and t_2, respectively. Computing the cost function (2.30) then has time complexity $O(t_2)$. In Algorithm 2.3.2 we construct a bipartite weighted graph with $2n_2$ vertices in time $O(n_2 n_1 t_2 + n_2^2)$. On that graph we perform a maximum weighted bipartite matching algorithm in $O(n_2^3)$ using the Hungarian algorithm [Kuh55].[3] We do this once for each row and route all vertices to their assigned rows. Then, we continue with Line 5 of Algorithm 2.3.1, resulting in a total time complexity for

[3] A tighter bound of $O(\sqrt{n}m \log(nC))$ is possible [GT89], for n, m, C the number of vertices, the number of edges, and absolute maximum integer edge weight, respectively. Our edge weights can be scaled to integers that are upper bounded by $O(n_2(n_1 + n_2))$.

running the permuter of $O(n_1\,(n_2 n_1 t_2 + n_2^3) + n_2 t_1)$.

2.3.2 Partial Token Swapping

The Token Swapping problem is similar to Routing via Matchings, but minimizes the total number of transpositions instead of the depth [Yam+14]. It follows that the induced permutation circuit is optimized for circuit size. For $\epsilon > 0$, a $(1+\epsilon)$-approximation algorithm is an algorithm that produces a solution within a factor $(1+\epsilon)$ of optimal for all valid inputs. Here, we define a generalized version of Token Swapping that allows for partial permutations, and then give a 4-approximation algorithm for this problem on connected simple graphs that generalizes a previous 4-approximation algorithm for total permutations [Mil+16].

Definition 2.9 (*Partial Token Swapping*)**.** We define Partial Token Swapping as an optimization problem. Given are a graph G and partial permutation $\pi\colon V(G) \rightharpoonup V(G)$. The objective is to find the smallest $k \in \mathbb{N}$ such that $\hat{\pi} = (u_1\ v_1)(u_2\ v_2)\dots(u_k\ v_k)$, for $\hat{\pi}$ some completion of π and $(u_i, v_i) \in E(G)$ for $i \in [k]$.

Analogous to the routing number, we define the *routing size* of $\pi\colon V(G) \rightharpoonup V(G)$ on G, $\mathrm{rs}(G,\pi)$, to be the minimum k in Definition 2.9, and the routing size of G as

$$\mathrm{rs}(G) \coloneqq \max_{\sigma \in \mathrm{Sym}(V(G))} \mathrm{rs}(G,\sigma)\,. \tag{2.32}$$

Token Swapping is the special case of Partial Token Swapping where π is constrained to be a total permutation. Partial Token Swapping also has an equivalent token-based formulation, similar to Partial Routing via Matchings.

The correspondence to minimal-size classical routing is immediate by replacing transpositions with SWAP gates.

The decision version of Token Swapping was first shown to be NP-complete [Mil+16] and hard for a model of parametrized complexity, parametrized by the number of swaps k [BMR17]. Furthermore, assuming the Exponential Time Hypothesis (ETH), Token Swapping cannot be solved in time $f(k)(|V(G)| + |E(G)|)^{o(k/\log k)}$ with f any computable function [BMR17].

Approximation Algorithm for Partial Token Swapping

We now describe a permuter that aims to minimize the circuit size. Miltzow et al. [Mil+16] gave a 4-approximation algorithm for Token Swapping. Here, we generalize their results to Partial Token Swapping and prove that our generalized algorithm is also a 4-approximation algorithm. For this section, we consider the token-based formulation of Partial Token Swapping (recall the notion of tokens introduced in Section 2.3.1).

The main idea of Miltzow et al. is to perform swaps that reduce the sum of all distances of tokens to their destinations. We use the following definitions from [Mil+16]: An *unhappy swap* is "an edge swap where one of the tokens swapped is already on its target and the other token reduces its distance to its target vertex (by one)", and a *happy swap chain* is a path of $\ell + 1$ distinct vertices $v_1 v_2 \dots v_\ell$, such that swapping all (v_i, v_{i+1}), for $i \in [\ell - 1]$, in increasing order strictly reduces the distances of all tokens in the chain to their destinations.

When considering a partial permutation, not all vertices have a token assigned to them. We add an extra step to the approximation algorithm for Token Swapping to make use of this: Before considering an unhappy swap, we first try to swap a token to a tokenless neighbor if it brings the token closer to its destination. We call this a *no-token swap*.

The approximation algorithm for Partial Token Swapping is specified in full in Algorithm 2.3.3.

```
input  : π : V(G) ⇀ V(G)
while  π ≠ id|_{dom(π)} :
    if  there exists a happy swap chain v_1 v_2 ... v_ℓ then
        Perform transpositions (v_1 v_2)(v_2 v_3) ... (v_{ℓ-1} v_ℓ)
    else if ∃v ∈ dom(π), ∃u ∈ N(v) \ dom(π) : d(u, π(v)) < d(v, π(v)) then
        Perform no-token swap (v u)                    // u has no token
    else
        There exists an unhappy swap; perform it
    Update π according to the transpositions that were performed
return The sequence of transpositions that was performed
```

Algorithm 2.3.3: Routing tokens on a connected simple graph G to their destinations while minimizing the number of transpositions. We add an extra step that performs no-token swaps to the algorithm of [Mil+16]. For $v \in V(G)$, $N(v) \subseteq V(G)$ denotes the set of *neighbors* of v. The partial permutation $\mathrm{id}|_{\mathrm{dom}(\pi)} \colon V(G) \rightharpoonup V(G)$ is the restriction of the identity function $\mathrm{id} \colon V(G) \leftrightarrow V(G)$ to $\mathrm{dom}(\pi)$ (so it is undefined outside of $\mathrm{dom}(\pi)$).

Theorem 2.10. *Given a simple connected graph G and $\pi \colon V(G) \rightharpoonup V(G)$, Algorithm 2.3.3 uses at most $4 \cdot \mathrm{rs}(G, \pi)$ transpositions.*

Proof. The proof is very similar to [Mil+16, Theorem 7] with some minor modifications to account for no-token swaps. Let

$$S := \sum_{v \in \mathrm{dom}(\pi)} d(v, \pi(v)) \,. \tag{2.33}$$

We know that $\mathrm{rs}(G, \pi) \geq S/2$ since each swap can only reduce S by two. A no-token swap reduces S by one. A happy swap chain of length ℓ reduces S by $\ell + 1$. As such, over the course of the algorithm,

$$\#(\text{happy swaps}) + \#(\text{no-token swaps}) \leq S \,. \tag{2.34}$$

For an unhappy swap, the token that is swapped away from its destination must next

be involved in a happy swap or a no-token swap, so

$$\#(\text{unhappy swaps}) \leq \#(\text{happy swaps}) + \#(\text{no-token swaps})\,. \tag{2.35}$$

Overall, we have

$$\begin{aligned} &\#(\text{unhappy swaps}) + \#(\text{happy swaps}) + \#(\text{no-token swaps}) && (2.36)\\ &\quad \leq 2\#(\text{happy swaps}) + 2\#(\text{no-token swaps}) && (2.37)\\ &\quad \leq 2S \leq 4 \cdot \mathrm{rs}(G, \pi)\,. && \square \end{aligned}$$

Miltzow et al. further showed that their algorithm for total permutations gives a 2-approximation algorithm when the graph is a tree. We now give an example showing that this is not the case for our modified algorithm when the permutation is partial. Consider the path graph P_n, for $n > 2$, and a partial permutation

$$\pi \coloneqq \{i \mapsto i+1 \mid i \in [n-2]\} \cup \{n \mapsto 1\}\,. \tag{2.38}$$

Trivially, the shortest product of transpositions implementing π is

$$\prod_{i=0}^{n-2} ((n-i)\ (n-1-i)) \tag{2.39}$$

of length $n-1$. However, the algorithm selects no-token swaps arbitrarily. In the worst case, it could select the sequence of transpositions

$$\left[\prod_{i=0}^{n-3} ((n-2-i)\ (n-1-i))\right] \cdot \left[\prod_{i=0}^{n-2} ((n-i)\ (n-1-i))\right] \cdot \left[\prod_{i=0}^{n-3} ((2+i)\ (3+i))\right] \tag{2.40}$$

of length $3n-5$. Therefore, in the limit we get an approximation ratio of $\lim_{n\to\infty}(3n-5)/(n-1) = 3$.

While (2.38) is only undefined on one input, we can modify π by removing $k = o(n)$ entries to make it harder to find an appropriate completion, since there are $(k+1)!$ possibilities. Then the algorithm still asymptotically achieves an approximation ratio of $\lim_{n\to\infty}(3n-5-2k)/(n-1) = 3$.

Of course, it is still possible that the algorithm could achieve better than a 4-approximation. We leave the best approximation ratio of our Partial Token Swapping algorithm (on trees and in general) as an open question.

Finally, we determine the time complexity of this permuter. Computing an all-to-all distance matrix takes time $\Theta(|V(G)|^3)$ using the Floyd-Warshall algorithm [Flo62], but this cost needs only to be incurred once for a graph so we do not include it. A happy or unhappy swap can be found in time $O(|E(G)|)$ by finding cycles in an auxiliary directed graph [Mil+16]. Similarly, finding no-token swaps has time complexity $O(|E(G)|)$. Therefore, we get a total time complexity of $O(S|E(G)|) \leq O(|V(G)|^2|E(G)|)$.

2.4 Placing Qubits on the Architecture

A *mapping* algorithm (or *mapper*) finds an assignment of circuit qubits to architecture vertices such that gates can be executed efficiently. We specify mappers in terms of the routing number $\mathrm{rt}(G_{\mathcal{A}}, \pi)$ (2.10) and the routing size $\mathrm{rs}(G_{\mathcal{A}}, \pi)$ (2.32), for $\pi\colon V(G_{\mathcal{A}}) \rightharpoonup V(G_{\mathcal{A}})$. In practice, we replace these quantities with the upper bounds that result from applying our permuters.

Mappers construct *placements* of circuit qubits onto qubits of the architecture. A placement is a bijective partial function $p\colon Q \rightharpoonup V(G_{\mathcal{A}})$. A mapper has access to the *current placement* $\hat{p}\colon Q \to V(G_{\mathcal{A}})$ provided by the circuit transformation. Given a placement p and the current placement $\hat{p}$, we can compute a partial permutation $p \circ \hat{p}^{-1}\colon V(G_{\mathcal{A}}) \rightharpoonup V(G_{\mathcal{A}})$ that implements p. All our mappers construct a placement p that is initially undefined everywhere and modify it until finished.

In the remainder of this section, we describe several specific mappers that we implement and evaluate. We describe mappers optimizing for circuit depth in Section 2.4.1 and for circuit size in Section 2.4.2. We also give an upper bound on the time complexity of the mappers as a function of the time complexity of the permuter, t_p.

2.4.1 Circuit Depth Mappers

In this section we discuss mappers that attempt to minimize the transformed circuit depth. Let L be the first layer of gates of the input circuit, and let M be a maximum matching in the architecture graph.

Greedy Depth Mapper

The *greedy depth mapper* iteratively places the highest-cost gate at its lowest-cost location, where cost is measured in terms of the routing number to achieve the placement. More precisely, we initialize the set of used vertices $U \leftarrow \emptyset$ and find a placement $p' \coloneqq \{q_1 \mapsto v_1, q_2 \mapsto v_2\}$ that attains the optimum

$$\max_{(q_1,q_2)\in \mathrm{tg}(L)} \min_{(v_1,v_2)\in M} \mathrm{rt}\big(G_{\mathcal{A}}, (p \cup \{q_1 \mapsto v_1, q_2 \mapsto v_2\}) \circ \hat{p}^{-1}\big)\,, \tag{2.41}$$

where we consider both orderings of edges from M, $(v,u),(u,v) \in M$, since edges are undirected. Then, we update $U \leftarrow U \cup \mathrm{dom}(p')$ and recompute M for the induced subgraph $G_{\mathcal{A}}[V(G_{\mathcal{A}}) \setminus U]$; we remove the gate associated to (q_1, q_2) from L; we set $p \leftarrow p \cup p'$; and we iterate until $\mathrm{tg}(L) = \emptyset$ or $M = \emptyset$. Finally, we return the placement p.

In this procedure, we perform at most $\min\{|L|, |M|\}$ iterations to place gates. In each iteration, we find a p' according to (2.41) in time $O(|L||M|t_p)$. Thus, the time

complexity for one call of the mapper is

$$O\Big(\min\{|L|,|M|\}\left(|L||M|t_p+\sqrt{|V(G_{\mathcal{A}})|}|E(G_{\mathcal{A}})|\right)\Big)\,, \tag{2.42}$$

where $O(\sqrt{|V(G_{\mathcal{A}})|}|E(G_{\mathcal{A}})|)$ is the complexity of computing a maximum matching [MV80].

Incremental Depth Mapper

Instead of trying to place (almost) all gates in L, the *incremental depth mapper* guarantees placement of only the lowest-cost gate, as given by the routing number, and incrementally improves the situation for the other gates. Specifically, we first find a placement $p_{\min} \coloneqq \{q_1 \mapsto v_1, q_2 \mapsto v_2\}$ that attains the optimum

$$c'_{\min} \coloneqq \min_{(q_1,q_2)\in \mathrm{tg}(L)} \min_{(v_1,v_2)\in E(G_{\mathcal{A}})} \mathrm{rt}(G_{\mathcal{A}}, (p\cup\{q_1\mapsto v_1, q_2\mapsto v_2\})\circ\hat{p}^{-1})\,, \tag{2.43}$$

where we consider both orderings of $E(G_{\mathcal{A}})$, $(u,v),(v,u)\in E(G_{\mathcal{A}})$. We set $p\leftarrow p_{\min}$ and define $U\coloneqq\{u,v\}$. Let $c_{\min}\coloneqq\max\{c'_{\min},1\}$.

We find a placement for the remaining two-qubit gates that (individually) does not exceed $c_{\min}$. We iterate in arbitrary order over $(q_1,q_2)\in\mathrm{tg}(L)$ and do the following: For $i\in[2]$, we construct a set of eligible vertices

$$U_i \coloneqq \left\{v\in V(G_{\mathcal{A}})\setminus U \mid \mathrm{rt}\big(G_{\mathcal{A}}, (p\cup\{q_i\mapsto v\})\circ\hat{p}^{-1}\big)\le c_{\min}\right\}. \tag{2.44}$$

Now we try to find $v_1^*\neq v_2^*$ as

$$(v_1^*,v_2^*)\coloneqq \operatorname*{arg\,min}_{(v_1,v_2)\in U_1\times U_2} d(v_1,v_2)\,. \tag{2.45}$$

If such (v_1^*,v_2^*) does not exist, we do not include q_1 and q_2 in p; otherwise, we set $p\leftarrow p\cup\{q_1\mapsto v_1^*, q_2\mapsto v_2^*\}$ and update $U\leftarrow U\cup\{v_1^*,v_2^*\}$. After iterating over all

gates in $\mathrm{tg}(L)$, we return p.

The time complexity of the incremental mapper is

$$O\big(|L|\left(|E(G_{\mathcal{A}})|t_p + |V(G_{\mathcal{A}})|t_p + |V(G_{\mathcal{A}})|^2\right)\big)\,. \tag{2.46}$$

This assumes we have access to the all-pairs distance matrix of the architecture graph, which can be precomputed in time $\Theta(|V(G_{\mathcal{A}})|^3)$ [Flo62] (independent of the input circuit).

2.4.2 Circuit Size Mappers

We now discuss mappers that optimize for circuit size. The behavior of such mappers is somewhat different from mappers optimizing for circuit depth. If there is any gate that can be performed without moving qubits, then there is no disadvantage to doing that immediately since it will have to be performed eventually. If there is any such gate, we simply return the empty placement. Thus we assume, for all mappers in this section, that there are no gates to be performed in-place.

Greedy Size Mapper

The *greedy size mapper* the same as the greedy depth mapper (Section 2.4.1), except that we replace $\mathrm{rt}(\cdot)$ with $\mathrm{rs}(\cdot)$ in (2.41).

Simple Size Mapper

The *simple size mapper* places only the lowest-cost gate at its lowest-cost location. More precisely, we find a placement $p := \{q_1 \mapsto v_1, q_2 \mapsto v_2\}$ that attains the optimum

$$\min_{(q_1,q_2)\in\mathrm{tg}(L)} \min_{(v_1,v_2)\in E(G_{\mathcal{A}})} \mathrm{rs}\big(G_{\mathcal{A}}, (p \cup \{q_1 \mapsto v_1, q_2 \mapsto v_2\}) \circ \hat{p}^{-1}\big) \tag{2.47}$$

where we consider all orderings of the edges of $E(G_{\mathcal{A}})$, and return p. Note that we have replaced $\mathrm{rt}(\cdot)$ with $\mathrm{rs}(\cdot)$ in (2.43). The time complexity of the simple size mapper is $O(|L||E(G_{\mathcal{A}})|t_p)$.

Extension Size Mapper

The *extension size mapper* first finds an initial placement p using (2.47). Let $c'_{\min}$ be the value attained at the optimum for (2.47). After finding the initial placement, we try to only place another gate if it is cheaper to place now rather than in a later call to the mapper.

Specifically, for the current p and $\hat{p}$, we define $\hat{p}'\colon Q \to V(G_{\mathcal{A}})$ as the placement after performing the permutation circuit constructed from transpositions achieving $\mathrm{rs}(G_{\mathcal{A}}, p \circ \hat{p}^{-1})$. Let $U \leftarrow \emptyset$ and $G_U = G_{\mathcal{A}}[V(G_{\mathcal{A}}) \setminus U]$. Now we define a heuristic for the number of saved transpositions, $s\colon Q \times Q \to \mathbb{N}$, as

$$\begin{aligned} s(q_1, q_2) \coloneqq \mathrm{rs}\big(G_{\mathcal{A}}, p \circ \hat{p}^{-1}\big) + \min_{(v_1,v_2)\in E(G_{\mathcal{A}})} \mathrm{rs}\Big(G_{\mathcal{A}}, \{q_1 \mapsto v_1, q_2 \mapsto v_2\} \circ (\hat{p}')^{-1}\Big) \\ - \min_{(u_1,u_2)\in E(G_U)} \mathrm{rs}\big(G_{\mathcal{A}}, (p \cup \{q_1 \mapsto u_1, q_2 \mapsto u_2\}) \circ \hat{p}^{-1}\big)\,, \end{aligned} \tag{2.48}$$

and we consider all orderings of the edges of $E(G_{\mathcal{A}})$ and $E(G_U)$.

The extension size mapper iterates the following. We find the gate $(q_1^*, q_2^*) \in \mathrm{tg}(L)$ attaining

$$s_{\max} \coloneqq \max_{(q_1,q_2)\in \mathrm{tg}(L)} s(q_1, q_2)\,, \tag{2.49}$$

and let $(u_1^*, u_2^*) \in E(G_U)$ be the edge attaining $s_{\max}$. If $s_{\max} \geq 0$, we set $p \leftarrow p \cup \{q_1^* \mapsto u_1^*, q_2^* \mapsto u_2^*\}$, remove the gate (q_1^*, q_2^*) from L, update $U \leftarrow U \cup \{v_1^*, v_2^*\}$, and iterate; otherwise, we stop and return p.

Calculating $s(q_1, q_2)$ for any $q_1, q_2 \in Q$ takes time $O(|E(G_{\mathcal{A}})|t_p)$. Therefore, the

total time complexity of the extension size mapper is

$$O\big(|L|^2|E(G_{\mathcal{A}})|t_p\big)\,. \tag{2.50}$$

Qiskit-based Mapper

Finally, we implement a mapper that is based on Qiskit's circuit transformation (described in Section 2.2.2). Since this is a mapper, we only execute one iteration of the circuit transformation: for the first layer L. We also do not modify the output circuit, but instead return the final $\hat{p}$ that would be induced by executing all SWAPs found during the mapping process.

We make three changes to Qiskit's circuit transformation. The first is that when minimizing S, instead of choosing a maximal set of SWAPs in every iteration, we choose only one SWAP along an edge $e \in E(G_{\mathcal{A}})$ that minimizes S. The second is that the upper bound on the number of iterations is raised to $|V(G_{\mathcal{A}})|^2$, since we only apply one SWAP per iteration. Thirdly, if no trial is successful, we fall back to the simple size mapper (Section 2.4.2) and return the placement it finds, which places only one gate in this iteration.

We now give the time complexity of the Qiskit mapper. First, we compute an all-to-all distance matrix in time $\Theta(|V(G_{\mathcal{A}})|^3)$ [Flo62], which we ignore since it is a one-time cost dependent only on the architecture. Each of the $O\big(|V(G_{\mathcal{A}})|^2\big)$ iterations has a time complexity of $O(|E(G_{\mathcal{A}})||L|)$. Thus, the Qiskit mapper has time complexity $O\big(|V(G_{\mathcal{A}})|^2|E(G_{\mathcal{A}})||L|\big)$.

2.5 Results

We implement the architecture-respecting circuit transformations introduced in Section 2.2.2 with a variety of mappers and appropriate permuters. We also implement

the greedy swap transformation described in Section 2.2.2. We check the validity of our implementations by testing closeness in fidelity of the original output state and that of the transformed circuit for random input states of 11 qubits on random circuits [SUC19] (described in the next section).

2.5.1 Evaluation Criteria

When testing the performance of these circuit transformations, each is allocated at most 8GB of RAM and 2 days to transform all circuits of a data point. For each data point we transform 10 random circuits and 1 QSP circuit. We consider a 2-day runtime acceptable, given that classical computational resources are plentiful compared to quantum ones. We generate the data on a heterogeneous cluster with Intel Opteron 2354 and Intel Xeon X5560 processors.

The Cartesian permuter (Line 9), the general size permuter (Section 2.3.2), and Qiskit's circuit transformation (Section 2.2.2) are randomized. We run multiple trials of these permuters and take the best result. Most of the time, trials produce equally good permutation circuits, although occasionally they deviate by a few SWAP gates. Our mappers run permuters $O(|L||E(G_{\mathcal{A}})|)$ times, so we do only 4 trials to quickly remove any bad outliers. In contrast, our circuit transformation only directly runs a permuter once per layer of gates, so in this case we perform a slower 100 trials in an attempt to save a few SWAPs. We leave the number of trials for Qiskit's circuit transformation at its default of 40.

We test the performance of circuit transformations for the *grid*, $P_{n_1} \times P_{n_2}$, using the permuter from Line 9 and the modular architecture, $\mathrm{Mod}(n_1, n_2)$, with the permuter from Line 9, for $n_1, n_2 \in \mathbb{N}$. For an N-qubit circuit, we set $n_1 = n_2 = \lceil \sqrt{N} \rceil$ so that there are enough qubits in the architecture to contain the circuit. By Corollary 2.8, we know that taking $n_1 = n_2$ minimizes the routing time for our routing strategy among all grids with the same number of qubits. It is less clear how to balance

parameters for the modular architecture since Corollary 2.7 does not depend on n_1 and n_2. For $n_1 \ll n_2$ or $n_2 \ll n_1$, less routing is needed, since many qubits are adjacent to one another. Thus, we take $n_1 = n_2$ in an attempt to consider a hard case. For some values of N, it may also be possible to find parameters $n_1' \neq n_2'$ such that $N \leq n_1' n_2' < \lceil \sqrt{N} \rceil^2 = n_1 n_2$, requiring fewer qubits. However, this introduces unwanted size-dependent behavior in our results when $|n_1' - n_2'| \gg 0$ for one circuit size and $n_1' \approx n_2'$ for the next, so we find it preferable to fix $n_1 = n_2$.

We compare the transformed circuits in terms of their *weighted depth* and *weighted size*. For both trapped-ion and superconducting qubits, two-qubit gates typically have longer execution times and lower fidelities than single-qubit gates [Lin+17]. Even among two-qubit gates there is a difference between execution times. Assuming fast local unitaries, the SWAP gate has 1–3 times the interaction cost of a CNOT depending on the physical interactions used to realize the gates [VHC02]. For simplicity, we assign unit cost for one-qubit gates, cost 10 for CNOT, and cost 30 for SWAP. We define the weighted size of a circuit as the sum of all gate weights and the weighted depth of a circuit as the maximum-weight path in the DAG of the circuit, where the weight of a path is the sum of the weights of the gates along it.

We consider two circuit families: random circuits and quantum signal processing (QSP) circuits [LC17]. Random circuits have been proposed for quantum computational supremacy experiments on near-term quantum devices [Boi+18; Bou+18]. Such proposals typically construct random circuits so that architecture constraints are automatically obeyed. For our purposes, random circuits provide a class of examples with little structure for circuit transformations to exploit, so we expect them to represent a hard case with large overhead. We generate a fixed set of 10 random circuits for various qubit counts. We set the number of circuit layers to 20. For each layer, we bin the qubits into pairs uniformly at random and assign each pair of qubits a Haar-random unitary from SU(4). Finally, we decompose each unitary into

the smallest possible number of CNOT + SU(2) gates [VW04]. This random circuit generator is provided by Qiskit [Abr+19].

We consider QSP circuits for Hamiltonian simulation as an example of a realistic quantum algorithm. We use the unoptimized circuits provided in [Chi+18], decomposed into Z rotations, CNOT gates, and single-qubit Clifford gates. The QSP algorithm requires precise angles that turn out to be expensive to compute. Therefore, [Chi+18] uses randomized angles instead, giving a circuit that does not correctly implement the Hamiltonian simulation. Nevertheless, the circuit corresponds to an accurate implementation of QSP, up to rotation angles, and can be used for benchmarking resources. Furthermore, the circuit transformations we construct are unaffected by those angles. We only consider one pair of *phased iterates* of the QSP algorithm ($V^\dagger_{\phi_i+\pi} V_{\phi_{i-1}}$ as in [Chi+18, Eq. 31]). A full QSP circuit for the architecture can be constructed by iterating the mapped circuit of such phased iterates, a permutation circuit between iterations, state preparation, and state unpreparation. The cost of the transformed phased iterates dominates all other costs of the construction, so the total cost can be estimated by taking our result times the number of iterations.

The circuit transformations from Section 2.2.2 are constructed from a permuter and a mapper. We denote such circuit transformations by tf: $\{\text{d,s}\} \times \mathcal{M}$, where $\mathcal{M}$ is the set of all mappers (see Table 2.2), "d" denotes an appropriate depth permuter (Section 2.3.1), and "s" denotes the general size permuter (Section 2.3.2). For example, by tf(d,greedy depth) we denote an architecture-respecting circuit transformation with a depth permuter for the architecture and the greedy depth mapper (Section 2.4.1).

2.5.2 Numerical Results

Figure 2.1 plots our results. We first consider the random circuit results. For the grid, we find that tf(d,incremental) shows much slower growth of weighted depth than circuit transformations that do not use depth-optimized permuters (Line 9). We

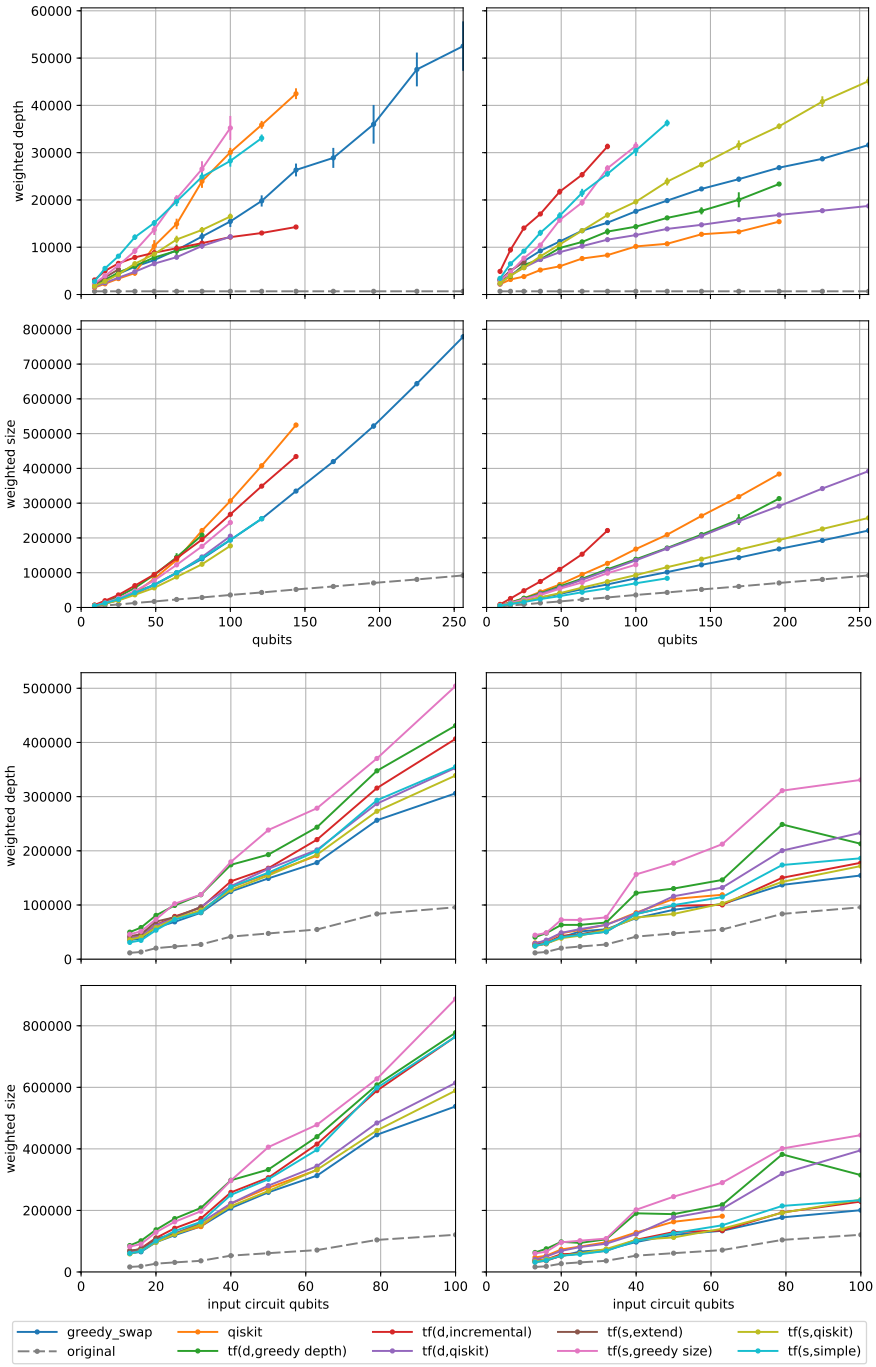


Figure 2.1: The weighted depth and weighted size for architecture-respecting random circuits (top two rows) and QSP circuits (bottom two rows) on the grid architecture (left column) and the modular architecture (right column). We generate fixed sets of 10 random circuits for increasing qubit counts and plot the mean and standard deviation for each data point. One QSP circuit is considered for each data point. The metrics for the original circuit are also given to make the overhead introduced in circuit transformations explicit; note that the original circuit does not respect the architecture constraints. The notation tf: $\{d,s\} \times \mathcal{M}$ indicates a circuit transformation constructed from either an appropriate depth ("d") permuter or the size ("s") permuter and one of our mappers (Table 2.2).

Abbreviation	Mapper name	Section
greedy depth	greedy depth mapper	2.4.1
incremental	incremental depth mapper	2.4.1
greedy size	greedy size mapper	2.4.2
simple	simple size mapper	2.4.2
extend	extension size mapper	2.4.2
qiskit	qiskit-based mapper	2.4.2

Table 2.2: The abbreviated names of the set of mappers $\mathcal{M}$ used to construct circuit transformations tf: $\{d,s\} \times \mathcal{M}$.

also note that tf(d,qiskit) performs much better than Qiskit's circuit transformation (Section 2.2.2), suggesting that depth-optimized permuters can offer a significant advantage. On the modular graph, Qiskit's circuit transformation is much better at minimizing the weighted depth, but tf(d,qiskit) starts closing the gap for larger sizes. Unfortunately, we do not know if tf(d,qiskit) performs better at larger sizes because Qiskit's circuit transformation is not fast enough to generate the relevant data. Up to 100 qubits tf(s,qiskit) achieves the best weighted size on grid architectures, and tf(s,simple) does best on modular architectures up to 121 qubits. For all sizes the greedy swap circuit transformation (Section 2.2.2) performs as one of the best at optimizing for weighted circuit size. The greedy swap circuit transformation is also able transform larger circuits within the time limit as expected from its lower time complexity.

For larger QSP circuits, the greedy circuit transformation (Section 2.2.2) is the clear winner in both weighted depth and weighted size, suggesting that it may be a good approach for practical quantum circuits. Surprisingly, tf(s,qiskit) also performs fairly well at minimizing the depth despite targeting the circuit size.

2.6 Conclusion and Future Work

We have specified various ways to efficiently transform general quantum circuits to respect architecture constraints while attempting to minimize the overhead. We investigated the classical routing problem and proposed Partial Routing via Matchings and Partial Token Swapping as models of our optimization objectives of minimizing the circuit depth and circuit size, respectively. We gave algorithms for Partial Routing via Matchings for the path graph, the complete graph, and for the generalized hierarchical products of graphs, and showed tighter bounds for certain partial permutations. We then gave more detailed analyses of special cases of the generalized hierarchical product that arise in proposed quantum architectures: the Cartesian product (e.g., for grid architectures) and the modular architecture. We also showed a 4-approximation algorithm for Partial Token Swapping on general graphs.

We constructed architecture-respecting circuit transformations with a variety of heuristic qubit placement strategies (called *mappers*). A mapper attempts to find suitable qubit placements on the architecture to execute the circuit succinctly. Given a permuter subroutine, our mappers can handle any connected simple graph. We also showed how to construct a circuit transformation from a permuter and a mapper.

Finally, we tested our circuit transformations against Qiskit's circuit transformation and a basic greedy strategy with large quantum circuits on a grid or modular architecture. When optimizing for weighted circuit size, our greedy circuit transformation was one of the best in all cases, though using our circuit transformations with algorithms for Partial Token Swapping sometimes gave a slight advantage. For the weighted circuit depth, the picture was more nuanced. We found that algorithms using Partial Routing via Matchings for classical routing could give good performance for random circuits, but Qiskit's circuit transformation and our greedy circuit transformation also performed well and gave the best results in some cases.

We would like to better understand what circuit transformations work best for which architectures, quantum algorithms, and objective functions. We also would like to use the tools of Partial Routing via Matchings and Partial Token Swapping to establish bounds on the overhead of specific quantum architectures. Ideally, we could use these tools and circuit transformations to design architectures that offer good performance subject to realistic hardware constraints and to compute realistic resource estimates for implementations of quantum algorithms.

Our mapper algorithms may be improved by including some form of lookahead to consider later layers of the given circuit, or by specializing mappers to particular architectures.

Modeling the architecture as a simple graph loses information about the underlying hardware. For example, in the IBM system the architecture edges have directionality indicating the control and target of CNOTs. In implementations of the modular architecture, the interconnecting links are probably much noisier and slower than local operations. In general, gate costs and times can vary significantly across a hardware implementation and sometimes even vary over time [Mur+19]. Adapting to variable costs and keeping track of operations performed asynchronously is challenging but could be worthwhile for architectures that support a mixture of fast and slow operations.

Finally, we hope that future progress on architecture-respecting circuit transformations will be facilitated by a suitable set of benchmarks of large quantum circuits. We publicly make available and license our source code, benchmark circuits, and results (in `TSV` format) [SUC19] and encourage others to do the same.

Chapter 3

Nearly optimal time-independent state reversal of a spin chain

This chapter is based on

[Bap+21b] Aniruddha Bapat, Eddie Schoute, Alexey V. Gorshkov, and Andrew M. Childs. "Nearly optimal time-independent reversal of a spin chain". In: *Physical Review Research* (2021). arXiv: 2003.02843v1 [quant-ph]. Forthcoming

3.1 Introduction

Quantum information transfer is a fundamental operation in quantum physics, and fast, accurate protocols for transferring quantum states across a physical system are likely to play a key role in the design of quantum computers [DiV00; Kim08]. For example, quantum information transfer can be used to establish long-range entanglement and is also useful for quantum routing. Extensive work has studied the implementation of various information transfer protocols, often via Hamiltonian dynamics on spin chains [Bos07].

Information transfer in Hamiltonian systems is governed by the spread of en-

tanglement and has close links to Lieb-Robinson bounds [LR72], entanglement area laws [ECP10], and algorithms for quantum simulation [Haa+18]. Fundamental limits to the rate of entanglement growth are set by bounds on the *asymptotic entanglement capacity* [Dür+01; Chi+03; CLV04; Ben+03] and more recent small incremental entangling theorems [Bra07; AMV13; Aud14; Mar+16]. We show that these limits can also be used to obtain lower bounds on the execution time of Hamiltonian protocols for information transfer. This raises the question of whether a protocol can achieve optimality by saturating the bound.

In this chapter, we propose the first *time-independent* protocol for state reversal using nearest-neighbor interactions that we expect has applications in noisy, connectivity-limited quantum devices. We show that the execution time of our protocol is nearly optimal, comparable to the time-dependent protocol given in [Rau05]. We also find, through simulations, that these reversal protocols have reduced error scaling in system size to noise due to static disorder caused by imperfect fabrication when compared to a SWAP-based protocol (see Section 3.4). In addition, our protocol does not require dynamical control but only engineered nearest-neighbor couplings, so we expect it to be more experimentally feasible on near-term quantum systems such as superconducting qubits [Kja+20] where dynamical control could be an additional source of noise.

Before presenting our state reversal protocol in more detail, let us elaborate on the claim that it is *nearly optimal*—specifically, that it has an evolution time within a factor $1.502(1 + 1/n)$ of the shortest possible. Under the normalization condition of interactions, our protocol achieves state reversal in time

$$t_n \coloneqq \pi\sqrt{(n+1)^2 - p(n)}/4\,, \tag{3.1}$$

where $p(n) \coloneqq n \pmod 2$. This is equivalent in time to a SWAP gate circuit of depth

$\sim n/3$. As state reversal using only SWAPs requires depth at least $n-1$ [ACG94], our protocol is faster than any SWAP-based protocol by an asymptotic factor of 3. Similarly, we can compare to other time-independent Hamiltonian protocols that use nearest-neighbor interactions: [Chr+04] implements state transfer in time $n\pi/4$ and [Alb+04] implements state reversal in time $n\pi/2$ but introduces relative phases in the state. Our time-independent protocol (and some time-dependent protocols [Rau05; FT06; KD15]) thus improve upon these previous protocols for state transfer and state reversal except for a subleading term.

We lower-bound the time for state reversal, which can generate entanglement across a bipartition, by using bounds on the asymptotic entanglement capacity in a more general model [Ben+03; Chi+03]. The asymptotic entanglement capacity bounds the rate at which entanglement can be generated by any evolution of a given bipartite Hamiltonian interspersed with arbitrary LOCC and with arbitrary finite local ancilla spaces. We give an explicit example of entanglement generated by state reversal and lower-bound the time using the capacity of a normalized two-qubit interaction in canonical form (1.2), even allowing for LOCC. Nonetheless, our state reversal protocol is able to nearly saturate this bound without classical communication, without ancillas, and with only nearest-neighbor interactions throughout the chain.

We propose a state reversal protocol with Hamiltonian of the form

$$H(\boldsymbol{J},\boldsymbol{h}) = J_0\sigma_x^1 + \sum_{k=1}^{n-1} J_k\sigma_x^k\sigma_x^{k+1} + J_n\sigma_x^n - \sum_{k=1}^{n} h_k\sigma_z^k, \tag{3.2}$$

where the coefficients $\boldsymbol{J},\boldsymbol{h}$ are engineered as follows. Letting

$$a_k := \pi\sqrt{(n+1)^2-(n+1-k)^2}/(4t_n)\,, \tag{3.3}$$

for $k\in\mathbb{N}$, our protocol is defined as (see also Figure 3.1)

Protocol 3.1. Let $J_k = a_{2k+1}, h_k = a_{2k}$ for all sites k, and let $H := H(\boldsymbol{J},\boldsymbol{h})$. Apply

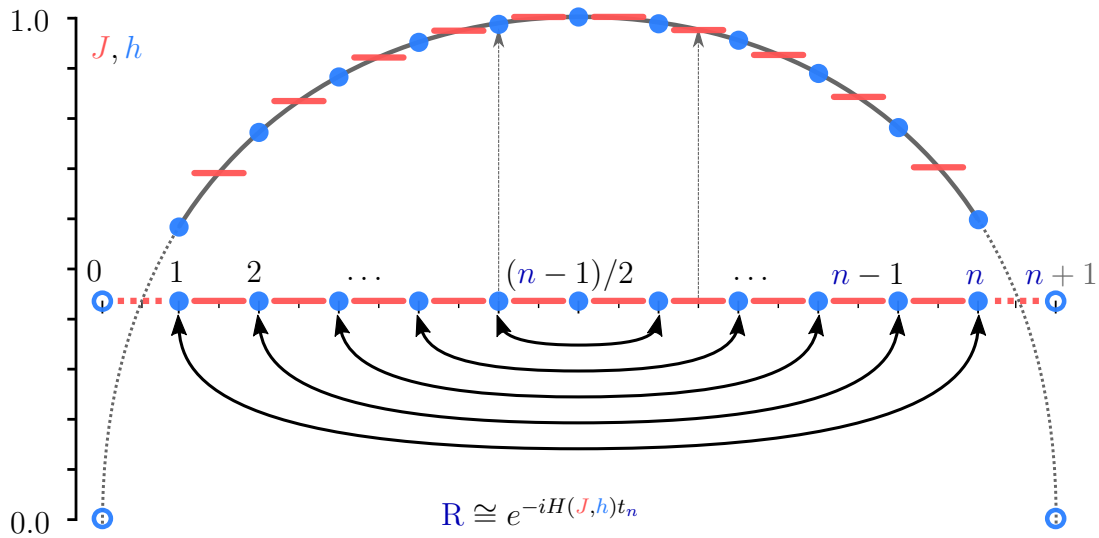


Figure 3.1: The state reversal operation R (depicted by arrows) and an illustration of our time-independent protocol to implement it. The nearest-neighbor $\sigma_x^k \sigma_x^{k+1}$ couplings (J_k, dashes) and on-site σ_z^k fields (h_k, dots) are plotted on the y-axis. Sites $0, n+1$ are ancilla qubits, which are not part of the protocol and are used purely in the analysis.

$U := e^{-it_n H}$ to the input state.

We show in the following sections that our protocol implements state reversal exactly, up to a global phase (we denote this equivalence by $\cong$). In other words,

Theorem 3.2. $U \cong \mathrm{R}$.

The rest of the chapter is organized as follows. In Section 3.2, we provide the proof of Theorem 3.2. Next, in Section 3.3, we give lower bounds on the time required to implement state reversal using normalized, local Hamiltonians. In Section 3.4, we carry out an analysis of reversal protocols under a noise model given by static disorder.

3.2 Proof and analysis of the protocol

We prove the correctness of our protocol (i.e., Theorem 3.2) by mapping the spin chain to a doubled chain of Majorana fermions via a Jordan-Wigner transformation, describing the action in the Majorana picture, and then mapping back to the spin

picture. To help with the analysis, we extend the chain with two ancillary sites $\{0, n+1\}$ called the *edge*, E, and refer to the sites $\{1, \ldots, n\}$ as the *bulk*, B. We define the transverse-field Ising model (TFIM) Hamiltonian

$$\widetilde{H} := \sum_{k=0}^{n} a_{2k+1} \sigma_x^k \sigma_x^{k+1} - \sum_{k=1}^{n} a_{2k} \sigma_z^k. \tag{3.4}$$

on the extended chain that reduces to H when the edge is initialized to state $|{+}{+}\rangle$. Similarly, we define $\widetilde{U} := e^{-i\widetilde{H}t_n}$. Note that the operator $\widetilde{H}$ (and hence $\widetilde{U}$) acts trivially on $|{+}{+}\rangle_E$, so this edge state does not change through the course of the evolution. (Our results also hold using the edge state $|{-}{-}\rangle_E$, which is equivalent to negating the sign of the longitudinal fields in (3.2).) We then prove that in the Heisenberg picture, Pauli matrices on site k map to the corresponding Pauli on site $n+1-k$ for all sites k in the chain.

First, we map to the doubled chain of Majorana fermionic operators by defining

$$\gamma_{2k} := P_{[0,k-1]} \cdot \sigma_x^k, \quad \gamma_{2k+1} := P_{[0,k-1]} \cdot \sigma_y^k \tag{3.5}$$

at each site, where we have used the notation $P_{[a,b]} := \prod_{j=a}^{b}(-\sigma_z^j)$ for the Jordan-Wigner parity string between sites a and b. The γ_k are Hermitian and satisfy the Majorana anti-commutation relations $\{\gamma_j, \gamma_k\} = 2\delta_{jk}$. We also see that $\sigma_z^k = -i\gamma_{2k}\gamma_{2k+1}$ and $\sigma_x^k \sigma_x^{k+1} = i\gamma_{2k+1}\gamma_{2k+2}$, leading (3.4) to take the form

$$\widetilde{H} = i \sum_{k=1}^{2n+1} a_k \gamma_k \gamma_{k+1} \,. \tag{3.6}$$

The Majoranas γ_0, γ_{2n+3} do not appear in the sum, since $a_0 = a_{2n+2} = 0$. In the following lemma, we show how $\widetilde{U}$ transforms the Majorana operators. Our main technique is an analogy with the dynamics of the y component of the spin operator for a spin $n + \frac{1}{2}$ particle, similar to [Alb+04; Chr+04]. Here, the same analogy provides

a protocol which gives state reversal on all spins in the chain without introducing relative phases.

Lemma 3.3. *The operation $\widetilde{U}$ acts on the Majorana operators as*

$$\widetilde{U}\gamma_k\widetilde{U}^\dagger = \begin{cases} \gamma_k & \text{if } k = 0, 2n+3, \\ (-1)^{k-1}\gamma_{2n+3-k} & \text{otherwise.} \end{cases} \tag{3.7}$$

Proof. For the first case, $\widetilde{H}$ has no overlap with operators γ_0 and γ_{2n+3}, so they are stationary under evolution by $\widetilde{H}$.

For the remaining cases, we use the analogy with a spin $s = n + \frac{1}{2}$ particle. The Heisenberg evolution of γ_k corresponds to the rotation of the S_z eigenstate $|s, k-s-1\rangle$ of magnetization $k - s - 1$. Observing that

$$\frac{i\pi}{4t_n}\langle s, m|S_y|s, m'\rangle = a_{s+m+1}(\delta_{m'(m+1)} - \delta_{m(m'+1)}) \tag{3.8}$$

(with $\hbar = 1$), we can express (3.6) in the bilinear form $\widetilde{H} = \frac{1}{2}\boldsymbol{\gamma}^\dagger A\boldsymbol{\gamma}$, for the vector $\boldsymbol{\gamma} := \begin{bmatrix}\gamma_1 & \gamma_2 & \dots & \gamma_{2n+2}\end{bmatrix}$ and the matrix $A := -\pi/(2t_n)S_y$ expressed in the S_z basis. Using the Majorana commutation relations, we have $\dot{\boldsymbol{\gamma}} = i[\widetilde{H}, \boldsymbol{\gamma}] = 2iA\boldsymbol{\gamma}$, so $\boldsymbol{\gamma}(t) = e^{2iAt}\boldsymbol{\gamma}(0)$. The Heisenberg evolution of γ_k under $\widetilde{H}$ for time t_n is exactly analogous to the (Schrödinger) time evolution of the state $|s, k-s-1\rangle$ under S_y for time π. A π-rotation under S_y maps

$$|s, -s+k-1\rangle \mapsto (-1)^{k-1}|s, s-k+1\rangle, \tag{3.9}$$

and correspondingly, $\gamma_k(t_n) = (-1)^{k-1}\gamma_{2n+3-k}$. □

Note that (3.9) can easily be verified for a spin-1/2 particle. Similarly, a spin-s particle may be viewed as a system of $2s$ spin-$\frac{1}{2}$ particles with maximal total spin. In this picture, a π-rotation under S_y corresponds to independent π-rotations of each

small spin. Since the state $|s, k-s-1\rangle$ is represented by a permutation-symmetric state with $k-1$ up spins, the π-rotation maps it to a state with $2s-(k-1)$ up spins and introduces a phase (-1) for each up spin, which is precisely (3.9).

