Week 3: Quantum Walk Algorithms

Stacey Jeffery

February 11, 2025

3.1 Introduction

Random walks on graphs are a type of process that can be used to model the behaviour of certain types of classical randomized algorithms. The large body of work on understanding the dynamics of random walks, such as the expected time after which the metaphorical walker has visited certain states (i.e. vertices), can then be leveraged to analyze the running time of these algorithms. Analogously, quantum walk algorithms are quantum algorithms that can be designed and analyzed based on random walks.

Broadly speaking, there are two types of quantum walk algorithms: discrete-time quantum walks, and continuous-time quantum walks. A continuous-time quantum walk is an algorithm based on a unitary evolution whose Hamiltonian is chosen according to a classical random walk, for example, as its *Laplacian*, L. Thus, Hamiltonian simulation techniques can be used to implement the algorithm. We will discuss this very briefly in Section 3.8.

Discrete-time quantum walks are quantum algorithms based on a product of two reflections. Some authors describe all quantum algorithms based on many repetitions of a product of two reflections as discrete-time quantum walks. We will see later that this is extremely general. A narrower definition of discrete-time quantum walks, which we will adopt, is a quantum algorithm using a product of two reflections that is designed using a classical random walk. We will see several types of quantum algorithms of this form, in Section 3.5 and Section 3.7.

One of the motivations for studying quantum walk algorithms is, particularly in the case of discrete-time quantum walk algorithms, it is sometimes sufficient to just design a purely classical random walk algorithm, and then apply a generic speedup, meaning that the reasoning needed to design an algorithm of this form is purely classical. Such algorithms are easy to design, but the generic speedups we know are at most quadratic. To extend these techniques to get super-polynomial speedups does require some more involved reasoning, but trying to bridge the gap between easy-to-design quantum walk algorithms, and fast quantum walk algorithms, to the extent that this is possible, is an active subject of research.

3.2 Random Walks on Graphs

Markov chains A *Markov chain* is an infinite sequence of random variables, $(X_t)_{t=0}^{\infty}$ on some state space V satisfying the *Markov property*:

$$\forall x_1, \ldots, x_t \in V,\ \Pr[X_t = x_t | X_1 = x_1, \ldots, X_{t-1} = x_{t-1}] = \Pr[X_t = x_t | X_{t-1} = x_{t-1}].$$

In words: the distribution of X_t depends only on X_{t-1}. A process that gives rise to such a chain is called a Markov process, and we will assume it is time-independent, so that it can be described by a transition matrix $P \in \mathbb{R}^{V \times V}$ where for all t, $\Pr[X_t = v | X_{t-1} = u] = P_{u,v}$, and an initial distribution for X_0, $\sigma \in \mathbb{R}^V$. Then for any $t \geq 0$, X_t has distribution σP^t. Random walks on graphs are a special case of Markov processes.

Random walks All graphs we consider walking on will be undirected. The edges may have positive weights, with the case of all weights being 1 (or simply equal) corresponding to the usual unweighted case.

Weighted undirected graph G:

- vertex set $V(G)$
- edge set $E(G)$
- weight function $\mathsf{w} : E(G) \to \mathbb{R}_{>0}$

It is convenient to fix an arbitrary direction for each edge, but we emphasize that the graph itself is not directed, meaning if u is reachable from v in one step, then v is also reachable from u in one step.

Let $\overrightarrow{E}(G)$ be a set s.t. $\forall\{u,v\} \in E(G)$, $(u,v) \in \overrightarrow{E}(G)$ xor $(v,u) \in \overrightarrow{E}(G)$.

For ease of notation, we overload the weight function w to act on directed edges, and also extend it to be 0 on pairs of vertices that are not connected by an edge.

Let $\mathsf{w}_{u,v} = \mathsf{w}_{v,u} = \mathsf{w}_{\{u,v\}}$ for all $\{u,v\} \in E(G)$.
Let $\mathsf{w}_{u,v} = 0$ $\forall\{u,v\} \notin E(G)$.

We define notation for the total weight on a vertex, which is a weighted version of the degree, and the total weight on the entire graph, which is a weighted version of the number of edges.

$\forall u \in V(G)$, let $\mathsf{w}_u = \sum_{v \in V(G)} \mathsf{w}_{u,v}$. Let $\mathcal{W}(G) = \displaystyle\sum_{(u,v) \in \overrightarrow{E}(G)} \mathsf{w}_{u,v} = \frac{1}{2} \sum_{u \in V(G)} \mathsf{w}_u$.

A random walk on G is a Markov process on $V(G)$ with transition matrix $P_{u,v} = \frac{\mathsf{w}_{u,v}}{\mathsf{w}_u}$.

Its *stationary distribution* is $\pi_u = \frac{\mathsf{w}_u}{2\mathcal{W}(G)}$, so

$$\pi_u P_{u,v} = \frac{\mathsf{w}_{u,v}}{2\mathcal{W}(G)} = \pi_v P_{v,u},$$

a condition known as *detailed balance*. Markov processes with detailed balance are called *reversible*. Random walks on graphs are precisely reversible Markov processes.

Exercise 3.2.1. *Show that $\pi P = \pi$, so π is a left 1-eigenvector of P (and this is precisely the property that makes π a* stationary distribution *of P). Bonus: Show that π is the* unique *left 1-eigenvector of P.*

Finally, we introduce some notation to describe the neighbours of a vertex.

Let $\Gamma(u) = \{v \in V(G) : \{u,v\} \in E(G)\}$ be the *neighbourhood* of u

3.3 Random Walk Algorithms vs. Quantum Walk Algorithms

A random walk algorithm is an algorithm that can be modelled as a random walk. That is, there is some graph G (relevant to the problem we want to solve), and we assume we can "walk" on it, meaning we can sample, for any current vertex u, a neighbour