Due to the signed reversal of the Majoranas in Lemma 3.3, the parity string $P_{[0,k]} = i^{b+1-a}\prod_{j=2a}^{2b+1}\gamma_j$ is (with the exception of γ_0) reflected about the center of the chain with an overall phase that exactly cancels when the product is reordered by increasing site index. The invariance of the edge Majoranas is crucial, as it provides a phase factor that cancels the state-dependent phases when we revert to the spin picture. In particular, we have the following lemma.

Lemma 3.4. *The operation $\widetilde{U}$ acts on the parity strings as $\widetilde{U}P_{[0,k]}\widetilde{U}^\dagger = i\sigma_x^0\sigma_x^{n+1}P_{[0,n-k]}$ for all k.*

Proof. Applying Lemma 3.3, we have

$$\widetilde{U}P_{[0,k]}\widetilde{U}^\dagger = i^{k+1}(-1)^{k(2k+1)}\gamma_0\prod_{j=1}^{2k+1}\gamma_{2n+3-j}\,. \tag{3.10}$$

$$= \gamma_0 P_{[0,n]}P_{[0,n-k]}\gamma_{2n+2} \tag{3.11}$$

where we reordered the product and used $P_{[n+1-k,n]} = P_{[0,n]}P_{[0,n-k]}$. From the Majorana anti-commutation relations and (3.5), the result follows. □

Now we prove the main theorem.

Proof of Theorem 3.2. $U \cong \mathrm{R}$ holds iff all bulk observables on the chain transform identically under U, R. For any operator $\mathcal{O}^k$ supported on bulk site $k \in \{1,\ldots,n\}$, we show that $U\mathcal{O}^kU^\dagger = \langle{+}{+}|\widetilde{U}\mathcal{O}^k\widetilde{U}^\dagger|{+}{+}\rangle_E = \mathcal{O}^{n+1-k}$. (Henceforth we drop the edge

subscript E.) By (3.5) and Lemmas 3.3 and 3.4, σ_x^k is mapped to

$$U\sigma_x^k U^\dagger = \langle ++|\widetilde{U}P_{[0,k-1]}\gamma_{2k}\widetilde{U}^\dagger|++\rangle \tag{3.12}$$

$$= -i\langle ++|\sigma_x^0\sigma_x^{n+1}P_{[0,n+1-k]}\gamma_{2n+3-2k}|++\rangle \tag{3.13}$$

$$= -i\sigma_z^{n+1-k}\sigma_y^{n+1-k} = \sigma_x^{n+1-k}\,. \tag{3.14}$$

Next, we use Lemma 3.4 to show that σ_z^k is mapped to

$$U\sigma_z^k U^\dagger = -\langle ++|\widetilde{U}P_{[0,k-1]}P_{[0,k]}\widetilde{U}^\dagger|++\rangle \tag{3.15}$$

$$= \langle ++|\sigma_x^0\sigma_x^{n+1}P_{[0,n+1-k]}\sigma_x^0\sigma_x^{n+1}P_{[0,n-k]}|++\rangle \tag{3.16}$$

$$= \sigma_z^{n+1-k}\,. \tag{3.17}$$

All other observables can be written in terms of the on-site Pauli operators σ_x^k, σ_z^k, so U is identical to R, up to global phase. □

3.3 Time lower bound

We now prove a lower bound on the optimal time, t^*, to implement state reversal using normalized local interactions. Let the entanglement entropy between systems A and B of a bipartite state $|\psi\rangle_{AB}$ be $\mathrm{E}(|\psi\rangle)$, defined as the local von Neumann entropy $S(\rho) := -\operatorname{Tr}(\rho\log_2\rho)$, for $\rho = \operatorname{Tr}_B(|\psi\rangle\langle\psi|)$. Then, the asymptotic entanglement capacity of a Hamiltonian H that couples systems A and B was shown to equal [Ben+03]

$$E_H = \sup_{|\psi\rangle\in\mathcal{H}_{AA'BB'}} \lim_{t\to 0} \frac{\mathrm{E}\big(e^{-iHt}|\psi\rangle\big) - \mathrm{E}(|\psi\rangle)}{t}\,, \tag{3.18}$$

where $\mathcal{H}_{AA'BB'}$ is the Hilbert space of the bipartite systems A and B with arbitrarily large ancilla spaces A' and B', respectively. In particular, for a Hamiltonian of the

form $\sigma_x \otimes \sigma_x$, [Dür+01; Chi+03] showed that

$$\alpha_{xx} := E_{\sigma_x \otimes \sigma_x} = 2 \max_y \sqrt{y(1-y)} \log_2 \frac{y}{1-y} \approx 1.912. \tag{3.19}$$

This is tighter than the more general small incremental entangling bound $E_H \leq \alpha \|H\| \log_2 d = 2$ for the conjectured $\alpha = 2$ [Bra07] (best known $\alpha = 4$ [Aud14]) and where the smallest dimension of A or B gives $d = 2$. Since E is invariant under local unitaries, a direct corollary is that $E_{\sigma_y \otimes \sigma_y} = E_{\sigma_z \otimes \sigma_z} = \alpha_{xx}$.

We now show that Protocol 3.1 is close to the shortest time possible.

Theorem 3.5. *It holds that* $\frac{t_n}{t^*(1+1/n)} \leq \alpha_{xx}\pi/4 < 1.502$.

Proof. We prove the time lower bound via an upper bound on the rate of increase of entanglement across a cut in the center of the chain (allowing differences of one qubit for odd n). Designate the left half of the cut as subsystem $\mathcal{A}$ and the right half as subsystem $\mathcal{B}$. $\mathcal{A}$ consists of subsystem A given by the qubit at site $\lfloor n/2 \rfloor$ adjacent to the cut, and subsystem A' consisting of the remaining qubits to the left of the cut as well as a finite but arbitrary number of ancilla systems that are not part of the chain. Similarly, $\mathcal{B}$ consists of subsystem B, the qubit at site $\lfloor n/2 \rfloor + 1$, and B', the remaining qubits in the right half with an arbitrary finite number of ancilla.

Consider Hamiltonians of the form $H(t) = K(t) + \bar{K}(t)$ specifying the evolution of the $\mathcal{AB}$ system, where $K(t)$ is a two-qubit Hamiltonian supported on systems AB (i.e., the cut edge), while $\bar{K}$ contains terms supported on AA' or BB' but not the cut edge AB. For brevity, we drop the time parameter t even though we allow the Hamiltonian to be time-dependent. We assume that K is expressed in canonical form (1.2) due to equivalence under local unitaries. Aside from its support, we make no assumptions about the form of $\bar{K}$ (so the resulting bound is more general than nearest-neighbor interactions). We call H satisfying these conditions *divisible* and also call protocols using divisible Hamiltonians divisible.

Observing that E_H is the supremum over a time derivative of the von Neumann entropy of $\rho = \mathrm{Tr}_{\mathcal{B}}(|\psi\rangle\langle\psi|)$, we have

$$E_H = \sup_{|\psi\rangle} \mathrm{Tr}\left(-\frac{d\rho}{dt}\log\rho - \rho\frac{d\log\rho}{dt}\right) \tag{3.20}$$

$$= \sup_{|\psi\rangle} \mathrm{Tr}\left(-\frac{d\rho}{dt}\log\rho\right). \tag{3.21}$$

The reduced density matrix ρ has time evolution

$$\frac{d\rho}{dt} = -i\,\mathrm{Tr}_{\mathcal{B}}([H, |\psi\rangle\langle\psi|]). \tag{3.22}$$

We substitute $H = \bar{K} + \sum_{j\in\{x,y,z\}} \mu_j\sigma_j \otimes \sigma_j$ in the commutator and substitute the time-dependence of ρ into (3.21). By linearity of the trace and sublinearity of the supremum, we get

$$E_H \leq E_{\bar{K}} + \sum_{j\in\{x,y,z\}} \mu_j E_{\sigma_j\otimes\sigma_j} \leq \alpha_{xx}\,, \tag{3.23}$$

where we observe that $E_{\bar{K}} = 0$ since $\bar{K}$ does not have support across the cut, and use the normalization condition $\sum_j |\mu_j| \leq 1$. This bound holds for all divisible Hamiltonians H, with nearest-neighbor Hamiltonians as a special case.

The entanglement generated by any divisble protocol can now be bounded in time. We observe that if the protocol contains local measurements then these cannot increase entanglement $\mathrm{E}(|\psi\rangle)$ and that feedback may be viewed as a particular time-dependence of H conditioned on measurement outcomes. Therefore, (3.23) bounds the total increase in entanglement across bipartition $\mathcal{AB}$ over a time t^* by

$$\mathrm{E}(|\psi(t^*)\rangle) - \mathrm{E}(|\psi(0)\rangle) \leq \alpha_{xx}t^* \tag{3.24}$$

for any initial state $|\psi(0)\rangle$ acted on by a divisible protocol and LOCC.

Finally, we give an explicit bound on the worst-case time of divisible state reversal

protocols by specifying an initial state. Let the system start in the product state $|\phi\rangle_{\mathcal{A}} \otimes |\phi\rangle_{\mathcal{B}}$ where each qubit forms a Bell state with a local ancilla not part of the chain. Clearly, $\mathrm{E}(|\phi\rangle_{\mathcal{A}} \otimes |\phi\rangle_{\mathcal{B}}) = 0$. We perform a reversal R on the chain and get the state $|\psi\rangle_{\mathcal{AB}} \coloneqq \mathrm{R}(|\phi\rangle_{\mathcal{A}} \otimes |\phi\rangle_{\mathcal{B}})$, which is maximally entangled, i.e., $\mathrm{E}(|\psi\rangle_{\mathcal{AB}}) = n$. Then, (3.24) gives the bound

$$t^* \geq \frac{\mathrm{E}(|\psi\rangle_{\mathcal{AB}}) - \mathrm{E}(|\phi\rangle_{\mathcal{A}} \otimes |\phi\rangle_{\mathcal{B}})}{\alpha_{xx}} \geq \frac{n}{\alpha_{xx}} \tag{3.25}$$

on any divisible state reversal protocol. Comparing this to our protocol time (3.1), we have

$$\frac{t_n}{t^*} \leq \frac{\alpha_{xx}\pi\sqrt{(n+1)^2 - p(n)}}{4n} \leq \frac{\alpha_{xx}\pi(1+1/n)}{4}\,. \qquad \square$$

3.4 Robustness of the protocol

Protocol 3.1 is exact, i.e., any input state $|\psi\rangle$ maps perfectly to the output $\mathrm{R}|\psi\rangle$, assuming the interaction coefficients are implemented exactly as prescribed. However, inherent in experimental systems is noise, and the usefulness of a given state transfer protocol is determined not only by the time of operation and fidelity under perfect implementation, but also resilience to noise. Here, we model imperfect fabrication as a static noise term on every coefficient in the Hamiltonian. We compare our time-independent protocol with a swap-based protocol for reversal (odd-even sort) and a gate-based protocol [Rau05].

Stochastic noise can be modeled as a perturbation to the Hamiltonian coefficients. For the case of disorder, we draw a single noise term for every coefficient from the normal distribution $\mathcal{N}$. We assume that the noise is multiplicative, so that the noise strength scales proportional to the magnitude of the coefficient. The perturbed

Hamiltonian H' for our time-independent protocol then looks like

$$H' = J'_0\sigma_x^1 + \sum_{k=1}^{n-1} J'_k\sigma_x^k\sigma_x^{k+1} + J'_n\sigma_x^n - \sum_{k=1}^{n} h'_k\sigma_z^k\,, \tag{3.26}$$

where $J'_i = J_i \cdot (1+\delta J_i), h'_i = h_i \cdot (1+\delta h_i)$, where $\delta h_i \sim \mathcal{N}(\delta_h), \delta J_i \sim \mathcal{N}(\delta_J)$ for specified standard deviations $\delta_h, \delta_J \geq 0$. Evolution under this Hamiltonian gives a noisy reversal $\mathrm{R}' \coloneqq e^{-iH'_i t_n}$ that reduces to R when $\delta_h = \delta_J = 0$. For swap and gate-based protocols, we compute an equivalent Hamiltonian formulation and similarly add noise terms.

A natural and widely used metric for the distinguishability of outputs of two quantum channels is the completely bounded trace norm (or diamond norm) [AKN98]. The computation of the diamond norm can be efficiently expressed as the solution to a semidefinite program [Wat09], making it a somewhat non-trivial quantity to compute. We consider unitary noise models, where the diamond distance is equivalent to a simpler notion of distinguishability, the *spectral distance*

$$\Delta \coloneqq \|\mathrm{R}' - \mathrm{R}\|\,, \tag{3.27}$$

where we take the spectral norm of the difference between perfect and noisy state reversals R and R'. In this case, the diamond distance is at most two times as large as the spectral distance [Kit97]. The distance Δ can be used to bound another common figure of merit, the *fidelity*

$$\mathrm{F}(\rho,\sigma) = \mathrm{Tr}\left(\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}}\right), \tag{3.28}$$

for output states ρ and σ evolved by a perfect and noisy reversal, respectively.

We will prove a bound on the minimum fidelity for completeness here but do not claim novelty of the result. First, we bound Δ by the minimum fidelity over pure

states as follows:

$$\Delta^2 = \|(\mathrm{R}-\mathrm{R}')^\dagger(\mathrm{R}-\mathrm{R}')\| \tag{3.29}$$

$$= \max_{|\psi\rangle} |\langle\psi|(\mathrm{R}-\mathrm{R}')^\dagger(\mathrm{R}-\mathrm{R}')|\psi\rangle| \tag{3.30}$$

$$= \max_{|\psi\rangle} |\langle\psi|2\mathbb{1} - \mathrm{R}^\dagger\,\mathrm{R}' - \mathrm{R}'^\dagger\,\mathrm{R}|\psi\rangle| \tag{3.31}$$

$$= \max_{|\psi\rangle} |2 - 2\,\mathrm{Re}\,\langle\psi|\,\mathrm{R}^\dagger\,\mathrm{R}'|\psi\rangle| \tag{3.32}$$

$$= 2 - \min_{|\psi\rangle} 2\,\mathrm{Re}\,\langle\psi|\,\mathrm{R}^\dagger\,\mathrm{R}'|\psi\rangle \tag{3.33}$$

$$\geq 2 - 2\min_{|\psi\rangle}|\langle\psi|R^\dagger R'|\psi\rangle|\,, \tag{3.34}$$

where we used the fact that for any unitary U, $\mathrm{Re}\,\langle\psi|U|\psi\rangle \leq 1$, and $\mathrm{Re}\,[z] \leq |z|$ for any $z \in \mathbb{C}$. Let $F_{\min}$ denote the worst-case fidelity over all input states. By the joint concavity of the fidelity [Wat18, Corollary 3.26], $F_{\min}$ is attained for a pure state, thus

$$F_{\min} = \min_{|\psi\rangle} \mathrm{F}(R'|\psi\rangle, R|\psi\rangle) = |\langle\psi|R^\dagger R'|\psi\rangle| \tag{3.35}$$

since $\mathrm{F}(|\phi_1\rangle, |\phi_2\rangle) = |\langle\phi_2|\phi_1\rangle|$ for pure states $|\phi_1\rangle$ and $|\phi_2\rangle$. It then follows from (3.34) that $F_{\min} \geq 1 - \frac{1}{2}\Delta^2$. We estimate the spectral distance dependence on noise and system size in the three candidate protocols [SB21]. For each protocol, we probe the distance as a function of similar on-site and coupling disorder $\delta = \delta_h = \delta_J$, and increasing number of spins n. The spectral distance is computed by exact diagonalization, taking time exponential in n, and it is possible to probe system sizes up to $n = 14$ with the resources available. At these sizes, we can already see differences between the protocols, shown in Figure 3.2. At each error rate δ, the swap protocol has the worst performance, the time-independent protocol performs better, and the gate-based protocol has the best performance. We note that the gate-based and time-independent protocols perform within a standard deviation of one another, but the SWAP protocol

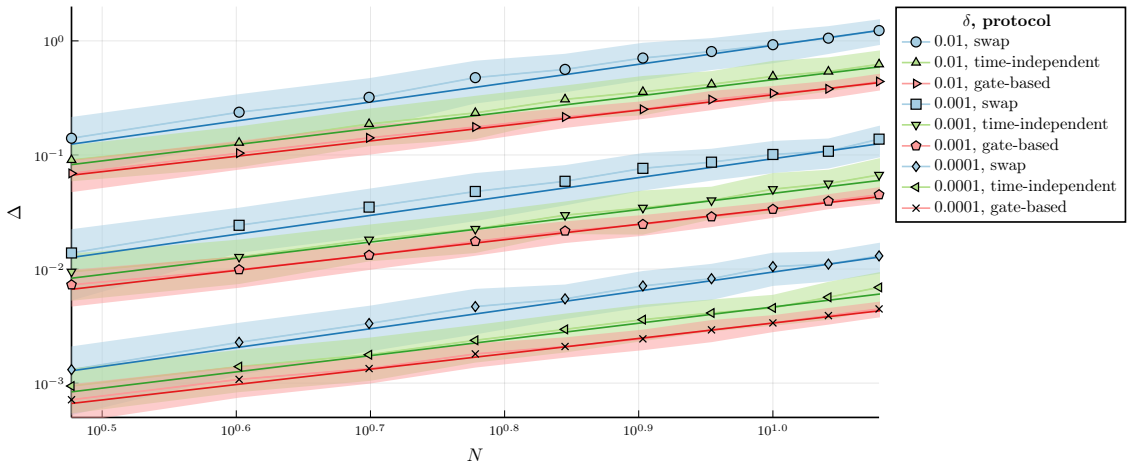


Figure 3.2: Spectral distance mean values with standard deviation (shaded region) for different protocols under varying strengths of noise. We take 100 samples for each data point and use a linear fit for a power law $\Delta = \exp(n^a \delta^b)$ controlled on the protocol, i.e., fitting $\log \Delta = a \log n + b \log \delta + O(1)$, to find (standard error) $a \approx 1.66(0.012)$ and $b \approx 0.994(0.0028)$ for the SWAP-based protocol. The b coefficient changes insignificantly for time-independent and gate-based protocols but the a coefficient is reduced by 0.31(0.016) for gate-based and 0.23(0.016) for time-independent protocols, indicating more robust scaling of these protocols in system size, relative to a SWAP-based protocol.

is significantly noisier. For example, at a threshold of $\Delta \leq 0.03$, the swap can reverse only up to 4 sites, while the time-independent protocol can successfully reverse 8 sites. Therefore, the specialized protocols for reversal improve upon SWAP-based protocols not only in runtime but also in accuracy.

The relative performance of time-independent and gate-based protocols (including the SWAP protocol) may not be captured by our simulations. Since the time-independent protocol is static, it derives its error primarily from imperfect engineering of the coupling strengths and interactions with the environment. Gate-based protocols, however, require dynamical control, which could be an additional source of noise. Since this noise source is likely to worsen the performance of discrete protocols, we cannot make a definite comparison between our protocol and gate-based protocols in our noise model.

3.5 Discussion

The time-dependent protocol in [Rau05] is closely related to our time-independent protocol, and both can be described within the same framework. In the time-dependent case, the state is evolved alternately under two restrictions of the Hamiltonian (3.2): $H(\mathbf{1}, \mathbf{0})$ (uniform Ising) and $H(\mathbf{0}, \mathbf{1})$ (uniform transverse field), each for time $\pi/4$, for a total of $n + 1$ rounds. In the Majorana picture, these Hamiltonians carry out a simultaneous braiding of neighboring Majoranas along even (resp. odd) edges of the doubled Majorana chain. The resulting map matches Lemma 3.3 exactly, implying that the two protocols are identical at the level of Majorana operators. Indeed, any protocol achieving the map in Lemma 3.3 is guaranteed to implement state reversal.

While a superconstant speedup is not possible in one dimension, our techniques suggest that routing protocols for higher-dimensional systems might be found by exploiting similar mappings between spins and fermions [YK07; Kit06; Che20]. While state transfer in these systems has been studied [Yao+13], the more general questions of upper and lower bounds on routing remain open, and our bounds based on the entanglement capacity might yield new insights.

Chapter 4

Quantum routing with fast reversals

This chapter is based on

[Bap+21a] Aniruddha Bapat, Andrew M. Childs, Alexey V. Gorshkov, Samuel King, Eddie Schoute, and Hrishee Shastri. "Quantum routing with fast reversals". In: *Quantum* 5 (Aug. 2021), p. 533. DOI: 10.22331/q-2021-08-31-533

4.1 Introduction

In this chapter, we give a Hamiltonian routing protocol for the path architecture graph. We also give a protocol on general graphs for routing sparse permutations, i.e. permutations with few non-trivial mappings. Rather than directly engineering a quantum routing protocol, we consider a hybrid strategy that leverages fast state reversal, R, (Chapter 3) to implement Hamiltonian quantum routing. The reversal operation can be implemented in SWAP-normalized time (3.1)

$$T(\mathrm{R}) \leq \frac{\sqrt{(n+1)^2 - p(n)}}{3} \leq \frac{n+1}{3}, \tag{4.1}$$

where $p(n) \in \{0, 1\}$ is the parity of n. The Hamiltonian protocol of Chapter 3 can be understood by looking at the time evolution of the site Majorana operators obtained by

a Jordan-Wigner transformation of the spin chain. In this picture, the protocol can be interpreted as the rotation of a fictitious particle of spin $n + 1/2$ whose magnetization components are in one-to-one correspondence with the Majoranas on the chain. A reversal corresponds to a rotation of the large spin by an angle of π. The gate-based reversal protocol [Rau05] is a special case of a quantum cellular automaton with a transition function given by the $(n + 1)$-fold product of nearest-neighbor controlled-Z (CZ) operations—an operation that can be done 3 times faster than a SWAP gate—and Hadamard operations. In an open spin chain, this process spreads out local Pauli observables at site i over the chain and "refocuses" them at site $n + 1 - i$ in $n + 1$ steps for every i. The ability to spread local observables (which is present in the gate-based and Hamiltonian protocols but not in SWAP-based protocols) may be key to obtaining a speedup over SWAP-based algorithms.

Routing using reversals has been studied extensively due to its applications in comparative genomics (where it is known as *sorting by reversals*) [BP93; KS95]. References [Ben+08; PS02; NNN05] present routing algorithms where, much like in our case, reversals have length-weighted costs. However, these models assume reversals are performed sequentially, while we assume independent reversals can be performed in parallel, where the total cost is given by the evolution time, akin to circuit depth. To our knowledge, results from the sequential case are not easily adaptable to the parallel setting and require a different approach.

Routing on paths is a fundamental building block for routing on more general graphs. For example, a two-dimensional grid graph is the Cartesian product of two path graphs, and the best known routing routine applies a path routing subroutine 3 times [ACG94]. A Hamiltonian routing protocol on the path of time cn, for a constant $c > 0$, would imply a Hamiltonian routing time of $3cn$ on the grid. A similar speedup follows for higher-dimensional grids. More generally, routing algorithms for the generalized hierarchical product of graphs can take advantage of faster routing of

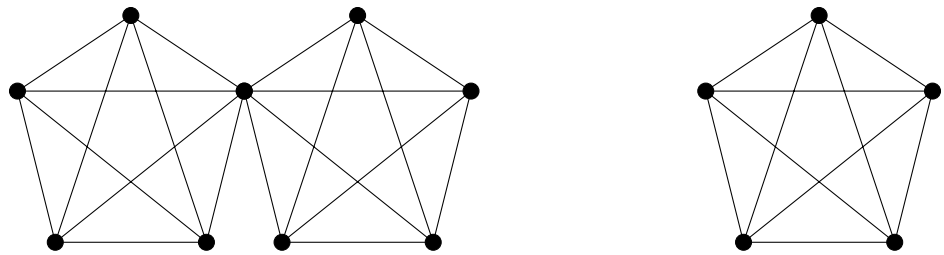
the path base graph (Theorem 2.4). For other graphs, it is open whether fast reversals can be used to give faster routing protocols for general permutations.

In this chapter, we present the following results on quantum routing using fast reversals. In Section 4.2, we give basic examples of using fast reversals to perform routing on general graphs to indicate the extent of possible speedup over SWAP-based routing, namely a graph for which routing can be sped up by a factor of 3, and another for which no speedup is possible. Section 4.3 presents algorithms for routing sparse permutations, where few qubits are routed, both for paths and for more general graphs. Here, we obtain the full factor 3 speedup over SWAP-based routing. Then, in Section 4.4, we prove the main result that there is a quantum routing algorithm for the path with worst-case constant-factor advantage over any SWAP-based routing scheme. Finally, in Section 4.5, we show that our algorithm has average-case routing time $2n/3 + o(n)$ and any SWAP-based protocol has average-case routing time at least $n - o(n)$.

4.2 Simple bounds on routing using reversals

Given the ability to implement a fast reversal R with cost given by (4.1), the largest possible asymptotic speedup of reversal-based routing over SWAP-based routing is a factor of 3. This is because the reversal operation, which is a particular permutation, cannot be performed faster than $n/3+o(n)$, and can be performed in time n classically using odd-even sort. As we now show, some graphs can saturate the factor of 3 speedup for general permutations, while other graphs do not admit any speedup over SWAPs.

Maximal speedup: For n odd, let K_n^* denote two complete graphs, each on $(n+1)/2$ vertices, joined at a single "junction" vertex for a total of n vertices (Figure 4.1a). Consider a permutation on K_n^* in which every vertex is sent to the other complete



(a) Joined graph K_9^*. **(b)** Complete graph K_5.

Figure 4.1: K_9^* admits the full factor of 3 speedup in the worst case when using reversals over SWAPs, whereas K_5 admits no speedup when using reversals over SWAPs.

subgraph, except that the junction vertex is sent to itself. To route with SWAPs, note that each vertex (other than that at the junction) must be moved to the junction at least once, and only one vertex can be moved there at any time. Because there are $(n+1)/2-1$ non-junction vertices on each subgraph, implementing this permutation requires a SWAP-circuit depth of at least $n-1$.

On the other hand, any permutation on K_n^* can be implemented in time $n/3+O(1)$ using reversals. First, perform a reversal on a path that connects all vertices with opposite-side destinations. After this reversal, every vertex is on the side of its destination and the remainder can be routed in at most 2 steps [ACG94]. The total time is at most $(n+1)/3+2$, exhibiting the maximal speedup by an asymptotic factor of 3.

No speedup: Now, consider the complete graph on n vertices, K_n (Figure 4.1b). Every permutation on K_n can be routed in at most time 2 using SWAPs [ACG94]. Consider implementing a 3-cycle on three vertices of K_n for $n \geq 3$ using reversals. Any reversal sequence that implements this permutation will take at least time 2. Therefore, no speedup is gained over SWAPs in the worst case.

We have shown that there exists a family of graphs that allows a factor of 3 speedup for any permutation when using fast reversals instead of SWAPs, and others where reversals do not grant any improvement. The question remains as to where the path

graph lies on this spectrum. Faster routing on the path is especially desirable since this task is fundamental for routing in more complex graphs.

4.3 An algorithm for sparse permutations

We now consider routing sparse permutations, where only a small number k of qubits are to be moved. For the path, we show that the routing time is at most $n/3 + O(k^2)$. More generally, we show that for a graph G of radius

$$r = \min_{u \in V(G)} \max_{v \in V(G)} d(u, v), \tag{4.2}$$

the routing time is at most $2r/3 + O(k^2)$. Our approach to routing sparse permutations using reversals is based on the idea of bringing all k qubits to be permuted to the center of the graph, rearranging them, and then sending them to their respective destinations.

4.3.1 Paths

A description of the algorithm on the path, called `MiddleExchange`, appears in Algorithm 4.3.1. Figure 4.2 presents an example of `MiddleExchange` for $k = 6$.

In Theorem 4.1, we prove that Algorithm 4.3.1 achieves a routing time of asymptotically $n/3$ when implementing a sparse permutation of $k = o(\sqrt{n})$ qubits on the path graph. First, let $\mathcal{S}_n$ denote the set of permutations on $\{1, \ldots, n\}$, so $|\mathcal{S}_n| = n!$. Then, for any permutation $\pi \in \mathcal{S}_n$ that acts on a set of labels $\{1, \ldots, n\}$, let π_i denote the destination of label i under π. We may then write $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$. Let $\bar{\mathrm{R}}$ denote an ordered series of reversals $\mathrm{R}_1, \ldots, \mathrm{R}_m$, and let $\bar{\mathrm{R}}_1 \mathbin{+\!\!+} \bar{\mathrm{R}}_2$ be the concatenation of two reversal series. Finally, let $S \cdot \mathrm{R}$ and $S \cdot \bar{\mathrm{R}}$ denote the result of applying R and $\bar{\mathrm{R}}$ to a sequence S, respectively, and let $|\mathrm{R}|$ denote the length of the reversal R, i.e., the number of vertices it acts on.

Input : π, a permutation

function `MiddleExchange`(π):

  identify the labels $x_1, \ldots, x_k \in [n]$ to be permuted, with $x_i < x_{i+1}$

  let t be the largest index for which $x_t \leq \lfloor n/2 \rfloor$, i.e. the last label x_t left of the median

  for $i = 1$ *to* $t - 1$ **:**

    join the labels $x_1, \ldots, x_i$ to x_{i+1} using at most one reversal R_i, not containing x_{i+1}

  for $j = k$ *to* $t + 2$ **:**

    join the labels $x_j, \ldots, x_k$ to x_{j-1} using at most one reversal R_j, not containing x_j

  using at most two reversals R_t and R_{t+1}, move the labels $x_1, \ldots, x_t$ and $x_{t+1}, \ldots, x_k$ to the median

  $\bar{\mathrm{R}} := \mathrm{R}_1, \ldots, \mathrm{R}_{t-1}, \mathrm{R}_k, \ldots, \mathrm{R}_{t+2}, \mathrm{R}_t, \mathrm{R}_{t+1}$ `// The reversals performed so far, in order`

  route the labels $x_1, \ldots, x_k$ such that after performing $\bar{\mathrm{R}}$ in reverse order, each label is at its destination

  perform $\bar{\mathrm{R}}$ in reverse order

Algorithm 4.3.1: MiddleExchange algorithm to sort sparse permutations on the path graph.

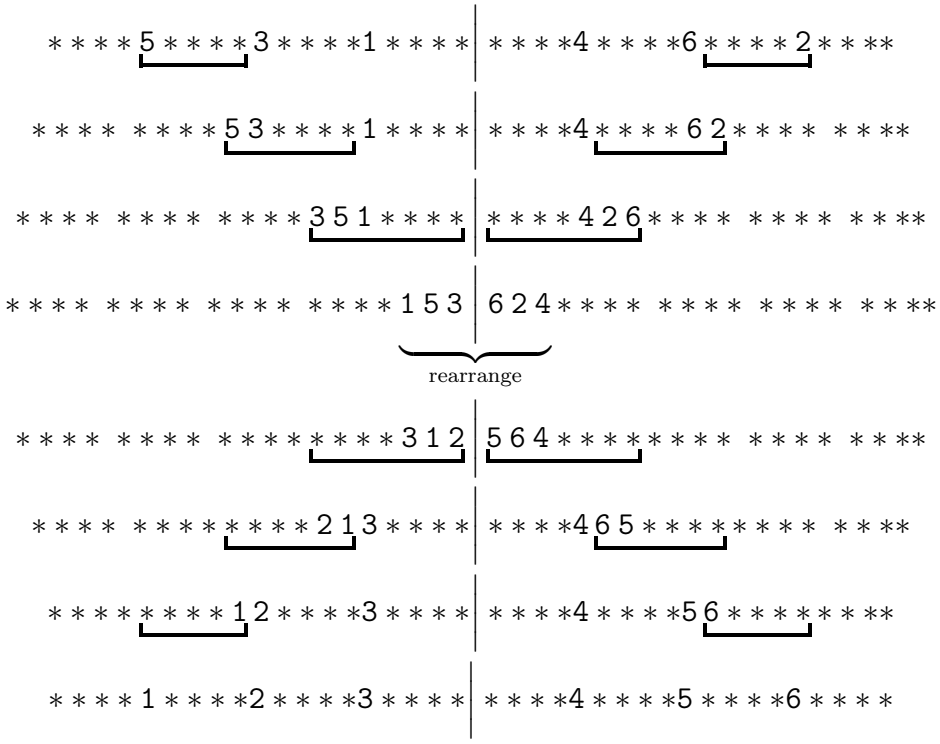


Figure 4.2: Example of `MiddleExchange` (Algorithm 4.3.1) on the path for $k = 6$.

Theorem 4.1. *Let $\pi \in \mathcal{S}_n$ with $k = |\{x \in [n] \mid \pi_x \neq x\}|$ (i.e., k elements are to be permuted, and $n-k$ elements begin at their destination). Then Algorithm 4.3.1 routes π in time at most $n/3 + O(k^2)$.*

Proof. Algorithm 4.3.1 consists of three steps: compression (Line 4–Line 8), inner permutation (Line 10), and dilation (Line 11). Notice that compression and dilation are inverses of each other.

Let us first show that Algorithm 4.3.1 routes π correctly. Just as in the algorithm, let $x_1, \ldots, x_k$ denote the labels $x \in [n]$ with $x_i < x_{i+1}$ such that $\pi_x \neq x$, that is, the elements that do not begin at their destination and need to be permuted. It is easy to see that these elements are permuted correctly: After compression, the inner permutation step routes x_i to the current location of the label π_{x_i} in the middle. Because dilation is the inverse of compression, it will then route every x_i to its correct destination. For the non-permuting labels, notice that they lie in the support of either no reversal or exactly two reversals, R_1 in the compression step and R_2 in the dilation step. Therefore R_1 reverses the segment containing the label and R_2 re-reverses it back into place (so $\mathrm{R}_1 = \mathrm{R}_2$). Therefore, the labels that are not to be permuted end up exactly where they started once the algorithm is complete.

Now we analyze the routing time. Let $d_i = x_{i+1} - x_i - 1$ for $i \in [k-1]$. As in the algorithm, let t be the largest index for which $x_t \leq \lfloor n/2 \rfloor$. Then, for $1 \leq i \leq t-1$, we have $|\mathrm{R}_i| = d_i + i$, and, for $t+2 \leq j \leq k$, we have $|\mathrm{R}_j| = d_{j-1} + k - j$. Moreover, we have $|\mathrm{R}_t| = \lfloor n/2 \rfloor - x_t - 1 + t$ and $|\mathrm{R}_{t+1}| = x_{t+1} - \lfloor n/2 \rfloor + k - t$. From all reversals in the first part of Algorithm 4.3.1, $\bar{\mathrm{R}}$, consider those that are performed on the left

side of the median (position $\lfloor n/2 \rfloor$ of the path). The routing time of these reversals is

$$\begin{aligned}
\frac{1}{3}\sum_{i=1}^{t}|\mathrm{R}_i| + 1 &= \frac{1}{3}\left(\lfloor n/2 \rfloor - x_t - 1\right) + \frac{1}{3}\sum_{i=1}^{t}\left(d_i + i + 1\right) \\
&= \frac{t(t+1)}{6} + \frac{1}{3}\left(\lfloor n/2 \rfloor - x_t - 1\right) + \sum_{i=1}^{t}\left(x_{i+1} - x_i\right) \\
&= O(t^2) + \frac{1}{3}\left(\lfloor n/2 \rfloor - x_1\right) \\
&\leq \frac{n}{6} + O(k^2).
\end{aligned} \tag{4.3}$$

By a symmetric argument, the same bound holds for the compression step on the right half of the median. Because both sides can be performed in parallel, the total cost for the compression step is at most $n/6 + O(k^2)$. The inner permutation step can be done in time at most k using odd-even sort. The cost to perform the dilation step is also at most $n/6 + O(k^2)$ because dilation is the inverse of compression. Thus, the total routing time for Algorithm 4.3.1 is at most $2(n/6 + O(k^2)) + k = n/3 + O(k^2)$. □

It follows that sparse permutations on the path with $k = o(\sqrt{n})$ can be implemented using reversals with a full asymptotic factor of 3 speedup.

4.3.2 General graphs

We now present a more general result for implementing sparse permutations on an arbitrary graph.

Theorem 4.2. *Let $G = (V, E)$ be a graph with radius r and π a permutation of vertices. Let $S = \{v \in V : \pi_v \neq v\}$. Then π can be routed in time at most $2r/3 + O(|S|^2)$.*

Proof. We route π using a procedure similar to Algorithm 4.3.1, consisting of the same three steps adapted to work on a spanning tree of G: compression, inner permutation, and dilation. Dilation is the inverse of compression and the inner permutation step can be performed on a subtree consisting of just $k = |S|$ nodes by using the Routing

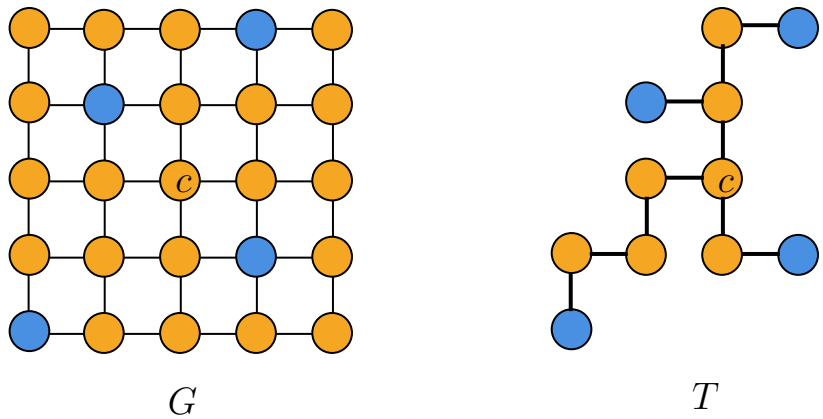


Figure 4.3: Illustration of the token tree $\mathcal{T}$ in Theorem 4.2 for a case where G is the 5×5 grid graph. Blue circles represent vertices in S and orange circles represent vertices not in S. Vertex c denotes the center of G. Red-outlined circles represent intersection vertices. In particular, note that one of the blue vertices is an intersection because it is the first common vertex on the path to c of two distinct blue vertices.

VIA MATCHINGS algorithm for trees in $3k/2 + O(\log k)$ time [Zha99]. It remains to show that compression can be performed in $r/3 + O(k^2)$ time.

We construct a *token tree* $\mathcal{T}$ that reduces the compression step to routing on a tree. Let c be a vertex in the *center* of G, i.e., a vertex with distance at most r to all vertices. Construct a shortest-path tree $\mathcal{T}'$ of G rooted at c, say, using breadth-first search. We assign a token to each vertex in S. Now $\mathcal{T}$ is the subtree of $\mathcal{T}'$ formed by removing all vertices $v \in V(\mathcal{T}')$ for which the subtree rooted at v does not contain any tokens, as depicted in Figure 4.3. In $\mathcal{T}$, call the first common vertex between paths to c from two distinct tokens an *intersection* vertex, and let $\mathcal{I}$ be the set of all intersection vertices. Note that if a token t_1 lies on the path from another token t_2 to c, then the vertex on which t_1 lies is also an intersection vertex. Since $\mathcal{T}$ has at most k leaves, $|\mathcal{I}| \leq k - 1$.

For any vertex v in $\mathcal{T}$, let the *descendants* of v be the vertices $u \neq v$ in $\mathcal{T}$ whose path on $\mathcal{T}$ to c includes v. Now let $\mathcal{T}_v$ be the subtree of $\mathcal{T}$ rooted at v, i.e., the tree composed of v and all of the descendants of v. We say that all tokens have been *moved up to* a vertex v if for all vertices u in $\mathcal{T}_v$ without a token, T_u also does not contain a

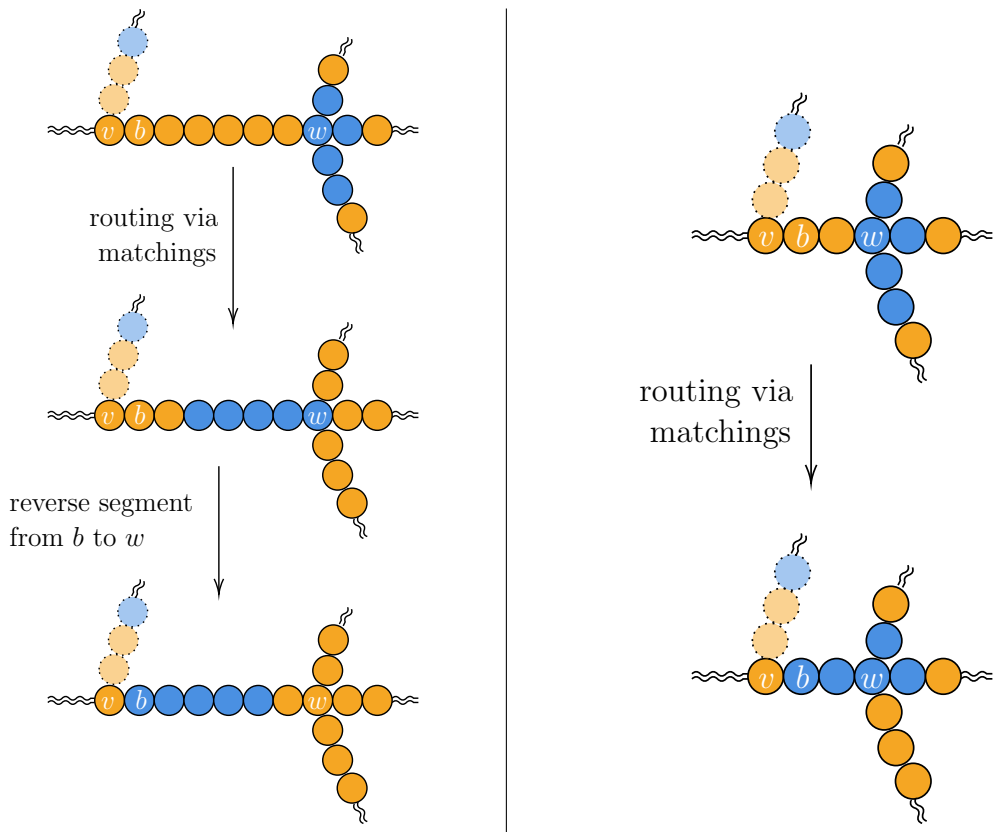
```
Input  : A vertex v in tree T
function MoveUpTo(v):
    if T_v contains only one token then
        Perform a reversal on the segment starting at the leaf node of T_v and
          ending at v.
        return
    for each child b of v :
        w := The intersection vertex in T_b closest to b          // includes b
        MoveUpTo(w)
        m := the number of tokens from S in T_b
        l(p) := the length of the path p from w to b in T_v
        if l(p) ≥ m then   // Enough room on p, form a path of tokens
          at b
            Route the m tokens in T_b to the first m vertices of p using
              Routing via Matchings.
            Perform a reversal on the segment starting at w and ending at b.
        else  // Not enough room on p, form a tree of tokens rooted
          at b
            Route the m tokens in T_b as close as possible to b using Routing
              via Matchings.
    if v has no token then
        Perform a reversal on the segment starting from v and ending at a
          vertex u in T_v such that u has a token in S and no descendent of u
          has a token in S, if such a u exists.
```

Algorithm 4.3.2: An algorithm that recursively moves all tokens from S that lie on T_v up to an intersection vertex v.

token. The compression step can then be described as moving tokens up to c.

We describe a recursive algorithm for doing so in Algorithm 4.3.2. The base case considers the trivial case of a subtree with only one token. Otherwise, we move all tokens on the subtrees of descendant b up to the closest intersection w using recursive calls as illustrated in Figure 4.4. Afterwards, we need to consider whether the path p between v and w has enough room to store all tokens. If it does, we use a Routing via Matchings algorithm for trees to route tokens from w onto p, followed by a reversal to move these tokens up to v. Otherwise, the path is short enough to move all tokens up to v by the same Routing via Matchings algorithm.

We now bound the routing time on $\mathcal{T}_{w_1}$ of `MoveUpTo(`w_1`)`, for any vertex $w_1 \in V(\mathcal{T})$.



(a) When $l(p) \geq m$. In this case, $l(p) = 7 \geq 5 = m$.

(b) When $l(p) < m$. In this case, $l(p) = 3 < 5 = m$.

Figure 4.4: An example of moving the m tokens in $\mathcal{T}_w$ up to b (Line 10–Line 14 in Algorithm 4.3.2).

First note that all operations on subtrees $\mathcal{T}_b$ of $\mathcal{T}_{w_1}$ are independent and can be performed in parallel. Let $w_1, w_2, \ldots, w_t$ be the sequence of intersection vertices that `MoveUpTo(·)` is recursively called on that dominates the routing time of `MoveUpTo(`w_1`)`. Let d_w, for $w \in V(\mathcal{T}_{w_1})$, be the distance of w to the furthest leaf node in $\mathcal{T}_w$. Assuming that the base case on Line 3 has not been reached, we have a routing time of

$$T(w_1) \leq T(w_2) + \frac{d_{w_1} - d_{w_2}}{3} + O(k), \tag{4.4}$$

where $O(k)$ bounds the time required to route $m \leq k$ tokens on a tree of size at most $2m$ following the recursive `MoveUpTo(`w_2`)` call [Zha99]. We expand the time cost $T(w_i)$ of recursive calls until we reach the base case of w_t to obtain

$$T(v) \leq T(w_t) + \sum_{i=1}^{t-1} \left(\frac{d_{w_i} - d_{w_{i+1}}}{3} + O(k) \right) = T(w_t) + \frac{d_{w_1} - d_{w_t}}{3} + t \cdot O(k) \tag{4.5}$$

$$\leq \frac{d_{w_1}}{3} + (t+1)O(k). \tag{4.6}$$

Since $d_v \leq r$ and $t \leq k$, this shows that compression can be performed in $r/3 + O(k^2)$ time. □

In general, a graph with radius r and diameter d will have $d/2 \leq r \leq d$. Using Theorem 4.2, this implies that for a graph G and a sparse permutation with $k = o(\sqrt{r})$, the bound for the routing time will be between $d/3 + o(d)$ and $2d/3 + o(d)$. Thus, for such sparse permutations, using reversals will always asymptotically give us a constant-factor worst-case speedup over any SWAP-only protocol since $\mathrm{rt}(G) \geq d$. Furthermore, for graphs with $r = d/2$, we can asymptotically achieve the full factor of 3 speedup.

```
Input  : π, a permutation of a contiguous subset of [n].
function GenericDivideConquer(BinarySorter, π):
    if |π| = 1 then
        return ∅
    B := BinaryLabeling(π)
    R̄ := BinarySorter(B)
    π := π · R̄
    R̄ = R̄ ⧺ GenericDivideConquer(BinarySorter, π[0, ⌊n/2⌋])
    R̄ = R̄ ⧺ GenericDivideConquer(BinarySorter, π[⌊n/2⌋ + 1, |π|])
    return R̄
```

Algorithm 4.4.1: Divide-and-conquer algorithm for recursively sorting π. `BinaryLabeling`(π) is a subroutine that uses (4.7) to transform π into a bitstring, and `BinarySorter` is a subroutine that takes as input the resulting binary string and returns an ordered reversal sequence $\bar{\mathrm{R}}$ that sorts it.

4.4 Algorithms for routing on the path

Our general approach to implementing permutations on the path relies on the divide-and-conquer strategy described in Algorithm 4.4.1. It uses a correspondence between implementing permutations and sorting binary strings, where the former can be performed at twice the cost of the latter. This approach is inspired by [PS02] and [Ben+08] who use the same method for routing by reversals in the sequential case.

First, we introduce a binary labeling using the indicator function

$$I(v) = \begin{cases} 0 & \text{if } v < n/2, \\ 1 & \text{otherwise.} \end{cases} \tag{4.7}$$

This function labels any permutation $\pi \in \mathcal{S}_n$ by a binary string

$$I(\pi) \coloneqq (I(\pi_1), I(\pi_2), \ldots, I(\pi_n)). \tag{4.8}$$

Let π be the target permutation, and σ any permutation such that $I(\pi\sigma^{-1}) = (0^{\lfloor n/2 \rfloor} 1^{\lceil n/2 \rceil})$. Then it follows that σ divides π into permutations π_L, π_R acting only

on the left and right halves of the path, respectively, i.e., $\pi = \pi_L \cdot \pi_R \cdot \sigma$. We find and implement σ via a binary sorting subroutine, thereby reducing the problem into two subproblems of length at most $\lceil n/2 \rceil$ that can be solved in parallel on disjoint sections of the path. Proceeding by recursion until all subproblems are on sections of length at most 1, the only possible permutation is the identity and π has been implemented. Because disjoint permutations are implemented in parallel, the total routing time is $T(\pi) = T(\sigma) + \max(T(\pi_L), T(\pi_R))$.

We illustrate Algorithm 4.4.1 with an example, where the binary labels are indicated below the corresponding destination indices:

$$
\begin{array}{ccccccccccccc}
\mathtt{7\,6\,0\,2\,5\,1\,3\,4} & \xrightarrow{\text{label}} & \underline{\mathtt{7\,6\,0\,2\,5\,1\,3\,4}} & \xrightarrow{\text{sort}} & \underline{\mathtt{0\,3\,1\,2\,5\,7\,6\,4}} & \xrightarrow{\text{label}} & \underline{\mathtt{0\,3\,1\,2\ \ 5\,7\,6\,4}} \\
 & & \mathtt{1\,1\,0\,0\,1\,0\,0\,1} & & \mathtt{0\,0\,0\,0\,1\,1\,1\,1} & & \mathtt{0\,1\,0\,1\ \ 0\,1\,1\,0} \\
 & & & & & & \downarrow \text{sort} \\
\underline{\mathtt{0\,1}}\ \ \underline{\mathtt{2\,3}}\ \ \underline{\mathtt{4\,5}}\ \ \underline{\mathtt{6\,7}} & \xleftarrow{\text{sort}} & \underline{\mathtt{0\,1}}\ \ \underline{\mathtt{3\,2}}\ \ \underline{\mathtt{5\,4}}\ \ \underline{\mathtt{6\,7}} & \xleftarrow{\text{label}} & \underline{\mathtt{0\,1\,3\,2\ \ 5\,4\,6\,7}} & & \\
\mathtt{0\,1\ \ 0\,1\ \ 0\,1\ \ 0\,1} & & \mathtt{0\,1\ \ 1\,0\ \ 1\,0\ \ 0\,1} & & \mathtt{0\,0\,1\,1\ \ 0\,0\,1\,1} & &
\end{array}
\tag{4.9}
$$

Each labeling and sorting step corresponds to an application of (4.7) and `BinarySorter`, respectively, to each subproblem. Specifically, in (4.9), we use TBS (Algorithm 4.4.2) to sort binary strings.

We present two algorithms for `BinarySorter`, which perform the work in our sorting algorithm. The first of these binary sorting subroutines is Tripartite Binary Sort (TBS, Algorithm 4.4.2). TBS works by splitting the binary string into nearly equal (contiguous) thirds, recursively sorting these thirds, and merging the three sorted thirds into one sorted sequence. We sort the outer thirds forwards and the middle third backwards which allows us to merge the three segments using at most

```
Input  : B, a binary string
function TripartiteBinarySort(B):
    if |B| = 1 then
        return ∅
    m_1 := ⌊|B|/3⌋
    m_2 := ⌊2|B|/3⌋
    R̄ := TripartiteBinarySort(B[0, m_1])
    R̄ := R̄ ++ TripartiteBinarySort(B[m_1 + 1, m_2] ⊕ 11...11)        // ⊕ being bitwise-XOR, so we sort the middle third backwards
    R̄ := R̄ ++ TripartiteBinarySort(B[m_2 + 1, |B|])
    i := index of first 1 in B
    j := index of last 0 in B
    B := B · R(i, j)
    return R̄ ++ R(i, j)
```

Algorithm 4.4.2: Tripartite Binary Sort (TBS). We let $\mathrm{R}(i,j)$ denote a reversal on the subsequence $S[i,j]$, i,j inclusive.

one reversal. For example, we can sort a binary string as follows:

$$
\begin{array}{c}
\texttt{010011100110100111101110011 01}\\
\texttt{010011100 011010011 110111001}\\
\text{TBS}\downarrow \qquad \text{TBS}\downarrow\text{backwards} \qquad \downarrow\text{TBS}\\
\texttt{000001111 111110000 000111111}\\
\texttt{00000}\underbrace{\texttt{111111110000000}}\texttt{11111}\\
\texttt{000000000000111111111111111},
\end{array}
\tag{4.10}
$$

where the arrows with TBS indicate recursive calls to TBS and the bracket indicates the reversal to merge the segments. Let `GDC(TBS)` denote Algorithm 4.4.1 when using TBS to sort binary strings, where `GDC` stands for `GenericDivideConquer`.

The second algorithm is an adaptive version of TBS (Algorithm 4.4.3) that, instead of using equal thirds, adaptively chooses the segments' length. Adaptive TBS considers every pair of partition points, $0 \le i \le j < n-1$, that would split the binary sequence

```
Input  : B, a binary string
function AdaptiveTripartiteBinarySort(B):
R̄ := ∅
for i = 0 to n − 2 :
    for j = i to n − 2 :
        R̄_0 = AdaptiveTripartiteBinarySort(B[0, i])
        c_0 := cost(R̄_0)
        R̄_1 = AdaptiveTripartiteBinarySort(B[i + 1, j])
        c_1 := cost(R̄_1)
        R̄_2 = AdaptiveTripartiteBinarySort(B[j + 1, n − 1])
        c_2 := cost(R̄_2)
        r := cost of merging reversal using i and j as partition points
        if R̄ = ∅ or max{c_0, c_1, c_2} + r < cost(R̄) then
            R̄ := R̄_0 ++ R̄_1 ++ R̄_2
return R̄
```

Algorithm 4.4.3: Adaptive TBS. For the sake of clarity, we implement an exhaustive search over all possible ways to choose the partition points. However, we note that the optimal partition points can be found in polynomial time by using a dynamic programming method [Ben+08].

into two or three sections: $B[0, i]$, $B[i + 1, j]$, and $B[j + 1, n - 1]$ (where $i = j$ corresponds to no middle section). For each pair, it calculates the minimum cost to recursively sort the sequence using these partition points. Since each section can be sorted in parallel, the total *sorting time* depends on the maximum time needed to sort one of the three sections and the cost of the final merging reversal. Let `GDC(ATBS)` denote Algorithm 4.4.1 when using Adaptive TBS to sort binary strings.

Notice that the partition points selected by TBS are considered by the Adaptive TBS algorithm and are selected by Adaptive TBS only if no other pair of partition points yields a faster sorting time. Thus, for any permutation, the sequence of reversals found by Adaptive TBS costs no more than that found by TBS. However, TBS is simpler to implement and will be faster than Adaptive TBS in finding the sorting sequence of reversals.

4.4.1 Worst-case bounds

In this section, we prove that all permutations of sufficiently large length n can be sorted in time strictly less than n using reversals. Let $n_x(b)$ denote the number of times character $x \in \{0,1\}$ appears in a binary string b, and let $T(b)$ (resp., $T(\pi)$) denote the best possible sorting time to sort b (resp., implement π) with reversals. Assume all logarithms are base 2 unless specified otherwise.

Lemma 4.3. *Let $b \in \{0,1\}^n$ such that $n_x(b) < cn + O(\log n)$, where $c \in [0, 1/3]$ and $x \in \{0,1\}$. Then, $T(b) \leq (c/3 + 7/18)\, n + O(\log n)$.*

Proof. To achieve this upper bound, we use TBS (Algorithm 4.4.2). There are $\lfloor \log_3 n \rfloor$ steps in the recursion, which we index by $j \in \{0, 1, \ldots, \lfloor \log_3 n \rfloor\}$, with step 0 corresponding to the final merging step. Let $|\mathrm{R}_j|$ denote the size of the longest reversal in recursive step j that merges the three sorted subsequences of size $n/3^{j+1}$. The size of the final merging reversal R_0 can be bounded above by $(c + 2/3)n + O(\log n)$ because $|\mathrm{R}_0|$ is maximized when every x is contained in the leftmost third if $x = 1$ or the rightmost third if $x = 0$. So we have

$$T(b) \leq \left(\sum_{j=0}^{\log_3 n} \frac{|\mathrm{R}_j|}{3} \right) + O(\log n) \leq \left(\frac{c}{3} + \frac{2}{9} \right) n + O(\log n) + \left(\sum_{j=1}^{\log_3 n} \frac{|\mathrm{R}_j|}{3} \right) + O(\log n) \tag{4.11}$$

$$\leq \left(\frac{c}{3} + \frac{7}{18} \right) n + O(\log n), \tag{4.12}$$

where we used $|\mathrm{R}_j| \leq n/3^j$ for $j \geq 1$. □

Now we can prove a bound on the cost of a sorting series found by Adaptive TBS for any binary string of length n.

Theorem 4.4. *For all bit strings $b \in \{0,1\}^n$ of arbitrary length $n \in \mathbb{N}$, $T(b) \leq (1/2 - \varepsilon)\, n + O(\log n) \approx 0.483n + O(\log n)$, where $\varepsilon = 1/3 - 1/\sqrt{10}$.*

Figure 4.5: Case 2 of Theorem 4.4. If there are few zeros and ones in the leftmost and rightmost thirds, respectively, we can shorten the middle section so that it can be sorted quickly. Then, because each of the outer thirds contain far more zeros than ones (or vice versa), they can both can be sorted quickly as well.

Proof. Let $b \in \{0,1\}^n$ for some $n \in \mathbb{N}$. Partition b into three sections $b = b_1 b_2 b_3$ such that $|b_1| = |b_3| = \lfloor n/3 \rfloor$ and $|b_2| = n - 2\lfloor n/3 \rfloor$. Since $\lfloor n/3 \rfloor = n/3 - d$ where $d \in \{0, 1/3, 2/3\}$, we write $|b_1| = |b_2| = |b_3| = n/3 + O(1)$ for the purposes of this proof. Recall that if segments b_1 and b_3 are sorted forwards and segment b_2 is sorted backwards, the resulting segment can be sorted using a single reversal, R (see the example in (4.10)). Then we have

$$T(b) \leq \max(T(b_1), T'(b_2), T(b_3)) + \frac{|\mathrm{R}| + 1}{3}, \tag{4.13}$$

where $T'(b_2)$ is the time to sort b_2 backwards using reversals.

We proceed by induction on n. For the base case, it suffices to note that every binary string can be sorted using reversals and, for finitely many values of $n \in \mathbb{N}$, any time needed to sort a binary string of length n exceeding $(1/2 - \varepsilon)\, n$ can be absorbed into the $O(\log n)$ term. Now assume $T(b) \leq (1/2 - \varepsilon)\, k + O(\log k)$ for all $k < n$, $b \in \{0,1\}^k$.

Case 1: $n_0(b_1) \geq 2\varepsilon n$ **or** $n_1(b_3) \geq 2\varepsilon n$**.** In this case, $|\mathrm{R}| \leq n - 2\varepsilon n$, so

$$T(b) \leq \frac{n - 2\varepsilon n + 1}{3} + \max(T(b_1), T'(b_2), T(b_3)) \leq \left(\frac{1}{2} - \varepsilon\right) n + O(\log n) \tag{4.14}$$

by the induction hypothesis.

Case 2: $n_0(b_1) < 2\varepsilon n$ **and** $n_1(b_3) < 2\varepsilon n$**.** In this case, adjust the partition such that

$|b_1| = |b_3| = n/3+2\varepsilon n/(3-6\varepsilon)-O(1)$ and consequently $|b_2| = n/3-4\varepsilon n/(3-6\varepsilon)+O(1)$, as depicted in Figure 4.5. In this adjustment, at most $2\varepsilon n/(3-6\varepsilon)$ zeros are added to the segment b_1 and likewise with ones to b_3. Thus, $n_1(b_3) \leq 2\varepsilon n + 2\varepsilon n/(3-6\varepsilon) = (1 + 1/(3-6\varepsilon))\, 2\varepsilon n$. Since $n = (3-6\varepsilon)|b_1| - O(1)$, we have

$$n_1(b_3) \leq \left(1 + \frac{1}{3-6\varepsilon}\right) 2\varepsilon((3-6\varepsilon)|b_1| - O(1)) = (2-3\varepsilon)4\varepsilon|b_1| - O(1). \tag{4.15}$$

Let $c = (2-3\varepsilon)4\varepsilon = 2/15$. Applying Lemma 4.3 with this value of c yields

$$T(b_3) \leq \left(\frac{2}{45} + \frac{7}{18}\right)|b_1| + O(\log(|b_1|)) = \left(\frac{1}{\sqrt{10}} - \frac{1}{6}\right) n + O(\log n). \tag{4.16}$$

Since $|b_1| = |b_3|$, we obtain the same bound $T(b_1) \leq (1/\sqrt{10} - 1/6)n + O(\log n)$ by applying Lemma 4.3 with the same value of c.

By the inductive hypothesis, $T'(b_2)$ can be bounded above by

$$T'(b_2) \leq \left(\frac{1}{2} - \varepsilon\right)\left(\frac{n}{3} - \frac{4\varepsilon}{3-6\varepsilon}n + O(1)\right) + O(\log n) = \left(\frac{1}{\sqrt{10}} - \frac{1}{6}\right) n + O(\log n). \tag{4.17}$$

Using (4.13) and the fact that $|\mathrm{R}| \leq n$, we get the bound

$$T(b) \leq \left(\frac{1}{\sqrt{10}} - \frac{1}{6}\right) n + O(\log n) + \frac{n+1}{3} = \left(\frac{1}{2} - \varepsilon\right) n + O(\log n)$$

as claimed. □

This bound on the cost of a sorting series found by Adaptive TBS for binary sequences can easily be extended to a bound on the minimum sorting sequence for any permutation of length n.

Corollary 4.5. *For a length-n permutation π, $T(\pi) \leq \left(1/3 + \sqrt{2/5}\right)n + O(\log^2 n) \approx 0.9658n + O(\log^2 n)$.*

Proof. To sort π, we turn it into a binary string b using (4.7). Then let $\mathrm{R}_1, \mathrm{R}_2, \ldots, \mathrm{R}_m$ be a sequence of reversals to sort b. If we apply the sequence to get $\pi' = \pi\, \mathrm{R}_1\, \mathrm{R}_2 \cdots \mathrm{R}_m$, every element of π' will be on the same half as its destination. We can then recursively perform the same procedure on each half of π', continuing down until every pair of elements has been sorted.

This process requires $\lfloor \log n \rfloor$ steps, and at step i, there are 2^i binary strings of length $\frac{n}{2^i}$ being sorted in parallel. This gives us the following bound to implement π:

$$T(\pi) \leq \sum_{i=0}^{\log n} T(b_i), \tag{4.18}$$

where $b_i \in \{0,1\}^{n/2^i}$. Applying the bound from Theorem 4.4, we obtain

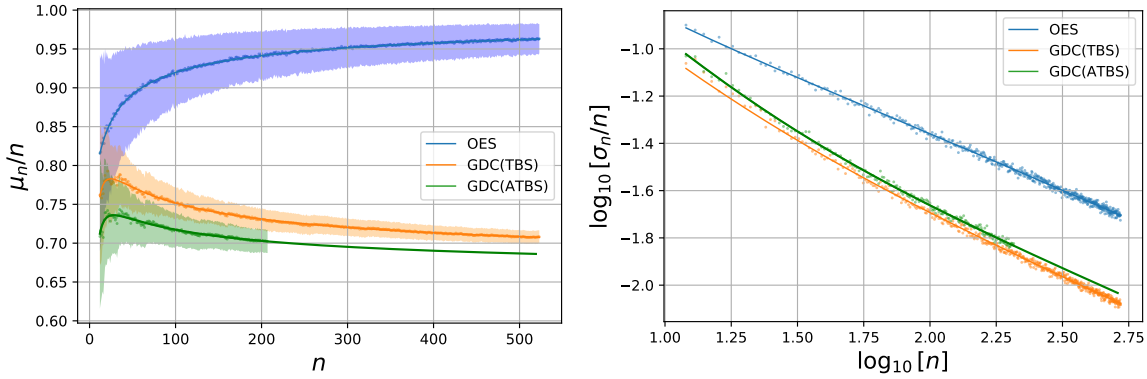
$$T(\pi) \leq \sum_{i=0}^{\log n} T(b_i) \leq \sum_{i=0}^{\log n} \left(\left(\frac{1}{6} + \frac{1}{\sqrt{10}} \right) \frac{n}{2^i} + O(\log(n/2^i)) \right) = \left(\frac{1}{3} + \sqrt{\frac{2}{5}} \right) n + O(\log^2 n).$$

□

4.5 Average-case performance

So far we have presented worst-case bounds that provide a theoretical guarantee on the speedup of quantum routing over classical routing. However, the bounds are not known to be tight, and may not accurately capture the performance of the algorithm in practice.

In this section we show better performance for the *average-case* routing time, the expected routing time of the algorithm on a permutation chosen uniformly at random from $\mathcal{S}_n$. We present both theoretical and numerical results on the average routing time of swap-based routing (such as odd-even sort) and quantum routing using TBS and ATBS. We show that on average, `GDC(TBS)` (and `GDC(ATBS)`, whose sorting time on any instance is at least as fast) beats swap-based routing by a constant factor 2/3. We have the following two theorems, whose proofs can be found in Appendices A.1



(a) Normalized mean routing time with std. deviation.

(b) Log normalized standard deviation of the routing time.

Figure 4.6: The mean routing time and fit of the mean routing time for odd-even sort (OES), and routing algorithms using Tripartite Binary Sort (`GDC(TBS)`) and Adaptive TBS (`GDC(ATBS)`). We exhaustively search for $n < 12$ and sample 1000 permutations uniformly at random otherwise. We show data for `GDC(ATBS)` only for $n \leq 207$ because it becomes too slow after that point. We find that the fit function $\mu_n = an + b\sqrt{n} + c$ fits the data with an $R^2 > 99.99\%$ (all three algorithms). For OES, the fit gives $a \approx 0.9999$; for `GDC(TBS)`, $a \approx 0.6599$; and for `GDC(ATBS)`, $a \approx 0.6513$. Similarly, for the standard deviation, we find that the fit function $\sigma_n^2 = an + b\sqrt{n} + c$ fits the data with $R^2 \approx 99\%$ (all three algorithms), suggesting that the normalized deviation of the performance about mean scales as $\sigma_n/n = \Theta(n^{-0.5})$ asymptotically.

and A.2, respectively.

Theorem 4.6. *The average routing time of any* SWAP*-based procedure is lower bounded by* $n - o(n)$.

Theorem 4.7. *The average routing time of* `GDC(TBS)` *is* $2n/3 + O(n^{\beta})$ *for a constant* $\beta \in \left(\frac{1}{2}, 1\right)$.

These theorems provide average-case guarantees, yet do not give information about the non-asymptotic behavior. Therefore, we test our algorithms on random permutations for instances of intermediate size.

Our numerics [KSS21] show that Algorithm 4.4.1 has an average routing time that is well-approximated by $c \cdot n + o(n)$, where $2/3 \lesssim c < 1$, using TBS or Adaptive TBS as the binary sorting subroutine, for permutations generated uniformly at random. Similarly, the performance of odd-even sort (OES) is well-approximated by $n + o(n)$. Furthermore, the advantage of quantum routing is evident even for fairly short paths. We demonstrate this by sampling 1000 permutations uniformly from $\mathcal{S}_n$ for $n \in [12, 512]$, and running OES and `GDC(TBS)` on each permutation. Due to computational constraints, `GDC(ATBS)` was run on sample permutations for lengths $n \in [12, 206]$. On an Intel i7-6700HQ processor with a clock speed of 2.60 GHz, OES took about 0.04 seconds to implement each permutation of length 512; `GDC(TBS)` took about 0.3 seconds; and, for permutations of length 200, `GDC(ATBS)` took about 6 seconds.

The results of our experiments are summarized in Figure 4.6. We find that the mean normalized time costs for OES, `GDC(TBS)`, and `GDC(ATBS)` are similar for small n, but the latter two decrease steadily as the lengths of the permutations increase while the former steadily increases. Furthermore, the average costs for `GDC(TBS)` and `GDC(ATBS)` diverge from that of OES rather quickly, suggesting that `GDC(TBS)` and `GDC(ATBS)` perform better on average for somewhat small permutations ($n \approx 50$) as well as asymptotically.

The linear coefficient a of the fit of μ_n for OES is $a \approx 0.9999 \approx 1$, which is consistent with the asymptotic bound proven in Theorems 4.6 and 4.7. For the fit of the mean time costs for `GDC(TBS)` and `GDC(ATBS)`, we have $a \approx 0.6599$ and $a \approx 0.6513$ respectively. The numerics suggest that the algorithm routing times agree with our analytics, and are fast for instances of realistic size. For example, at $n = 100$, `GDC(TBS)` and `GDC(ATBS)` have routing times of $\sim 0.75n$ and $0.72n$, respectively. On the other hand, OES routes in average time $> 0.9n$. For larger instances, the speedup approaches the full factor of 2/3 monotonically. Moreover, the fits of the standard deviations suggest $\sigma_n/n = \Theta(1/\sqrt{n})$ asymptotically, which implies that as permutation length increases, the distribution of routing times gets relatively tighter for all three algorithms. This suggests that the average-case routing time may indeed be representative of typical performance for our algorithms for permutations selected uniformly at random.

4.6 Conclusion

We have shown that our algorithm, `GDC(ATBS)` (i.e., Generic Divide-and-Conquer with Adaptive TBS to sort binary strings), uses the fast state reversal primitive to outperform any SWAP-based protocol when routing on the path in the worst and average case. Recent work shows a lower bound on the time to perform a reversal on the path graph of n/β, where $\beta \approx 4.5$ (Theorem 3.5). Thus we know that the Hamiltonian routing time in the cannot be improved by more than a factor β over SWAPs, even with new techniques for implementing reversals. However, it remains to understand the fastest possible routing time on the path. Clearly, this is also lower bounded by n/β. Our work could be improved by addressing the following two open questions: (i) how fast can state reversal be implemented, and (ii) what is the fastest way of implementing a general permutation using state reversal?

We believe that the upper bound in Corollary 4.5 can likely be decreased. For example, in the proof of Lemma 4.3, we use a simple bound to show that the reversal sequence found by `GDC(TBS)` sorts binary strings with fewer than cn ones sufficiently fast for our purposes. It is possible that this bound can be decreased if we consider the reversal sequence found by `GDC(ATBS)` instead. Additionally, in the proof of Theorem 4.4, we only consider two pairs of partition points: one pair in each case of the proof. This suggests that the bound in Theorem 4.4 might be decreased if the full power of `GDC(ATBS)` could be analyzed.

Improving the algorithm itself is also a potential avenue to decrease the upper bound in Corollary 4.5. For example, the generic divide-and-conquer approach in Algorithm 4.4.1 focused on splitting the path exactly in half and recursing. An obvious improvement would be to create an adaptive version of Algorithm 4.4.1 in a manner similar to `GDC(ATBS)` where instead of splitting the path in half, the partition point would be placed in the optimal spot. It is also possible that by going beyond the divide-and-conquer approach, we could find faster reversal sequences and reduce the upper bound even further.

Chapter 5

Bounding the quantum-classical routing separation

This chapter is based on work in

[Bap+] Aniruddha Bapat, Andrew Childs, Alexey Gorshkov, and Eddie Schoute. "Bounding the quantum routing advantage". In preparation

5.1 Introduction

In this chapter, we explore the extent to which genuinely quantum operations can accelerate quantum routing, as opposed to SWAP gates in classical routing. The relative power in routing of the models we explore is depicted in Figure 1.1. We investigate gate-based quantum routing with arbitrary two-qubit unitaries and Hamiltonian (quantum) routing by Hamiltonian evolution with norm-bounded interactions in Section 5.2. We first explore the limits of these more powerful models by lower bounding the gate-based routing depth by $2/c(G) - 1$, where $c(G)$ is the vertex expansion of the graph G, which is the smallest number of vertices adjacent to a vertex set relative its size. This confirms the intuition that a small vertex cut negatively impacts the

routing circuit depth. Second, we show a looser lower bound on the routing time in the Hamiltonian model of $2/(\alpha \cdot h(G)) - 1$, where $0 < \alpha \leq 4$ is the constant of small incremental entangling [Aud14] and $h(G)$ is the edge expansion of G, which is the smallest edge boundary size of a vertex set relative to the number of vertices. The edge expansion is generally related to small edge cuts in the graph. Along the way, we show time lower bounds on state preparation reminiscent of area laws [ECP10; Gon+17]. Third, in Section 5.3, we prove a general upper bound on (classical) routing on simple connected graphs, allowing us to prove conditions on the spectral gap of the Laplacian of G that rule out a superpolynomial separation between the (worst-case) classical and Hamiltonian routing times. This excludes interaction graphs with poor expansion properties. Finally, we give an example of superconstant, ($\Omega(\sqrt{n})$ for n the number of vertices) quantum-classical separation for routing in a strengthened model of (non-adaptive) Hamiltonian routing with constant ancilla resources in Section 5.4.

5.2 Quantum Routing

In this section, we introduce quantum routing and prove lower bounds dependent on graph expansion properties. We model the architecture constraints by the architecture graph G.

5.2.1 Gate-based Quantum Routing

First, we consider routing in the gate model of quantum computation, which we call gate-based quantum routing. We define the *gate-based quantum routing number* $\mathrm{qrt}(G)$ as

$$\mathrm{qrt}(G) \coloneqq \max_{\pi} \mathrm{qrt}(G, \pi), \tag{5.1}$$

where π is a permutation of the qubits and $\mathrm{qrt}(G, \pi)$ is the minimum depth of an architecture-respecting circuit that implements the permutation π. We do not

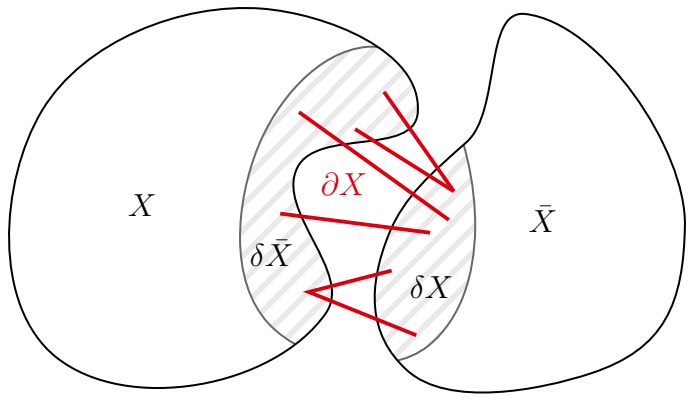


Figure 5.1: A graph can be partitioned into two sets of vertices X and $\bar{X}$. The vertex boundary δX of X is the set of vertices outside of X directly connected to X, and similarly for $\delta\bar{X}$ and $\bar{X}$. The edge boundary ∂X (red) of X (and $\bar{X}$) is the set of edges that connect X to $\bar{X}$.

limit routing circuits to a particular gate set. If necessary, any such circuit may be approximated by a universal gate set at a polylogarithmic overhead by the Solovay-Kitaev theorem. This implies that single-qubit gates are free since they can be absorbed into adjacent two-qubit gates.

We briefly prove a diameter lower bound on gate-based quantum routing.

Theorem 5.1.

$$\mathrm{qrt}(G) \geq \mathrm{diam}(G). \tag{5.2}$$

Proof. Consider two vertices $u, v \in V(G)$ at a distance $\mathrm{diam}(G)$ and a circuit $\mathcal{C}$ of two-qubit unitaries with depth D acting on G. Any local operator acting on u evolved in the Heisenberg picture under $\mathcal{C}$ will have no support on vertices further than distance D. In order to swap u and v, all of the support of that Heisenberg-evolved operator must be on v, which implies $D \geq \mathrm{diam}(G)$. Therefore, $\mathrm{qrt}(G) \geq \mathrm{diam}(G)$. □

We will see that the gate-based quantum routing number can, just like classical routing, be bounded by a vertex bottleneck in the interaction graph. A vertex bottleneck naturally *partitions* the graph G into two distinct subgraphs of G. We can define a partition of G by a vertex subset $X \subseteq V(G)$ and its *vertex complement*

$\bar{X} \coloneqq V(G) \setminus X$; see Figure 5.1. The set X neighbors a set of vertices which we define as the *vertex boundary* of X,

$$\delta X \coloneqq \{v \in \bar{X} \mid \{u, v\} \in E(G),\, u \in X\} \tag{5.3}$$

and are a vertex cut in G. Now we define the *vertex expansion* (or vertex isoperimetric number) as [Chu]

$$c(G) \coloneqq \min_{X \subseteq V(G) : |V(G)|/2} \frac{|\delta X|}{|X|}. \tag{5.4}$$

Intuitively, this lower bounds how many vertices neighbor any small enough set X. Therefore, the number of vertices in the induced subgraph $G[X \cup N(X)]$, for $N(X)$ the neighborhood of X, grows (or "expands") by at least a factor of $1 + c(G)$.

To prove a lower bound on quantum gate-based routing, we relate routing to the task of generating entanglement. We can quantify the entanglement of pure state ρ on a bipartite joint system $X\bar{X}$, consisting of the subsystems X and $\bar{X}$, by the Neumann entropy of the reduced density operator $\rho_X \coloneqq \mathrm{Tr}_{\bar{X}}(\rho)$, defined as

$$S(\rho_X) \coloneqq -\mathrm{Tr}(\rho_X \log \rho_X). \tag{5.5}$$

(The function $\log(x)$ denotes the logarithm base 2 unless specified and we denote the natural logarithm by $\ln(x)$.) We will refer to the von Neumann entropy as "the entropy" and denote $S_X(\rho) \coloneqq S(\rho_X)$. For completeness, we list some elementary properties of the von Neumann entropy that will be useful later and can be easily verified.

Lemma 5.2. *For a state ρ on a joint system $X\bar{X}$, the following statements about the entropy hold:*

1. *If ρ is a pure state, then the entropy is symmetric, i.e.*

$$S_X(\rho) = S_{\bar{X}}(\rho). \tag{5.6}$$

2. *The entropy is invariant under change of basis, i.e.*

$$S(U\rho U^\dagger) = S(\rho). \tag{5.7}$$

3. *The entropy is invariant under local unitaries U_X on X and $U_{\bar{X}}$ on $\bar{X}$, i.e.*

$$S_X((U_X \otimes U_{\bar{X}})\rho(U_X \otimes U_{\bar{X}})^\dagger) = S_X(\rho). \tag{5.8}$$

The maximum change in entropy for any unitary respecting the interaction constraints of an interaction graph G is bounded by the smallest vertex boundary size for each partitioning. We need only consider the unitary acting across the partitioning by the invariance of the entropy under local unitaries (Lemma 5.2). We formalize this in the following Lemma derived from small total entangling (STE) [Mar+16].

Lemma 5.3 (*Small Total Entangling* (STE))**.** *Let U be a unitary acting nontrivially only the joint subsystem $\delta X \delta \bar{X}$, then the change in the entropy can be bounded by*

$$\left|S_X(U\rho U^\dagger) - S_X(\rho)\right| \leq 2\min(|\delta X|, |\delta \bar{X}|). \tag{5.9}$$

Proof. The unitary U is a local unitary on the joint subsystem $X\delta X$; Lemma 5.2 implies

$$S_{X\delta X}(U\rho U^\dagger) = S_{X\delta X}(\rho). \tag{5.10}$$

Now, assume w.l.o.g. that $|\delta\bar{X}| \leq |\delta X|$, then

$$
\begin{aligned}
&|S_X(U\rho U^\dagger) - S_X(\rho)| && (5.11)\\
&\quad = |S_X(U\rho U^\dagger) - S_{X\delta X}(U\rho U^\dagger) + S_{X\delta X}(\rho) - S_X(\rho)| && (5.12)\\
&\quad \leq |S_X(U\rho U^\dagger) - S_{X\delta X}(U\rho U^\dagger)| + |S_{X\delta X}(\rho) - S_X(\rho)| && (5.13)\\
&\quad \leq S_{\delta X}(U\rho U^\dagger) + S_{\delta X}(\rho) && (5.14)\\
&\quad \leq 2|\delta X|, && (5.15)
\end{aligned}
$$

where we used the triangle inequality; subadditivity of the entropy; and $S(\rho) \leq |\delta X|$ for any state ρ on system δX consisting of qubits. The case where $|\delta X| \leq |\delta\bar{X}|$ is analogous. □

We can saturate this bound in several special cases. A SWAP gate can saturate this bound when the subsystems δX and $\delta\bar{X}$ are single qubits that are maximally entangled with the remainder of $\bar{X}$ and X, respectively. Furthermore, if we assume there is sufficient connectivity, then we can saturate this bound in higher dimensions too: Let

$$|\delta X| = |\delta\bar{X}| \leq \min(|X|, |\bar{X}|)/2 \tag{5.16}$$

and be maximally entangled with the remainder of $\bar{X}$ and X, respectively. Then, if we exchange δX with $\delta\bar{X}$ through SWAPs, the entropy will have increased by $2|\delta X|$, saturating the bound.

We now prove a lower bound on the time required for *state preparation* of entangled states. We define state preparation as the task of preparing some target state ρ given an initial state ρ_0. A special case of state preparation is routing a particular state. Intuitively, if the change in entanglement between initial state ρ_0 and final state ρ is $|S_X(\rho) - S_X(\rho_0)|$, then a simple argument from STE gives a circuit depth lower bound of $|S_X(\rho) - S_X(\rho_0)|/(2|\delta X|)$, and similar arguments have been used with the

entanglement capacity [Ben+03; Eld+20]. However, this does not account for the time required to entangle the boundary subsystem with the bulk subsystem. A careful accounting gives us the following Lemma, which we show can be saturated in some cases later.

Lemma 5.4. *Given an initial state ρ_0 and a target state ρ on the bipartite system consisting of X and $\bar{X}$, let us define the entanglement entropy increase*

$$\Delta S_Z \coloneqq |S_Z(\rho) - S_Z(\rho_0)|, \tag{5.17}$$

for any subsystem Z. Then any gate-based unitary circuit $\mathcal{C}$ for preparing ρ from ρ_0 restricted by an interaction graph G must have depth

$$d \geq \frac{\Delta S_X + \Delta S_{\bar{X}}}{2\min(|\delta X|, |\delta \bar{X}|)} - 1 \tag{5.18}$$

for any $X \subsetneq V(G)$.

Proof. Assume w.l.o.g. $|\delta X| \leq |\delta \bar{X}|$. The change in entropy $\Delta S_{\bar{X}}$ is closely related to that of its bulk system $Y \coloneqq \bar{X} \setminus \delta X$. We claim that $\Delta S_{\bar{X}} \leq \Delta S_Y + 2|\delta X|$. First, the entropy of the target state can be upper bounded using subadditivity and $S_{\delta X}(\cdot) \leq |\delta X|$ as

$$\Delta S_{\bar{X}} = S_{\bar{X}}(\rho) - S_{\bar{X}}(\rho_0) \leq S_Y(\rho) - S_{\bar{X}}(\rho_0) + |\delta X|. \tag{5.19}$$

The triangle inequality $S_{\bar{X}}(\rho_0) \geq |S_Y(\rho_0) - S_{\delta X}(\rho_0)|$ then gives

$$\Delta S_{\bar{X}} \leq S_Y(\rho) - S_{\bar{X}}(\rho_0) + |\delta X| \leq \Delta S_Y + 2|\delta X| \tag{5.20}$$

as claimed.

We can decompose $\mathcal{C}$ into a sequence of disjoint unitaries U_i on the boundary

between X and δX and unitaries V_i on the boundary between Y and δX, where $i \in \mathbb{N}$. For operations U_i and V_i to be simultaneous, they must act on disjoint subsets $\delta X_i, \delta X_i' \subseteq \delta X$, respectively. Between each application of $U_i V_i$, there are local unitary operations within X, δX, and $\bar{X} \setminus \delta X$, labelled as O_i, that we allow to be performed instantaneously. The circuit can thus be decomposed as

$$\mathcal{C} = O_d U_d V_d \dots O_1 U_1 V_1 O_0. \tag{5.21}$$

We lower bound d by using the change in entropy and STE. First, we note that the operations O_i cannot change the entropy of the respective subsystems. By Lemma 5.3, U_i can change the entropy of X by at most $2|\delta X_i|$ and V_i can change the entropy of Y by at most $2|\delta X_i'|$. Therefore, we have two inequalities that must be satisfied:

$$\Delta S_X \leq 2 \sum_{i=1}^{d} |\delta X_i| \tag{5.22}$$

$$\text{and} \quad \Delta S_Y \leq 2 \sum_{i=1}^{d} |\delta X_i'|. \tag{5.23}$$

Summing these together and noting $|\delta X_i| + |\delta X_i'| \leq |\delta X|$ we get

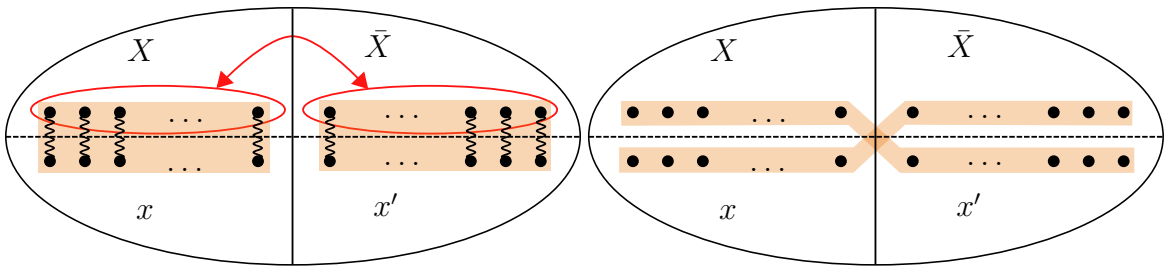
$$\Delta S_X + \Delta S_Y \leq 2 \sum_{i=1}^{d} |\delta X_i| + |\delta X_i'| \leq 2d|\delta X|. \tag{5.24}$$

Now, by (5.20), we get

$$d \geq \frac{\Delta S_X + \Delta S_Y}{2|\delta X|} \geq \frac{\Delta S_X + \Delta S_{\bar{X}}}{2|\delta X|} - 1 \tag{5.25}$$

as claimed.

The proof when $|\delta X| > |\delta \bar{X}|$ follows analogously by considering $\delta \bar{X}$ instead of δX and $X \setminus \delta \bar{X}$ instead of Y. □



(a) Before routing **(b)** After routing

Figure 5.2: For the proof of Theorem 5.5, we consider a bipartite system consisting of X and $\bar{X}$ with $|X| \leq |\bar{X}|$. The subsystems X and $\bar{X}$ consist of qubits (represented by vertices) augmented with one ancilla each in ancilla spaces x and x', respectively. We initialize each qubit-ancilla pair in a Bell state (wavy line). Now, the entropy of subsystem Xx is 0. We then perform routing, exchanging X with a subset of $\bar{X}$ (in red), and increase the entropy of Xx to $2|X|$. Lemma 5.3 shows that the entanglement increase for each layer of gates is bounded by the vertex boundary size $|\delta X|$, therefore lower-bounding the circuit depth and $\mathrm{qrt}(G)$.

We can saturate the Lemma up to the additive -1 constant when there is sufficient connectivity within $\bar{X}$, and between X and δX. A sufficient condition is that it is maximally connected within $\bar{X}$, and between X and δX. Let the initial state ρ_0 be $|X|/2$ Bell pairs in X and $|\bar{X}|/2$ Bell pairs in $\bar{X}$. The ends of $2|\delta X|$ Bell pairs prepared in X and $\bar{X}$ can be exchanged by concurrent SWAPs between X and δX. This will increase the entropy of X by $2|\delta X|$ as expected. However, we now spend one time step to swap δX with ends of fresh Bell pairs in Y using concurrent SWAPs. This does not increase the entanglement in X or $\bar{X}$, just within Y. Iterating, first, SWAPs between X and δX then, second, between δX and Y, we saturate the bound as claimed.

The lower bound on the gate-based quantum routing number is a simple corollary of Lemma 5.4 by preparing an appropriate initial state. See Figure 5.2 for an illustration of the proof concept.

Theorem 5.5. *For any architecture graph G,*

$$\mathrm{qrt}(G) \geq \max_{X \subseteq V(G):|V(G)|/2} \frac{2|X|}{\min(|\delta X|, |\delta \bar{X}|)} - 1 \geq \frac{2}{c(G)} - 1. \tag{5.26}$$

Proof. Let us augment the graph G with one ancilla qubit for each qubit $v \in V(G)$, initialized in a Bell pair with v. Since these ancilla are not connected with the main graph, they cannot help with routing. When describing sets X in the following, we consider the original qubits and their ancilla together. Then the entropy $S_X(\rho_0) = S_{\bar{X}}(\rho_0) = 0$ for any X since the reduced state is pure.

The gate-based quantum routing number $\mathrm{qrt}(G)$ maximizes over all permutations of the vertices. To show a lower bound, it suffices to pick a permutation π that routes all vertices $v \in X$ to $\bar{X}$ arbitrarily and routes $|X|$ vertices $u \in \bar{X}$ to X arbitrarily. Let the resulting state be our target state ρ. This gives $S_X(\rho) = S_{\bar{X}}(\rho) = 2|X|$. By Lemma 5.4, the depth of any circuit performing this state preparation and routing task is lower bounded by

$$\mathrm{qrt}(G, \pi) \geq \frac{\Delta S_X + \Delta S_{\bar{X}}}{2\min(|\delta X|, |\delta \bar{X}|)} - 1 = \frac{2|X|}{\min(|\delta X|, |\delta \bar{X}|)} - 1. \tag{5.27}$$

We now maximize over all X with $|X| \leq V(G)/2$ to lower bound the gate-based quantum routing number

$$\mathrm{qrt}(G) \geq \mathrm{qrt}(G, \pi) \geq \max_{X \subseteq V(G)} \frac{2|X|}{\min(|\delta X|, |\delta \bar{X}|)} - 1 \geq \frac{2}{c(G)} - 1, \tag{5.28}$$

where we used that $\min(|\delta X|, |\delta \bar{X}|) \leq |\delta X|$. □

One simple consequence is that gate-based quantum routing on the *star graph*, $S_n \coloneqq K_{1,n}$, the complete bipartite graph, is no faster than classical routing up to a constant factor. A trivial classical routing strategy has a depth upper bounded by $3n/2$, whereas we have $c(S_n) \leq 2/n$ so that $\mathrm{qrt}(S_n) \geq n - 1$ This is a consequence of the small vertex cut in the star graph.

5.2.2 Hamiltonian Routing

In this section, we will consider a stronger model for quantum routing, namely using two-qubit Hamiltonians with fast local operations. We define the *Hamiltonian routing time*, $\mathrm{hqrt}(G)$, as

$$\mathrm{hqrt}(G) \coloneqq \max_{\pi} \mathrm{hqrt}(G, \pi), \tag{5.29}$$

where π is a permutation of qubits and $\mathrm{hqrt}(G, \pi)$ is the minimum evolution time of an architecture-respecting evolution under our normalization condition for the canonical form (1.2). The shortest CNOT time in this model is $\pi/4$ and the shortest SWAP time is $3\pi/4$ [VHC02]. Any two-qubit unitary takes at most $3\pi/4$ since any such gate can be decomposed into at most 3 CNOT gates and single-qubit rotations [VW04]. Therefore, we renormalize by the SWAP time so that $\mathrm{hqrt}(G, \pi) \leq \mathrm{qrt}(G, \pi)$ for any permutation π.

We will see that the Hamiltonian routing time can be lower bounded by an edge cut in the graph G. An edge cut partitions G given by two vertex subsets $X \subseteq V(G)$ and $\bar{X}$. The edges leaving X form the *edge boundary* of X

$$\partial X \coloneqq \{(x, \bar{x}) \in E \mid x \in X, \bar{x} \in \delta X\} = \partial \bar{X} \tag{5.30}$$

and are an edge cut. Now we define the *edge expansion* (or edge isoperimetric number or Cheeger constant) as

$$h(G) \coloneqq \min_{X \subseteq V(G) : |X| \leq |V(G)|/2} \frac{|\partial X|}{|X|}. \tag{5.31}$$

Intuitively, this corresponds to a lower bound on how many edges leave any small enough set X. Therefore, the number of edges in the induced subgraph $G[X \cup N(X)]$, for $N(X)$ the neighborhood of X, grows (or "expands") by at least $1 + h(G)$.

We show a lower bound of $\mathrm{hqrt}(G) = \Omega(1/h(G))$. The edge expansion is always

larger than the vertex expansion, i.e. $h(G) \geq c(G)$, giving a weaker lower bound than on gate-based quantum routing. In particular, the star graph has $h(S_n) = \Theta(1)$ so our lower bound gives $\mathrm{hqrt}(S_n) = \Omega(1/h(S_n)) = \Omega(1)$. Given that $\mathrm{qrt}(S_n) = \Omega(n)$, a large separation is still possible between Hamiltonian routing and gate-based quantum routing.

To prove the lower bound on Hamiltonian routing, we use the continuous analogue of STE, the small incremental entangling (SIE) theorem adapted to our setting, conjectured by Kitaev [Bra07] and first proven in [AMV13].

Lemma 5.6 (*Small Incremental Entangling* (SIE))**.** *Given a finite joint system $X\bar{X}$, any Hamiltonian H with support only on $\delta X \delta \bar{X}$ and any initial pure state ρ, then the entanglement capacity, $\Gamma(H, \rho)$, is bounded by*

$$\Gamma(H, \rho) \coloneqq \frac{dS_X(\rho(t))}{dt} \leq \alpha \|H\| \log d, \tag{5.32}$$

where $\rho(t) = U(t)\rho U(t)^\dagger$ for $U(t) = e^{-iHt}$, $0 < \alpha \leq 4$ is a constant, and $d = \min(|\delta X|, |\delta \bar{X}|)$.

It is conjectured that $\alpha = 2$ [Bra07] but the best known bound has reached $\alpha = 4$ [Aud14]. No generality is lost by assuming pure states: We can add a purification system C as ancilla system w.l.o.g. to X. The resulting state on the joint system $X\bar{X}C$ is pure and bounded by SIE. Since including C as ancilla system can only increase the rate of entanglement growth (we can always ignore it), we have that the entanglement growth is also bounded for mixed states on $X\bar{X}$.

We can derive another expression for $\Gamma(H,\rho)$ by defining $\rho_X(t) \coloneqq \mathrm{Tr}_{\bar{X}}(\rho(t))$ and

$$\Gamma(H,\rho) = -\frac{d}{dt}\,\mathrm{Tr}(\rho_X(t)\log\rho_X(t)) \tag{5.33}$$

$$= -\,\mathrm{Tr}\left(\frac{d\rho_X(t)}{dt}\log\rho_X(t)\right) \tag{5.34}$$

$$= -i\,\mathrm{Tr}(\mathrm{Tr}_{\bar{X}}([H,\rho])\log\rho_X(t)), \tag{5.35}$$

where we used the Schrödinger equation $i\frac{d\rho}{dt} = [H,\rho]$ (setting $\hbar = 1$). It is easy to see that (5.35) is linear in H.

The evolution of a system with interaction graph G, for any $X \subseteq V(G)$, can be described by a Hamiltonian $H = H_X + H_{\bar{X}} + H_{\delta X \delta \bar{X}}$, where H_Y only has support on the subsystem of vertices $Y \subseteq V(G)$. Operations local to X or $\bar{X}$ do not generate entanglement, therefore we have that

$$\Gamma(H,\rho) = \Gamma(H_{\delta X \delta \bar{X}},\rho). \tag{5.36}$$

We can see that this is true by first explicitly computing

$$\mathrm{Tr}_{\bar{X}}([H_{\bar{X}},\rho]) = 0 \tag{5.37}$$

because the partial trace is cyclic on the $\bar{X}$ subsystem. Second,

$$\Gamma(H_X,\rho) = -\,\mathrm{Tr}([H_X,\rho_X(t)]\log\rho_X(t)) = 0 \tag{5.38}$$

because $\log\rho_X(t)$ commutes with $\rho_X(t)$ and using the cyclic property of the trace. By linearity, (5.36) holds and we can restrict ourselves to consider only Hamiltonians of the form $H_{\delta X \delta \bar{X}}$.

Now we can bound the entanglement capacity of any edge cut in the graph as given by the edge boundary for a vertex subset X.

Theorem 5.7. *Given any $X \subseteq V(G)$, and any pure initial state ρ, the entanglement capacity of a Hamiltonian H with support only on the joint subsystem $\delta X \delta \bar{X}$ can by bounded by*

$$\Gamma(H, \rho) = \frac{dS_X(\rho(t))}{dt} \leq \alpha |\partial X|, \tag{5.39}$$

for α the constant of SIE.

Proof. We write $H = \sum_{e \in \partial X} H^{(e)}$, for $H^{(e)}$ two-qubit Hamiltonian interactions acting only on the endpoints of e. By linearity,

$$\Gamma(H, \rho) = \Gamma\left(\sum_{e \in \partial X} H^{(e)}, \rho\right) = \sum_{e \in \partial X} \Gamma\big(H^{(e)}, \rho\big). \tag{5.40}$$

We bound each term by SIE (Lemma 5.6)

$$\sum_{e \in \partial X} \Gamma\big(H^{(e)}, \rho\big) \leq \alpha \sum_{e \in \partial X} \big\|H^{(e)}\big\|. \tag{5.41}$$

By unitary similarity (which the norm is invariant under), we can rewrite each term in canonical form (1.2) and apply our normalization condition such that $\sum_{e \in \partial X} \|H^{(e)}\| \leq |\partial X|$. □

Given this relation of entanglement capacity to edge cuts in the graph, we show a lower bound on the time to perform state preparation in the Hamiltonian model dependent on the edge cut.

Corollary 5.8. *Given an initial pure state ρ_0 and target pure state ρ on a bipartite system $X\bar{X}$, let us define the entanglement entropy increase $\Delta S_X \coloneqq S_X(\rho) - S_X(\rho_0)$. Then any Hamiltonian unitary evolution from ρ_0 to ρ restricted by interaction graph G must have evolution time*

$$t \geq \frac{\Delta S_X}{\alpha |\partial X|}. \tag{5.42}$$

Proof. Follows from Theorem 5.7. □

A lower bound on Hamiltonian routing is then easy since routing a particular state is a special case of state preparation.

Theorem 5.9. *For any graph G,*

$$\mathrm{hqrt}(G) \geq \frac{8}{3\pi} \frac{1}{\alpha \cdot h(G)}. \tag{5.43}$$

Proof. We prepare the same initial state as in Theorem 5.5, where we have one half of a Bell pair at each vertex $v \in V(G)$. To show a lower bound, it suffices to pick a permutation π that routes all vertices $v \in X$ to $\bar{X}$ arbitrarily and routes $|X|$ vertices $u \in \bar{X}$ to X arbitrarily. Let the resulting state be our target state ρ. This gives $\Delta S_X = S_X(\rho) = 2|X|$. Corollary 5.8 and SWAP time normalization imply that the time to implement this state preparation and routing task is lower bounded by

$$\mathrm{hqrt}(G, \pi) \geq \frac{4}{3\pi} \frac{\Delta S_X}{\alpha |\partial X|} = \frac{8}{3\pi} \frac{|X|}{\alpha |\partial X|}. \tag{5.44}$$

We now maximize over all X to lower bound the Hamiltonian routing time

$$\mathrm{hqrt}(G) \geq \mathrm{hqrt}(G, \pi) \geq \max_X \frac{8}{3\pi} \frac{|X|}{\alpha |\partial X|} = \frac{8}{3\pi} \frac{1}{\alpha \cdot h(G)} \tag{5.45}$$

as claimed. □

We match previous results on entanglement area laws for dynamics [Gon+17, Theorem 1] on lattices, which also have a dependency on the edge cut. For example, the distinction between edge cut and vertex cut is significant for lattices of superconstant dimension. Generally, it holds that $|\partial X| \leq \max_v d_v |\delta X|$, for d_v the degree of $v \in V(G)$. It remains an open question whether Hamiltonian routing can be lower bounded by $\Omega(1/c(G))$. We show in Chapter 5 that is not the case for a stronger model of Hamiltonian routing.

Another case that has been well-studied is the path graph, P_n. Here, *odd-even*

sort [Knu98] gives a simple classical routing algorithm that upper bounds the circuit depth by n. It is easy to see that $c(P_n) \leq 2/n$ so that $\mathrm{qrt}(P_n) \geq n-1$, matching the diameter lower bound (Theorem 5.1). Thus, significant improvement over classical routing on the path is only possible in the Hamiltonian routing model: We have $h(P_n) \leq 2/n$, giving $\mathrm{hqrt}(P_n) \geq 4n/(3\pi\alpha)$, slightly weaker (even if $\alpha = 2$) than a specialized bound of $4n/(3\pi\alpha_0) \approx 0.222n$, for $\alpha_0 \approx 1.912$, based on the entanglement capacity (Theorem 3.5) Indeed, Chapter 4 shows $\mathrm{hqrt}(P_n) \leq (1-\varepsilon)n + O(\log^2 n)$, for constant $\varepsilon \approx 0.034$, so that $\mathrm{hqrt}(P_n) < \mathrm{qrt}(P_n)$ for large enough n.

5.3 Comparison with Classical Routing

For some graph families fast classical routing algorithms are already known [ACG94; CSU19]. An example are grid graphs $P_L \times P_L$ with side length $L \times L$, where we know $\mathrm{rt}(P_L \times P_L) \leq 3L$ and we can exclude a superconstant quantum advantage simply by $\mathrm{hqrt}(G) = \Omega(\mathrm{diam}(G))$ from Lieb-Robinson bounds [LR72].

In this section, we compare our quantum routing results with general bounds on classical routing. This will allow us to obtain more conditions for a superpolynomial separation. Our results and proofs are generalizations of results in [ACG94] from regular graphs to irregular graphs.

In classical routing we route a permutation π in multiple time steps. We first assign to each vertex v a *token* labeled $\pi(v)$. Then, in each time step, we may perform SWAP gates on neighboring vertices to exchange their tokens with the constraint that each vertex participates in at most one SWAP. Routing terminates when each token has been moved to its destination vertex. The difficulty of classical routing on G is characterized by the *routing number* (Chapter 2)

$$\mathrm{rt}(G) \coloneqq \max_{\pi} \mathrm{rt}(G, \pi), \tag{5.46}$$

where $\mathrm{rt}(G,\pi)$ is defined as the minimal number of time steps needed to implement permutation π. It follows that $\mathrm{qrt}(G,\pi) \leq \mathrm{rt}(G,\pi)$ for any permutation π.

5.3.1 General Classical Routing

Our classical routing algorithm consists of performing SWAPs along a number of special random walks. We upper bound how many SWAPs along these walks act on the same subspace at the same time, leading to high parallelism on graphs with a large first non-zero eigenvalue of the (normalized) graph Laplacian.

Let the adjacency matrix of a simple graph G be

$$A(u,v) = \begin{cases} 1 & \text{if } (v,u) \in E(G) \\ 0 & \text{otherwise,} \end{cases} \tag{5.47}$$

for $v, u \in V(G)$, and let the diagonal matrix $T(v,v) = d_v$, for $d_v = \|Av\|_1$ the degree of v, and 0 otherwise. Then the (normalized) graph Laplacian is defined as $\mathcal{L} = \mathbb{1} - T^{1/2}AT^{1/2}$ or, explicitly, [Chu]

$$\mathcal{L}(u,v) = \begin{cases} 1 & \text{if } u = v \text{ and } d_v \neq 0, \\ -\frac{1}{\sqrt{d_u d_v}} & \text{if } u \text{ and } v \text{ are adjacent,} \\ 0 & \text{otherwise.} \end{cases} \tag{5.48}$$

The Laplacian $\mathcal{L}$ is symmetric and positive semidefinite [Chu]. The stationary distribution $T^{1/2}\mathbf{1}$, for $\mathbf{1}$ the constant one function, is the zero eigenfunction of $\mathcal{L}$, i.e., $\mathcal{L}T^{1/2}\mathbf{1} = \mathbf{0}$. Let the eigenvalues of $\mathcal{L}$ be ordered $0 = \lambda_0 \leq \lambda_1 = \lambda(G) \leq \cdots \leq \lambda_n$, then the following holds for $\lambda(G)$ [Chu, Lemmas 1.7 and 1.9]:

Lemma 5.10. *We have:*

1. *For $n \geq 2$,*

$$\lambda(K_n) \leq \frac{n}{n-1}. \tag{5.49}$$

2. *For a graph G that is not the complete graph, we have $\lambda(G) \leq 1$.*

3. *If G is connected, then*

$$\lambda(G) \geq \frac{1}{2\,\mathrm{diam}(G)|E(G)|}. \tag{5.50}$$

We assume $n \geq 2$ and show a general bound on the (classical) routing number In this section without attempting to minimize the constants. We will use random walks $X = v_1 v_2 \ldots$, for $v_i \in V$, with transition probabilities $P(u, v) = \mathrm{P}[x_{i+1} = v \mid x_i = u]$. These probabilities form the transition matrix P of the simple random walk on G, so that $P = AT^{-1}$, i.e.,

$$P(u, v) = \begin{cases} 1/d_u & \text{if } u \text{ and } v \text{ are adjacent,} \\ 0 & \text{otherwise.} \end{cases} \tag{5.51}$$

Note that we default to right multiplication with the transition matrix so that our probability distributions can be interpreted as column vectors. Therefore, the probability that a random walk starting at u is at v after $i \in \mathbb{N}$ steps is given by

$$\mathrm{P}[x_i = v \mid x_0 = u] = \boldsymbol{e}(v)^{\mathsf{T}} P^i \boldsymbol{e}(u), \tag{5.52}$$

where $\boldsymbol{e}(v)$ is the column vector with a 1 in position v and 0 otherwise.

Now, consider the probability that, with a random walk starting at each vertex, too many walks will visit the same vertex around the same time. We illustrate this with Figure 5.3, where two walks $W(u)$ and $W(v)$, starting at vertices u and v, visit a red vertex $W(v)_i = W(u)_j$ too close to one another if $|i - j| < 5$. We show that with high probability, this does not occur too often for random walks of length $l \geq \ln(n)/\lambda(G)$. This is a generalization of [ACG94, Lemma 2] to irregular graphs, where we explicitly bound the dependence on the degree for walks to intersect. In particular, the entries

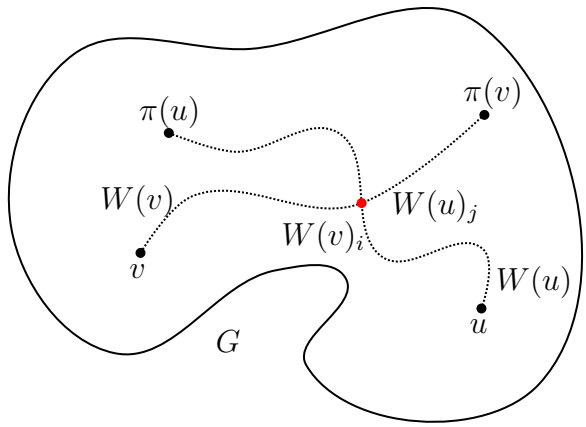


Figure 5.3: Drawn are two walks: the walk $W(v)$ from v to $\pi(v)$ and the walk $W(u)$ from u to $\pi(u)$, for some permutation π of the vertices. Walks may intersect at a vertex (red) such that the i-th step of $W(v)$ and the j-th step of $W(u)$ are the same, i.e., $W(v)_i = W(v)_j$. When we perform transpositions along random walks to perform routing, these may interfere when they intersect at similar times. We bound the total number of walk intersecting where $|i-j| < 5$ for any vertex when considering random walks of sufficient length. Given that random walks do not intersect too much, we can use them to route simultaneously on arbitrary graphs.

of $T/\min_v d_v$ in (5.60) are bounded by the *degree ratio*

$$d_* := \frac{\max_v d_v}{\min_v d_v}. \tag{5.53}$$

Lemma 5.11. *Let G be a connected simple graph on n vertices and suppose $l \geq \ln(n)/\lambda(G)$. For every $v \in V(G)$, let $W(v)$ denote a random walk of length l starting at vertex v. Let $I(v)$ denote the total number of other walks $W(u)$ such that there exist indices $0 \leq i, j \leq l, |i-j| < 5$, so that $W(v)_i = W(u)_j$. Then, with probability at most n^{-125} there is no vertex $v \in V(G)$ with $I(v) > 90ld_*$.*

Proof. We wish to bound $I(v)$ for any $v \in V(G)$. Let us introduce the indicator random variable depending on the random walks $W(v)$

$$X_{uv} = \begin{cases} 1 & \text{if } W(u) \text{ intersects } W(v), \\ 0 & \text{otherwise,} \end{cases} \tag{5.54}$$

where we define that $W(u)$ intersects $W(v)$ if there exist indices $0 \leq i, j \leq l, |i-j| < 5$, such that $W(v)_i = W(u)_j$. We include the random walk starting at v in the total which only increases the expectation of $I(v)$, then the expected value of $I(v)$ over random walks is

$$\mathbf{E}[I(v)] \leq \mathbf{E}\left[\sum_u X_{uv}\right] = \sum_u \mathrm{P}[X_{uv} = 1] \tag{5.55}$$

$$= \sum_u \mathrm{P}\left[\bigvee_{i\in[l]} \bigvee_{j:|i-j|<5} W(v)_i = W(u)_j\right] \tag{5.56}$$

$$\leq \sum_u \sum_{i\in[l]} \sum_{j:|i-j|<5} \mathrm{P}\Big[W(v)_i = W(u)_j\Big]. \tag{5.57}$$

Recall (5.52), then

$$\sum_{u,i,j} \mathrm{P}\Big[W(v)_i = W(u)_j\Big] = \sum_u \sum_{i\in[l]} \boldsymbol{e}(W(v)_i)^\mathsf{T} \sum_{j:|i-j|<5} P^j \boldsymbol{e}(u) \tag{5.58}$$

$$= \sum_{i\in[l]} \boldsymbol{e}(W(v)_i)^\mathsf{T} \sum_{j:|i-j|<5} P^j \mathbf{1}, \tag{5.59}$$

where we note the inequality $\boldsymbol{x}^\mathsf{T}\mathbf{1} = \sum_\ell \boldsymbol{x}_\ell \leq \sum_\ell c_\ell \boldsymbol{x}_\ell$, for $\boldsymbol{x}_\ell \geq 0$, when we scale by $c_\ell \geq 1$, so

$$\sum_{i\in[l]} \boldsymbol{e}(W(v)_i)^\mathsf{T} \sum_{j:|i-j|<5} P^j \mathbf{1} \leq \sum_{i\in[l]} \boldsymbol{e}(W(v)_i)^\mathsf{T} \sum_{j:|i-j|<5} P^j \frac{T\mathbf{1}}{\min_v d_v} \tag{5.60}$$

$$\leq 9 \sum_{i\in[l]} \boldsymbol{e}(W(v)_i)^\mathsf{T} \frac{T\mathbf{1}}{\min_v d_v} \tag{5.61}$$

$$\leq 9 l d_*, \tag{5.62}$$

since $PT\mathbf{1} = T\mathbf{1}$ and $T(i,i) \leq \max_v d_v$.

We now bound the tail probablity of $I(v)$. We use the multiplicative Chernoff bound, which states that for random variable $Y = \sum_i Y_i$, where Y_i are independent

random variables, and mean μ, $\mathrm{P}[X > (1+\delta)\mu] \leq \left(e^{\delta}/(1+\delta)^{1+\delta}\right)^{\mu}$ for any $\delta > 0$. We can identify $I(v) = \sum_{u \neq v} X_{uv}$ and see that the Chernoff bound applies since walks are independent (note that they may depend on v). Setting $\delta = 9$, we have

$$\mathrm{P}[I(v) > 90ld_*] \leq \mathrm{P}[I(v) > 10\,\mathbf{E}[I(v)]] < \left(\frac{e^9}{10^{10}}\right)^{9ld_*} \tag{5.63}$$

$$\leq n^{9\ln(e^9/10^{10})d_*/\lambda(G)} \tag{5.64}$$

$$\leq n^{-126} \tag{5.65}$$

by the Chernoff bound, $d_* \geq 1$, and $\lambda(G) \leq n/(n-1)$ with $n^{-1/(n-1)} \leq 1$. Since there are n vertices, the probability that there exists a vertex with $I(v) > 90ld_*$ is at most n^{-125}. The Lemma follows from the contrapositive. □

Now we show that if we "glue" together random walks starting at the vertices at opposite ends of a permutation of order two then these two walks will, with high probability, have few intersections. This is an adaptation of [ACG94, Lemma 3] to irregular graphs, using our previous Lemma.

Lemma 5.12. *Let G denote a simple connected graph on n vertices and let σ be a permutation of order two on $V(G)$. Put $l = \frac{10}{\lambda(G)} \ln n$. Then there is a set of n walks $W(v)$, one for each $v \in V(G)$ and of length $2l$, where both $W(v)$ and $W(\sigma(v))$ have, as endpoints, v and $\sigma(v)$ and traverse the same set of edges (in opposite directions) satisfying the following: If $I(v)$ denotes the total number of other walks $W(u)$ such that there exist two indices $0 \leq i, j \leq l, |i-j| < 5$, for which $W(v)_i = W(u)_j$ or $W(v)_i = W(u)_{2l-j}$, then $I(v) \leq 400ld_*$ for all v with high probability.*

Proof. We first prove that it is possible to construct n walks satisfying the conditions of the Lemma and that are close to random [BFU94]. Let us define the probablity of a random walk starting at $v \in V(G)$ ending at a vertex $w \in V(G)$, where w is a

random variable, after $t \in \mathbb{N}$ steps, then

$$P_v^{(t)}(w) \coloneqq \mathrm{P}[W(v)_t = w] \tag{5.66}$$

$$= \sum_{v_2,\ldots,v_{t-1} \in V(G)} \mathrm{P}[W(v) = v, v_2, \ldots, v_{t-1}, w]. \tag{5.67}$$

We now define the *relative pointwise distance*, $\Delta \colon \mathbb{N} \to \mathbb{R}$, of $P_v^{(t)}(w)$ to the stationary distribution as [Chu]

$$\Delta(t) \coloneqq \max_{v,w} \frac{\left| P_v^{(t)}(w) - \pi(w) \right|}{\pi(w)}. \tag{5.68}$$

By [Chu, Theorem 1.16] we have that all paths of length l are close to stationary, i.e.,

$$\Delta(l) \leq 2 \exp\left(\frac{-2l\lambda(G)}{2 + \lambda(G)} \right) \frac{|E(G)|}{\min_x d_x} < 2n^{-10} \frac{|E(G)|}{\min_x d_x} \leq 2n^{-8}, \tag{5.69}$$

where we used $\lambda(G) > 0$ and $\min_x d_x \geq 1$. Then, we can compare the statistics of a truly random walk $W(v)$ of length l to a random walk where we condition on the last vertex being $w \in V$, which is sampled according to the stationary distribution. Let us call this a *conditioned walk*. The probability of a particular random walk $W(v)$ can be related to the conditioned walk by

$$\mathrm{P}[W(v)] = \sum_w \mathrm{P}[W(v) \,|\, W(v)_l = w] P_v^{(l)}(w) \tag{5.70}$$

$$\leq \sum_w \mathrm{P}[W(v) \,|\, W(v)_l = w](1 + 2n^{-8})\pi(w) \tag{5.71}$$

$$= 2n^{-8} + \sum_w \mathrm{P}[W(v) \,|\, W(v)_l = w]\pi(w) \tag{5.72}$$

and the corresponding lower bound can be derived similarly. Therefore, a conditioned walk has vanishing deviation in n from a random walk.

By putting such a conditioned walk from v to w back to back with a conditioned walk from $\sigma(v)$ to w, in reverse order, we have shown the existence of walks from each

v to $\sigma(v)$ that are close to random walks. Given that Lemma 5.11 holds for any $W(v)$, we have that $I(v) > 180ld_*$ for no walk $W(v)$ with high probability. The result follows from the contrapositive. □

The existence of walks between opposite ends of an order-two permutation with few intersections leads to a classical routing algorithm that divides the walks into disjoint sets that do not intersect. This adapts [Chu, Theorem 4.10] to the irregular graph setting using our previous Lemmas.

Theorem 5.13. *Let σ denote a permutation of order two on the vertex set of a connected graph G. Then*

$$\mathrm{rt}(G,\sigma) \leq \frac{80\,000 d_*}{\lambda(G)^2} \ln^2 n = O\left(\frac{d_*}{\lambda(G)^2} \log^2 n\right). \tag{5.73}$$

Proof. Let $l = \frac{10}{\lambda(G)} \ln n$. We want to show $\mathrm{rt}(G,\sigma) = O(l^2 d_*)$. Let $W(v)$ be a system of walks of length $2l$ satisfying Lemma 5.12. Let H be the graph whose vertices are the walks $W(v)$ and in which $W(v)$ and $W(u)$ are adjacent if there exist two indices $0 \leq i, j \leq l, |i-j| < 5$, so that $W(v)_i = W(u)_j$ or $W(v)_i = W(u)_{2l-j}$. By Lemma 5.12, the maximum degree of H is at most $400ld_*$ with high probability, hence it is $(400ld_*)$-colorable. We can therefore divide the walks $W(v)$ into $400ld_*$ sets of independent walks of length $2l$.

We now present the routing algorithm. For each set of independent walks we sequentially do the following. For step i, with $1 \leq i \leq l$, we flip tokens along the edges numbered i and $2l-1-i$ in each of the walks. After l steps the tokens at either end of the walk will have been exchanged and the tokens not involved in any walk have not moved. After repeating this for all independent sets, all tokens have reached their destinations. □

Now it is easy to generalize to all permutations.

Corollary 5.14. *For every connected simple graph G,*

$$\mathrm{rt}(G) \leq \frac{160\,000 d_*}{\lambda(G)^2} \ln^2 n = O\left(\frac{d_*}{\lambda(G)^2} \log^2 n\right). \tag{5.74}$$

Proof. Any permutation of $V(G)$ can be written as a product of two permutations of order two. Use Theorem 5.13 to route each sequentially and obtain the result. □

Corollary 5.14 provides novel upper bounds in certain special cases of graphs. Of particular interest are irregular graphs where $d_*/\lambda(G)^2 = o(n)$. One such example is an Erdös-Rényi graph $G_{n,p}$, which is an n-vertex graph where each edge is independently present with some probability p. Hoffman, Kahle, and Paquette [HKP19] showed that for $p \geq (1+\delta)\log n/n$, for constant $\delta > 0$, there is a constant $C(\delta)$ such that

$$|1 - \lambda(G)| < \frac{C(\delta)}{\sqrt{p(n-1)}} = O\left(\frac{1}{\sqrt{n}}\right) \tag{5.75}$$

with high probability. Thus, we have that $\lambda(G) = \Omega(1)$ with high probability for such p and large enough n. Moreover, the degree ratio $d_* \to 1$ for $n \to \infty$ with high probability, though it does not exactly equal 1 for finite n, giving some irregularity. Under these conditions, Corollary 5.14 shows that $\mathrm{rt}(G_{n,p}) = O(\log^2 n)$ with high probability.

5.3.2 Conditions for a Superpolynomial Separation

To compare our upper bound on the classical routing number and the Hamiltonian routing time lower bound, we bound the Hamiltonian routing time in terms of the spectral gap $\lambda(G)$. To do so, we use the *Cheeger inequality* [Che71; Chu] that we state here without proof.

Lemma 5.15 (Cheeger inequality)**.** *For any connected graph G,*

$$2h_G \geq \lambda(G) > \frac{h_G^2}{2}, \tag{5.76}$$

where the Cheeger constant, h_G, is defined as

$$h_G \coloneqq \min_{X \subseteq V(G):|X| \leq |V(G)|/2} \frac{|\partial X|}{\sum_{x \in X} d_x}. \tag{5.77}$$

The edge expansion $h(G)$ relates to h_G as

$$h(G) = \min_X \frac{|\partial X|}{|X|} \leq \min_X \frac{|\partial X|}{\sum_{x \in X} d_x} \max_v d_v = h_G \max_v d_v, \tag{5.78}$$

where $X \subseteq V(G)$ and $|X| \leq |V(G)|/2$. We now rewrite the Hamiltonian routing lower bound, Theorem 5.5, in terms of spectral properties.

Lemma 5.16. *For a connected graph G, we have that*

$$\mathrm{hqrt}(G) \geq \frac{8}{3\pi\alpha \max_v d_v} \sqrt{\frac{1}{2\lambda(G)}}. \tag{5.79}$$

Proof. Given Theorem 5.9, (5.78), and Lemma 5.15, we have

$$\begin{aligned} \mathrm{hqrt}(G) &\geq \frac{8}{3\pi\alpha \cdot h(G)} && (5.80) \\ &\geq \frac{8}{3\pi\alpha \cdot h_G \max_v d_v} && (5.81) \\ &> \frac{8}{3\pi\alpha \max_v d_v} \sqrt{\frac{1}{2\lambda(G)}} && (5.82) \end{aligned}$$

as claimed. □

Interaction graphs G of common quantum architectures have bounded degree [IBM18; Aru+19; KMW02]. We have that Corollary 5.14 gives $\mathrm{rt}(G) = O(\log^2(n)/\lambda(G)^2)$ and

Lemma 5.16 gives $\mathrm{hqrt}(G) = \Omega(1/\sqrt{\lambda(G)})$. Therefore, the Hamiltonian routing time and classical routing number are polynomially related up to log factors in this case.

A simple way to bound the slowdown when a classical routing algorithm is used instead of a Hamiltonian routing algorithm is the ratio of the routing times. By Corollary 5.14 and Lemma 5.16, we have that

$$\frac{\mathrm{rt}(G)}{\mathrm{hqrt}(G)} = O\left(\frac{d_* \max_v d_v}{\lambda(G)^{3/2}} \log^2 n\right). \tag{5.83}$$

By performing routing on the spanning tree of G, we have that $\mathrm{rt}(G) = O(n)$ [ACG94], and, trivially, $\mathrm{hqrt}(G) = \Omega(1)$. Therefore, (5.83) is nontrivial if

$$d_* \max_v d_v = o\left(\frac{n}{\log^2 n} \lambda(G)^{3/2}\right). \tag{5.84}$$

Moreover, it is possible to bound the classical routing number by a polynomial in the Hamiltonian routing time when $\lambda(G)$ is sufficiently small.

Theorem 5.17. *For a simple connected graph G, $\mathrm{rt}(G) = O(\mathrm{poly}(\mathrm{hqrt}(G)))$ if*

$$\frac{1}{\lambda(G)} = \Omega\left(\mathrm{poly}(d_*, \log n) \max_v d_v\right). \tag{5.85}$$

Proof. We wish to show when $\mathrm{rt}(G)$ is some polynomial of $\mathrm{qrt}(G)$, i.e. $\mathrm{rt}(G) = O(\mathrm{hqrt}(G)^k)$ for constant $k \geq 1$. By Corollary 5.14 and Lemma 5.16, this happens when

$$\frac{\log \mathrm{rt}(G)}{\log \mathrm{hqrt}(G)} = O\left(\frac{\log d_* + \log\log n}{\log \frac{1}{\lambda(G) \max_v d_v}} + 1\right) \tag{5.86}$$

can be upper bounded by a constant. A sufficient condition is

$$\frac{1}{\lambda(G) \max_v d_v} = \Omega(\mathrm{poly}(d_*, \log n)) \tag{5.87}$$

as claimed. □

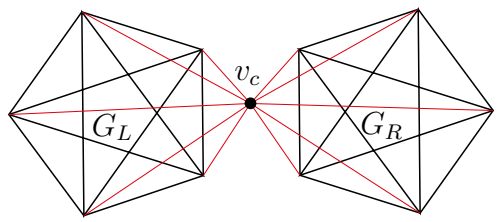


Figure 5.4: The vertex barbell graph B_{2n}, for $n = 5$, that consists of two complete graphs connected on a vertex. We have that $\lambda(G) \leq 2/n$ by Lemma 5.15.

We call the type of function that bounds $\mathrm{rt}(G)$ depending on $\mathrm{hqrt}(G)$ the *separation*. Theorem 5.17 does not rule out a superpolynomial separation between classical and Hamiltonian routing on, for example, expander graphs with $\lambda(G) = \Omega(1)$. We note some cases of expander graphs where still no superpolynomial separation is possible. Graphs G of bounded degree have a diameter $\mathrm{diam}(G) = \Omega(\log n)$, giving a lower bound of $\mathrm{hqrt}(G) = \Omega(\log n)$. So, for constant $\lambda(G)$, we have $\mathrm{rt}(G) = O(\log^2 n)$ and only a quadratic separation is possible between Hamiltonian and classical routing.

Additionally, some expander graphs with small diameter are unsuitable for separating gate-based quantum routing from classical routing. The star graph, S_n, has $\lambda(S_n) = 1$ [Chu] but is a poor vertex expander since $c(S_n) = O(n^{-1})$, giving $\mathrm{qrt}(S_n) = \Theta(n)$. Hamiltonian quantum routing, however, could still provide an advantage in this case, since our lower bound on $\mathrm{hqrt}(S_n)$ is trivial. We take a first step in resolving this question in the next section.

5.4 Toward a Separation

We have given necessary conditions for a superpolynomial separation between Hamiltonian and classical routing, but we are not even aware of any super-constant separation. In this section we will show that such a separation is possible in Hamiltonian routing *with ancilla* by considering a vertex bottleneck. This also shows that the Hamiltonian routing with ancilla cannot be lower bounded by $\Omega(1/c(G))$.

We show a separation on a graph B_{2n}, for $n \in \mathbb{N}$, that we call the *vertex barbell*

graph (see Figure 5.4). It consists of two complete graphs G_L, G_R, of n vertices each and a central vertex v_c where each complete graph is fully connected with v_c, forming two complete graphs of size $n+1$ joined at a vertex. We have that $\mathrm{qrt}(B_{2n}) = \Theta(()n)$ since Theorem 5.5 with $c(B_{2n}) \leq 1/n$ implies the lower bound and a trivial SWAP routing strategy implies the upper bound.

Toward the separation, we consider the stronger model of *Hamiltonian routing with ancilla*. This model is based on Hamiltonian routing with two additional assumptions: (i) each qubit has one associated ancilla qubit available, and (ii) the ancilla can only perform a SWAP with its associated qubit in negligible time cost[1]. Let us denote the *Hamiltonian routing time with ancilla* as

$$\mathrm{hqrt_a}(G) \coloneqq \max_{\pi} \mathrm{hqrt_a}(G, \pi), \tag{5.88}$$

where $\mathrm{hqrt_a}(G, \pi)$ is the routing time in the Hamiltonian routing with ancilla model of π on graph G. As a point of comparison, we may define a modified gate-based quantum routing number $\mathrm{qrt}_a(G)$ analogously. Due to the vertex bottleneck, we still have that $\mathrm{qrt}_a(B_{2n}) = \Theta(n)$.

In Hamiltonian routing with ancilla, we can use a protocol for fast state transfer [Guo+19] to implement a routing protocol for the hard case on B_{2n}.

Theorem 5.18. *Given a vertex barbell graph B_{2n} and a permutation π that permutes all vertices from G_L to G_R and vice versa, we have*

$$\mathrm{hqrt_a}(B_{2n}, \pi) = O(\sqrt{n}). \tag{5.89}$$

[1]Instead of fast SWAPs, we could consider the graph B_{4n}, where half the qubits in G_L and G_R are ancilla. In Hamiltonian routing with ancilla, the ancilla are only connected locally.

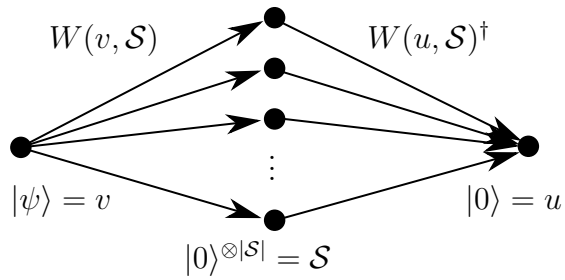


Figure 5.5: In our routing protocol for the vertex barbell graph, we transfer the state $|\psi\rangle$ on qubit v to u by using the intermediate qubits $\mathcal{S}$ as ancilla (in the $|0\rangle^{\otimes|\mathcal{S}|}$ state). The operation $W(v,\mathcal{S})$ creates a W-state dependent on $|\psi\rangle$ over $\mathcal{S}$ in time $\pi/(2\sqrt{|\mathcal{S}|})$ [Guo+19]. Since it is unitary, we can use its inverse to transfer the state to u. We repeat this procedure in G_R to transfer the state to its destination.

Proof. We define a Hamiltonian to construct a W-state [Guo+19]

$$W(x,\mathcal{S}) \coloneqq \sum_{v\in\mathcal{S}} c_x^\dagger c_v + \text{h.c.}, \tag{5.90}$$

where $\mathcal{S} \subseteq V(B_{2n})$, $x \in V(B_{2n}) \setminus \mathcal{S}$, and $c_y = |0\rangle_y\langle 1|_y$ (resp. $c_y^\dagger$), for $y \in V(B_{2n})$, are annihilation (resp. creation) operators. For a time $T = \pi/(2\sqrt{|\mathcal{S}|})$ and state $|\psi\rangle = a_0|0\rangle + a_1|1\rangle$ on x, we have that

$$e^{-iW(x,\mathcal{S})T}(a_0|0\rangle_x + a_1|1\rangle_x)|0\rangle_{\mathcal{S}} = |0\rangle_x(a_0|0\rangle_{\mathcal{S}} + a_1|W\rangle_{\mathcal{S}}), \tag{5.91}$$

where $|W\rangle \coloneqq \frac{1}{\sqrt{|\mathcal{S}|}}\sum_{v\in\mathcal{S}} c_v^\dagger|0\rangle_{\mathcal{S}}$ is the W-state over the qubits $\mathcal{S}$ (an equal superposition over Hamming weight 1 strings).

The protocol is then as follows. We first use (fast) SWAPs between each qubit and its ancilla so all vertices in the graph are in the initial state $|0\rangle$. We now pick some vertex $v \in V(G_L)$ and show how to route the qubit to $\pi(v)$. We SWAP the qubit at v with its ancilla to return to initial state at v. Then, we use $W(v, V(G_L)\setminus\{v\})$ to construct a W-state on G_L, followed by the inverse operation $W(v_c, V(G_L)\setminus\{v\})^\dagger$, which sends the state from v to the central vertex v_c in total time $2T$. We repeat this process to transfer the qubit from v_c to $\pi(v)$. Then, we SWAP the qubit at $\pi(v)$ with

its ancilla. If the qubit that is now at $\pi(v)$ needs to be routied, we follow an analogous procedure and send it to $\pi(\pi(v))$. If it does not (it was an ancilla qubit), we pick some other vertex in $V(G_L)$ that still needs to be routed. We iterate in this way, alternating a vertex from G_R, then G_L, until all vertices are routed to their destination ancilla. Finally, we simultaneously SWAP all qubits with their ancilla to finish the routing. The total time is $4T \cdot 2n = O(\sqrt{n})$. □

We can now generalize the algorithm to all permutations on B_{2n}.

Corollary 5.19. *Given a vertex barbell graph B_{2n}, we have*

$$\mathrm{hqrt_a}(B_{2n}) = O(\sqrt{n}) \tag{5.92}$$

Proof. Let π be any permutation of the vertices $V(G)$. First, we trivially route v_c to $\pi(v_c)$ and $\pi^{-1}(v_c)$ to v_c in $O(1)$ time using SWAPs. Then we route all vertices that are permuted only within G_L or G_R in $O(1)$ time using swaps since $\mathrm{rt}(K_n) \leq 2$ [ACG94]. Finally, consider the vertex barbell subgraph of the remaining vertices that need to route between G_L and G_R plus v_c. This can be routed in $O(\sqrt{n})$ time by Theorem 5.18. □

This shows a quadratic separation

$$\mathrm{qrt}(B_{2n}) \geq \mathrm{qrt}_a(B_{2n}) = \Omega(\mathrm{hqrt_a}(B_{2n})^2) \tag{5.93}$$

and $\mathrm{hqrt_a}(B_{2n}) \notin \Omega(c(B_{2n}))^{-1}) = \Omega(n)$.

5.5 Conclusion

In this chapter we have extended the scope of routing from classical routing and introduced gate-based and Hamiltonian models of routing, investigated lower bounds,

and also separations. We showed some conditions on the spectrum for when a superpolynomial separation can be excluded, in particular for graphs with poor expansion. Another case that can be excluded are expander graphs with $\lambda(G) = \Omega(1)$ but bounded degree since these can exhibit at most a quadratic separation between Hamiltonian and classical routing.

One prominent open question is whether the star graph S_n, which has $\lambda(S_n) = 1$, can exhibit a separation. We showed $\mathrm{qrt}(S_n) \geq n - 1$, whereas we are only able to show that a trivial lower bound on Hamiltonian routing, $\mathrm{hqrt}(S_n) = \Omega(1)$, holds. We showed an $\Omega(\sqrt{n})$ separation for a strengthened model of Hamiltonian routing with ancilla. Therefore, the star graph seems like a good first case to investigate whether a superconstant separation exists between gate-based quantum routing and Hamiltonian routing.

Along the way, we showed bounds on the depth and time required to perform state preparation tasks through entropic arguments. Piroli, Styliaris, and Cirac [PSC21] showed LOCC circuit lower bounds on state preparation and inspired us to show similar state preparation results to lower bound routing. In fact, since the entropy is non-increasing under LOCC, we have that STE, SIE, our state preparation bounds, and thus our routing bounds generalize to models including LOCC. These models give a stronger class of quantum routing (see Figure 1.1) and would allow, e.g., teleportation to bridge long distances. Trivially, this can exceed the Lieb-Robinson velocity [LR72] and would invalidate simple lower bounds based on the diameter of the graph (Theorem 5.1). It is an open question how much stronger these models of routing are.

Chapter 6

Surface code compilation via edge-disjoint paths

This chapter is based on

[BKS21] Michael Beverland, Vadym Kliuchnikov, and Eddie Schoute. *Surface code compilation via edge-disjoint paths*. Oct. 21, 2021. arXiv: 2110.11493 [quant-ph]

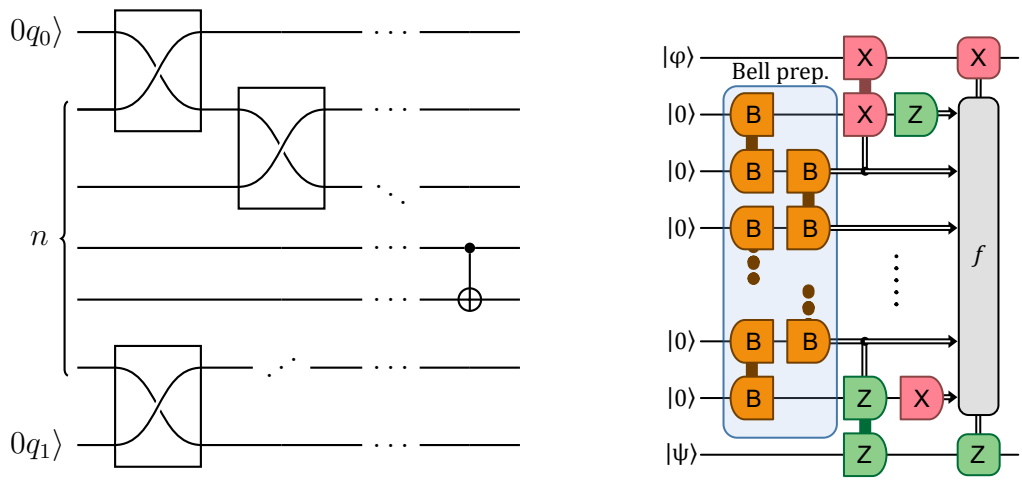
6.1 Introduction

Quantum hardware will always be somewhat faulty and subject to decoherence, due to inevitable fabrication imperfections and the impossibility of completely isolating physical systems. For large computations it becomes a certainty that faults will occur among the many qubits and operations involved. *Fault-tolerant quantum computation* (FTQC) can be implemented despite this by encoding the information in a quantum error correcting code and applying logical operations which are carefully designed to process the encoded information with an acceptably low effective error rate.

The surface code [Kit03; BK98] provides a promising approach to implement FTQC.

Firstly, it can be implemented using geometrically local operations on a patch of qubits in a 2D grid, which is the natural setting for many hardware platforms including superconducting [Fow+12; Cha+20a] and Majorana [Kar+17] qubits. Secondly, the logical qubits it encodes remain protected even for relatively high noise rates, with a threshold of around 1% [WFH11]. Thirdly, a sufficiently general set of elementary logical operations can be performed fault tolerantly on qubits encoded in the surface code using *lattice surgery* [Hor+12]. By tiling the plane with surface code patches, a 2D grid of logical qubits is formed, where the elementary operations are geometrically local; see Figure 1.2. When combined with magic state distillation [BK05] these operations become universal for quantum computing. Indeed this approach, which we will refer to as the *surface code architecture*, is seen as among the most promising by many research groups and companies working in quantum computing [Fow+12; Cha+20b; YK17; FC16].

In this work, we seek to minimize the resources required to fault-tolerantly implement a quantum algorithm using the surface code architecture, which we will refer to as the *surface code compilation problem*. For concreteness, we will assume that the input quantum algorithm is expressed as a quantum circuit composed of preparations and destructive measurements of individual qubits in the Z or X basis, controlled-not (CNOT), Pauli-X, -Y, and -Z, Hadamard (H), Phase (S) and T gates. Our results can be easily generalized to broader classes of input quantum circuits. The output is the quantum algorithm executed using the elementary logical surface code operations shown in Figure 1.2. Ultimately, we would like to minimize the *physical space-time cost*, which is the product of the number of physical qubits and the time required to run an algorithm. To avoid implementation details, we instead minimize the more abstract *logical space-time cost*, which is the number of logical qubits (the circuit width) multiplied by the number of logical time steps (the circuit depth) of the algorithm expressed in elementary surface code operations. The logical and physical space-time



(a) With SWAP operations **(b)** With Bell pair

Figure 6.1: Application of a CNOT(q_0, q_1) on distant qubits using surface code operations in two ways. (a) Using a SWAP-based approach requires $\Omega(n)$ depth using operations from Figure 1.2, while (b) generating and consuming a Bell pair [LO17] can be implemented in constant depth. The classical function f computes Pauli corrections on the output qubits.

costs are expected to be 1-to-1 and monotonically related (see Appendix B.2), such that minimizing the former should minimize the latter.

A well-established approach to implement surface code compilation is known as sequential Pauli-based computation [Lit19], where non-Clifford operations are implemented by injection using Pauli measurements, and Clifford operations are conjugated through the circuit until the end. The circuit that is run in this approach then consists of a sequence of high-weight Pauli measurements which can have overlapping support leading them to be measured one after the other. For large input circuits this can be problematic because highly parallel input circuits can become serialized with prohibitive runtimes.

A major challenge to solve the surface code compilation problem is that quantum algorithms typically involve operations between logical qubits that are far apart when laid out in a 2D grid. One approach to deal with a long-range gate is to perform classical routing around until the pair of interacting qubits are next to one-another (Chapter 2). However, this can result in a deep circuit, see Figure 6.1a. A more

efficient approach is to create long-range entanglement by producing Bell pairs, which for example can be used to implement a long-range CNOT with a constant-depth circuit [Bri+98; LO17; Jav+17] (see Figure 6.1b). Both of these approaches can be implemented with the elementary operations of the surface code.

Moreover, algorithms typically consist of many long-range operations that can ideally be performed in parallel. For architecture-respecting circuit transformations based on classical routing, this can be done by considering a permutation of the logical qubits which is implemented by a sequence of SWAPs [Lao+18; Mur+19; ZW19]. Finding these SWAP circuits reduces to a routing problem on graphs (Chapter 2 and [SHT19]). There are efficient classical routing algorithms that solve this problem for certain families of graphs, however finding the minimal depth solution for a general graph is NP-hard [BR17].

In this chapter, we provide a solution to the surface code compilation problem which generalizes the use of entanglement for long-range CNOTs discussed above to the implementation of many long-range operations in parallel. In particular, we propose *Edge-Disjoint Paths Compilation* (EDPC), which is a computationally efficient classical algorithm tailored to the elementary operations of the surface code. We find evidence that our EDPC algorithm significantly outperforms other approaches by performing a detailed cost analysis for the execution of a set of quantum circuits benchmarks.

EDPC reduces the problem of executing quantum circuits to problems in graph theory. Logical qubits correspond to graph vertices, and there is an edge between qubits if the elementary surface code operation can be applied between them. We show how to perform multiple long-range CNOTs in constant depth along a set of edge-disjoint paths (EDP) in the graph. In other words, long-range CNOTs can be performed simultaneously, in one round, if their controls and targets are connected by edge-disjoint paths. This leads to the well-studied problem of finding maximum EDP sets between pairs of vertices [Kle96]. Therefore, this implementation of long-range

CNOTs along with the elementary operations allow compilation of Clifford operations.

The final operations that complete our gate set for universal quantum computation with the surface code are T gates. The T gates are not natural operations on the surface code, but can be implemented fault-tolerantly by consuming specialized resource states, called "magic states". These magic states can be produced using a highly-optimized process called magic state distillation, which we assume occurs independently of the computation on our code. We assume that logical magic states are available in a specified region of the grid. EDPC reduces magic state delivery to simple MAX FLOW instances that have known efficient algorithm [FF56].

The outline of the chapter is as follows. In Section 6.2, we construct key higher-level components from the basic surface code operations in Figure 1.2 including simple long-range operations. These long-range operations allow us to perform many parallel CNOT operations given vertex-disjoint and edge-disjoint paths that connect the data qubits in Section 6.3. Because of its importance to the algorithms, there we also compare the state of the art graph algorithms for finding vertex-disjoint or edge-disjoint sets of paths and analyze their relation to our algorithms. We complete our gate set by giving an algorithm for efficient remote rotations using magic states at the boundary in Section 6.4. Putting parallel long-range CNOT and remote rotations together, we construct our circuit compilation algorithm, EDPC, in Section 6.5. Finally, we compare the performance of EDPC to prior surface code compilation work in Section 6.6, note its connections to network coding [BH20], and give numerical results comparing the space-time performance with a SWAP-based compilation algorithm.

6.2 Key circuit components from surface code operations

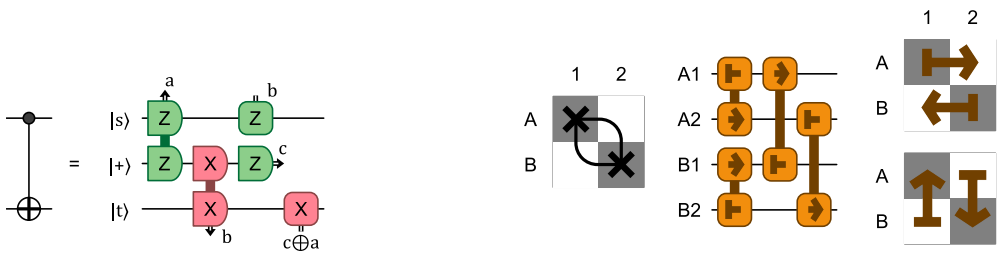
Recall that our goal in this work is to develop an efficient classical compilation algorithm which re-expresses a quantum algorithm into one that uses the elementary operations of the surface code with a low logical space-time cost. In Appendix B.1 we give an overview of the surface code and justify the resource costs of the elementary operations shown in Figure 1.2. The initial quantum algorithm is assumed to be expressed as a circuit diagram involving preparations and measurements of individual qubits in the computational basis, controlled-not (CNOT), Pauli-X, -Y, and -Z, Hadamard (H), Phase (S) and T gates. In this section we build and calculate the cost of some key circuit components from the elementary surface code operations in Figure 1.2. The contents of this section are reproductions or straightforward extensions of previously-known circuits.

6.2.1 Single-qubit operations

Some of the operations of the input circuit can be implemented directly with elementary surface code operations, namely the preparation and measurement of individual qubits in the measurement basis, and the Hadamard gate (provided three neighboring ancillary patches are available as ancillas, see Figure 1.2). Pauli operations do not need to be implemented at all since they can be commuted through Clifford gates and arbitrary Pauli gates [Kni05] and can therefore be tracked classically and merged with the final measurements. For this reason, while we occasionally explicitly provide the Pauli corrections where instructive, we often show equivalence of two circuits only up to Pauli corrections. The remaining single-qubit operations in the input circuit, namely the S and T gates, can be implemented using magic states and is addressed in Section 6.4.

6.2.2 Local CNOT and SWAP gates

An important circuit component is the CNOT gate, which can be implemented as shown in Figure 6.2a [ZBL08]. The qubits involved in this example are stored in adjacent patches, i.e., it is local. Another useful operation is a SWAP of a pair of qubits stored in nearby patches. The surface code's move operation shown in Figure 1.2 gives a straightforward way to implement this as shown in Figure 6.2b. With these implementations, the CNOT requires one ancilla patch, while SWAP requires two. Both are depth 2.



(a) CNOT gate **(b)** SWAP gate

Figure 6.2: A CNOT gate can be implemented in depth 2 using ZZ and XX joint measurements with a $|+\rangle$ ancilla state, followed by classically controlled Pauli corrections. The SWAP gate can be implemented using four move operations and two ancillas in depth 2.

6.2.3 Long-range CNOT using SWAP gates

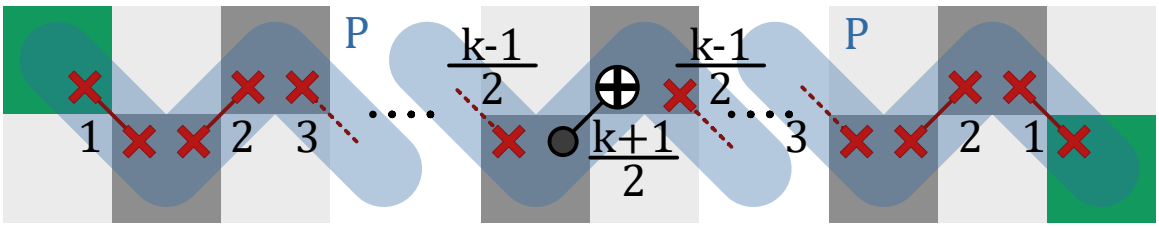


Figure 6.3: A non-local CNOT can be implemented using SWAPs which takes depth $2\lceil\frac{k-1}{2}\rceil$ using a zig-zag of ancilla patches along the path P of length k. The figure shows the case when k is odd and SWAPs depth is $2(k-1)$. The patches on the path can store other logical information, which will simply be moved during the SWAP gates. The patches adjacent to the path are ancillas which are used to implemented the SWAP gates.

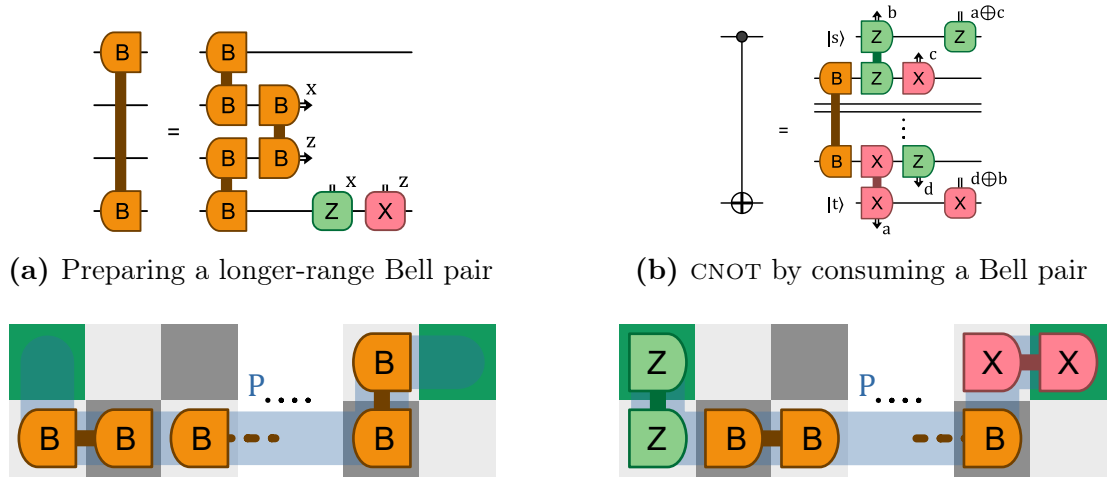
Typical input circuits for surface code compilation will involve CNOT operations on pairs of qubits that are far apart after layout. A very intuitive approach to apply a long-range CNOT(q_1, q_2) gate is shown in Figure 6.3. This involves making use of SWAP gates to first move the qubits q_1 and q_2 so that they are near one another, and then use the local CNOT gate in Figure 6.2a. Let the path $P = v_1 v_2 \dots v_k$, for $k \in \mathbb{N}$, where $v_1 = q_1$ and $v_k = q_2$. As each swap has depth 2, we get a circuit of depth $2\lceil \frac{k-1}{2} \rceil$ since we can perform SWAPs on either end simultaneously. Afterwards, the two qubits are adjacent and we simply perform a CNOT in depth 2.

A lower bound on the depth it takes to perform a long-range CNOT gate using swaps is proportional to the length of the shortest q_1-q_2 path. To move a qubit k patches using SWAPs takes depth exactly $2k$. Therefore, to move control and target to the middle of the shortest path connecting them, it must take time proportional to at least half the length of the path.

6.2.4 Long-range CNOT using a Bell pair

A circuit component that we make extensive use of in this chapter is the long-range CNOT using a Bell pair [LO17]. This allows us to apply CNOTs in depth 2 between any pair of qubits (provided there is a path of ancilla qubits which connects them).

To understand the construction, we first show in Figure 6.4a how to prepare a longer-range Bell pair from two Bell pairs. By iterating this construction one can form a circuit to prepare a long-range Bell pair at the ends of any path of adjacent ancilla patches in depth 2. Next, we show in Figure 6.4b how to implement a CNOT operation between qubits stored in patches neighboring a pair of patches storing a Bell pair. Putting these together, using a path of ancilla patches between a pair of qubits, a long-range CNOT can be implemented in depth 2 in a two-step circuit shown in Figure 6.4c and Figure 6.4d respectively. This approach can be used to implement the CNOT in depth 2 circuit using any path from the control to the target qubit which



(a) Preparing a longer-range Bell pair

(b) CNOT by consuming a Bell pair

(c) Preparing and consuming a Bell pair for long-range CNOT I

(d) Preparing and consuming a Bell pair for long-range CNOT II

Figure 6.4: A long-range CNOT can be implemented in depth 2 by first preparing a Bell pair. (a) Joining Bell-pairs with Bell measurements. This can be iterated to form a long-range Bell pair along any path of ancillas in depth 2. (b) A Bell pair can be used to apply a CNOT. (c,d) The first and second steps of a depth-2 circuit that implements a CNOT between a pair of patches at the end of a path of ancilla patches by preparing and consuming a Bell pair.

starts with a vertical edge and ends with a horizontal edge. There is also flexibility in the precise arrangement of the Bell pairs and Bell measurements along the path using the circuits in Appendix B.3.

Note that here we have focused on implementing a long-range CNOT by constructing and consuming a Bell pair. However a similar strategy (of first preparing a long-range Bell pair in the patches at the ends of a path of ancillas) can be used to implement other long-range operations such as teleportation.

6.3 Parallel long-range CNOTs using Bell pairs

Here, we generalize the use of Bell pairs from the setting of compiling an individual non-local CNOT gate into surface code operations to the setting in which a set of parallel non-local CNOT gates are compiled. In Figure 1.2 and the circuit components in Section 6.2, ancilla qubits are used to perform some operations on data qubits. To

consider the compilation on large sets of qubits, we must specify the location of data and ancilla qubits: here we assume a 1 data to 3 ancilla qubit ratio, as illustrated in Figure 6.6.

In Section 6.3.1 we discuss some relevant background on sets of vertex-disjoint paths (VDP) and sets of edge-disjoint paths (EDP) in graphs. Then in Section 6.3.2 we define the *VDP subroutine* and the *EDP subroutine* that apply parallel CNOT gates at the ends of a particular type of VDP or EDP set. In Section 6.3.3, we show how to use the EDP subroutines to compile a parallel CNOT circuit, consisting of disjoint CNOT operations, to the surface code architecture and prove bounds on the performance of this approach.

6.3.1 Vertex-disjoint paths (VDP) and edge-disjoint paths (EDP)

In Section 6.2.4 we saw that a long-range CNOT could be implemented with the use of a Bell pair produced with a path of ancilla qubits connecting the control and target of the CNOT. A barrier to implement multiple CNOTs simultaneously can arise when an ancilla resides in the paths associated with multiple different CNOTs. This motivates us to review some relevant theoretical background concerning sets of paths on graphs.

Given a graph G, a set of paths $\mathcal{P}$ is said to be a *vertex-disjoint-path* (VDP) set if no pair of paths in $\mathcal{P}$ share a vertex, and an *edge-disjoint-path* (EDP) set if no pair of paths in $\mathcal{P}$ share an edge. Note that a set of vertex-disjoint paths is also edge-disjoint. Further consider a set of *terminal pairs* $\mathcal{T} = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ for terminals $s_i, t_i \in V(G)$, the vertices of G, and $i \in [k]$. We then say that a set of paths $\mathcal{P}$ is a *VDP set for* $\mathcal{T}$ (respectively an *EDP set for* $\mathcal{T}$) if $\mathcal{P}$ is a VDP set (respectively an EDP set), and each path in $\mathcal{P}$ connects a distinct pair in $\mathcal{T}$. These path sets do not necessarily connect all pairs in $\mathcal{T}$. In what follows, we pay special attention to the square grid graph (see Figure 6.8a). The grid graph is relevant for qubits in the

surface code as shown in Figure 1.2, where the vertices correspond to code patches and edge connect vertices associated with adjacent patchs[1].

The problems of finding a maximum (cardinality) VDP set for $\mathcal{T}$ or a maximum EDP set for $\mathcal{T}$ have been well-studied and there are known efficient algorithms capable of finding approximate solutions to each. Unfortunately, on grids it is particularly hard to approximate the maximum VDP set. In particular, for $N := |V(G)|$ there exist terminal sets for which no efficient algorithm can find an approximate solution to within a $2^{O(\log^{1-\epsilon} N)}$ factor of the maximum set size for any $\epsilon > 0$, unless NP $\subseteq$ RTIME($N^{\text{poly}\log N}$) [CKN18]. However, efficient algorithms are available if one is willing to accept a looser approximation to the optimal solution. For example, a simple greedy algorithm is an $O(\sqrt{N})$-approximation algorithm for finding the maximum VDP set [KS04; KT06a], i.e., it produces a VDP set to within an $O(\sqrt{N})$ multiplicative factor of the optimal solution for any graph, not just the grid. For grids, the best efficient algorithm that is known is an $\tilde{O}(N^{1/4})$-approximation algorithm [CK15], where $\tilde{O}(\cdot)$ hides logarithmic factors of $O(\cdot)$.

The situation is better for approximation algorithms of the maximum EDP set: There is a $\Theta(\sqrt{N})$-approximation algorithm [CKS06] for any graph, and on grids Aumann and Rabani [AR95] showed an $O(\log N)$-approximation algorithm that was later improved to an $O(1)$-approximation algorithm [KT95; Kle96]. In practice, these algorithms can be technical to implement and can have large constant prefactors in their solutions that can be prohibitive for the instance sizes that we consider. A simple greedy algorithm forms a $O(\sqrt{N})$-approximation algorithm [KS04] for finding a maximum EDP set on the two-dimensional grid and does not suffer from the constant prefactors of the asymptotically superior alternatives. The dominant runtime complexity of this greedy algorithm is mainly in finding shortest paths for each terminal

[1]Later we will consider a modification of the square grid graph because our algorithms require some further restrictions on the paths, for example preventing them from passing through those vertices associated with data qubits. It is unclear if all of the results in this section also apply for these modified graphs.

pair, giving a $O(|\mathcal{T}|N \log N)$, runtime upper bound by Dijkstra's algorithm[2].

It is informative to consider the comparative size of the maximum EDP and VDP sets for the same terminal set $\mathcal{T}$. Since any VDP set is also an EDP set, the size of the maximum VDP set for $\mathcal{T}$ cannot be larger than the maximum EDP set for $\mathcal{T}$. Moreover, one can construct some cases of $\mathcal{T}$ on the grid [Kle96] in which the maximum EDP set is a factor $\sqrt{N}$ larger than the maximum VDP set [Kle96]. For example, consider the set of terminal pairs $\mathcal{T} = \{((i,1),(L,i)) \mid i \in [L]\}$ of an $L \times L$ grid graph, where vertex (i,j) denotes the vertex in row i and column j. All terminals can be connected by edge-disjoint paths but the maximum VDP set is of size one.

In Section 6.3.2, we show that both VDP and EDP sets for $\mathcal{T}$ can be used to form constant-depth compilation subroutines for disjoint CNOT circuits. Ultimately, as will become clear in Section 6.3.2, each path in the EDP or VDP sets for $\mathcal{T}$ allows us to implement one more CNOT gate in parallel by a compilation subroutine. In this work, we focus on EDPs rather than VDPs for two main reasons. Firstly, as mentioned above, better approximation algorithms exist for finding maximum EDP sets than for finding maximum VDP sets on the grid. Although, in practice, we make use of the greedy $O(\sqrt{N})$-approximation algorithm for finding maximum EDP sets in this work. Secondly, as was also mentioned above, the maximum EDP set is at least as large as the maximum VDP set.

An important open problem that could ultimately influence the performance of the surface code compilation algorithm we present in this work is whether an alternative approximation algorithm for finding maximum EDP sets can be used that performs better in practical instances.

[2]It may be possible to improve the runtime by using a decremental dynamic all-pair shortest path algorithm; it may be quicker to maintain a data structure for all shortest paths that can quickly be updated when edges are removed.

6.3.2 Long-range CNOT subroutines using VDP and EDP

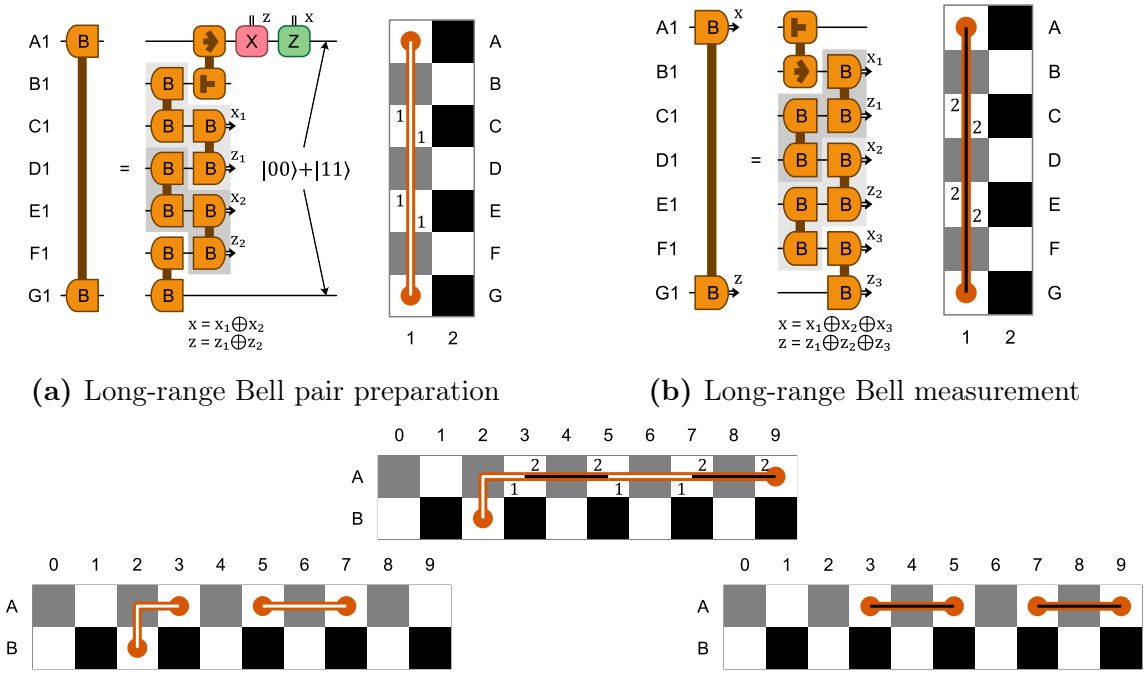
Here we present one of our main technical contributions, namely a description of how to implement a set of long-range CNOTs at the end of VDP and EDP sets using surface code operations. This is central to our overall surface code compilation algorithm presented in Section 6.5.

Consider the $L \times L$ square grid graph G (see Figure 6.8a), which consists of vertices $V(G) = [L] \times [L]$, for $[L] \coloneqq \{1, \ldots, L\}$ and undirected edges

$$E(G) = \{((i,j),(i,j+1)) \mid i \in [L], j \in [L-1]\} \\ \cup \{((i,j),(i+1,j)) \mid i \in [L-1], j \in [L]\}. \quad (6.1)$$

Here, vertices correspond to qubits stored in surface code patches, and edges connect qubits on adjacent patches (see Figure 1.2). We color the vertices of G with three colors: black, grey, and white (see Figure 6.6). All vertices with both even row and even column index are colored black and correspond to data qubits (where data qubits correspond to qubits in the input circuit). The vertices (corresponding to ancilla qubits) with both odd row and odd column index are colored white, and all remaining vertices are colored grey. This gives us a $1:3$ data qubit to ancilla qubit ratio. We set n to equal the number of black vertices, i.e., the number of data qubits.

Due to the designation of some vertices as data qubits and others as ancilla vertices in our layout, and due to the asymmetry of two-qubit operations along horizontal and vertical edges in Figure 1.2, we add some restrictions to the paths we consider. We define an *operator path* to be a path $P = v_1 v_2 \ldots v_k$, for $k \in \mathbb{N}$, such that v_1 and v_k correspond to data qubits and its *interior* $v_2 \ldots v_{k-1}$ are all ancilla qubits. Moreover, v_1 to v_2 must be a vertical edge, and v_{k-1} to v_k must be a horizontal edge. Then an *operator VDP (resp. EDP) set* is a set of vertex-disjoint (resp. edge-disjoint) operator paths. In addition, we require that the ends of the paths in the operator EDP set do



(a) Long-range Bell pair preparation

(b) Long-range Bell measurement

(c) Two-stage Bell preparation using segments

Figure 6.5: (a) For segments marked in white, we use long-range Bell pair preparation in depth 2. (b) For segments marked in black, we then use long-range Bell pair measurement in depth 2. (c) The Bell measurements in stage 2 stitch together the Bell pairs made in phase 1, resulting in a Bell pair in the qubits at the ends of the full path.

not overlap. With the coloring assignments of the grid graph G, it is easy to see that the first and last vertex of an operator path are colored black. In what follows, we show how we can implement CNOTs between the data qubits at the ends of the paths in an operator VDP (EDP) set in constant depth.

First consider an operator VDP set $\mathcal{P}$. It is straightforward to see that we can simultaneously apply long-range CNOTs along each $P \in \mathcal{P}$ as in Figure 6.4 in depth 2. We call this the *vertex-disjoint paths subroutine* (VDP subroutine).

Now consider an operator EDP set $\mathcal{P}$. An EDP set can have intersecting paths, and the ancilla qubits at intersections appear in multiple paths, preventing us from simultaneously producing Bell pairs at their ends. We circumvent this by producing Bell pairs across a path in two stages by splitting the path into segments; see Figure 6.5.

We will show that $\mathcal{P}$ can be *fragmented* into two VDP sets $\mathcal{P}_1$ and $\mathcal{P}_2$ that, together, form $\mathcal{P}$. More precisely, each path $P \in \mathcal{P}$ can be built by composing paths contained in $\mathcal{P}_1$ and $\mathcal{P}_2$ such that each path in either $\mathcal{P}_1$ or $\mathcal{P}_2$ appears in precisely one path in $\mathcal{P}$. We say that the paths in $\mathcal{P}_1$ and $\mathcal{P}_2$ are *segments* of paths in $\mathcal{P}$. This forms the basis of the *edge-disjoint paths subroutine* (EDP subroutine), which is presented in Algorithm 6.3.1 and illustrated with an example in Figure 6.6.

```
Input  : An operator EDP set 𝒫
P₁, P₂ ← fragment 𝒫 in two VDP sets of segments          // Theorem 6.2
for segment P ∈ P₁ :
    if P connects two data qubits then
        execute long-range CNOT along P
    else
        execute phase 1 operation along P (Figure 6.5a, or 6.7b, or 6.7c)
for segment P ∈ P₂ :
    if P connects two data qubits then
        execute long-range CNOT along P
    else
        execute phase 2 operation along P (Figure 6.5b, or 6.7d, or 6.7e)
```

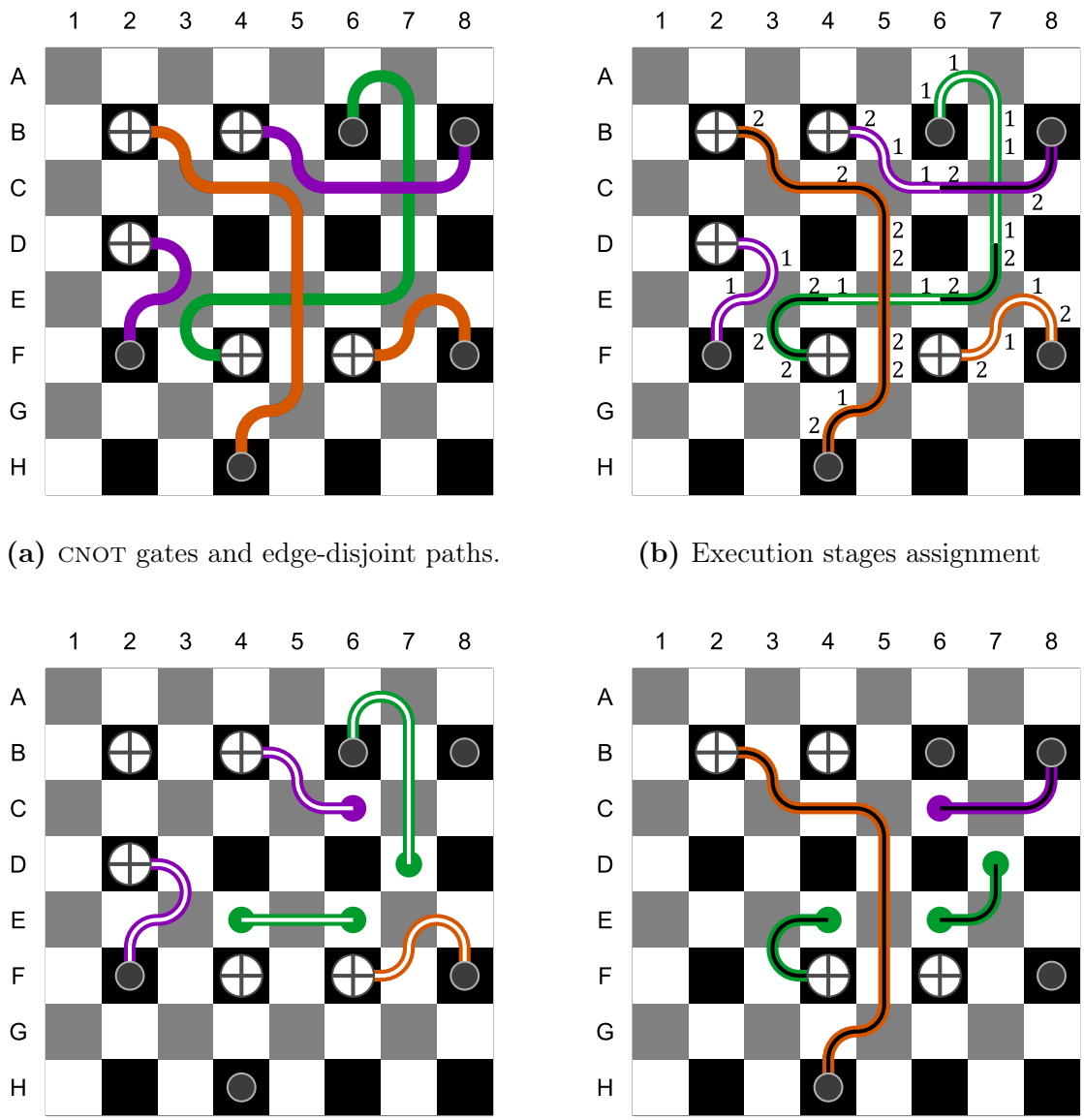
Algorithm 6.3.1: *EDP subroutine*: to apply CNOTs to the data qubits at the endpoints of a set of edge-disjoint paths $\mathcal{P}$, where the interior of each path is supported on ancilla qubits. The depth is at most 4.

We show the following Lemma, which restricts the adjacency of *crossing* vertices. As will become clear later, the adjacent crossing vertices impose systems of constraints on fragmenting $\mathcal{P}$, and their restricted adjacency of any operator EDP set ensures a fragmentation into two VDP sets always exists.

Lemma 6.1. *Given an operator EDP set $\mathcal{P}$, a* crossing *vertex is a vertex contained in more than one path in $\mathcal{P}$. Let the set of crossing vertices be V_c, then the induced subgraph $G[V_c]$ contains only three kinds of connected components:*

1. *Isolated vertices.*

2. *A* horizontal path*, where each vertex (i, j) in the connected component can only be adjacent to $(i-1, j)$ and $(i+1, j)$.*



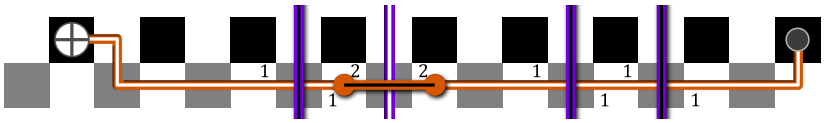
(a) CNOT gates and edge-disjoint paths.

(b) Execution stages assignment

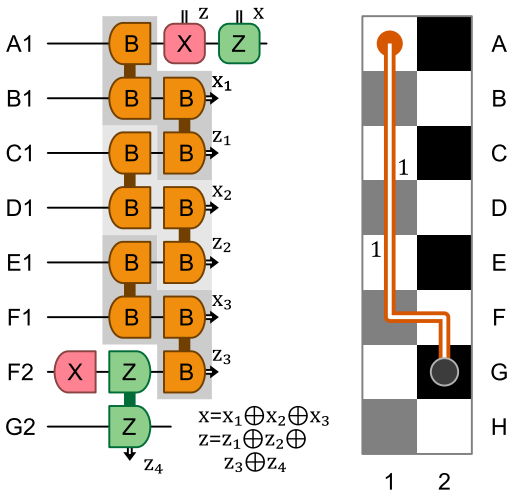
(c) First stage. State preparation.

(d) Second stage. Measurements.

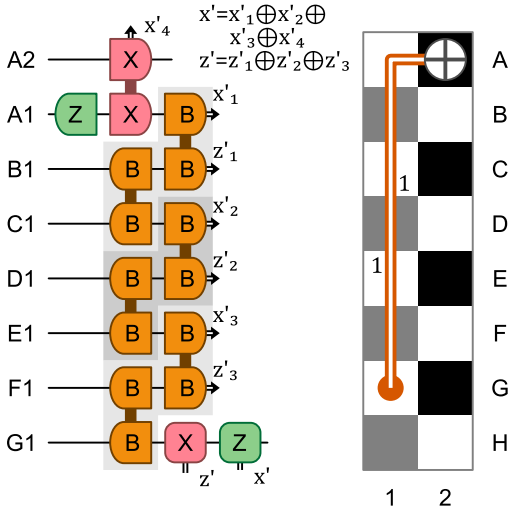
Figure 6.6: The EDP subroutine implements a set of parallel CNOTs connected by an operator EDP set. We assume a qubit ratio of 1 to 3 of data (black) to ancilla (gray and white). (a) The input to the EDP subroutine is a set of CNOTs and an associated EDP set. (b) We fragment the EDP set into two VDP sets consisting of segments of the original paths, and implement the compiled circuit over two depth-2 stages, one for each of these sets. (c) During the first stage we prepare a Bell pair between the ends of the segments in the first VDP set. (d) During the second stage we perform joint Bell measurements between the ends of segments in the second VDP set, producing long-range Bell pairs on ancillas adjacent to the control and target of each CNOT. Then, long-range CNOTs can easily be applied by using the long-range Bell pairs (Section 6.2.4). See Figure 6.7 for further details of the long-range operations used here.



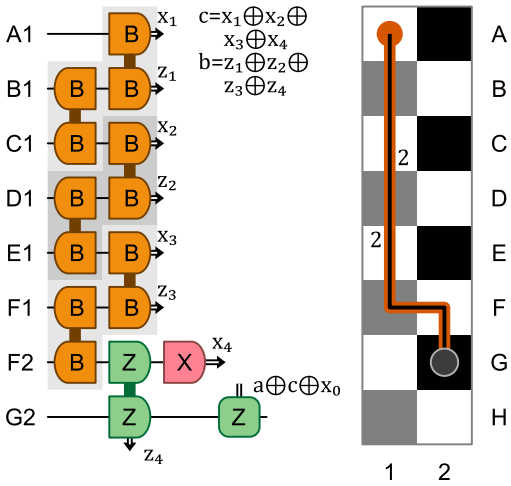
(a) Long-range CNOT in two stages



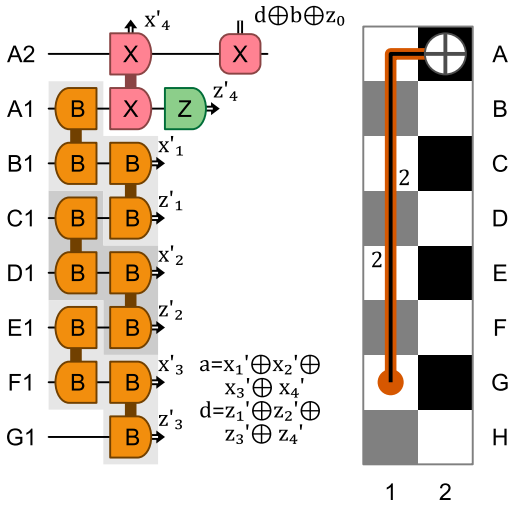
(b) Long-range X prep. with ZZ meas.



(c) Long-range Z prep. with XX meas.



(d) Long-range teleport with ZZ meas.



(e) Long-range teleport with XX meas.

Figure 6.7: Detailed implementation of the steps in Figure 6.6. For each segment that is scheduled in phase 1, we use (b) and (c); and for each supbath that is scheduled in phase 2, we use (d) and (e). In (d) variables x_0 and z_0 equal to the total parity of all long-range Bell measurements applied during stage 2 on the CNOT path. Each of these operations takes depth 2. (d) and (e) share the variables a and c.

3. *A* vertical path*, where each vertex* (i, j) *in the connected component can only be adjacent to* $(i, j-1)$ *and* $(i, j+1)$*.*

Proof. We consider all possible colors of a vertex (i, j) in a connected component of $G[V_c]$. Black vertices cannot be crossing vertices by definition of an operator EDP set so cannot be contained in V_c. It is then easy to see that white vertices in V_c satisfy the Lemma.

Therefore, the only relevant case is when (i, j) is a grey vertex. The vertices $(i+1, j)$ and $(i, j+1)$ are white and $(i+1, j+1)$ is black. We show that these white vertices cannot both be crossing vertices. Suppose that they are, then both edges between the white vertices and the black vertex, $((i+1, j), (i+1, j+1))$ and $((i, j+1), (i+1, j+1))$, are in $\mathcal{P}$. This is a contradiction with the fact that the interior of operator EDP paths cannot contain a black vertex so it must be at the end of two paths, but an operator EDP set cannot contain two paths ending at the same vertex. By the same argument applied to the other white neighbors of (i, j) we see that only $(i-1, j)$ and $(i+1, j)$ or $(i, j-1)$ and $(i, j+1)$ can both be crossing vertices, and the claim follows. □

We now prove that $\mathcal{P}$ can be fragmented.

Theorem 6.2. *We can fragment an operator EDP set* $\mathcal{P}$ *to produce vertex-disjoint sets of segments* $\mathcal{P}_1$ *and* $\mathcal{P}_2$*. If* $\mathcal{P}$ *is vertex-disjoint, then* $\mathcal{P}_1 = \mathcal{P}$ *and* $\mathcal{P}_2 = \emptyset$*.*

Proof. We assign edges for inclusion in segments in $\mathcal{P}_1$ or $\mathcal{P}_2$ by an edge labelling $l(e)\colon E(G) \to \{1, 2\}$. Given a labelling of all edges e in the paths of $\mathcal{P}$, we can assign edges $l(e) = b$ to segments in $\mathcal{P}_b$. Therefore, given a labelling of all edges in paths in $\mathcal{P}$, it is easy to construct $\mathcal{P}_1$ and $\mathcal{P}_2$. We now label all edge in the paths in $\mathcal{P}$ and prove that their labelling guarantees the vertex-disjointness property of $\mathcal{P}_1$ and $\mathcal{P}_2$.

We constrain the labeling around every crossing vertex v so that the VDP property is satisfied. Clearly, v is contained in the interior of exactly two paths, P_1 and P_2. Let

v be contained in edges e_1 and e_1' of P_1, and edges e_2 and e_2' of P_2, then we impose the constraints

$$l(e_1) = l(e_1') \tag{6.2}$$

$$l(e_2) = l(e_2') \tag{6.3}$$

$$l(e_1) \neq l(e_2) \tag{6.4}$$

guaranteeing the vertex-disjointness of segments at v since a segment of P_1 must span both e_1 and e_1', and a segment of P_2 must span both e_2 and e_2' with a different label.

We show there always exists a feasible solution given these constraints. If we consider the graph $G[V_c]$ induced by crossing vertices V_c, then we see that every connected component in $G[V_c]$ gives a system of constraints. The adjacency of $G[V_c]$, by Lemma 6.1, is such that each system has one degree of freedom, which we decide arbitrarily.

Finally, for every vertex disjoint path $P \in \mathcal{P}$, assign $l(e) = 1$ to all edges e in P. All remaining edges can be labeled arbitrarily. □

The depth of a CNOT circuit produced by the EDP subroutine for an operator EDP set $\mathcal{P}$ is at most 4. If $\mathcal{P}$ happens to be vertex-disjoint, then the depth is 2 since all paths are assigned to phase 1 by Theorem 6.2.

6.3.3 Compiling parallel CNOT circuits with the EDP subroutine

In this section we consider how to compile input parallel CNOT circuits using the EDP subroutine. We define the terminal pairs $\mathcal{T} \subseteq V(G) \times V(G)$ to be the pairs of control and target qubits for each CNOT gate in the parallel CNOT circuit. To use the EDP subroutine, we need to find operator EDP sets $\mathcal{P}_1, \ldots, \mathcal{P}_k$ that connect all terminal

pairs in $\mathcal{T}$. We will refer to any such set $\{\mathcal{P}_1, \ldots, \mathcal{P}_k\}$ as a $\mathcal{T}$*-operator set.* The depth of the compiled implementation is minimized when the size k of the $\mathcal{T}$-operator set is minimized.

There are reasons to believe that the compilation strategy for parallel CNOT-circuits formed by finding a minimal $\mathcal{T}$-operator set and applying the EDP subroutine should produce low-depth output circuits. For sparse input circuits, i.e. those with a small number of CNOTs, one can expected a small $\mathcal{T}$ operator set to exist, giving a low depth output. On the other hand, we prove in Theorem 6.3 that there are dense CNOT circuits for which the EDP subroutine with a minimal size $\mathcal{T}$-operator set produces a compiled circuit with optimal depth (up to a constant multiplicative factor).

Theorem 6.3. *Let a parallel input* CNOT *circuit with corresponding terminal pairs* $\mathcal{T}$ *be given such that* $|\mathcal{T}| = n/2$*. Furthermore, let the* n *qubits of the input circuit be embedded in a grid among* $3n$ *ancilla qubits according to the layout in Figure 6.6. For simplicity, we assume* n *is both even and the square of an integer. Then the following holds:*

1. *For all such* $\mathcal{T}$ *we can find* $\mathcal{T}$*-operator set of size* $2\sqrt{n} - 1$ *in polynomial time.*

2. *There exists a parallel* CNOT *circuit with corresponding* $\mathcal{T}$ *that needs depth* $\Omega(\sqrt{n})$ *to be implemented on the surface code architecture.*

Proof. To show (1), for each CNOT we construct an operator path and argue that all such paths can be grouped into $O(\sqrt{n})$ disjoint EDP sets. For simplicity, in the following, we specify paths by a sequence of key vertices, with each consecutive pair of key vertices connected by the shortest path (which is a horizontal or a vertical line).

We now construct an operator path for each CNOT, where the associated control vertex is $v = (v_x, v_y) \in V(G)$ and the target vertex is $u = (u_x, u_y) \in V(G)$. We can always form an operator path to connect u and v given by the following sequence of five key vertices v, $(v_x, v_y - 1)$, $(u_x - 1, v_y - 1)$, $(u_x - 1, u_y)$, u. The path consists of

one horizontal end segment, one vertical end segment, one horizontal interior segment, and one vertical interior segment.

Having assigned $n/2$ such paths, one to each CNOT, we now claim that any of these operator paths can share an edge with at most $2(\sqrt{n}-1)$ of the others. Since the operator paths have distinct endpoints, two different paths cannot collide on either of their end segments $v, (v_x, v_y - 1)$ and on $(u_x - 1, u_y), u$. Therefore pairs of these operator paths can only share an edge on their interior segments. The vertical interior segment of the operator path from v to u can share an edge with at most $\sqrt{n}-1$ other paths. To see this, consider an operator path from v' to u' which shares at least one vertical edge with the operator path from v to u. Explicitly, the segment $(v_x, v_y - 1)$, $(u_x - 1, v_y - 1)$ shares an edge with the segment $(v'_x, v'_y - 1)$, $(u'_x - 1, v'_y - 1)$, which implies that $v_y = v'_y$. Since the terminals are unique, there can only be $\sqrt{n} - 1$ other CNOTs with target in the control v_y where the control of the given CNOT is. An analogous argument applies for horizontal segments, such that the operator path from u to v can share an edge with at most $2\sqrt{n} - 2$ other operator paths.

Let us construct a graph H where each vertex represents an operator path as constructed above. We connect two vertices in H if the associated paths share an edge. Every vertex in H has degree at most $2\sqrt{n} - 2$, therefore, H is $2\sqrt{n} - 1$-colorable using the greedy polynomial time coloring algorithm. Each set of paths associated with vertices of a given color allow us to construct an edge-disjoint set of operator paths $\mathcal{P}$. We perform the EDP subroutine using $\mathcal{P}$ in turn for each color to implement all CNOTs in $\mathcal{C}$ in $O(\sqrt{n})$ depth for any CNOT circuit.

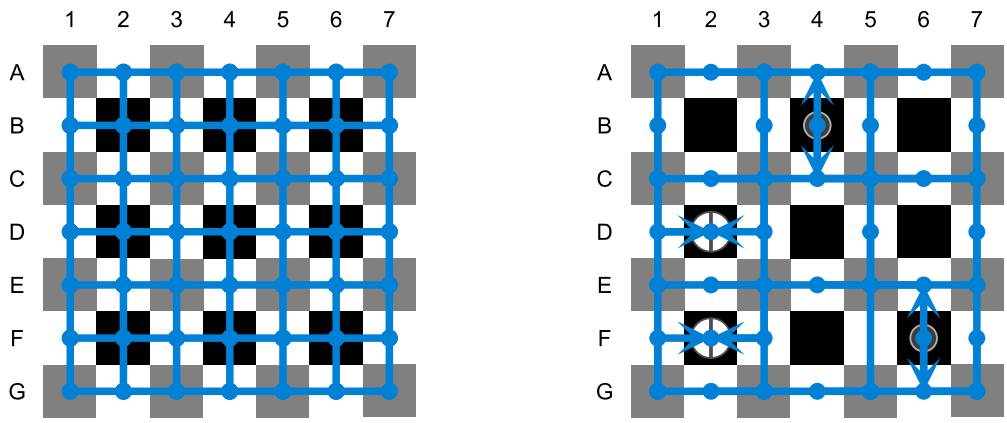
For the purposes of showing the lower bound (2) in the theorem statement, we give an example of an input CNOT circuit that on a particular initial state generates a lot of entanglement between the left and right half of the grid. Let the black vertices on the left of the grid be $X_C = \{(2i, 2j) \mid i \in [L/4], j \in [L/2]\}$ and let X_T be the remaining black vertices on the right of the grid, then we set $\mathcal{T}$ to be an arbitrary

pairing of all vertices from X_C and X_T. It is easy to see that $\mathcal{T}$ corresponds to a CNOT circuit, with controls in X_C and targets in X_T. We initialize the data qubits to a product state $|+\rangle^{n/2}|0\rangle^{n/2}$, where the $|+\rangle$ states are on qubits corresponding to controls X_C and the $|0\rangle$ states are on targets X_T. After applying the CNOTs to each qubit, we obtain a state where each qubit in X_C forms a Bell pair with a qubit in X_T. Therefore, the von Neumann entropy of the reduced state on X is increased by the circuit from 0 to $n/2$.

The lower bound follows an approach similar to that in Delfosse, Beverland, and Tremblay [DBT21] from a bound on the entanglement entropy generation of operations across a cut in the grid. Let $V_L = \{(i,j) \mid i \in [L/2], j \in [L]\}$ be the set of vertices on the left of the grid, and let $\bar{V}_L = V(G) \setminus V_L$ be its complement. Since the entanglement entropy is nonincreasing under local operations and classical communication (LOCC), we allow arbitrary LOCC in 0 depth on the subsystems corresponding V_L and $\bar{V}_L$. The subsystems of V_L and $\bar{V}_L$ can only interact on a boundary subsystem corresponding to the vertices neighboring $\bar{V}_L$ or, respectively, V_L. We can simulate any elementary surface code operation acting on the boundary subsystem by a CNOT followed by LOCC in constant depth, so it suffices to lower bound the depth for unitary operations. Since the vertex boundary is of size $O(\sqrt{n})$, the entanglement entropy generated by any unitary U acting on the boundary subsystem is bounded by $O(\sqrt{n})$ [Ben+03, Lemma 1], so a circuit depth of $\Omega(\sqrt{n})$ is required to prepare the Bell pairs, as claimed. □

In practice, it can be difficult to find minimal-size $\mathcal{T}$-operator sets. However, when the minimal size $\mathcal{T}$-operator set is k, in the following theorem we show that a $\mathcal{T}$-operator set $\{\mathcal{P}_1, \ldots, \mathcal{P}_l\}$ with size at most $l = O(k \log|\mathcal{T}|)$ can be found by a greedy algorithm that finds iteratively finds the maximum operator EDP set for remaining terminals in $\mathcal{T}$.

Theorem 6.4. *On the grid of n vertices, the greedy algorithm for finding $\mathcal{T}$-operator*



(a) Grid graph **(b)** Operator graph

Figure 6.8: The graphs used in this chapter. (a) The grid graph where each surface code patch corresponds to a vertex and is connected to its neighbors. (b) The operator graph for a set of terminal pairs $\mathcal{T}$ that correspond with a parallel CNOT circuit. EDP sets for $\mathcal{T}$ on this graph are also operator EDP sets.

sets repeats the following two steps , for $i = 1, \ldots,$ until there are no more terminal pairs to connect:

1. *find a maximum operator EDP set $\mathcal{P}_i$,*
2. *remove all terminal pairs in $\mathcal{P}_i$ from $\mathcal{T}$.*

The set $\{\mathcal{P}_1, \ldots, \mathcal{P}_k\}$ is a $\mathcal{T}$-operator set and is an $O(\log|\mathcal{T}|)$-approximation algorithm for finding minimum-size $\mathcal{T}$-operator sets.

Proof. We base our proof on [AR95]. Assume that the minimum-size $\mathcal{T}$-operator set is $\{\mathcal{Q}_1, \ldots, \mathcal{Q}_K\}$ for some size K. Then there is an operator EDP set $\mathcal{Q}_i$, for $i \in [K]$, such that $|\mathcal{Q}_i| \geq |\mathcal{T}|/K$. Therefore, the number of unconnected terminal pairs is reduced by at least a factor $(1 - 1/K)$ each iteration and it will require at most $O(K \log|\mathcal{T}|)$ iterations to connect all terminal pairs [Joh74]. □

To make use of Theorem 6.4 we would ideally like to have an algorithm to find maximum operator EDP sets on the grid, however the efficient algorithms we discussed in Section 6.3.1 are to find approximate maximum EDP sets on the grid. Fortunately,

we find an equivalence between operator EDP sets on the grid and EDP sets on a graph that we call *the $\mathcal{T}$-operator graph* (see Figure 6.8b). The $\mathcal{T}$-operator graph is a copy of the grid graph but with all vertices corresponding to control qubits in $\mathcal{T}$ only having vertical outgoing edges, and with all vertices corresponding to target qubits in $\mathcal{T}$ only having horizontal incoming edges, and all remaining vertices corresponding to data qubits are removed. An EDP set for terminal pairs $\mathcal{T}$ on the $\mathcal{T}$-operator graph is an operator EDP set on the grid. It is easy to see that a maximum operator EDP set for $\mathcal{T}$ on the grid is equivalent to a maximum EDP set for $\mathcal{T}$ on the $\mathcal{T}$-operator graph. Using an approximation algorithm for finding the maximum operator EDP set also still gives approximation guarantees for minimizing the $\mathcal{T}$-operator set, as shown in the following Corollary.

Corollary 6.5. *The greedy algorithm for finding minimum $\mathcal{T}$-operator sets, but with a κ-approximation algorithm for finding maximum operator EDP sets, gives an $O(\kappa \log|\mathcal{T}|)$-approximation algorithm for finding minimum $\mathcal{T}$-operator sets.*

Proof. We modify the proof of Theorem 6.4 such that every iteration we connect a $(1-\kappa/K)$ fraction of unconnected terminal pairs using the κ-approximation algorithm for finding maximum operator EDP sets. Therefore we obtain a $O(\kappa \log|\mathcal{T}|)$-approximation algorithm for findining minimum $\mathcal{T}$-operator sets. □

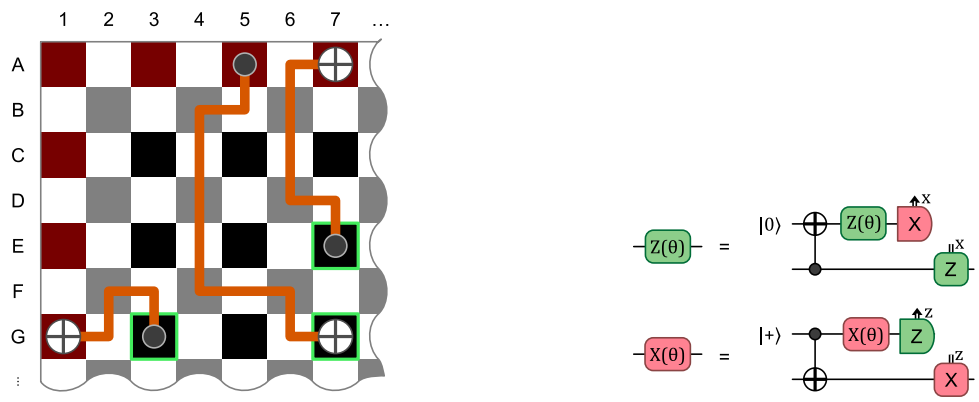
The equivalence between operator EDP sets on the grid and EDP sets on the $\mathcal{T}$-operator graph motivates us to seek an efficient algorithm to find approximate maximum EDP sets on the $\mathcal{T}$-operator graph as a key part of our EDPC algorithm. The algorithms we discussed in Section 6.3.1 come close to doing this, but some of them are intended for finding approximate maximum EDP sets on the grid rather than on the $\mathcal{T}$-operator graph and even if they are adapted, the guarantees of the size of the approximate minimum EDP sets they produce may not apply in the case of the $\mathcal{T}$-operator graph. The algorithms described in Refs. [AR95; KT95] for

finding approximate maximum EDP sets on the grid do not directly apply to the operator graph. While it seems straightforward to adapt the $O(\log n)$-approximation algorithm [AR95], the algorithms in [AR95; KT95] are complex to implement and have large constant-factor overheads, which can make them impractical on small instance sizes. In our EDPC algorithm implementation, we instead use the simple greedy algorithm [KS04], which applies to any graph and approximates the maximum EDP set size up to a factor $O(\sqrt{n})$ and does not have a large constant overhead. We leave it as an open question to find better approximation algorithms for finding maximum operator EDP sets.

6.4 Remote rotations with magic states

Thus far we have discussed the surface code compilation of all the input circuit operations listed in Section 6.1 except for the single-qubit rotation gates $S = Z(\pi/4)$ and $T = Z(\pi/8)$. In this section we design a subroutine for the compilation of parallel rotation circuits. The S and T gates can be implemented by using specially prepared *magic states* $|S\rangle$ and $|T\rangle$, respectively. Magic states can be prepared using a highly-optimized process known as *magic state distillation* [Kni04], which distills many faulty magic states that are easy to prepare into fewer robust states. Still, producing both $|S\rangle$ and $|T\rangle$ involves considerable overhead. The $|S\rangle$ state is used to apply the S-gate in a 'catalytic' fashion, whereby the state $|S\rangle$ is returned afterwards. On the other hand, the state $|T\rangle$ is consumed to apply the T-gate. The reason for this distinction is rooted in the fact that the S-gate is Clifford but the T-gate is non-Clifford.

In this work, we do not address the mechanism by which magic states are produced, but instead assume that these states are provided at specific locations where they can be used to implement gates. More specifically, we assume rotation gates S and T (and also Clifford variations of these such as $X(\pi/8) = T_x$ and $X(\pi/4) = S_x$) can



(a) Long-range CNOTs for diagonal gates

(b) Remote $Z(\theta)$ and $X(\theta)$

Figure 6.9: We assume the capability of performing S and T gates at the boundary qubits (red) where it is easy for us to supply the requisite S and T magic states. We can then execute S or T gates in the Z or X basis for our circuit by using long range CNOTs and the circuits in Figure 6.9b. For example, to execute S or T on qubits G3, E7 and HTH on G7, we apply long-range CNOTs between pairs (G3,G1), (E7,A7), (A5,G7) and then execute S or T on G1, A7, HTH on A5. We can continue applying other Clifford gates to qubits G3, E7, and G7 right after performing the long-range CNOT, without waiting for the Z correction, since we can propagate the correction through Clifford operations.

be applied as a resource on specific ancilla qubits $B \subseteq V(G_{\mathcal{A}})$ at the boundary of a large array of logical qubits (Figure 6.9a). This will allow sufficient space outside the boundary where highly-optimized magic state distillation and synthesis circuits can be implemented. Because a large number of magic states are used in the computation, we consider having magic state distillation adjacent to and concurrent with computation we are concerned with in this chapter to be a reasonable allocation of resources.

We need a technique to apply *remote rotations* to data qubits which can be far from the boundary making use of the rotations that can be performed at the boundary. We make use of the property that any Z rotation (including T or S) has the same action when applied to either qubit in the state $\alpha|00\rangle + \beta|11\rangle$. In particular, these two qubits need not be close to one another, so we can apply Z rotations *remotely*. A similar notion holds for X-rotations (including $T_x = HTH$ or $S_x = HSH$) and $\alpha|{+}{+}\rangle + \beta|{-}{-}\rangle$. Given a qubit q that needs to perform a Z rotation requiring a

magic state, we apply a *remote Z-rotation* (Figure 6.9b): by performing a long-range $\textsc{cnot}(q, q')$ to a boundary ancilla $q' \in B$ prepared in $|0\rangle$. Therefore we can apply the Z rotation remotely and use an X measurement on q' to collapse the state back to one logical qubit. Similarly, for a qubit q that needs to perform an X rotation requiring a magic state, we apply a long-range $\textsc{cnot}(q', q)$ to an ancilla q' prepared in $|+\rangle$ on the boundary, giving

$$\textsc{cnot}|+\rangle(\alpha|+\rangle + \beta|-\rangle) = \alpha|{+}{+}\rangle + \beta|{-}{-}\rangle\,. \tag{6.5}$$

Therefore we apply the X rotation remotely and collapse the state back by a single-qubit Z measurement of q'.

The task of compiling a parallel rotation circuit therefore reduces to applying a set of CNOT gates from the boundary to the sites of the rotation gates. This can be achieved by finding an appropriate EDP set and running the EDP subroutine of Algorithm 6.3.1. Compared to the task of finding an EDP set for parallel CNOT gates of Section 6.3, there is one simplifying condition here: Any boundary qubit can be used for each CNOT when applying remote rotations. As we explain below, we can find the maximum EDP set for the compilation of remote rotations by solving the following (unit) MAX FLOW problem [KT06b].

Definition 6.6 (MAX FLOW)**.** Given a directed graph G and source and sink vertices $s, t \in V(G)$, we wish to find a flow for all edges of G, $f(e)\colon E(G) \to \mathbb{R}$, that is skew symmetric, $f((u, v)) = -f((v, u))$, and, for $v \in V(G) \setminus \{s, t\}$, must respect the constraints

$$f(e) \leq 1 \tag{6.6}$$

$$\text{and} \sum_{u:(v,u)\in E(G)} f((v, u)) = 0 \tag{6.7}$$

such that the outgoing source flow $|f| := \sum_{u:(s,u)\in E(G)} f((s,u))$ is maximized.

To understand why this yields a maximum EDP, we first point out that a solution for which f has binary values provides an EDP set by building paths from those edges e for which $f(e) = 1$. Moreover, this EDP set must be maximum, because a larger EDP set would imply a larger flow than f, which is the maximum flow by definition. Indeed the Ford-Fulkerson algorithm [FF56] solves Max Flow in runtime bounded by $O(|E(G)||f|)$ and finds flow values $f(e) \in \{0,1\}$ on all $e \in E(G)$ because of the unit capacity constraints, $f(e) \leq 1$. Therefore, f corresponds to a maximum EDP set [KT06b, Section 7.6].

The *remote rotation subroutine* in Algorithm 6.4.1 finds subsets of parallel single-qubit rotations that can be performed in depth 4. If only a subset of the given gates is executed, this subroutine may need to be iterated to execute all gates.

Input : A set of data qubits $W \subseteq V(G_{\mathcal{A}})$ that wish to perform gates requiring magic states

1 Create virtual vertices s and t
2 $G' = (V(G_{\mathcal{A}}) \cup \{s,t\}, \{(s,s') \mid s' \in W\} \cup \{(t',t) \mid t' \in B\})$
3 $f \leftarrow$ solve Max Flow on G' using the Ford-Fulkerson algorithm
4 $\mathcal{P} \leftarrow$ construct edge-disjoint set of s–t paths from f
5 remove s and t from each $P \in \mathcal{P}$
6 `execute` remote rotations at boundary with EDP subroutine given $\mathcal{P}$

Algorithm 6.4.1: *Remote rotation subroutine*: executes a subset of single qubit rotations that require magic states at the boundary. For the set of qubits $W \subseteq V(G_{\mathcal{A}})$ that wish to perform a rotation, we find a maximum set of edge-disjoint paths $\mathcal{P}$ connecting W to the boundary data qubits $B \subseteq V(G_{\mathcal{A}})$ by a Max Flow reduction. Using the EDP subroutine (Algorithm 6.3.1), we can perform remote rotations (Figure 6.9) on the subset of qubits connected to the boundary by $\mathcal{P}$ in depth 4.

One could consider a number of generalizations and variations of this compilation subroutine for parallel rotation circuits. For instance, when the number of rotation gates is small, it may be useful to find VDP sets rather than EDP sets so that the VDP subroutine rather than the EDP subroutine can be applied. There is a different

reduction to MAX FLOW in this case which can be obtained by replacing each vertex with two vertices, one with incoming edge and one with outgoing edges, connected by a directed edge with capacity 1. This guarantees only one flow can pass through every vertex.

Although we do not consider other single-qubit rotations in our input circuit for compilation, it is worth noting that any single-qubit rotation gate $Z(\theta)$ can be approximately synthesized to arbitrary precision [RS16] using $|S\rangle$ and $|T\rangle$ states along with the surface code operations shown in Figure 1.2. The approach used to apply S and T gates shown in Figure 6.9a can also be used to apply any rotation $Z(\theta)$ within the grid of surface codes by synthesizing the rotation at the boundary. However, if one considers more general rotations in the input circuit, the time needed for synthesis at the boundary will need to be accounted for and accommodated by other aspects of the overall surface code compilation algorithm. Another extension that can be considered is if multi-qubit diagonal gates are allowed in the input circuit. We show how X and Z rotations generalize to multi-qubit diagonal gates in Appendix B.4, although we do not use this in our surface code compilation algorithm.

6.5 EDPC surface code compilation algorithm

In this section we construct the EDPC algorithm for compiling universal input circuits into surface code operations by combining subroutines Algorithm 6.3.1 and Algorithm 6.4.1 for compiling long-range CNOTs and Z/X rotations respectively. First we provide a more formal definition of surface code compilation:

Definition 6.7 (Surface code compilation)**.** Consider an input quantum circuit of operations $\mathcal{C} = g_1 g_2 \dots g_\ell$, which is a list of length ℓ of operations g_i for $i \in [\ell]$, consisting of: state preparation in X or Z basis; the single-qubit operators X, Y, Z, H, S, T, $S_x = HSH$, $T_x = HTH$; CNOT operations; and X, Z-measurements. Then a

```
Input  : Circuit C with Paulis commuted to the end and merged with
         measurement
while available operations in C :
    execute all available state preparation, measurement, and Hadamard
    G_m ← {available operations g ∈ C | g is S, T, S_x, or T_x}
    G_c ← {available operations g ∈ C | g is CNOT}
    while G_m ∪ G_c ≠ ∅ :
        G ← surface code grid graph
        Paths P_m ← from Algorithm 6.4.1 on G with operations G_m
        Remove edges in P_m from G
        P_c ← approximate max operator EDP set on G with operations G_c
        execute remote rotations along P_m and long-range CNOTs along P_c
          using EDP subroutine
        Remove executed rotations from G_m and CNOTs from G_c
```

Algorithm 6.5.1: The EDPC algorithm for surface code compilation of a circuit $\mathcal{C} = g_1 \dots g_\ell$. An operation g_i is *available* if it has not been executed and all operations g_j with overlapping support, for $j < i$, are executed.

surface code compilation produces an equivalent output circuit $\mathcal{O}$ in terms of surface code operations (Figure 1.2) on a grid of surface codes with S, T, S_x, and T_x rotations applied only at the grid's boundary.

We specify our surface code compilation algorithm EDPC in Algorithm 6.5.1, which attempts to minimize the space-time cost of a circuit expressed in surface code operations. Note that the input circuit is considered to be a sequence of operations rather than a series of time steps that specify the operations in each time step, such that l is the number of operations of the input circuit, not the depth.

In what follows, we find upper bounds on both the classical runtime and on the depth of the output circuits of EDPC given an input circuit with depth D acting on n qubits. It is useful to note that each of the D layers of the input circuit can be decomposed into a set of parallel CNOTs followed by a set of parallel rotations, each acting on at most n qubits. First we address the set of parallel CNOTs.

To compile a set of parallel CNOTs, our implementation of EDPC uses a greedy $O(\sqrt{n})$-approximation algorithm for finding maximum operator EDP sets, and then

applies the EDP subroutine. The greedy algorithm has a runtime bounded by $O(n^2 \log n)$ (Section 6.3.1) and the EDP subroutine contributes a $O(1)$-depth portion to the output circuit. The number of iterations the EDP subroutine is used is upper bounded by $O(n)$ for any set of parallel CNOTs because at least one CNOT is compiled every time. Therefore, the CNOTs in all D layers of the input circuit contribute at most $O(Dn^3 \log n)$ to the runtime and $O(Dn)$ to the depth of the output circuit. Note that this is a trivial upper bound on the depth of the output circuit that would be achieved by a compilation algorithm that simply sequentially implemented long-range CNOT gates.

To compile a set of parallel rotations, EDPC finds the max-flow using the Ford-Fulkerson algorithm, which has a runtime bounded by $O(n^2)$ [FF56], and contributes a $O(1)$-depth portion to the output circuit. We show that at most $O(\sqrt{n})$ iterations of this max-flow algorithm are needed to compile any set of parallel rotations.

Lemma 6.8. *Given a circuit of k parallel rotations, EDPC outputs a circuit of depth at most $O(\sqrt{k})$.*

Proof. The remote rotation subroutine finds a maximum flow connecting the data qubits performing rotations to the boundary where every additional unit of flow is one more rotation executed. This maximum flow is equal to the minimum edge-cut separating the data qubits from the boundary [FF56]. The boundary of a rectangle containing k vertices on the grid is of size at least $\Omega(\sqrt{k})$, giving a minimum cut size of $\Omega(\sqrt{k})$. Thus, at most $O(\sqrt{k})$ iterations are necessary to implement all remote rotations, as claimed. □

Using this Lemma, we see that the parallel rotations in all D layers of the input circuit contribute at most $O(Dn^{5/2} \log n)$ to the runtime and $O(D\sqrt{n})$ to the depth of the output circuit. Overall, the CNOT compilation dominates both of these costs such that EDPC runs in time $O(Dn^3 \log n)$ and produces a depth $O(Dn)$ output circuit.

There are various modifications of EDPC that are worth considering. If the $O(\sqrt{n})$-approximation greedy algorithm for finding maximum operator EDP sets was improved to a κ-approximation algorithm, for $\kappa = o(\sqrt{n})$, then the bound on the compiled circuit depth improves to $O(D\kappa\sqrt{n})$. Another point of consideration is that EDPC leans heavily on finding operator EDP paths and the EDP subroutine, but a similar surface code compilation algorithm could be constructed from operator VDP paths and the VDP subroutine instead. We believe that the larger maximum EDP sets should allow EDPC to apply more gates simultaneously (see Section 6.3.1), especially if the improved approximation algorithms on grids can be adapted to finding approximately maximum operator EDP sets. Both of these features can give asymptotic improvements at only a $2\times$ depth increase over the VDP subroutine. However, it is not difficult to construct instances where a VDP-based approach would give a lower depth, motivating a more nuanced trade-off between our EDP-based approach and a VDP-based approach. Another modification that could be beneficial would be to relax the requirement to execute all available gates before moving on to the next set. This could increase the number of long-range gates that are performed in parallel but would require careful scheduling with Hadamard gate execution, which may block some paths.

6.6 Comparison of EDPC with existing approaches

In this section, we compare EDPC with other approaches in the literature. We first mention some of the features and short-comings of the well-established approach of Pauli-based computation Section 6.6.1. Then we address a more recently proposed compilation approach based on network coding in Section 6.6.2. In Section 6.6.3 we specify a SWAP-based compilation algorithm based on Chapter 2 and use this as a benchmark for numerical studies of the performance of EDPC in Section 6.6.4.

6.6.1 Surface code compilation by Pauli-based computation

One well-established surface code compilation approach is known as *Pauli-based computation*, which is described in [Lit19]. For an algorithm expressed in terms of Clifford and T gates, Pauli-based computation first involves re-expressing the algorithm as a sequence of joint multi-qubit Pauli measurements along with additional ancilla qubits prepared in T states. This re-expressed circuit has no Clifford operations, and the circuit depth can be straight-forwardly deduced from the input circuit since each T gate results in two [CC21] joint Pauli measurements[3]. This re-expression of the circuit essentially comes from first replacing each T gate by a small gate teleportation circuit consisting of an ancilla in a T state and a two-qubit joint Pauli measurement, and then commuting all Clifford operations to the end of the circuit. The main advantage of the Pauli-based computation approach is that all Cliffords are removed from the input circuit.

That said, this approach has a major drawback. When a Clifford circuit is commuted through a two-qubit joint Pauli measurement, it is transformed into Pauli measurements which can have support on all logical qubits. Therefore, the resulting circuit may contain measurements with large overlapping support that need to be performed sequentially (even when the T gates in the input circuit were acting on disjoint qubits during the same time step). The sequential nature of the joint measurements causes a fixed rate of T-state consumption that does not grow with the number of logical qubits. A limited rate of T-state consumption can lead to a large space-time cost for circuits with many T gates per time step.

A modified version of this Pauli-based computation compilation algorithm can be used to implement more T gates in parallel [Lit19, Section 5.1]. However, as highlighted in [CC21, Section V.A], this results in a significant increase of total logical

[3] In the scheme presented in [Lit19] only one joint Pauli measurement is needed per T gate, but additional features are required of the surface code such as twist defects which were avoided in [CC21], and which we have avoided in this chapter.

space-time cost when compared to the standard Pauli-based computation compilation algorithm, even when disregarding the increased T-factory costs that would be needed to achieve a higher T state production rate.

In contrast with Pauli-based computation, one of our goals when designing the EDPC algorithm was to maintain the parallelism present in the input circuit, such that input circuits with higher numbers of T gates per time step are compiled to circuits with a higher T-state consumption rate.

6.6.2 Surface code compilation by network coding

Another approach to surface code compilation, based on the field known as *linear network coding* [Ahl+00], can be built from the framework put forward in Beaudrap and Herbert [BH20]. Similar to our EDPC algorithm, the essential idea in this compilation scheme is to generate sets of Bell pairs in order to implement operations acting on pairs of distant qubits.

In the abstract setting of network coding [LL04], one is given a directed graph G_{NC} and a set of terminal pairs $\mathcal{T} = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ for source terminals $s_i \in V(G_{\mathrm{NC}})$ and target terminals $t_i \in V(G_{\mathrm{NC}})$ for $i \in [k]$. Messages are passed through edges according to a linear rule. Namely, the value of the message associated with an edge is given as a specific linear combination of the values of those edges which are directed at the edge's head. One can consider the task of "designing a linear network code" by specifying the linear function at each edge in the graph such that when any messages are input via the source vertices $s_1, \ldots, s_k$, then those same messages are copied over to the corresponding output via the target vertices $t_1, \ldots, t_k$.

A number of works have considered how linear network coding theory can be applied to the quantum setting [LOW10; Kob+09; Kob+11; SGI12; HPE19]. Beaudrap and Herbert [BH20] gives a construction for a constant-depth circuit to generate Bell pairs across the terminal pairs $\mathcal{T}$ on a set of ancilla qubits corresponding to the vertices of

G_{NC} with CNOTs allowed on the edges of G_{NC}. This is similar to, but not precisely the same scenario as we consider for surface code compilation in this chapter since the basic operations are CNOTs rather than the elementary operations of the surface code, and since only ancilla qubits are considered without any data qubits. However, it should be quite straightforward to modify the approach in Beaudrap and Herbert [BH20] to form a surface code compilation algorithm. For example, one could use a layout similar to that which we use for EDPC in Figure 6.6, with G_{NC} corresponding to a connected subset of ancilla qubits among a set of data qubits. The Bell-pairs produced by the linear network coding approach could then be used to compile long-range operations between data qubits.

In such a network coding based compilation algorithm, the task of compiling an input circuit into surface code operations would largely rely on subroutines for (1) identifying $\mathcal{T}$ to implement the circuit's long-range gates, and (2) designing a linear network code for $\mathcal{T}$. A major barrier to forming a usable compilation algorithm with linear network coding is that we are unaware of the existence of any efficient algorithm to design linear network codes, or even to identify if a given terminal pair set admits any linear network code. If such linear network code can be found efficiently, then it is yet unclear if a surface code compilation algorithm based on network coding would outperform EDPC.

6.6.3 Surface code compilation by SWAP

Here we specify a SWAP-based compilation algorithm, stated in Algorithm 6.6.1, which we use to benchmark our EDPC against in Section 6.6.4. We assume the 1-to-1 ancilla-to-data qubit ratio as illustrated in Figure 6.10. This is more qubit-efficient than the 3-to-1 ratio we use for EDPC, and it allows the swap gadget in Figure 6.2b to be implemented between diagonally neighboring data qubits.

The first step of the SWAP-based compilation algorithm is to assign each of the

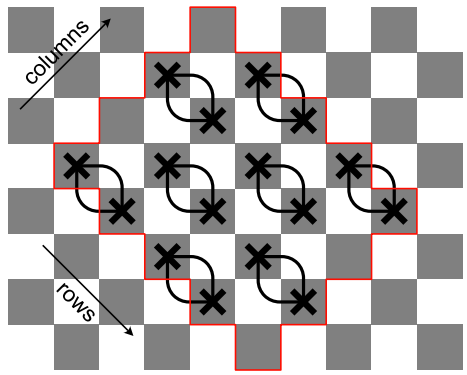


Figure 6.10: On a rotated $L_1 \times L_2$ grid (here, 4×5), we can implement an odd-even pattern of swaps on data qubits (gray) using ancillas (white). Row-wise and column-wise SWAPs used in SWAP routing on a grid [ACG94] can be modified as shown above so that ancilla used for SWAPs do not overlap. Therefore, any arbitrary permutation on a rotated grid of can be implemented in depth $4(L_1 + 1) + 2(L_2 + 1)$.

input circuit's qubits to a data qubit in the layout. Then, the gates in the input circuit are collected together into sets of disjoint gates. Before each set of gates, a permutation built from SWAP-gates is applied, which re-positions the qubits so that the gates in the set can be applied locally. We assume that the available local operations are the same as for our EDPC algorithm. In particular, we assume that the rotation gates (S, T, S_x and T_x) can only be implemented at the boundary and that other single-qubit operations are performed as described in Section 6.2.1. One exception is that we make the simplifying assumption that the Hadamard can be performed without the need of three ancilla patches to simplify our analysis – this assumption could lead to an underestimate of the resources required for this SWAP-based compilation algorithm.

There are two main components of our SWAP-based algorithm which remain to be specified: how the permutations are implemented, and how we choose to separate the input circuit into a sequence of sets of disjoint gates. To permute the positions of data qubits, sequences of SWAP operations are used. As was shown in Ref. [SHT19], any permutation of the n vertices in a square grid can be achieved in at most $3\sqrt{n}$ rounds of nearest-neighbor swaps. To do this involves three stages, with the first and

Input : A circuit $\mathcal{C}$ with all Paulis commuted to the end and merged with measurement

function `cost(`*mapping* π*, vertices* v_1, v_2`):`
  return depth and edge attaining $\min_{e \in \mathcal{M}}$ `depth(route(`$\pi + \{v_1 \mapsto e_1, v_2 \mapsto e_2\}$`))`
while *available gates in* $\mathcal{C}$ **:**
  $\mathcal{G} \leftarrow$ available gates in $\mathcal{C}$
  `execute` all Hadamards and measurements in $\mathcal{G}$
  $G \leftarrow$ surface code grid graph
  $\pi \leftarrow$ empty mapping of $V(G) \to V(G)$
  `// Start modification for operations requiring magic states`
  Set $B \subseteq V(G)$ as the set of boundary vertices
  $\mathcal{G}_m \leftarrow \{g \in \mathcal{G} \mid g \text{ is } S, T, S_x, T_x\}$
  while $\mathcal{G}_m \neq \emptyset$ *and* $B \neq \emptyset$ **:**
    $g \leftarrow$ pop random gate from $\mathcal{G}_m$
    $\pi \leftarrow \pi + \{v \mapsto u\}$, for closest $u \in B$ to v
    remove u from B and G
  `// End modification`
  $\mathcal{M} \leftarrow$ maximum matching of G
  $\mathcal{G}_c = \{g \in \mathcal{G} \mid g \text{ is CNOT}\}$
  while $G_c \neq \emptyset$ *and* $\mathcal{M} \neq \emptyset$ **:**
    $g^*, e^* \leftarrow \max_{g \in \mathcal{G}_c}$ `cost(`$\pi + \{v_1 \mapsto e_1, v_2 \mapsto e_2\}$`)` for v_1, v_2 current location of g
    $\pi \leftarrow \pi + \{v_1 \mapsto e_1, v_2 \mapsto e_2\}$, for v_1, v_2 current location of g^*
    remove g^* from $\mathcal{G}_c$
    remove e^* from $\mathcal{M}$
  `execute` the SWAPs found by `route(`π`)`
  `execute` gates on qubits mapped by π since they are now local

Algorithm 6.6.1: SWAP *compilation*: We construct an algorithm based on the *greedy depth* mapper algorithm from [CSU19]. Let us implicitly define `route(`π`)`, for mapping π, which finds a SWAP circuit for implementing partial permutations [CSU19]. We can compute the required partial permutation from the current mapping of qubits, and the given future mapping π.

third stages each involving rounds of SWAP-gates within rows only, and the second stage involving rounds of SWAP-gates within columns only. A round of SWAP-gates within either rows only or within columns only are implemented with surface code operations as shown in Figure 6.10.

There is considerable freedom in how to collect together gates from the input circuit into sets of disjoint gates. In our implementation in Algorithm 6.6.1, we use the *greedy depth* mapper algorithm from [CSU19], with a small modification to ensure that S and T gates are performed at the boundary. This algorithm also incorporates some further optimizations as described in Ref. [CSU19], including a partial mapping of qubits to locations, leaving the remaining qubits to go anywhere in an attempt to minimize the SWAP circuit depth.

6.6.4 Numerical results

Here we compare our EDPC compilation algorithm (Algorithm 6.5.1) and the SWAP-based compilation algorithm (Algorithm 6.6.1) when applied to a number of different input circuits.

We first consider randomly generated circuits of different gate densities. Our random circuits are built up of sequential sets of disjoint CNOTs that act on randomly selected pairs of data qubits. The density n_{CNOT} of a circuit is how many of the data qubits are involved in a CNOT gate in any such set. Therefore, $n_{\text{CNOT}} = 0.1n$ means that 10% of all qubits (n) are performing a CNOT gate in each set. Our random circuits consist of a sequence of 20 sets of randomly selected disjoint CNOTs at various densities. For each data point, we sample 10 random circuits and plot the mean space-time cost in Figure 6.11 with the standard error of the mean in the shaded region. The runtime of the SWAP protocol was bounded by 2 days, which was insufficient for larger instances of these random circuits at high densities.

We also consider a more structured input circuit, namely implementing half of a

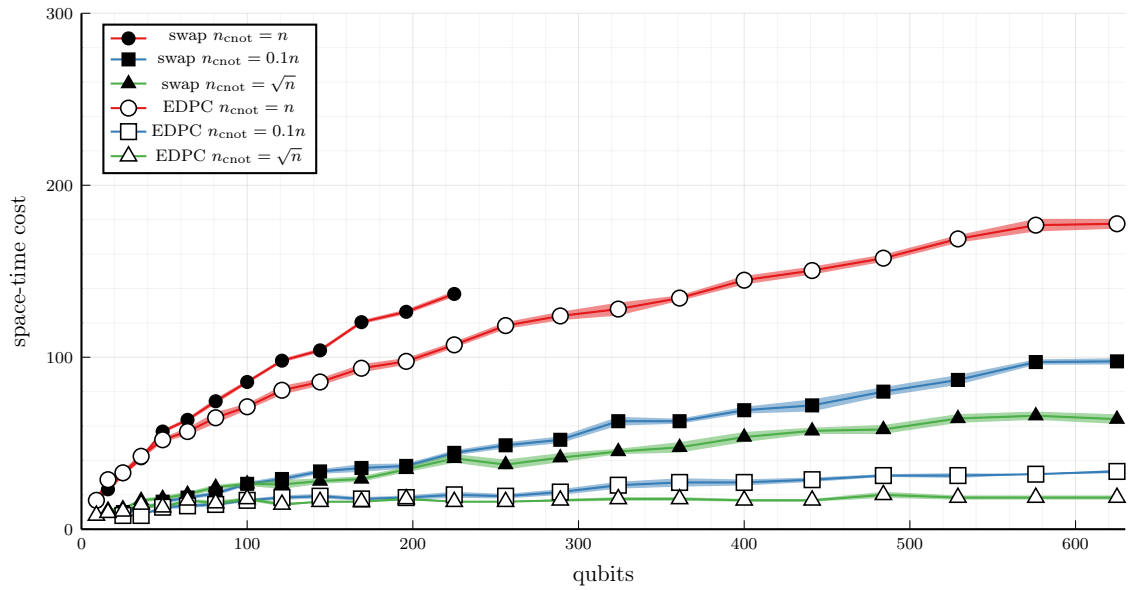
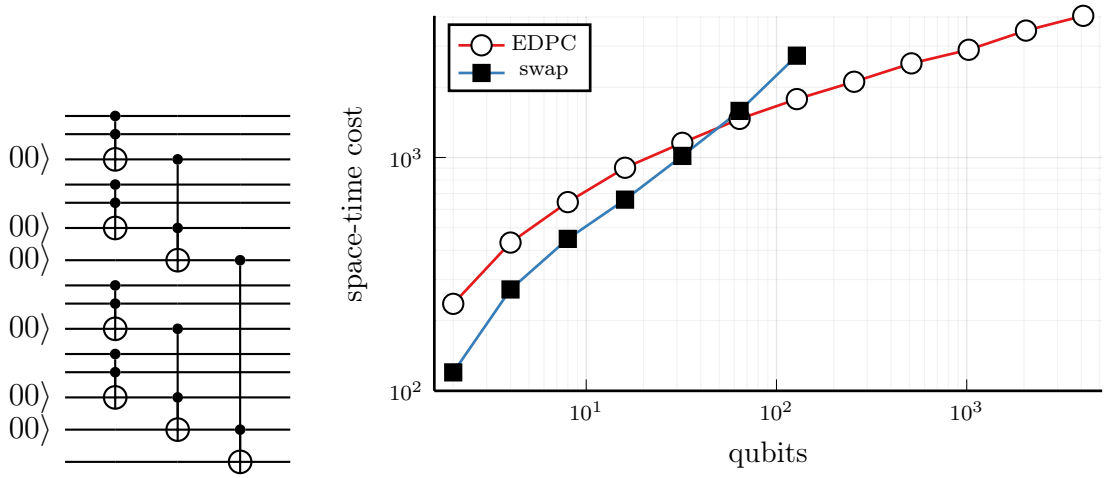


Figure 6.11: Space-time cost of a randomly sampled set of disjoint CNOTs with standard error of the mean (shaded region) compiled to the surface code using EDPC and SWAP compilation. We generate 10 random circuits for each number of qubits (n) consisting of a set of disjoint CNOTs of varying density; the number of randomly selected qubits involved in a CNOT is given by n_{CNOT}. At all densities we see improved performance and scaling using EDPC.

multi-controlled-X gate, C^{k}NOT. We consider decompositions of C^{k}NOT for k integer powers of 2, but only compile the first half of the circuit, given in Figure 6.12a. A T-efficient implementation of C^{k}NOT uses measurement and feedback for uncomputation [Jon13], which are not captured in our model (see Section 6.7). We plot the space-time cost of compiling the half C^{k}NOT in Figure 6.12b. We see that the dependence on the number of qubits k is worse for SWAP-based compilation, and results in a larger space-time cost starting at 64 qubits. Unfortunately, the SWAP-based compilation is quite slow: we ran the algorithm for at most 3 days and 9 hours at each data point and were only able to obtain results up to 128 qubits. However, the data we were able to obtain indicates a cross-over for compiling C^{k}NOT circuits. The SWAP-based compilation has better space-time performance for small instances, while EDPC has a better space-time performance for compiling large C^{k}NOT circuits.



(a) Half C^8NOT **(b)** Compilation of half C^kNOT

Figure 6.12: We compare the space-time cost of compiling a T-gate optimized circuit decomposition for a half C^kNOT circuit to the surface code using EDPC and SWAP compilation. We see in the log-log plot (b) that dependence of the space-time cost on n gives a higher scaling dependence in the case of SWAP compilation than EDPC. This results a lower space-time cost for EDPC starting from 64 qubits.

6.7 Conclusion

In this chapter, we have introduced the EDPC algorithm for the compilation of input quantum circuits into operations which can be implemented fault-tolerantly with the surface code. At the heart of this algorithm is the EDP subroutine, which can implement both sets of parallel long-range CNOT gates and sets of parallel rotations in constant depth, using existing efficient graph algorithms to find sets of edge-disjoint paths. EDPC has advantages over other compilation approaches including Pauli-based computation, network coding based compilation, and SWAP-based compilation. We find numerically that EDPC significantly outperforms SWAP-based circuit compilation in the space-time cost of random CNOT circuits for a broad range of instances, and for larger C^kNOT gates. However, many details of EDPC can be improved, as it is only a first step towards using long-range operations for surface code compilation.

We have argued how the EDP subroutine of the EDPC algorithm can benefit from a rigorous understanding of maximum EDP sets on paths. It seems likely that the

$O(\log n)$-approximation algorithm for finding maximum EDP sets on grids [AR95] can be modified to give an algorithm that restricts to those paths which are needed for the EDP subroutine (which we call operator paths). In our implementation, we have used a simple greedy $O(\sqrt{n})$-approximation algorithm for this task. Any $O(\text{poly}\log n)$-approximation algorithm for finding maximum operator EDP sets would show optimal compilation of worst-case dense parallel CNOT circuits up to logarithmic factors, i.e. a depth bounded by $\tilde{O}(\sqrt{n})$. A more practical issue is to find approximation algorithms with improved asymptotic performance and reasonable constant prefactors.

The runtime complexity of EDPC for an input circuit of depth D acting on n qubits is $O(Dn^3 \log n)$. This is much lower than the known upper bound for the SWAP-based compilation in Section 6.6.3, which was found to be $O(Dn^5)$ in Childs, Schoute, and Unsal [CSU19]. We found that our implementation of the SWAP-based compilation implementation runtime is much slower than that of EDPC on small instances, and found that the SWAP-based algorithm had impractically long runtimes when applied to circuits beyond a few hundred qubits, the regime of large-scale applications of quantum algorithms[Rei+17; GE21]. Potential ways to further improve EDPC's runtime include using a dynamical decremental all-pair shortest path algorithm in the greedy approximation of the maximum EDP set, or by finding faster approximation algorithms for finding the maximum EDP set.

Any diagonal gates in the Z (or X) basis can be performed remotely on the boundary, including CCZ gates [GF19] (see Appendix B.4). Therefore, our results on applying $Z(\theta)$ rotations can be extended to diagonal gates, which will benefit circuit depth.

Even with the capability to perform long-range operations it may still be helpful to localize the quantum information on some part of the architecture such as by permuting the data qubits. In particular, the size of the EDP set is bounded above by the minimum edge cut separating the terminals. Therefore, it may be beneficial

to first redistribute quantum information where it is needed to ensure large EDP solutions exist. It is straightforward to construct a long-range move of a data qubit to an ancilla in depth 2 from a long-range CNOT, by performing the CNOT targeting a $|0\rangle$ ancilla state and measuring the source in the X basis up to Pauli corrections. It also is straightforward to adapt the EDP subroutine to perform sets of these long-range moves along operator paths, now ending at the ancilla, in depth 4. The depth to permute only a few qubits a long distance can be improved significantly by this technique. For example, a SWAP of the two corners of an $L \times L$ grid architecture takes $O(1)$ depth using long-range moves, as opposed to $\Omega(L)$ depth classical routing. It remains an open question how to trade off permuting data qubits (using SWAPs or long-range moves) and directly using long-range CNOTs.

We have assumed that classical feedback is not present in the input circuit for clarity of presentation. The EDPC algorithm we have described could be readily extended to the setting of classical feedback in the input circuit to form a "just-in-time" surface code compilation algorithm. To do so, a larger computation would be broken up into a sequence of circuit executions without classical feedback, where prior measurement results specify the next circuit to compile and execute.

Chapter 7

Conclusion

We showed bounds on the depth and time for increasingly stronger models of quantum routing (see Table 7.1 for an overview) and we showed how routing can be used to construct efficient architecture-respecting circuit transformations. One major open question is if there is a superconstant routing separation between Hamiltonian or gate-based quantum routing and classical routing. We only exhibited an $\Omega(\sqrt{n})$ separation on the vertex barbell graph in the stronger Hamiltonian routing with ancilla model. The gap between our lower bounds on Hamiltonian routing and classical routing are notably large in some simple examples. The star graph, S_n, is one example where we only obtain a lower bound of $\mathrm{hqrt}(S_n) = \Omega(1)$ but have $\mathrm{qrt}(S_n) = \Omega(n)$. Another path toward showing a separation is to investigate generalizations of state reversal to higher dimensional grids.

We have also seen how the ability to perform LOCC on surface codes significantly affect what circuit transformations are possible. We showed that it is not necessary to perform routing for long-range operations on the lattice surgery surface code since parallel long-range operations can be implemented in constant depth, if they can be connected by an edge-disjoint set of operator paths. However, performance of EDPC, which solely uses long-range operations and no routing, does not rule out that an

Model	Routing Bound	Reference
Classical	$\mathrm{rt}(\Pi_{\boldsymbol{v}}(G_1, G_2), \sigma) \leq \left\lceil \frac{\deg(\pi)}{\mathrm{ham}(\boldsymbol{v})} \right\rceil (\mathrm{rt}(G_1) + \mathrm{rt}(G_2)) + \mathrm{rt}(G_2)$	Theorem 2.4
	4-approximation algorithm of $\mathrm{rs}(G, \sigma)$	Theorem 2.10
	$\mathrm{rt}(G) = O\left(\frac{d_*}{\lambda_1^2} \log^2 \lvert V(G) \rvert\right)$	Corollary 5.14
Gate-based	$\mathrm{qrt}(G) \geq \dfrac{2}{c(G)} - 1$	Theorem 5.5
	$\implies \mathrm{qrt}(B_{2n}) = \Theta(n)$	Section 5.4
Hamiltonian	$\mathrm{hqrt}(P_n, \text{reversal}) \leq \dfrac{n+1}{3}$	Chapter 3
	$\mathrm{hqrt}(P_n) \geq \dfrac{4}{3\pi} \dfrac{n}{\alpha}$	Theorem 3.5
	$\mathrm{hqrt}(P_n) \leq (1-\varepsilon)n + O(\log^2 n)$, for $\varepsilon \approx 0.034$	Corollary 4.5
	$\mathrm{hqrt}(G) \geq \dfrac{8}{3\pi} \dfrac{1}{\alpha \cdot h(G)}$	Theorem 5.9
Ham.+ancilla	$\mathrm{hqrt}_{\mathrm{a}}(B_{2n}) = O(\sqrt{n})$	Corollary 5.19

Table 7.1: Summarized results on increasingly stronger models of routing bounding the depth and time (and one entry for the size, $\mathrm{rs}(G, \pi)$). In some cases, the routing bound is dependent on σ, which is an arbitrary permutations of $V(G)$. Special cases are the hierarchical product of graphs G_1 and G_2 with boolean vector $\boldsymbol{v}$, $\Pi_{\boldsymbol{v}}(G_1, G_2)$, the path graph, P_n, and the vertex barbell graph, B_{2n}. We reference where these results were shown in this dissertation. A time-dependent state reversal protocol was known [Rau05], Chapter 3 presents a novel time-independent protocol. The strongest model we consider is Hamiltonian routing with ancilla (denoted "Ham.+ancilla"), which shows an $\Omega(\sqrt{n})$ separation from $\mathrm{qrt}(B_{2n})$.

approach which intermingles long-range operations with routing, e.g. for clusters of operations, would give improved performance on realistic circuits.

It is an open question how much stronger quantum routing using LOCC is. By distributing Bell pairs, we can implement routing through teleportation. This model seems quite strong since measurement-based computation [Bri+09], whose graph states can be constructed in constant depth, is possible in this setting. However, distributing entanglement using graph states can be challenging [HPE19; Adc+20]. If linear network coding is possible on the architecture graph, we also obtain improved entanglement distribution [LOW10; Kob+09; BH20].

Our architecture-respecting circuit transformations in Chapter 2 rely on intelligent qubit mapping that decides where to route qubits. We gave heuristics for qubit mapping that are informed by the classical routing depth, but these heuristics only perform local optimization. Even though the general task is NP-hard [MFM08], it is an open question whether efficient (approximation) algorithms can be designed for specific architectures that will take more of the structure of the circuit into account and are able to improve on benchmarks such as [TC21b].

Another ongoing area of research is circuit optimization and how it can be used in conjunction with architecture-respecting circuit transformations. A common optimization is combining SWAPs from classical routing with preceding gates [ZW19; Siv+20; TC21a; LB21]. Entire CNOT+Rz circuits can also efficiently be resynthesized to architecture-respecting circuits [KG20; NGM20]. And the ZX-calculus has shown promise in circuit optimization [KW20; Dun+20] and can find optimized architecture-respecting circuits [Cow+20].

Optimization can even be applied over the choice of gate set. This is in apparent disregard of the Solovay-Kitaev theorem [Kit97], which tells us that elementary gate sets are equivalent up to polylogarithmic factors. But the (even constant) factors in overhead matter in the non-asymptotic regime, such that optimizing pulse sequences

directly can give large improvements [Shi+19; Lao+21; Jur+21]. In some cases, quantum computers naturally have access to powerful operations. For example, ion trap quantum computer can perform global operations that can implement multi-controlled-NOT gates with k controls in depth $3k/2$ [Grz+21].

Appendix A

Average Hamiltonian routing time on paths

A.1 Average routing time using only SWAPs

In this section, we prove Theorem 4.6. First, define the infinity distance $d_\infty \colon \mathcal{S}_n \to \mathbb{N}$ to be $d_\infty(\pi) = \max_{1 \le i \le n} |\pi_i - i|$. Note that $0 \le d_\infty(\pi) \le n-1$. Finally, define the set of permutations of length n with infinity distance at most k to be $B_{k,n} = \{\pi \in \mathcal{S}_n : d_\infty(\pi) \le k\}$.

The infinity distance is crucially tied to the performance of odd-even sort, and indeed, any classical routing algorithm. For any permutation π of length n, the routing number is bounded below by $d_\infty(\pi)$, since the element furthest from its destination must be swapped at least $d_\infty(\pi)$ times, and each of those SWAPs must occur sequentially. To show that the average routing time of any SWAP-based protocol is asymptotically at least n, we first show that $|B_{(1-\varepsilon)n,n}|/n! \to 0$ for all $0 < \varepsilon \le 1/2$.

Schwartz and Vontobel [SV17] present an upper bound on $|B_{k,n}|$ that was proved in [Klø08] and [TS10]:

Lemma A.1. *For all* $0 < r < 1$, $|B_{rn,n}| \leq \Phi(rn, n)$, *where*

$$\Phi(k,n) = \begin{cases} ((2k+1)!)^{\frac{n-2k}{2k+1}} \prod_{i=k+1}^{2k} (i!)^{2/i} & \text{if} \quad 0 < k/n \leq \frac{1}{2} \\ (n!)^{\frac{2k+2-n}{n}} \prod_{i=k+1}^{n-1} (i!)^{2/i} & \text{if} \quad \frac{1}{2} \leq k/n < 1. \end{cases} \tag{A.1}$$

Proof. Note that $r = k/n$. For the case of $0 < r \leq 1/2$, refer to [Klø08] for a proof. For the case of $1/2 \leq r < 1$, refer to [TS10] for a proof. □

Lemma A.2.

$$n! = \Theta\left(\sqrt{n}\left(\frac{n}{e}\right)^n\right) \tag{A.2}$$

Proof. This follows from well-known precise bounds for Stirling's formula:

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\frac{1}{12n+1}} \leq n! \leq \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\frac{1}{12n}} \tag{A.3}$$

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n \leq n! \leq \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e \tag{A.4}$$

(see for example [Rob55]). □

With Lemmas A.1 and A.2 in hand, we proceed with the following theorem:

Theorem A.3. *For all* $0 < \varepsilon \leq 1/2$, $\lim_{n\to\infty} |B_{(1-\varepsilon)n,n}|/n! = 0$. *In other words, the proportion of permutations of length* n *with infinity distance less than* $(1-\epsilon)n$ *vanishes asymptotically.*

Proof. Lemma A.1 implies that $|B_{(1-\varepsilon)n,n}|/n! \leq \Phi((1-\varepsilon)n, n)/n!$. The constraint $0 < \varepsilon \leq 1/2$ stipulates that we are in the regime where $1/2 \leq r < 1$, since $r = 1 - \varepsilon$. Then we use Lemma A.2 to simplify any factorials that appear. Substituting (A.1) and simplifying, we have

$$\frac{\Phi\left((1-\varepsilon)n, n\right)}{n!} = \frac{\prod_{i=(1-\varepsilon)n+1}^{n-1} (i!)^{2/i}}{(n!)^{2\varepsilon - 2/n}} = O\left(\frac{e^{2\varepsilon n-2}}{n^{2\varepsilon n-2}} \prod_{i=(1-\varepsilon)n+1}^{n-1} \frac{i^{2+1/i}}{e^2}\right). \tag{A.5}$$

We note that $i^{1/i}$ terms can be bounded by

$$\prod_{i=(1-\varepsilon)n+1}^{n-1} i^{\frac{1}{i}} \leq \prod_{i=(1-\varepsilon)n+1}^{n-1} n^{\frac{1}{(1-\varepsilon)n}} \leq n^{\frac{\varepsilon}{1-\varepsilon}} \leq n \tag{A.6}$$

since $\varepsilon \leq 1/2$. Now we have

$$\begin{aligned}
O\left(\frac{e^{2\varepsilon n-2}}{n^{2\varepsilon n-2}} \prod_{i=(1-\varepsilon)n+1}^{n-1} \frac{i^{2+1/i}}{e^2}\right) &= O\left(\frac{n}{n^{2\varepsilon n-2}} \prod_{i=(1-\varepsilon)n+1}^{n-1} i^2\right) && \text{(A.7)}\\
&= O\left(\frac{n}{n^{2\varepsilon n-2}} \left(\frac{(n-1)!}{((1-\varepsilon)n+1)!}\right)^2\right) && \text{(A.8)}\\
&= O\left(\frac{n}{n^{2\varepsilon n-2} e^{2\varepsilon n}} \frac{(n-1)^{2n-1}}{((1-\varepsilon)n+1)^{2(1-\varepsilon)n+2}}\right) && \text{(A.9)}\\
&= O\left(\frac{n}{n^{2\varepsilon n-2} e^{2\varepsilon n}} \frac{n^{2n}}{((1-\varepsilon)n)^{2(1-\varepsilon)n}}\right) && \text{(A.10)}\\
&= O\left(\frac{n^3}{\exp\left((\ln(1-\varepsilon)(1-\varepsilon)+\varepsilon)2n\right)}\right). && \text{(A.11)}
\end{aligned}$$

Since $\ln(1-\varepsilon)(1-\varepsilon)+\varepsilon > 0$ for $\varepsilon > 0$, this vanishes in the limit of large n. □

Now we prove the theorem.

Proof of Theorem 4.6. Let $\bar{T}$ denote the average routing time of any SWAP-based protocol. Consider a random permutation π drawn uniformly from $\mathcal{S}_n$. Due to Theorem A.3, π will belong in $B_{(1-\varepsilon)n,n}$ with vanishing probability, for all $0 < \varepsilon \leq 1/2$. Therefore, for any fixed $0 < \varepsilon \leq 1/2$ as $n \to \infty$, $(1-\varepsilon)n < \mathbb{E}\left[d_\infty(\pi)\right]$. This translates to an average routing time of at least $n - o(n)$ because we have, asymptotically, $(1-\varepsilon)n \leq \bar{T}$ for all such ε. □

A.2 Average routing time using TBS

In this section, we prove Theorem 4.7, which characterizes the average-case performance of TBS (Algorithm 4.4.2). This approach consists of two steps: a recursive call on

three equal partitions of the path (of length $n/3$ each), and a merge step involving a single reversal.

We denote the uniform distribution over a set S as $\mathcal{U}(S)$. The set of all n-bit strings is denoted $\mathbb{B}^n$, where $\mathbb{B} = \{0, 1\}$. Similarly, the set of all n-bit strings with Hamming weight k is denoted $\mathbb{B}^n_k$. For simplicity, assume that n is even. We denote the runtime of TBS on $b \in \mathbb{B}^n$ by $T(b)$.

When running `GDC(TBS)` on a given permutation π, the input bit string for TBS is $b = I(\pi)$, where the indicator function I is defined in (4.7). We wish to show that, in expectation over all permutations π, the corresponding bit strings are quick to sort. First, we show that it suffices to consider uniformly random sequences from $\mathbb{B}^n_{n/2}$.

Lemma A.4. *If $\pi \sim \mathcal{U}(\mathcal{S}_n)$, then $I(\pi) \sim \mathcal{U}(\mathbb{B}^n_{n/2})$.*

Proof. We use a counting argument. The number of permutations π such that $I(\pi) \in \mathbb{B}^n_{n/2}$ is $(n/2)!(n/2)!$, since we can freely assign index labels from $\{1, 2, \ldots, n/2\}$ to the 0 bits of $I(\pi)$, and from $\{n/2 + 1, \ldots, n\}$ to the 1 bits of $I(\pi)$. Therefore, for a uniformly random π and arbitrary $b \in \mathbb{B}^n_{n/2}$,

$$\mathrm{P}[I(\pi) = b] = \frac{(n/2)!(n/2)!}{n!} = \frac{1}{\binom{n}{n/2}} = \frac{1}{|\mathbb{B}^n_{n/2}|}. \tag{A.12}$$

Therefore, $I(\pi) \sim \mathcal{U}(\mathbb{B}^n_{n/2})$. □

While $\mathbb{B}^n_{n/2}$ is easier to work with than $\mathcal{S}_n$, the constraint on the Hamming weight still poses an issue when we try to analyze the runtime recursively. To address this, Lemma A.5 below shows that relaxing from $\mathcal{U}(\mathbb{B}^n_{n/2})$ to $\mathcal{U}(\mathbb{B}^n)$ does not affect expectation values significantly.

We give a recursive form for the runtime of TBS. We use the following convention for the substrings of an arbitrary n-bit string a: if a is divided into 3 segments, we label the segments $a_{0.0}, a_{0.1}, a_{0.2}$ from left to right. Subsequent thirds are labeled

analogously by ternary fractions. For example, the leftmost third of the middle third is denoted $a_{0.10}$, and so on. Then, the runtime of TBS on string a can be bounded by

$$T(a) \leq \max_{i \in \{0,1,2\}} T(a_{0.i}) + \frac{n_1(a_{0.0}) + n_1(\overline{a_{0.2}}) + n/3 + 1}{3}, \tag{A.13}$$

where $\overline{a}$ is the bitwise complement of bit string a and $n_1(a)$ denotes the Hamming weight of a. Logically, the first term on the right-hand side is a recursive call to sort the thirds, while the second term is the time taken to merge the sorted subsequences on the thirds using a reversal. Each term $T(a_{0.i})$ can be broken down recursively until all subsequences are of length 1. This yields the general formula

$$T(b) \leq \frac{1}{3}\left(\sum_{r=1}^{\lceil \log_3(n) \rceil} \max_{i \in \{0,1,2\}^{r-1}} \{n_1(a_{0.i0}) + n_1(\overline{a_{0.i2}})\} + n/3^r + 1\right), \tag{A.14}$$

where $i \in \emptyset$ indicates the empty string.

Lemma A.5. *Let $a \sim \mathcal{U}(\mathbb{B}^n)$ and $b \sim \mathcal{U}(\mathbb{B}^n_{n/2})$. Then*

$$\mathbf{E}[T(b)] \leq \mathbf{E}[T(a)] + \widetilde{O}(n^\alpha) \tag{A.15}$$

where $\alpha \in (\frac{1}{2}, 1)$ is a constant.

The intuition behind this lemma is that by the law of large numbers, the deviation of the Hamming weight from $n/2$ is subleading in n, and the TBS runtime does not change significantly if the input string is altered in a subleading number of places.

Proof. Consider an arbitrary bit string a, and apply the following transformation. If $n_1(a) = k \geq n/2$, then flip $k - n/2$ ones chosen uniformly randomly to zero. If $k < n/2$, flip $n/2 - k$ zeros to ones. Call this stochastic function $f(a)$. Then, for all a, $f(a) \in \mathbb{B}^n_{n/2}$, and for a random string $a \sim \mathcal{U}(\mathbb{B}^n)$, we claim that $f(a) \sim \mathcal{U}(\mathbb{B}^n_{n/2})$. In other words, f maps the uniform distribution on $\mathbb{B}^n$ to the uniform distribution on

$\mathbb{B}^n_{n/2}$.

We show this by calculating the probability $\mathrm{P}[f(a) = b]$, for arbitrary $b \in \mathbb{B}^n_{n/2}$. A string a can map to b under f only if a and b disagree in the same direction: if, WLOG, $n_1(a) \geq n_1(b)$, then a must take value 1 wherever a, b disagree (and 0 if $n_1(a) \leq n_1(b)$). We denote this property by $a \succeq b$. The probability of picking a uniformly random a such that $a \succeq b$ with x disagreements between them is $\binom{n/2}{x}$, since $n_0(b) = n/2$. Next, the probability that f maps a to b is $\binom{n/2+x}{x}$. Combining these, we have

$$\mathrm{P}[f(a) = b] = \sum_{x=-n/2}^{n/2} \mathrm{P}\Big[a \succeq b \text{ and } n_1(a) = \frac{n}{2} + x\Big] \cdot \mathrm{P}\Big[f(a) = b \mid a \succeq b \text{ and } n_1(a) = \frac{n}{2} + x\Big], \tag{A.16}$$

$$= \sum_{x=-n/2}^{n/2} \frac{\binom{n/2}{|x|}}{2^n} \cdot \frac{1}{\binom{n/2+|x|}{|x|}}, \tag{A.17}$$

$$= \frac{1}{\binom{n}{n/2}} \sum_{x=-n/2}^{n/2} \frac{\binom{n}{n/2-x}}{2^n}, \tag{A.18}$$

$$= \frac{1}{\binom{n}{n/2}} = \frac{1}{|\mathbb{B}^n_{n/2}|}. \tag{A.19}$$

Therefore, $f(a) \sim \mathcal{U}(\mathbb{B}^n_{n/2})$. Thus, f allows us to simulate the uniform distribution on $\mathbb{B}^n_{n/2}$ starting from the uniform distribution on $\mathbb{B}^n$.

Now we bound the runtime of TBS on $f(a)$ in terms of the runtime on a fixed a. Fix some $\alpha \in (\frac{1}{2}, 1)$. We know that $n_1(f(a)) = n/2$, and suppose $|n_1(a) - n/2| \leq n^\alpha$. Since $f(a)$ differs from a in at most n^α places, then at level r of the TBS recursion (see (A.14)), the runtimes of a and $f(a)$ differ by at most $1/3 \cdot \min\{2n/3^r, n^\alpha\}$. This is because the runtimes can differ by at most two times the length of the subsequence.

Therefore, the total runtime difference is bounded by

$$\Delta T \leq \frac{1}{3} \sum_{r=1}^{\lceil \log_3(n) \rceil} \min\left\{\frac{2n}{3^r}, n^\alpha\right\}, \tag{A.20}$$

$$= \frac{1}{3} \left(\sum_{r=1}^{\lceil \log_3(2n^{1-\alpha}) \rceil} n^\alpha + 2 \sum_{r=\lceil \log_3(2n^{1-\alpha}) \rceil + 1}^{\lceil \log_3(n) \rceil} \frac{n}{3^r} \right), \tag{A.21}$$

$$= \frac{1}{3} \left(n^\alpha \log(2n^\alpha/3) + 2 \sum_{s=0}^{\lfloor \log_3(n^\alpha/2) \rfloor - 1} 3^s \right) \tag{A.22}$$

$$= \frac{1}{3} \left(n^\alpha \log(2n^\alpha/3) + n^\alpha/2 - 1 \right) = \widetilde{O}(n^\alpha). \tag{A.23}$$

On the other hand, if $|n_1(a) - n/2| \geq n^\alpha/2$, we simply bound the runtime by that of OES, which is at most n.

Now consider $a \sim \mathcal{U}(\mathbb{B}^n)$ and $b = f(a) \sim \mathcal{U}(\mathbb{B}^n_{n/2})$. Since $n_1(a)$ has the binomial distribution $\mathcal{B}(n, 1/2)$, where $\mathcal{B}(k, p)$ is the sum of k Bernoulli random variables with success probability p, the Chernoff bound shows that deviation from the mean is exponentially suppressed, i.e.,

$$\mathrm{P}[|n_1(a) - n/2| \geq n^\alpha] = \exp(-O(n^{2\alpha-1})). \tag{A.24}$$

Therefore, the deviation in the expectation values is bounded by

$$|\mathbf{E}[T(f(a))] - \mathbf{E}[T(a)]| \leq n \exp(-O(n^{2\alpha-1})) + c(1 - \exp(-O(n^{2\alpha-1})))n^\alpha \log(n) = \widetilde{O}(n^\alpha), \tag{A.25}$$

where c is a constant. Finally, we conclude that

$$\mathbf{E}[T(b)] \leq \mathbf{E}[T(a)] + \widetilde{O}(n^\alpha) \tag{A.26}$$

as claimed. □

Next, we prove the main result of this section, namely, that the runtime of `GDC(TBS)` is $2n/3$ up to additive subleading terms.

Proof of Theorem 4.7. We first prove properties for sorting a random n-bit string $a \sim \mathcal{U}(\mathbb{B}^n)$ and then apply this to the problem of sorting $b \sim \mathcal{U}(\mathbb{B}^n_{n/2})$ using Lemmas A.4 and A.5.

The expected runtime for TBS can be calculated using the recursive formula in (A.14):

$$\mathbf{E}[T(a)] \leq \frac{1}{3}\left(\sum_{r=1}^{\log_3(n)} \mathbf{E}\left[\max_{i\in\{0,1,2\}^{r-1}} \{n_1(a_{0.i0}) + n_1(\overline{a_{0.i2}})\}\right] + n/3^r + 1\right). \tag{A.27}$$

The summand contains an expectation of a maximum over Hamming weights of i.i.d. uniformly random substrings of length $n/3^r$, which is equivalent to a binomial distribution $\mathcal{B}(n/3^r, 1/2)$ where we have $n/3^r$ Bernoulli trials with success probability $1/2$. Because of independence, if we sample $X_1, X_2 \sim \mathcal{B}(n/3^r, 1/2)$, then $X_1 + X_2 \sim \mathcal{B}(2n/3^r, 1/2)$. Using Lemma A.6 with $m = 3^{r-1}$, the expected maximum can be bounded by

$$\frac{n}{3^r} + O\left(\sqrt{(n/3^r)\log(3^{r-1}n/3^r)}\right) = \frac{n}{3^r} + \tilde{O}\big(n^{1/2}\big) \tag{A.28}$$

since the second term is largest when $r = O(1)$. Therefore,

$$\mathbf{E}[T(a)] \leq \frac{1}{3}\left(\sum_{r=1}^{\log_3(n)} \frac{2n}{3^r}\right) + \tilde{O}\big(n^{1/2}\big) = \frac{n}{3} + \tilde{O}\big(n^{1/2}\big). \tag{A.29}$$

Lemma A.5 then gives $\mathbf{E}[T(b)] \leq \frac{n}{3} + \tilde{O}(n^{\alpha})$.

The routing algorithm `GDC(TBS)` proceeds by calling TBS on the full path, and then in parallel on the two disjoint sub-paths of length $n/2$. We show that the distributions of the left and right halves are uniform if the input permutation is

sampled uniformly as $\pi \sim \mathcal{U}(\mathcal{S}_n)$. There exists a bijective mapping g such that $g(\pi) = (b, \pi_L, \pi_R) \in \mathbb{B}^n_{n/2} \times \mathcal{S}_{n/2} \times \mathcal{S}_{n/2}$ for any $\pi \in \mathcal{S}_n$ since

$$|\mathcal{S}_n| = n! = \binom{n}{n/2} \left(\frac{n}{2}\right)! \left(\frac{n}{2}\right)! = \left|\mathbb{B}^n_{n/2} \times \mathcal{S}_{n/2} \times \mathcal{S}_{n/2}\right|. \tag{A.30}$$

In particular, g can be defined so that b specifies which entries are taken to the first $n/2$ positions—say, without changing the relative ordering of the entries mapped to the first $n/2$ positions or the entries mapped to the last $n/2$ positions—and π_L and π_R specify the residual permutations on the first and last $n/2$ positions, respectively. Given $g(\pi) = (b, \pi_L, \pi_R)$, TBS only has access to b. After sorting, TBS can only perform deterministic permutations $\mu_L(b), \mu_R(b) \in \mathcal{S}_{n/2}$ on the left and right halves, respectively, that depend only on b. Thus TBS performs the mappings $\pi_L \mapsto \pi_L \circ (\mu_L(b))$ and $\pi_R \mapsto \pi_R \circ (\mu_R(b))$ on the output. Now it is easy to see that when $\pi_L, \pi_R \sim \mathcal{U}(\mathcal{S}_{n/2})$, the output is also uniform because the TBS mapping is independent of the relative permutations on the left and right halves.

More generally, we see that a uniform distribution over permutations $\mathcal{U}(\mathcal{S}_n)$ is mapped to two uniform permutations on the left and right half, respectively. Symbolically, for, $\pi \sim \mathcal{U}(\mathcal{S}_n)$, we have that

$$g(\pi) = (b, \pi_L, \pi_R) \sim \mathcal{U}(\mathbb{B}^n_{n/2} \times \mathcal{S}_{n/2} \times \mathcal{S}_{n/2}) = \mathcal{U}(\mathbb{B}^n_{n/2}) \times \mathcal{U}(\mathcal{S}_{n/2}) \times \mathcal{U}(\mathcal{S}_{n/2}). \tag{A.31}$$

As shown earlier, given uniform distributions over left and right permutations, the output is also uniform. By induction, all permutations in the recursive steps are uniform.

We therefore get a sum of expected TBS runtime on bit strings of lengths $n/3^r$, i.e.,

$$\sum_{r=1}^{\log_2 n} \mathbf{E}[T(b_r)] \leq \sum_{r=1}^{\log_2 n} \mathbf{E}[T(a_r)] + \tilde{O}\left(\left(\frac{n}{2^{r-1}}\right)^{\alpha}\right) \leq \frac{2n}{3} + \tilde{O}(n^{\alpha}) \tag{A.32}$$

where, by Lemma A.4 and the uniformity of permutations in recursive calls, we need only consider $b_r \sim \mathcal{U}(\mathbb{B}^{n/2^r}_{n/2^{r-1}})$ and we bound the expected runtime using Lemma A.5 with $a_r \sim \mathcal{U}(\mathbb{B}^{n/2^{r-1}})$. $\square$

We end with a lemma about the order statistics of binomial random variables used in the proof of the main theorem.

Lemma A.6. *Given m i.i.d. samples from the binomial distribution $X_i \sim B(n,p)$ with $i \in [m]$, and $p \in [0,1]$, the maximum $Y = \max_i X_i$ satisfies*

$$\mathbf{E}[Y] < pn + O\left(\sqrt{n\log(mn)}\right). \tag{A.33}$$

Proof. We use Hoeffding's inequality for the Bernoulli random variable $X \sim \mathcal{B}(n,p)$, which states that

$$\mathrm{P}[X \geq (p+\epsilon)n] \leq \exp(-2n\epsilon^2) \quad \forall \varepsilon \geq 0. \tag{A.34}$$

Pick $\epsilon = \sqrt{\frac{c}{2n}\log(mn)}$, where $c > 0$ is a constant. For this choice, we have

$$\mathrm{P}[X_i \geq (p+\epsilon)n] \leq \left(\frac{1}{mn}\right)^c \tag{A.35}$$

for every $i = 1, \ldots, m$. Then the probability that $Y < (p+\epsilon)n$ is identical to the probability that $X_i < (p+\epsilon)n$ for every i, which for i.i.d X_i is given by

$$\mathrm{P}[Y < (p+\epsilon)n] = \mathrm{P}[X < (p+\epsilon)n]^m > \left(1 - \frac{1}{(mn)^c}\right)^m. \tag{A.36}$$

Using Bernoulli's inequality ($(1+x)^r \geq 1 + rx$ for $x \geq -1$), we can simplify the above bound to

$$\mathrm{P}[Y < (p+\epsilon)n]^m > 1 - m^{1-c}n^{-c}. \tag{A.37}$$

Finally, we bound the expected value of Y by an explicit weighted sum over its range:

$$\mathbf{E}[Y] = \sum_{k=0}^{n} \mathrm{P}[Y = k] \cdot k \tag{A.38}$$

$$= \sum_{k=0}^{\lfloor (p+\epsilon)n \rfloor} \mathrm{P}[Y = k] \cdot k + \sum_{k=\lfloor (p+\epsilon)n \rfloor + 1}^{n} \mathrm{P}[Y = k] \cdot k \tag{A.39}$$

$$\leq \sum_{k=0}^{\lfloor (p+\epsilon)n \rfloor} \mathrm{P}[Y = k]) \cdot k + n \cdot \sum_{k=\lfloor (p+\epsilon)n \rfloor + 1}^{n} \mathrm{P}[Y = k] \tag{A.40}$$

$$\leq \sum_{k=0}^{\lfloor (p+\epsilon)n \rfloor} \mathrm{P}[Y = k] \cdot k + (mn)^{1-c} \tag{A.41}$$

$$\leq (p + \epsilon)n + (mn)^{1-c}. \tag{A.42}$$

Since $(mn)^{1-c} < 1$ for $c > 1$,

$$\mathbf{E}[Y] < \left\lceil pn + 1 + \sqrt{\frac{cn}{2} \log(mn)} \right\rceil = pn + O(\sqrt{n \log(mn)}) \tag{A.43}$$

as claimed. □

Appendix B

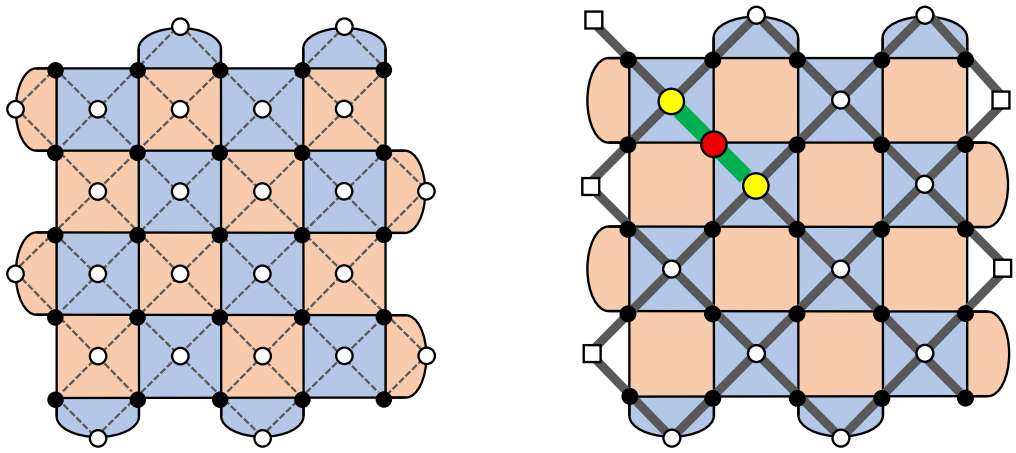
Surface code compilation

B.1 Surface code architecture

Here we review some basic details of the surface code focusing on the elementary logical operations shown in Figure 1.2. This is intended as a high-level overview to provide some intuition of how the logical operations in Figure 1.2 arise and what their resource costs are. For more thorough reviews of surface codes see Refs. [Bom13; Fow+12; Bro+17].

To implement the surface code, we assume *physical qubits* are laid out on the vertices of a 2D grid, with nearest-neighbor interactions allowed. A single surface code patch encodes a single *logical qubit* in $2d^2 - 1$ physical qubits, where the odd parameter d is known as the *code distance* which corresponds to the level of noise protection; see Figure B.1(a). For clarity, within this section of the appendix we refer to physical qubits and logical qubits explicitly, however in other sections we often drop the word "logical" when referring to logical qubits for brevity.

We designate every odd physical qubit as a *data physical qubit* in the patch, and every even physical qubit as an *ancilla physical qubit* to facilitate a *stabilizer* measurement; see Figure B.1(a). The code space of a surface code consists of those

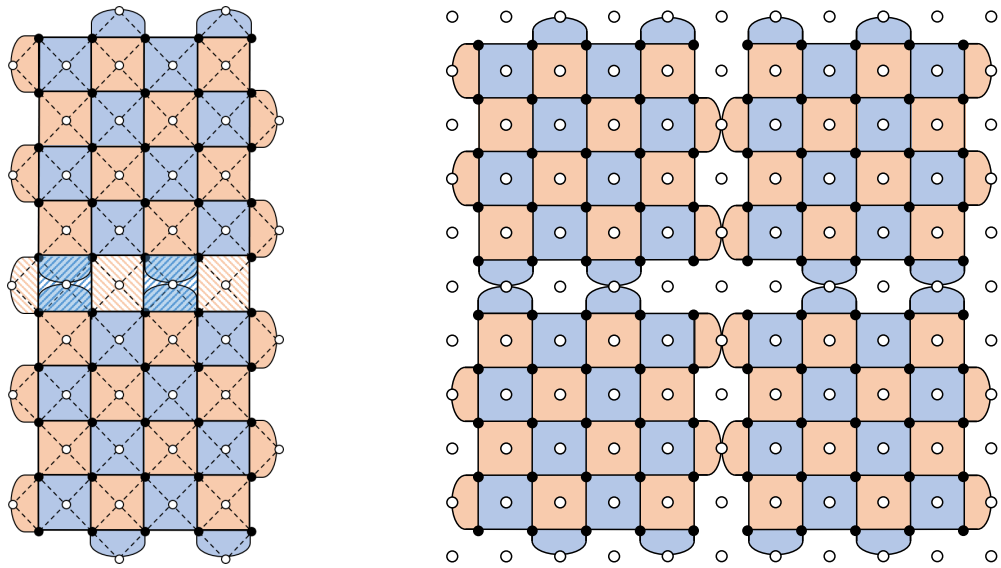


(a) Surface code patch **(b)** X-Stabilizer decoding graph

Figure B.1: (a) A $d = 5$ surface code patch implemented in a grid of data physical qubits (black disks), and ancilla physical qubits (white disks). Error correction is implemented with single-qubit operations and CNOT between pairs of qubits connected by a dashed edge. Z and X type stabilizers are associated with alternating red and blue faces. (b) A decoding graph that is defined by associating an edge with each qubit and a vertex for each stabilizer. If stabilizers are measured perfectly, Z errors on data qubits (marked in red) can be corrected by finding a minimum weight matching (green edges) of vertices associated with unsatisfied X stabilizers (yellow disks).

states of the data physical qubits which are simultaneous $+1$ eigenstates of the set of stabilizer generators. The stabilizer generators can be associated with faces and are either $X \otimes X \otimes X \otimes X$ or $Z \otimes Z \otimes Z \otimes Z$ operators for the bulk (interior) of the code or $X \otimes X$ or $Z \otimes Z$ operators on the boundary. We can see that the logical Z operator, Z_L, defined as any path of single-qubit Z operators on physical qubits connecting the rough boundaries, commutes with all stabilizers. Similarly, the logical X operator, X_L, is a path of X operators connecting the smooth boundaries.

For quantum error correction, it is necessary to repeatedly measure stabilizer generators. Stabilizer generators can be measured by running small circuits consisting of the preparation of the ancilla physical qubit, CNOTs between the ancilla physical qubit and the data physical qubits, followed by measurement of the ancilla physical qubit. Error correction can be performed by associating qubits with edges and stabilizer generators with vertices of a so-called decoding graph; see Figure B.1b. A classical



(a) Logical $Z_L \otimes Z_L$ measurement **(b)** Patches tiling the plane

Figure B.2: (a) A logical $Z_L \otimes Z_L$ measurement is performed by lattice surgery in the following steps: (i) Stop measuring the weight-two stabilizers along the horizontal boundary between the patches. (ii) Reliably measure the bulk faces for a single vertically-extended patch. Note that $Z_L \otimes Z_L$ can be inferred from the product of the outcomes of the newly measured red faces. This temporarily merges the patches to form a single extended surface code patch. (iii) Reliably measure once more the weight-two faces along the horizontal boundary between the patches. This separates the pair of patches. (b) Two types of patches tile the plane, with $Z_L \otimes Z_L$ measurements possible between vertically neighboring patches, and $X_L \otimes X_L$ measurements possible between horizontally neighboring patches.

algorithm known as a decoder is used to infer a set of edges (specifying the support of the X or Z correction) given a subset of vertices (corresponding to unsatisfied Z or X stabilizers, that is stabilizer generators with measurement outcome -1). Figure B.1b shows an example of this in the setting of perfect stabilizer measurements, although this can be generalized to handle faulty measurements by repeating measurements.

Logical operations can be implemented fault-tolerantly on logical qubits encoded in surface codes. For example, a destructive logical X measurement of a patch is implemented by measuring all data qubits in the X basis, and then using a decoder to process the physical outcomes and reliably identify the logical measurement outcome. Another important logical operation is the non-destructive measurement of a logical

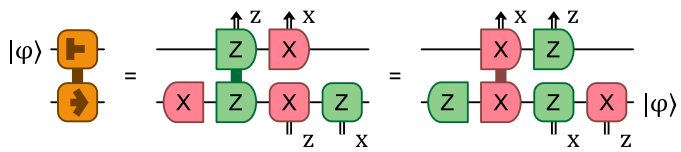
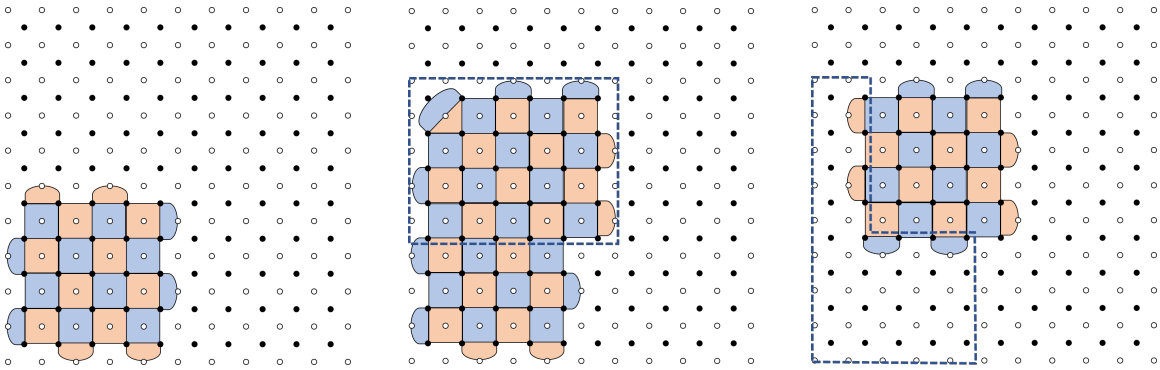


Figure B.3: The move operation can be implemented in depth 1 by local and neighboring Pauli measurements. A horizontal move can be implemented by preparing a single-qubit patch in $|0\rangle$, applying joint XX measurement, and then measuring the original patch in the Z basis (up to Pauli corrections). The vertical move follows from applying a Hadamard to the source qubit $|\phi\rangle$ and a Hadamard on the output. Simplifying the circuit gives the right-hand side in the Figure, with a ZZ measurement that is available vertically.

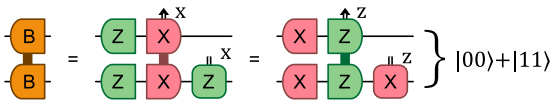
joint Pauli operator using an approach known as lattice surgery [Hor+12] as shown in Figure B.2a. To simplify lattice surgery by lining up the boundary stabilizers of neighboring patches, we consider a tiling of the plane using two versions of distance d surface code patches as shown in Figure B.2b which forms a grid of logical qubits. Logical $Z_L \otimes Z_L$ can be measured between vertical neighbor patches while $X_L \otimes X_L$ can be measured between horizontal neighbor patches.

The allowed fault-tolerant logical operations that we assume throughout Chapter 6 and the resources they require are listed in Figure 1.2. These are largely based on the rules specified in [Lit19]. Here we justify the resource requirements for the logical operations in Figure 1.2 not covered in [Lit19] on a distance-d surface code. For space analysis, we work in units of full surface code patches such that if any qubits from a patch are needed to implement an operation the full patch is counted. We show how to implement the operations in terms of more elementary Pauli measurements. The move operation can be implemented in depth 1 with the target qubit as ancilla, as shown in Figure B.3. The Hadamard can be implemented in depth three with three ancilla patched along with the move operation as shown in Figure B.4. Finally, Bell measurement and preparation can be implemented in depth 1 as shown in Figure B.5. There are alternatives to these implementations of logical operations which can lead to lower space-time cost. For example the Hadamard could be performed using just one

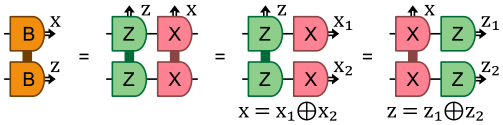


(a) Transversal Hadamard **(b)** Extended patch **(c)** Shrunk patch

Figure B.4: Implementation of a Hadamard operation in depth 3 with three ancilla patches. (a) A transverse Hadamard is applied in depth 0 to each physical data qubit, which switches the arrangement of X and Z stabilizer generators compared to the standard configuration. (b) The patch is extended in depth 1 so that a segment of the standard boundary type is introduced on the right. (c) The patch is shrunk into a standard surface code patch of the form of the top-left corner of the region (see Figure B.2b) in depth 1, but with its location shifted by a (code distance) d-independent amount. This allows us to shift the patch into the top-left corner in 0 depth (not shown). Then we move the logical qubit to the bottom-left corner in depth 1.



(a) Bell pair preparation



(b) Bell measurement

Figure B.5: We can implement Bell preparation and measurement in terms of single and two-qubit Pauli measurements in depth 1 as given in Figure B.5 [Lit19]. (a) A Bell pair can be prepare from a (horizontal) joint XX measurement of $|00\rangle$ or a (vertical) joint ZZ measurement of $|{+}{+}\rangle$, up to Pauli corrections. (b) A destructive Bell measurement can be implemented by a joint XX measurement followed by individual Z basis measurements, or by a joint ZZ measurement followed by individual X basis measurements.

logical ancilla patch if each patch was padded with extra qubits. We do not explore these alternatives here, but note that our EDPC algorithm can still be applied if these alternatives are used.

B.2 Logical space time cost as a proxy for physical space time cost

Here we provide a justification for our use of logical space time cost as a proxy for physical space time cost. As we have seen in Figure 1.2 and Appendix B.1, logical operations implemented with the surface code require physical time that scales as d and physical space that scales as d^2. For a logical circuit written in terms of a total of A_{logical} elementary logical operations implemented using surface codes of distance d, the physical space-time cost A_{physical} is approximately

$$A_{\text{physical}} \sim A_{\text{logical}} d^3. \tag{B.1}$$

The probability of any of these elementary operations resulting in a logical failure scales as $p_{\text{fail}} \sim (p/p^*)^{d/2}$, where the fixed system parameters are the physical error rate p, and the fault-tolerant threshold for the surface code p^*. Moreover, we assume $p_{\text{fail}} \sim 1/A_{\text{logical}}$ to ensure that the logical circuit is reliable with as small a code distance as possible. This suggests that the code distance behaves as

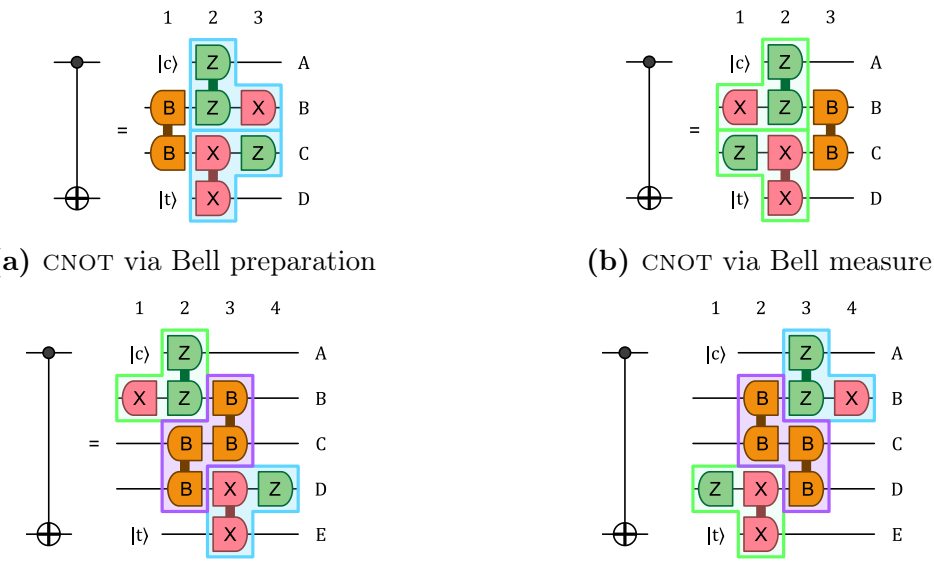
$$d \sim \frac{2 \log A_{\text{logical}}}{\log p^* - \log p}. \tag{B.2}$$

Therefore we see that the physical and logical space time costs are monotonically related, i.e.,

$$A_{\text{physical}} \sim A_{\text{logical}} (\log A_{\text{logical}})^3. \tag{B.3}$$

B.3 CNOT via Bell operations

We list more variations of the standard CNOT gate (Figure 6.2a) that use intermediate Bell preparation and measurements on ancillas in Figure B.6. By choosing the right subcircuit, we see that the long-range operations in Figure 6.7 implement a CNOT gate.



(a) CNOT via Bell preparation

(b) CNOT via Bell measure

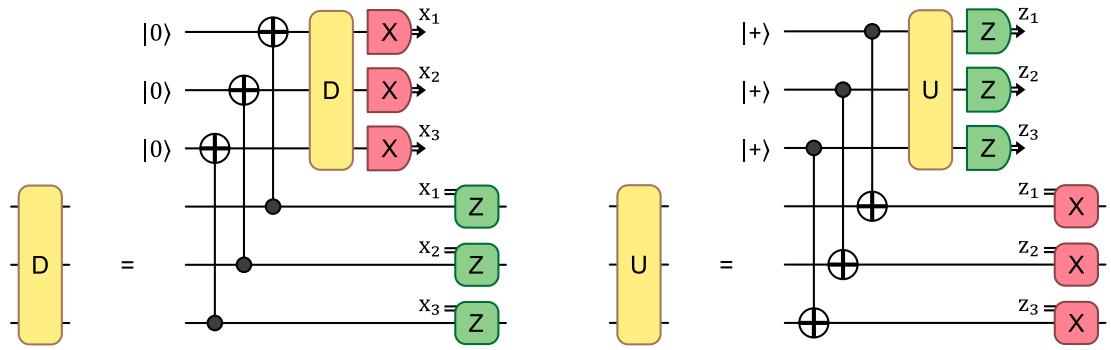
(c) CNOT with control first

(d) CNOT with target first

Figure B.6: Various implementations of a CNOT gate with intermediate ancilla qubits and Bell operations. In particular, we are able to apply the control and the target either before (green) or after (teal) Bell preparation and measurement steps, while keeping the depth at 2.

B.4 Remote execution of diagonal gates

A gate D diagonal on k source qubits in the computational basis can be executed on k ancilla by first entangling these ancilla qubits using CNOTs. We call this *remote* execution. Let the computational basis be $|\ell\rangle$, for $\ell \in [2^k]$, then $D|\ell\rangle = \exp(i\phi_\ell)|\ell\rangle$. We saw one use for remote gates in applying rotations at the boundary requiring magic states (Section 6.4).



(a) Diagonal gate in computational basis **(b)** Diagonal gate in Hadamard basis

Figure B.7: (a) Any k-qubit gate diagonal in the computational basis can be remotely executed on k dedicated ancilla by first using CNOTs. We use this technique to apply remote $Z(\theta)$ rotations (Figure 6.9b) with magic states at the boundary. (b) Similarly, gates diagonal in the Hadamard basis also have a remote implementation. Since the Pauli corrections can be commuted through Clifford circuits, Clifford circuits can be executed immediately after executing the CNOT operations with no need to wait on the remote operations.

We execute D remotely as follows (see Figure B.7). First, we initialize the ancilla in the state $|0\rangle^{\otimes k}$. Let the source qubits be in some pure state $\sum_\ell \alpha_\ell |\ell\rangle$, for $\alpha_\ell \in \mathbb{C}$. then we apply k transversal CNOT gates controlled on source qubits so that the overal state becomes $\sum_\ell \alpha_\ell |\ell\rangle \otimes |\ell\rangle$. We now apply D to the ancilla instead

$$(\mathbb{1} \otimes D) \sum_\ell \alpha_\ell |\ell\rangle \otimes |\ell\rangle = \sum_\ell \alpha_\ell \exp(i\phi_\ell) |\ell\rangle \otimes |\ell\rangle. \tag{B.4}$$

We now disentangle the ancilla by measuring them in the X basis. Let the measurement give outcomes $\boldsymbol{x} \in \{0,1\}^k$, then the state on the source qubits is mapped to

$$\sum_\ell \alpha_\ell \exp(i\phi_\ell)(-1)^{(\boldsymbol{x},\ell)} |\ell\rangle, \tag{B.5}$$

where $(\boldsymbol{x}, \ell)$ is the inner product modulo 2 between $\boldsymbol{x}$ and the binary representation of ℓ. Applying a Z correction to each qubit $j \in [k]$ controlled on measurement result $\boldsymbol{x}_j$ maps the state to $\sum_\ell \alpha_\ell \exp(i\phi_\ell)|\ell\rangle$ as required.

This technique can be extended to any unitary operator U since it can be unitarily diagonalized as $U = VDV^\dagger$ by the spectral theorem, for V unitary and D diagonal operators. A particularly simple case are unitary operators that are diagonal in the Hadamard basis, where $V = H^{\otimes k}$. We write $U = H^{\otimes k} D H^{\otimes k}$ on the source qubits and apply remote execution of D using our techniques above. We then simplify the circuit to obtain Figure B.7b.

Bibliography

[Abr+19] Héctor Abraham et al. *Qiskit: An Open-source Framework for Quantum Computing*. Comp. software. 2019. DOI: 10.5281/zenodo.2562110.

[AMV13] Karel Van Acoleyen, Michaël Mariën, and Frank Verstraete. "Entanglement Rates and Area Laws". In: *Physical Review Letters* 111.17 (Oct. 2013). DOI: 10.1103/physrevlett.111.170501.

[Adc+20] Jeremy C. Adcock, Sam Morley-Short, Axel Dahlberg, and Joshua W. Silverstone. "Mapping graph state orbits under local complementation". In: *Quantum* 4 (Aug. 2020), p. 305. DOI: 10.22331/q-2020-08-07-305.

[AKN98] Dorit Aharonov, Alexei Kitaev, and Noam Nisan. "Quantum circuits with mixed states". In: *Proceedings of the thirtieth annual ACM symposium on Theory of computing - STOC '98*. ACM Press, 1998. DOI: 10.1145/276698.276708.

[Ahl+00] Rudolf Ahlswede, Ning Cai, Shuo-Yen Robert Li, and Raymond W. Yeung. "Network information flow". In: *IEEE Transactions on Information Theory* 46.4 (July 2000), pp. 1204–1216. DOI: 10.1109/18.850663.

[AKS83] M. Ajtai, J. Komlós, and E. Szemerédi. "An $O(n \log n)$ sorting network". In: *Proceedings of the fifteenth annual ACM symposium on Theory of computing - STOC '83*. ACM Press, 1983. DOI: 10.1145/800061.808726.

[Alb+04] Claudio Albanese, Matthias Christandl, Nilanjana Datta, and Artur Ekert. "Mirror Inversion of Quantum States in Linear Registers". In: *Physical Review Letters* 93.23 (Nov. 2004). DOI: 10.1103/physrevlett.93.230502.

[ACG94] Noga Alon, F. R. K. Chung, and R. L. Graham. "Routing Permutations on Graphs via Matchings". In: *SIAM Journal on Discrete Mathematics* 7.3 (May 1994), pp. 513–530. ISSN: 0895-4801. DOI: 10.1137/s0895480192236628.

[ANI+21] MD SAJID ANIS et al. *Qiskit: An Open-source Framework for Quantum Computing*. 2021. DOI: 10.5281/zenodo.2573505.

[Aru+19] Frank Arute et al. "Quantum supremacy using a programmable superconducting processor". In: *Nature* 574.7779 (Oct. 2019), pp. 505–510. DOI: 10.1038/s41586-019-1666-5.

[Aud14] Koenraad M. R. Audenaert. "Quantum skew divergence". In: *Journal of Mathematical Physics* 55.11 (Nov. 2014), p. 112202. DOI: 10.1063/1.4901039.

[AR95] Yonatan Aumann and Yuval Rabani. "Improved Bounds for All Optical Routing". In: *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '95. 3600 University City Science Center, Philadelphia, PA, United States: Society for Industrial and Applied Mathematics, 1995, pp. 567–576. ISBN: 0898713498.

[BP93] V. Bafna and P. A. Pevzner. "Genome rearrangements and sorting by reversals". In: *Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science*. 1993, pp. 148–157. DOI: 10.1137/S0097539793250627.

[BR17] Indranil Banerjee and Dana Richards. "New Results on Routing via Matchings on Graphs". In: *Fundamentals of Computation Theory*. FCT: International Symposium on Fundamentals of Computation Theory (Bordeaux, France, Sept. 11–13, 2017). Ed. by Ralf Klasing and Marc Zeitoun. Lecture Notes in Computer Science 10472. Berlin, Germany: Springer, 2017, pp. 69–81. DOI: 10.1007/978-3-662-55751-8_7.

[BRS19] Indranil Banerjee, Dana Richards, and Igor Shinkar. "Sorting Networks on Restricted Topologies". In: *SOFSEM 2019: Theory and Practice of Computer Science*. SOFSEM: International Conference on Current Trends in Theory and Practice of Informatics (Nový Smokovec, Slovaki, Jan. 27–30, 2019). Lecture Notes in Computer Science 11376. Cham, Switzerland: Springer Nature Switzerland AG, 2019, pp. 54–66. ISBN: 978-3-030-10801-4. DOI: 10.1007/978-3-030-10801-4_6.

[Bap+] Aniruddha Bapat, Andrew Childs, Alexey Gorshkov, and Eddie Schoute. "Bounding the quantum routing advantage". In preparation.

[Bap+21a] Aniruddha Bapat, Andrew M. Childs, Alexey V. Gorshkov, Samuel King, Eddie Schoute, and Hrishee Shastri. "Quantum routing with fast reversals". In: *Quantum* 5 (Aug. 2021), p. 533. DOI: 10.22331/q-2021-08-31-533.

[Bap+18] Aniruddha Bapat, Zachary Eldredge, James R. Garrison, Abhinav Deshpande, Frederic T. Chong, and Alexey V. Gorshkov. "Unitary entanglement construction in hierarchical networks". In: *Physical Review A* 98.6 (Dec. 26, 2018). DOI: 10.1103/PhysRevA.98.062328.

[Bap+21b] Aniruddha Bapat, Eddie Schoute, Alexey V. Gorshkov, and Andrew M. Childs. "Nearly optimal time-independent reversal of a spin chain". In: *Physical Review Research* (2021). arXiv: 2003.02843v1 [quant-ph]. Forthcoming.

[Bar+09] L. Barrière, C. Dalfó, M. A. Fiol, and M. Mitjana. "The generalized hierarchical product of graphs". In: *Discrete Mathematics* 309.12 (June 2009), pp. 3871–3881. ISSN: 0012-365X. DOI: 10.1016/j.disc.2008.10.028.

[Bea+13] R. Beals, S. Brierley, O. Gray, A. W. Harrow, S. Kutin, N. Linden, D. Shepherd, and M. Stather. "Efficient distributed quantum computing". In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 469.2153 (Feb. 20, 2013). DOI: 10.1098/rspa.2012.0686.

[BH20] Niel de Beaudrap and Steven Herbert. "Quantum linear network coding for entanglement distribution in restricted architectures". In: *Quantum* 4 (Nov. 2020), p. 356. DOI: 10.22331/q-2020-11-01-356.

[Ben+08] Michael A. Bender, Dongdong Ge, Simai He, Haodong Hu, Ron Y. Pinter, Steven Skiena, and Firas Swidan. "Improved bounds on sorting by length-weighted reversals". In: *Journal of Computer and System Sciences* 74.5 (2008), pp. 744–774. ISSN: 0022-0000. DOI: 10.1016/j.jcss.2007.08.008.

[Ben+02] C. H. Bennett, J. I. Cirac, M. S. Leifer, D. W. Leung, N. Linden, S. Popescu, and G. Vidal. "Optimal simulation of two-qubit Hamiltonians using general local operations". In: *Physical Review A* 66.1 (July 2002). DOI: 10.1103/physreva.66.012305.

[Ben+03] C. H. Bennett, A. W. Harrow, D. W. Leung, and J. A. Smolin. "On the capacities of bipartite hamiltonians and unitary gates". In: *IEEE Transactions on Information Theory* 49.8 (July 2003), pp. 1895–1911. DOI: 10.1109/tit.2003.814935.

[BKS21] Michael Beverland, Vadym Kliuchnikov, and Eddie Schoute. *Surface code compilation via edge-disjoint paths*. Oct. 21, 2021. arXiv: 2110.11493 [quant-ph].

[Boi+18] Sergio Boixo, Sergei V. Isakov, Vadim N. Smelyanskiy, Ryan Babbush, Nan Ding, Zhang Jiang, Michael J. Bremner, John M. Martinis, and Hartmut Neven. "Characterizing quantum supremacy in near-term devices". In: *Nature Physics* 14.6 (Apr. 23, 2018), pp. 595–600. DOI: 10.1038/s41567-018-0124-x.

[Bom13] H. Bombin. "Topological Codes". In: *Quantum Error Correction*. Ed. by Daniel A. Lidar and Todd A. Brun. Cambridge: Cambridge University Press, Nov. 1, 2013. ISBN: 9780521897877. arXiv: 1311.0277 [quant-ph].

[Bom+21] Hector Bombin, Isaac H. Kim, Daniel Litinski, Naomi Nickerson, Mihir Pant, Fernando Pastawski, Sam Roberts, and Terry Rudolph. "Interleaving: Modular architectures for fault-tolerant photonic quantum computing". Mar. 15, 2021. arXiv: 2103.08612 [quant-ph].

[BMR17] Édouard Bonnet, Tillmann Miltzow, and Paweł Rzążewski. "Complexity of Token Swapping and its Variants". In: *Algorithmica* 80.9 (Oct. 2017), pp. 2656–2682. DOI: 10.1007/s00453-017-0387-0.

[Bos03] Sougato Bose. "Quantum Communication through an Unmodulated Spin Chain". In: *Physical Review Letters* 91.20 (Nov. 2003). DOI: 10.1103/physrevlett.91.207901.

[Bos07] Sougato Bose. "Quantum communication through spin chain dynamics: an introductory overview". In: *Contemporary Physics* 48.1 (Jan. 2007), pp. 13–30. DOI: 10.1080/00107510701342313.

[Bou+18] Adam Bouland, Bill Fefferman, Chinmay Nirkhe, and Umesh Vazirani. "On the complexity and verification of quantum random circuit sampling". In: *Nature Physics* (Oct. 29, 2018). DOI: 10.1038/s41567-018-0318-2.

[BK98] S. B. Bravyi and A. Yu. Kitaev. *Quantum codes on a lattice with boundary*. Nov. 20, 1998. arXiv: quant-ph/9811052v1 [quant-ph]. Pre-published.

[Bra07] Sergey Bravyi. "Upper bounds on entangling rates of bipartite Hamiltonians". In: *Physical Review A* 76.5 (Nov. 2007). DOI: 10.1103/physreva.76.052319.

[BK05] Sergey Bravyi and Alexei Kitaev. "Universal quantum computation with ideal Clifford gates and noisy ancillas". In: *Physical Review A* 71.2 (Feb. 2005). DOI: 10.1103/physreva.71.022316.

[Bre+16] Teresa Brecht, Wolfgang Pfaff, Chen Wang, Yiwen Chu, Luigi Frunzio, Michel H. Devoret, and Robert J. Schoelkopf. "Multilayer microwave integrated quantum circuits for scalable quantum computing". In: *npj Quantum Information* 2.16002 (Feb. 23, 2016). DOI: 10.1038/npjqi.2016.2.

[Bri+09] H. J. Briegel, D. E. Browne, W. Dür, R. Raussendorf, and M. Van den Nest. "Measurement-based quantum computation". In: *Nature Physics* 5.1 (Jan. 2009), pp. 19–26. DOI: 10.1038/nphys1157.

[Bri+98] H.-J. Briegel, W. Dür, J. I. Cirac, and P. Zoller. "Quantum Repeaters: The Role of Imperfect Local Operations in Quantum Communication". In: *Phys. Rev. Lett.* 81 (26 Dec. 1998), pp. 5932–5935. DOI: 10.1103/PhysRevLett.81.5932.

[BFU94] Andrei Z. Broder, Alan M. Frieze, and Eli Upfal. "Existence and Construction of Edge-Disjoint Paths on Expander Graphs". In: *SIAM Journal on Computing* 23.5 (Oct. 1994), pp. 976–989. DOI: 10.1137/s0097539792232021.

[Bro+17] Benjamin J. Brown, Katharina Laubscher, Markus S. Kesselring, and James R. Wootton. "Poking Holes and Cutting Corners to Achieve Clifford Gates with the Surface Code". In: *Physical Review X* 7.2 (May 2017), p. 021029. DOI: 10.1103/physrevx.7.021029.

[CC21] Christopher Chamberland and Earl T. Campbell. "Universal quantum computing with twist-free and temporally encoded lattice surgery". 2021. arXiv: 2109.02746 [quant-ph].

[Cha+20a] Christopher Chamberland, Guanyu Zhu, Theodore J. Yoder, Jared B. Hertzberg, and Andrew W. Cross. "Topological and Subsystem Codes on Low-Degree Graphs with Flag Qubits". In: *Physical Review X* 10 (2020). DOI: 10.1103/PhysRevX.10.011022.

[Cha+20b] Rui Chao, Michael E. Beverland, Nicolas Delfosse, and Jeongwan Haah. "Optimization of the surface code design for Majorana-based qubits". In: *Quantum* 4 (Oct. 2020), p. 352. ISSN: 2521-327X. DOI: 10.22331/q-2020-10-28-352.

[Che71] Jeff Cheeger. "A Lower Bound for the Smallest Eigenvalue of the Laplacian". In: *Problems in Analysis: A Symposium in Honor of Salomon Bochner (PMS-31)*. Ed. by Robert C. Gunning. Princeton University Press, 1971. DOI: 10.1515/9781400869312.

[CKS06] Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd. "An $O(\sqrt{n})$ Approximation and Integrality Gap for Disjoint Paths and Unsplittable Flow". In: *Theory of Computing* 2.1 (2006), pp. 137–146. DOI: 10.4086/toc.2006.v002a007.

[Che20] Yu-An Chen. "Exact bosonization in arbitrary dimensions". In: *Physical Review Research* 2.3 (Sept. 2020), p. 033527. DOI: 10.1103/physrevresearch.2.033527.

[Chi+03] A. M. Childs, D. W. Leung, F. Verstraete, and G. Vidal. "Asymptotic entanglement capacity of the Ising and anisotropic Heisenberg interactions". In: *Quantum Information and Computation* 3.2 (Mar. 2003), pp. 97–105. DOI: 10.26421/qic3.2.

[CLV04] A. M. Childs, D. W. Leung, and G. Vidal. "Reversible Simulation of Bipartite Product Hamiltonians". In: *IEEE Transactions on Information Theory* 50.6 (June 2004), pp. 1189–1197. DOI: 10.1109/tit.2004.828069.

[Chi+18] Andrew M. Childs, Dmitri Maslov, Yunseong Nam, Neil J. Ross, and Yuan Su. "Toward the first quantum simulation with quantum speedup". In: *Proceedings of the National Academy of Sciences* 115.38 (Sept. 18, 2018), pp. 9456–9461. DOI: 10.1073/pnas.1801723115.

[CSU19] Andrew M. Childs, Eddie Schoute, and Cem M. Unsal. "Circuit Transformations for Quantum Architectures". In: *14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019)*. Ed. by Wim van Dam and Laura Mančinska. Vol. 135. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 3:1–3:24. ISBN: 978-3-95977-112-2. DOI: 10.4230/LIPIcs.TQC.2019.3.

[CM11] Byung-Soo Choi and Rodney van Meter. "On the Effect of Quantum Interaction Distance on Quantum Addition Circuits". In: *ACM Journal on Emerging Technologies in Computing Systems* 7.3 (Aug. 2011), pp. 1–17. DOI: 10.1145/2000502.2000504.

[CM12] Byung-Soo Choi and Rodney van Meter. "A $\Theta(\sqrt{N})$-depth Quantum Adder on the 2D NTC Quantum Computer Architecture". In: *J. Emerg. Technol. Comput. Syst.* 8.3 (Aug. 3, 2012), 24:1–24:22. ISSN: 1550-4832. DOI: 10.1145/2287696.2287707.

[Chr+05] Matthias Christandl, Nilanjana Datta, Tony C. Dorlas, Artur Ekert, Alastair Kay, and Andrew J. Landahl. "Perfect transfer of arbitrary states in quantum spin networks". In: *Physical Review A* 71.3 (Mar. 2005). DOI: 10.1103/physreva.71.032312.

[Chr+04] Matthias Christandl, Nilanjana Datta, Artur Ekert, and Andrew J. Landahl. "Perfect State Transfer in Quantum Spin Networks". In: *Physical Review Letters* 92.18 (May 2004). DOI: 10.1103/physrevlett.92.187902.

[Chu] Fan Chung. *Spectral Graph Theory: revised and improved*. URL: http://www.math.ucsd.edu/~fan/research/revised.html (visited on 10/29/2020). Repr. of *Spectral Graph Theory*. American Mathematical Society, Dec. 1996. ISBN: 0821803158. DOI: 10.1090/cbms/092.

[CK15] Julia Chuzhoy and David H. K. Kim. "On Approximating Node-Disjoint Paths in Grids". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015)*. Ed. by Naveen Garg, Klaus Jansen, Anup Rao, and José D. P. Rolim. Vol. 40. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2015, pp. 187–211. ISBN: 978-3-939897-89-7. DOI: 10.4230/LIPIcs.APPROX-RANDOM.2015.187.

[CKN18] Julia Chuzhoy, David H. K. Kim, and Rachit Nimavat. "Almost polynomial hardness of node-disjoint paths in grids". In: *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing - STOC 2018*. ACM Press, 2018. DOI: 10.1145/3188745.3188772.

[CBC21] Laura Clinton, Johannes Bausch, and Toby Cubitt. "Hamiltonian simulation algorithms for near-term quantum hardware". In: *Nature Communications* 12.1 (Aug. 2021). DOI: 10.1038/s41467-021-25196-0.

[Cow+20] Alexander Cowtan, Silas Dilkes, Ross Duncan, Will Simmons, and Seyon Sivarajah. "Phase Gadget Synthesis for Shallow Circuits". In: *Electronic Proceedings in Theoretical Computer Science*. Electronic Proceedings for Theoretical Computer Science 318 (May 2020), pp. 213–228. DOI: 10.4204/eptcs.318.13.

[Cro+19] Andrew W. Cross, Lev S. Bishop, Sarah Sheldon, Paul D. Nation, and Jay M. Gambetta. "Validating quantum computers using randomized model circuits". In: *Physical Review A* 100.3 (Sept. 2019). DOI: 10.1103/physreva.100.032328.

[DBT21] Nicolas Delfosse, Michael E. Beverland, and Maxime A. Tremblay. *Bounds on stabilizer measurement circuits and obstructions to local implementations of quantum LDPC codes*. 2021. eprint: arXiv:2109.14599.

[DW18] Trithep Devakul and Dominic J. Williamson. "Universal quantum computation using fractal symmetry-protected cluster phases". In: *Physical Review A* 98 (2 Aug. 2018), p. 022332. DOI: 10.1103/PhysRevA.98.022332.

[DiV00] David P. DiVincenzo. "The Physical Implementation of Quantum Computation". In: *Fortschritte der Physik* 48.9-11 (Sept. 2000), pp. 771–783. DOI: 10.1002/1521-3978(200009)48:9/11<771::AID-PROP771>3.0.CO;2-E.

[Dun+20] Ross Duncan, Aleks Kissinger, Simon Perdrix, and John van de Wetering. "Graph-theoretic Simplification of Quantum Circuits with the ZX-calculus". In: *Quantum* 4 (June 2020), p. 279. DOI: 10.22331/q-2020-06-04-279.

[Dür+01] W. Dür, G. Vidal, J. I. Cirac, N. Linden, and S. Popescu. "Entanglement Capabilities of Nonlocal Hamiltonians". In: *Physical Review Letters* 87.13 (Sept. 2001). DOI: 10.1103/physrevlett.87.137901.

[ECP10] J. Eisert, M. Cramer, and M. B. Plenio. "Colloquium: Area laws for the entanglement entropy". In: *Reviews of Modern Physics* 82.1 (Feb. 2010), pp. 277–306. DOI: 10.1103/revmodphys.82.277.

[Eld+20] Zachary Eldredge, Leo Zhou, Aniruddha Bapat, James R. Garrison, Abhinav Deshpande, Frederic T. Chong, and Alexey V. Gorshkov. "Entanglement bounds on the performance of quantum computing architectures". In: *Physical Review Research* 2.3 (Aug. 2020), p. 033316. DOI: 10.1103/physrevresearch.2.033316.

[Fey82] Richard P. Feynman. "Simulating physics with computers". In: *International Journal of Theoretical Physics* 21.6-7 (June 1982), pp. 467–488. DOI: 10.1007/bf02650179.

[FT06] Joseph Fitzsimons and Jason Twamley. "Globally Controlled Quantum Wires for Perfect Qubit Transport, Mirroring, and Computing". In: *Physical Review Letters* 97.9 (Sept. 2006). DOI: 10.1103/physrevlett.97.090502.

[Flo62] Robert W. Floyd. "Algorithm 97: Shortest path". In: *Communications of the ACM* 5.6 (June 1962), p. 345. DOI: 10.1145/367766.368168.

[FF56] L. R. Ford and D. R. Fulkerson. "Maximal Flow Through a Network". In: *Canadian Journal of Mathematics* 8 (1956), pp. 399–404. DOI: 10.4153/cjm-1956-045-5.

[Fos+15] Michael Foss-Feig, Zhe-Xuan Gong, Charles W. Clark, and Alexey V. Gorshkov. "Nearly Linear Light Cones in Long-Range Interacting Quantum Systems". In: *Physical Review Letters* 114.15 (Apr. 2015), p. 157201. DOI: 10.1103/physrevlett.114.157201.

[Fow+12] Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. "Surface codes: Towards practical large-scale quantum computation". In: *Physical Review A* 86.3 (2012), p. 032324. DOI: 10.1103/PhysRevA.86.032324.

[FPK08] C. Di Franco, M. Paternostro, and M. S. Kim. "Perfect State Transfer on a Spin Chain without State Initialization". In: *Physical Review Letters* 101.23 (Dec. 2008). DOI: 10.1103/physrevlett.101.230502.

[FC16] Amir Fruchtman and Iris Choi. *Technical Roadmap for Fault-Toleran Quantum Computing*. Research rep. University of Oxford, Sept. 2016. URL: https://nqit.ox.ac.uk/content/technical-roadmap-fault-tolerant-quantum-computing.html (visited on 09/15/2021).

[GT89] Harold N. Gabow and Robert E. Tarjan. "Faster Scaling Algorithms for Network Problems". In: *SIAM Journal on Computing* 18.5 (Oct. 1989), pp. 1013–1036. DOI: 10.1137/0218069.

[GE21] Craig Gidney and Martin Ekerå. "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits". In: *Quantum* 5 (Apr. 2021), p. 433. ISSN: 2521-327X. DOI: 10.22331/q-2021-04-15-433.

[GF19] Craig Gidney and Austin G. Fowler. *Flexible layout of surface code computations using AutoCCZ states*. May 22, 2019. arXiv: 1905.08916 [quant-ph].

[Gon+17] Zhe-Xuan Gong, Michael Foss-Feig, Fernando G. S. L. Brandão, and Alexey V. Gorshkov. "Entanglement Area Laws for Long-Range Interacting Systems". In: *Physical Review Letters* 119.5 (July 2017). DOI: 10.1103/physrevlett.119.050501.

[Gre+13] Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. "Quipper: A Scalable Quantum Programming Language". In: *ACM SIGPLAN Notices* 48.6 (June 2013), pp. 333–342. ISSN: 0362-1340. DOI: 10.1145/2499370.2462177.

[Grz+21] Nikodem Grzesiak, Andrii Maksymov, Pradeep Niroula, and Yunseong Nam. *Efficient quantum programming using EASE gates on a trapped-ion quantum computer*. July 15, 2021. arXiv: 2107.07591 [quant-ph].

[Guo+19] Andrew Y. Guo, Minh C. Tran, Andrew M. Childs, Alexey V. Gorshkov, and Zhe-Xuan Gong. "Signaling and scrambling with strongly long-range interactions". In: *Physical Review A* 102.1 (July 8, 2019). DOI: 10.1103/physreva.102.010401.

[Haa+18] Jeongwan Haah, Matthew Hastings, Robin Kothari, and Guang Hao Low. "Quantum Algorithm for Simulating Real Time Evolution of Lattice Hamiltonians". In: *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, Oct. 2018, pp. 350–360. DOI: 10.1109/FOCS.2018.00041.

[HPE19] F. Hahn, A. Pappa, and J. Eisert. "Quantum network routing and local complementation". In: *npj Quantum Information* 5.1 (Sept. 2019). DOI: 10.1038/s41534-019-0191-6.

[Hir+09] Yuichi Hirata, Masaki Nakanishi, Shigeru Yamashita, and Yasuhiko Nakashima. "An Efficient Method to Convert Arbitrary Quantum Circuits to Ones on a Linear Nearest Neighbor Architecture". In: *2009 Third International Conference on Quantum, Nano and Micro Technologies*. IEEE, Feb. 2009. DOI: 10.1109/icqnm.2009.25.

[HKP19] Christopher Hoffman, Matthew Kahle, and Elliot Paquette. "Spectral Gaps of Random Graphs and Applications". In: *International Mathematics Research Notices* (May 2019). DOI: 10.1093/imrn/rnz077.

[HK73] John E. Hopcroft and Richard M. Karp. "An $n^{(5/2)}$ Algorithm for Maximum Matchings in Bipartite Graphs". In: *SIAM Journal on Computing* 2.4 (Dec. 1973), pp. 225–231. DOI: 10.1137/0202019.

[Hor+12] Clare Horsman, Austin G. Fowler, Simon Devitt, and Rodney Van Meter. "Surface code quantum computing by lattice surgery". In: *New Journal of Physics* 14.12 (Dec. 2012), p. 123011. DOI: 10.1088/1367-2630/14/12/123011.

[IBM18] IBM Q Team. *IBM Q Experience Devices*. 2018. URL: https://quantumexperience.ng.bluemix.net/qx/devices (visited on 10/09/2018).

[IBM21] *IBM Quantum*. 2021. URL: https://quantum-computing.ibm.com/ (visited on 09/12/2021).

[Ito+20] Toshinari Itoko, Rudy Raymond, Takashi Imamichi, and Atsushi Matsuo. "Optimization of quantum circuit mapping using gate transformation and commutation". In: *Integration* 70 (Jan. 2020), pp. 43–50. DOI: 10.1016/j.vlsi.2019.10.004.

[Ito+19] Toshinari Itoko, Rudy Raymond, Takashi Imamichi, Atsushi Matsuo, and Andrew W. Cross. "Quantum circuit compilers using gate commutation rules". In: *ASP-DAC '19: Proceedings of the 24th Asia and South Pacific Design Automation Conference*. 24th Asia and South Pacific Design Automation Conference (Tokyo, Japan, Jan. 21, 2019–Jan. 24, 2014). Ed. by Toshiyuki Shibuya. New York NY, United States: ACM Press, Jan. 2019, pp. 191–196. ISBN: 978-1-4503-6007-4. DOI: 10.1145/3287624.3287701.

[JWB03] Dominik Janzing, Pawel Wocjan, and Thomas Beth. *Identity check is QMA-complete*. May 9, 2003. arXiv: quant-ph/0305050v1.

[Jav+12] Ali Javadi-Abhari, Arvin Faruque, Mohammad J Dousti, Lukas Svec, Oana Catu, Amlan Chakrabati, Chen-Fu Chiang, Seth Vanderwilt, John Black, and Fred Chong. *Scaffold: Quantum programming language*. Tech. rep. TR-934-12. Princeton University, Department of Computer Science, 2012.

[Jav+17] Ali Javadi-Abhari, Pranav Gokhale, Adam Holmes, Diana Franklin, Kenneth R. Brown, Margaret Martonosi, and Frederic T. Chong. "Optimized surface code communication in superconducting quantum computers". In: *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, Oct. 2017. DOI: 10.1145/3123939.3123949.

[Joh74] David S. Johnson. "Approximation algorithms for combinatorial problems". In: *Journal of Computer and System Sciences* 9.3 (Dec. 1974), pp. 256–278. DOI: 10.1016/s0022-0000(74)80044-9.

[Jon13] Cody Jones. "Low-overhead constructions for the fault-tolerant Toffoli gate". In: *Physical Review A* 87.2 (Feb. 2013), p. 022328. DOI: 10.1103/physreva.87.022328.

[Jur+21] Petar Jurcevic et al. "Demonstration of quantum volume 64 on a superconducting quantum computing system". In: *Quantum Science and Technology* 6.2 (Mar. 2021), p. 025020. DOI: 10.1088/2058-9565/abe519.

[KS05] Peter Karbach and Joachim Stolze. "Spin chains as perfect quantum state mirrors". In: *Physical Review A* 72.3 (Sept. 2005). DOI: 10.1103/physreva.72.030301.

[Kar+17] Torsten Karzig, Christina Knapp, Roman M. Lutchyn, Parsa Bonderson, Matthew B. Hastings, Chetan Nayak, Jason Alicea, Karsten Flensberg, Stephan Plugge, Yuval Oreg, Charles M. Marcus, and Michael H. Freedman. "Scalable designs for quasiparticle-poisoning-protected topological quantum computation with Majorana zero modes". In: *Phys. Rev. B* 95 (23 June 2017), p. 235305. DOI: 10.1103/PhysRevB.95.235305.

[KS95] J. Kececioglu and D. Sankoff. "Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement". In: *Algorithmica* 13.1-2 (Feb. 1, 1995), pp. 180–210. ISSN: 0178-4617. DOI: 10.1007/BF01188586.

[KMW02] D. Kielpinski, C. Monroe, and D. J. Wineland. "Architecture for a large-scale ion-trap quantum computer". In: *Nature* 417.6890 (June 2002), pp. 709–711. DOI: 10.1038/nature00784.

[Kim08] H. J. Kimble. "The quantum internet". In: *Nature* 453.7198 (June 2008), pp. 1023–1030. DOI: 10.1038/nature07127.

[KSS21] Samuel King, Eddie Schoute, and Hrishee Shastri. *reversal-sort*. 2021. URL: https://gitlab.umiacs.umd.edu/amchilds/reversal-sort.

[KG20] Aleks Kissinger and Arianne Meijer-van de Griend. "CNOT circuit extraction for topologically-constrained quantum memories". In: *Quantum Information & Computation* 20.7&8 (June 1, 2020), pp. 581–596. arXiv: 1904.00633v2 [quant-ph].

[KW20] Aleks Kissinger and John van de Wetering. "PyZX: Large Scale Automated Diagrammatic Reasoning". In: *Electronic Proceedings in Theoretical Computer Science* 318 (May 2020), pp. 229–241. DOI: 10.4204/eptcs.318.14.

[Kit97] A Yu Kitaev. "Quantum computations: algorithms and error correction". In: *Russian Mathematical Surveys* 52.6 (Dec. 1997), pp. 1191–1249. DOI: 10.1070/rm1997v052n06abeh002155.

[Kit03] A.Yu. Kitaev. "Fault-tolerant quantum computation by anyons". In: *Annals of Physics* 303.1 (Jan. 2003), pp. 2–30. DOI: 10.1016/s0003-4916(02)00018-0.

[Kit06] Alexei Kitaev. "Anyons in an exactly solved model and beyond". In: *Annals of Physics* 321.1 (2006), pp. 2–111.

[Kja+20] Morten Kjaergaard, Mollie E. Schwartz, Jochen Braumüller, Philip Krantz, Joel I-Jan Wang, Simon Gustavsson, and William D. Oliver. "Superconducting Qubits: Current State of Play". In: *Annual Review of Condensed Matter Physics* 11.1 (Mar. 2020), pp. 369–395. DOI: 10.1146/annurev-conmatphys-031119-050605.

[KT95] J. Kleinberg and E. Tardos. "Disjoint paths in densely embedded graphs". In: *Proceedings of IEEE 36th Annual Foundations of Computer Science*. IEEE Comput. Soc. Press, 1995. DOI: 10.1109/sfcs.1995.492462.

[KT06a] Jon Kleinberg and Éva Tardos. In: *Algorithm Design*. 1st ed. Boston: Pearson/Addison-Wesley, 2006. Chap. 11. ISBN: 0321295358.

[KT06b] Jon Kleinberg and Éva Tardos. In: *Algorithm Design*. 1st ed. Boston: Pearson/Addison-Wesley, 2006. Chap. 7. ISBN: 0321295358.

[Kle96] Jon M. Kleinberg. "Approximation algorithms for disjoint paths problems". PhD thesis. Massachusetts Institute of Technology. Dept. of Electrical Engineering and Computer Science, 1996. URL: http://hdl.handle.net/1721.1/11013.

[Kløø8] Torleiv Kløve. *Spheres of Permutations under the Infinity Norm–Permutations with limited displacement*. Research rep. 376. Department of Informatics, University of Bergen, Norway, 2008. URL: http://www.ii.uib.no/publikasjoner/texrap/pdf/2008-376.pdf.

[Kni04] E. Knill. "Fault-Tolerant Postselected Quantum Computation: Schemes". Feb. 2004. arXiv: quant-ph/0402171 [quant-ph].

[Kni05] E. Knill. "Quantum computing with realistically noisy devices". In: *Nature* 434.7029 (Mar. 2005), pp. 39–44. DOI: 10.1038/nature03350.

[Knu98] Donald E. Knuth. *The Art of Computer Programming: Sorting and Searching*. 2nd ed. Vol. 3. Addison-Wesley, 1998. 800 pp. ISBN: 0201896850.

[Kob+09] Hirotada Kobayashi, François Le Gall, Harumichi Nishimura, and Martin Rötteler. "General Scheme for Perfect Quantum Network Coding with Free Classical Communication". In: *Automata, Languages and Programming*. Springer Berlin Heidelberg, 2009, pp. 622–633. DOI: 10.1007/978-3-642-02927-1_52.

[Kob+11] Hirotada Kobayashi, François Le Gall, Harumichi Nishimura, and Martin Rötteler. "Constructing quantum network coding schemes from classical nonlinear protocols". In: *2011 IEEE International Symposium on Information Theory Proceedings*. IEEE, July 2011. DOI: 10.1109/isit.2011.6033701.

[KS04] Stavros G. Kolliopoulos and Clifford Stein. "Approximating disjoint-path problems using packing integer programs". In: *Mathematical Programming* 99.1 (Jan. 2004), pp. 63–87. DOI: 10.1007/s10107-002-0370-6.

[Kuh55] H. W. Kuhn. "The Hungarian method for the assignment problem". In: *Naval Research Logistics Quarterly* 2.1-2 (Mar. 1955), pp. 83–97. DOI: 10.1002/nav.3800020109.

[KD15] P. Kumar and S. Daraeizadeh. "Parity-based mirror inversion for efficient quantum state transfer and computation in nearest-neighbor arrays". In: *Physical Review A* 91.4 (Apr. 2015). DOI: 10.1103/physreva.91.042310.

[Kun87] Manfred Kunde. "Optimal sorting on multi-dimensionally mesh-connected computers". In: *STACS 87*. STACS 1987. Ed. by Franz J. Brandenburg, Guy Vidal-Naquet, and Martin Wirsing. Springer Berlin Heidelberg, 1987, pp. 408–419. DOI: 10.1007/bfb0039623.

[Kut06] Samuel A. Kutin. *Shor's algorithm on a nearest-neighbor machine*. Aug. 31, 2006. arXiv: quant-ph/0609001v1.

[KMS07] Samuel A. Kutin, David Petrie Moulton, and Lawren M. Smithline. "Computation at a distance". In: *Chicago Journal of Theoretical Computer Science* 13.1 (Jan. 26, 2007), pp. 1–17. DOI: 10.4086/cjtcs.2007.001.

[Lao+18] L. Lao, B. van Wee, I. Ashraf, J. van Someren, N. Khammassi, K. Bertels, and C. G. Almudever. "Mapping of lattice surgery-based quantum circuits on surface code architectures". In: *Quantum Science and Technology* 4.1 (Sept. 12, 2018), p. 015005. DOI: 10.1088/2058-9565/aadd1a.

[LB21] Lingling Lao and Dan Browne. *2QAN: A quantum compiler for 2-local qubit Hamiltonian simulation algorithms*. Aug. 4, 2021. arXiv: 2108.02099 [quant-ph].

[Lao+21] Lingling Lao, Prakash Murali, Margaret Martonosi, and Dan Browne. "Designing Calibration and Expressivity-Efficient Instruction Sets for Quantum Computing". In: *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, June 2021. DOI: 10.1109/isca52012.2021.00071.

[LL04] April Rasala Lehman and Eric Lehman. "Complexity Classification of Network Information Flow Problems". In: *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '04. New Orleans, Louisiana: Society for Industrial and Applied Mathematics, 2004, pp. 142–150. ISBN: 089871558X.

[LOW10] Debbie Leung, Jonathan Oppenheim, and Andreas Winter. "Quantum Network Communication—The Butterfly and Beyond". In: *IEEE Transactions on Information Theory* 56.7 (July 2010), pp. 3478–3490. DOI: 10.1109/tit.2010.2048442.

[LDX19] Gushu Li, Yufei Ding, and Yuan Xie. "Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices". In: *ASPLOS '19: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM International Conference on Architectural Support for Programming Languages and Operating Systems. Ed. by Iris Bahar, Maurice Herlihy, Emmett Witchel, and Alvin R. Lebeck. New York NY, United States: The Association for Computing Machinery, 2019, pp. 1001–1014. ISBN: 978-1-4503-6240-5. DOI: 10.1145/3297858.3304023.

[LR72] Elliott H. Lieb and Derek W. Robinson. "The finite group velocity of quantum spin systems". In: *Communications in Mathematical Physics* 28 (1972), pp. 251–257. DOI: 10.1007/bf01645779.

[LSJ15] Chia-Chun Lin, Susmita Sur-Kolay, and Niraj K. Jha. "PAQCS: Physical Design-Aware Fault-Tolerant Quantum Circuit Synthesis". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 23.7 (July 2015), pp. 1221–1234. DOI: 10.1109/tvlsi.2014.2337302.

[Lin+17] Norbert M. Linke, Dmitri Maslov, Martin Roetteler, Shantanu Debnath, Caroline Figgatt, Kevin A. Landsman, Kenneth Wright, and Christopher Monroe. "Experimental comparison of two quantum computing architectures". In: *Proceedings of the National Academy of Sciences* 114.13 (Mar. 28, 2017), pp. 3305–3310. DOI: 10.1073/pnas.1618020114.

[Lit19] Daniel Litinski. "A Game of Surface Codes: Large-Scale Quantum Computing with Lattice Surgery". In: *Quantum* 3 (Mar. 2019), p. 128. DOI: 10.22331/q-2019-03-05-128.

[LO17] Daniel Litinski and Felix von Oppen. "Braiding by Majorana tracking and long-range CNOT gates with color codes". In: *Physical Review B* 96.20 (Nov. 2017). DOI: 10.1103/physrevb.96.205413.

[LC17] Guang Hao Low and Isaac L. Chuang. "Optimal Hamiltonian Simulation by Quantum Signal Processing". In: *Physical Review Letters* 118.1 (Jan. 2017). DOI: 10.1103/physrevlett.118.010501.

[LWD15] Aaron Lye, Robert Wille, and Rolf Drechsler. "Determining the minimal number of swap gates for multi-dimensional nearest neighbor quantum circuits". In: *The 20th Asia and South Pacific Design Automation Conference*. ASP-DAC '15: 20th Asia and South Pacific Design Automation Conference (Chiba/Tokyo, Japan, Jan. 19–22, 2015). New York NY, United States: IEEE, Jan. 2015, pp. 178–183. DOI: 10.1109/aspdac.2015.7059001.

[Mar+16] Michaël Mariën, Koenraad M. R. Audenaert, Karel Van Acoleyen, and Frank Verstraete. "Entanglement Rates and the Stability of the Area Law for the Entanglement Entropy". In: *Communications in Mathematical Physics* 346.1 (Aug. 2016), pp. 35–73. DOI: 10.1007/s00220-016-2709-5.

[MFM08] D. Maslov, S. M. Falconer, and M. Mosca. "Quantum Circuit Placement". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27.4 (Mar. 21, 2008), pp. 752–763. DOI: 10.1109/tcad.2008.917562.

[Mas07] Dmitri Maslov. "Linear depth stabilizer and quantum Fourier transformation circuits with no auxiliary qubits in finite-neighbor quantum architectures". In: *Physical Review A* 76.5 (Nov. 2007). DOI: 10.1103/physreva.76.052310.

[Met+06] Tzvetan S. Metodi, Darshan D. Thaker, Andrew W. Cross, Frederic T. Chong, and Isaac L. Chuang. "Scheduling physical operations in a quantum information processor". In: *Quantum Information and Computation IV*. Ed. by Eric J. Donkor, Andrew R. Pirich, and Howard E. Brandt. SPIE, May 12, 2006, p. 6244. DOI: 10.1117/12.666419.

[MV80] Silvio Micali and Vijay V. Vazirani. "An $O(\sqrt{|V|}|E|)$ algoithm for finding maximum matching in general graphs". In: *21st Annual Symposium on Foundations of Computer Science (sfcs 1980)*. IEEE, Oct. 1980. DOI: 10.1109/sfcs.1980.12.

[Mil+16] Tillmann Miltzow, Lothar Narins, Yoshio Okamoto, Günter Rote, Antonis Thomas, and Takeaki Uno. "Approximation and Hardness of Token Swapping". In: *24th Annual European Symposium on Algorithms (ESA 2016): Algorithms* (Aarhus). Ed. by Piotr Sankowski and Christos Zaroliagis. Vol. 57. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl-Leibniz-Zentrum für Informatik: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Feb. 16, 2016, 66:1–66:15. ISBN: 978-3-95977-015-6. DOI: 10.4230/LIPIcs.ESA.2016.66.

[MK13] C. Monroe and J. Kim. "Scaling the Ion Trap Quantum Processor". In: *Science* 339.6124 (Mar. 8, 2013), pp. 1164–1169. DOI: 10.1126/science.1231298.

[Mon+14] C. Monroe, R. Raussendorf, A. Ruthven, K. R. Brown, P. Maunz, L.-M. Duan, and J. Kim. "Large-scale modular quantum-computer architecture with atomic memory and photonic interconnects". In: *Physical Review A* 89.2 (Feb. 2014). DOI: 10.1103/physreva.89.022317.

[Mur+19] Prakash Murali, Jonathan M. Baker, Ali Javadi Abhari, Frederic T. Chong, and Margaret Martonosi. "Noise-Adaptive Compiler Mappings for Noisy Intermediate-Scale Quantum Computers". In: *ASPLOS '19: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM International Conference on Architectural Support for Programming Languages and Operating Systems. Ed. by Iris Bahar, Maurice Herlihy, Emmett Witchel, and Alvin R. Lebeck. New York NY, United States: The Association for Computing Machinery, 2019, pp. 1015–1029. ISBN: 978-1-4503-6240-5. DOI: 10.1145/3297858.3304075.

[Mur+20] Prakash Murali, Dripto M. Debroy, Kenneth R. Brown, and Margaret Martonosi. "Architecting Noisy Intermediate-Scale Trapped Ion Quantum Computers". In: *ISCA '20: Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture*. IEEE, May 2020. ISBN: 9781728146614. DOI: 10.1109/isca45697.2020.00051.

[NGM20] Beatrice Nash, Vlad Gheorghiu, and Michele Mosca. "Quantum circuit optimizations for NISQ architectures". In: *Quantum Science and Technology* 5.2 (Mar. 2020), p. 025010. DOI: 10.1088/2058-9565/ab79b1.

[NNN05] Thach Cam Nguyen, Hieu Trung Ngo, and Nguyen Bao Nguyen. "Sorting by Restricted-Length-Weighted Reversals". In: *Genomics, Proteomics & Bioinformatics* 3.2 (2005), pp. 120–127. ISSN: 1672-0229. DOI: 10.1016/S1672-0229(05)03016-0.

[NNH99] Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer Berlin Heidelberg, 1999. DOI: 10.1007/978-3-662-03811-6.

[PS16] M. Pedram and A. Shafaei. "Layout Optimization for Quantum Circuits with Linear Nearest Neighbor Architectures". In: *IEEE Circuits and Systems Magazine* 16.2 (2016), pp. 62–74. ISSN: 1531-636X. DOI: 10.1109/MCAS.2016.2549950.

[Pin+21] J. M. Pino, J. M. Dreiling, C. Figgatt, J. P. Gaebler, S. A. Moses, M. S. Allman, C. H. Baldwin, M. Foss-Feig, D. Hayes, K. Mayer, C. Ryan-Anderson, and B. Neyenhuis. "Demonstration of the trapped-ion quantum CCD computer architecture". In: *Nature* 592.7853 (Apr. 2021), pp. 209–213. DOI: 10.1038/s41586-021-03318-4.

[PS02] Ron Pinter and Steven Skiena. "Genomic sorting with length-weighted reversals". In: *Genome informatics. International Conference on Genome Informatics* 13 (Jan. 2002), pp. 103–11. DOI: 10.11234/gi1990.13.103.

[PSC21] Lorenzo Piroli, Georgios Styliaris, and J. Ignacio Cirac. *Quantum Circuits assisted by LOCC: Transformations and Phases of Matter*. Mar. 24, 2021. arXiv: 2103.13367 [quant-ph].

[Pre18] John Preskill. "Quantum Computing in the NISQ era and beyond". In: *Quantum* 2 (Aug. 2018), p. 79. DOI: 10.22331/q-2018-08-06-79.

[Rau05] Robert Raussendorf. "Quantum computation via translation-invariant operations on a chain of qubits". In: *Physical Review A* 72.5 (Nov. 13, 2005). DOI: 10.1103/physreva.72.052301.

[Rau+19] Robert Raussendorf, Cihan Okay, Dong-Sheng Wang, David T. Stephen, and Hendrik Poulsen Nautrup. "Computationally Universal Phase of Quantum Matter". In: *Physical Review Letters* 122.9 (Mar. 2019). DOI: 10.1103/physrevlett.122.090501.

[Rei+17] Markus Reiher, Nathan Wiebe, Krysta M. Svore, Dave Wecker, and Matthias Troyer. "Elucidating reaction mechanisms on quantum computers". In: *Proceedings of the National Academy of Sciences* 114.29 (2017), pp. 7555–7560. ISSN: 0027-8424. DOI: 10.1073/pnas.1619152114.

[Rob55] Herbert Robbins. "A remark on Stirling's formula". In: *The American Mathematical Monthly* 62.1 (1955), pp. 26–29. DOI: 10.2307/2315957.

[RS16] Neil J. Ross and Peter Selinger. "Optimal ancilla-free Clifford+T approximation of z-rotations". In: *Quantum Information and Computation* 16.11&12 (Sept. 2016), pp. 901–953. DOI: 10.26421/QIC16.11-12-1.

[SWD11] Mehdi Saeedi, Robert Wille, and Rolf Drechsler. "Synthesis of quantum circuits for linear nearest neighbor architectures". In: *Quantum Information Processing* 10.3 (June 2011), pp. 355–377. ISSN: 1573-1332. DOI: 10.1007/s11128-010-0201-2.

[SGI12] Takahiko Satoh, François Le Gall, and Hiroshi Imai. "Quantum network coding for quantum repeaters". In: *Physical Review A* 86.3 (Sept. 2012). DOI: 10.1103/physreva.86.032331.

[SB21] Eddie Schoute and Aniruddha Bapat. *Noisy State Reversal*. July 20, 2021. URL: https://gitlab.umiacs.umd.edu/amchilds/noisy-state-reversal.

[SUC19] Eddie Schoute, Cem Unsal, and Andrew Childs. *arct: architecture-respect circuit transformations*. Comp. software. 2019. URL: https://gitlab.umiacs.umd.edu/amchilds/arct.

[SV17] M. Schwartz and P. O. Vontobel. "Improved Lower Bounds on the Size of Balls Over Permutations With the Infinity Metric". In: *IEEE Transactions on Information Theory* 63.10 (2017), pp. 6227–6239. DOI: 10.1109/TIT.2017.2697423.

[SSP14] Alireza Shafaei, Mehdi Saeedi, and Massoud Pedram. "Qubit placement to minimize communication overhead in 2D quantum architectures". In: *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*. ASP-DAC '14: 19th Asia and South Pacific Design Automation Conference (Tokyo, Japan, Jan. 20–23, 2014). Ed. by Toshiyuki Shibuya. New York NY, United States: IEEE, Jan. 2014. DOI: 10.1109/aspdac.2014.6742940.

[Shi+05] Tao Shi, Ying Li, Zhi Song, and Chang-Pu Sun. "Quantum-state transfer via the ferromagnetic chain in a spatially modulated field". In: *Physical Review A* 71.3 (Mar. 2005). DOI: 10.1103/physreva.71.032309.

[Shi+19] Yunong Shi, Nelson Leung, Pranav Gokhale, Zane Rossi, David I. Schuster, Henry Hoffmann, and Frederic T. Chong. "Optimized Compilation of Aggregated Instructions for Realistic Quantum Computers". In: *ASPLOS '19: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM International Conference on Architectural Support for Programming Languages and Operating Systems. Ed. by Iris Bahar, Maurice Herlihy, Emmett Witchel, and Alvin R. Lebeck. New York NY, United States: The Association for Computing Machinery, 2019. ISBN: 978-1-4503-6240-5. DOI: 10.1145/3297858.3304018.

[Sir+19] Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Caroline Collange, and Fernando Magno Quintão Pereira. "Qubit allocation as a combination of subgraph isomorphism and token swapping". In: *Proceedings of the ACM on Programming Languages* 3.OOPSLA (Oct. 2019), pp. 1–29. DOI: 10.1145/3360546.

[Siv+20] Seyon Sivarajah, Silas Dilkes, Alexander Cowtan, Will Simmons, Alec Edgington, and Ross Duncan. "t|ket>: A Retargetable Compiler for NISQ Devices". In: *Quantum Science and Technology* 6.1 (Nov. 2020), p. 014003. DOI: 10.1088/2058-9565/ab8e92.

[Ste+11] M. Steffen, D. P. DiVincenzo, J. M. Chow, T. N. Theis, and M. B. Ketchen. "Quantum computing: An IBM perspective". In: *IBM Journal of Research and Development* 55.5 (Sept. 2011), 13:1–13:11. DOI: 10.1147/jrd.2011.2165678.

[SHT19] Damian S. Steiger, Thomas Häner, and Matthias Troyer. "Advantages of a modular high-level quantum programming framework". In: *Microprocessors and Microsystems* 66 (Apr. 2019), pp. 81–89. DOI: 10.1016/j.micpro.2019.02.003.

[Ste+19] David T. Stephen, Hendrik Poulsen Nautrup, Juani Bermejo-Vega, Jens Eisert, and Robert Raussendorf. "Subsystem symmetries, quantum cellular automata, and computational phases of quantum matter". In: *Quantum* 3 (May 2019), p. 142. DOI: 10.22331/q-2019-05-20-142.

[TS10] I. Tamo and M. Schwartz. "Correcting Limited-Magnitude Errors in the Rank-Modulation Scheme". In: *IEEE Transactions on Information Theory* 56.6 (2010), pp. 2551–2560. DOI: 10.1109/TIT.2010.2046241.

[TC20] Bochen Tan and Jason Cong. "Optimal Layout Synthesis for Quantum Computing". In: (July 30, 2020). arXiv: 2007.15671v1 [cs.AR].

[TC21a] Bochen Tan and Jason Cong. "Optimal Qubit Mapping with Simultaneous Gate Absorption". In: *2021 International Conference on Computer Aided Design*. Sept. 14, 2021. arXiv: 2109.06445 [cs.ET]. Forthcoming.

[TC21b] Bochen Tan and Jason Cong. "Optimality Study of Existing Quantum Computing Layout Synthesis Tools". In: *IEEE Transactions on Computers* 70.9 (Sept. 2021), pp. 1363–1373. DOI: 10.1109/tc.2020.3009140.

[VW04] Farrokh Vatan and Colin Williams. "Optimal quantum circuits for general two-qubit gates". In: *Physical Review A* 69.3 (Mar. 22, 2004). DOI: 10.1103/physreva.69.032315.

[Ven+18] Davide Venturelli, Minh Do, Eleanor Rieffel, and Jeremy Frank. "Compiling quantum circuits to realistic hardware architectures using temporal planners". In: *Quantum Science and Technology* 3.2 (Feb. 2018), p. 025004. DOI: 10.1088/2058-9565/aaa331.

[VHC02] G. Vidal, K. Hammerer, and J. I. Cirac. "Interaction Cost of Nonlocal Gates". In: *Physical Review Letters* 88.23 (May 24, 2002), p. 237902. DOI: 10.1103/PhysRevLett.88.237902.

[WFH11] David S. Wang, Austin G. Fowler, and Lloyd C. L. Hollenberg. "Surface code quantum computing with error rates over 1%". In: *Physical Review A* 83.2 (Feb. 2011), p. 020302. DOI: 10.1103/physreva.83.020302.

[Wat09] John Watrous. "Semidefinite programs for completely bounded norms". Jan. 2009. arXiv: 0901.4709 [quant-ph].

[Wat18] John Watrous. *The Theory of Quantum Information*. Cambridge University Press, 2018. ISBN: 1107180562.

[Web+20] Mark Webber, Steven Herbert, Sebastian Weidt, and Winfried K. Hensinger. "Efficient Qubit Routing for a Globally Connected Trapped Ion Quantum Computer". In: *Advanced Quantum Technologies* 3.8 (July 2020), p. 2000027. DOI: 10.1002/qute.202000027. Pre-published.

[Wil+08] Robert Wille, Daniel Große, Lisa Teuber, Gerhard W. Dueck, and Rolf Drechsler. "RevLib: An Online Resource for Reversible Functions and Reversible Circuits". In: *38th International Symposium on Multiple Valued Logic*. ISMVL 2008 (Dallas TX, United States). IEEE, May 2008, pp. 220–225. DOI: 10.1109/ismvl.2008.43.

[Wil+16] Robert Wille, Oliver Keszocze, Marcel Walter, Patrick Rohrs, Anupam Chattopadhyay, and Rolf Drechsler. "Look-ahead schemes for nearest neighbor optimization of 1D and 2D quantum circuits". In: *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. ASP-DAC '16: 21st Asia and South Pacific Design Automation Conference (Macao, Jan. 25–28, 2016). New York NY, United States: IEEE, Jan. 2016, pp. 292–297. DOI: 10.1109/aspdac.2016.7428026.

[WLD14] Robert Wille, Aaron Lye, and Rolf Drechsler. "Exact Reordering of Circuit Lines for Nearest Neighbor Quantum Architectures". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33.12 (Dec. 2014), pp. 1818–1831. DOI: 10.1109/tcad.2014.2356463.

[Yam+14] Katsuhisa Yamanaka, Erik D. Demaine, Takehiro Ito, Jun Kawahara, Masashi Kiyomi, Yoshio Okamoto, Toshiki Saitoh, Akira Suzuki, Kei Uchizawa, and Takeaki Uno. "Swapping Labeled Tokens on Graphs". In: *Fun with Algorithms: FUN 2014. Lecture Notes in Computer Science*. Springer International Publishing, 2014, pp. 364–375. DOI: 10.1007/978-3-319-07890-8_31.

[YK07] Hong Yao and Steven A. Kivelson. "Exact Chiral Spin Liquid with Non-Abelian Anyons". In: *Physical Review Letters* 99.24 (Dec. 2007), p. 247203. DOI: 10.1103/physrevlett.99.247203.

[Yao+11] N. Y. Yao, L. Jiang, A. V. Gorshkov, Z.-X. Gong, A. Zhai, L.-M. Duan, and M. D. Lukin. "Robust Quantum State Transfer in Random Unpolarized Spin Chains". In: *Physical Review Letters* 106.4 (Jan. 2011). DOI: 10.1103/physrevlett.106.040505.

[Yao+13] N.Y. Yao, C.R. Laumann, A.V. Gorshkov, H. Weimer, L. Jiang, J.I. Cirac, P. Zoller, and M.D. Lukin. "Topologically protected quantum state transfer in a chiral spin liquid". In: *Nature Communications* 4.1 (Mar. 2013), pp. 1–8. DOI: 10.1038/ncomms2531.

[YK17] Theodore J. Yoder and Isaac H. Kim. "The surface code with a twist". In: *Quantum* 1 (Apr. 2017), p. 2. ISSN: 2521-327X. DOI: 10.22331/q-2017-04-25-2.

[Zha99] Louxin Zhang. "Optimal Bounds for Matching Routing on Trees". In: *SIAM Journal on Discrete Mathematics* 12.1 (Jan. 1999), pp. 64–77. DOI: 10.1137/s0895480197323159.

[ZLF20] Xiangzhen Zhou, Sanjiang Li, and Yuan Feng. "Quantum Circuit Transformation Based on Simulated Annealing and Heuristic Search". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.12 (Dec. 2020), pp. 4683–4694. DOI: 10.1109/tcad.2020.2969647.

[ZBL08] Oded Zilberberg, Bernd Braunecker, and Daniel Loss. "Controlled-NOT gate for multiparticle qubits and topological quantum computation based on parity measurements". In: *Physical Review A* 77.1 (Jan. 2008), p. 012327. DOI: 10.1103/physreva.77.012327.

[ZPW18] Alwin Zulehner, Alexandru Paler, and Robert Wille. "An Efficient Methodology for Mapping Quantum Circuits to the IBM QX Architectures". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (June 7, 2018), pp. 1–1. DOI: 10.1109/tcad.2018.2846658.

[ZW19] Alwin Zulehner and Robert Wille. "Compiling SU(4) quantum circuits to IBM QX architectures". In: *ASP-DAC '19: Proceedings of the 24th Asia and South Pacific Design Automation Conference*. 24th Asia and South Pacific Design Automation Conference (Tokyo, Japan, Jan. 21, 2019–Jan. 24, 2014). Ed. by Toshiyuki Shibuya. New York NY, United States: ACM Press, Jan. 2019, pp. 185–190. ISBN: 978-1-4503-6007-4. DOI: 10.1145/3287624.3287704.

Index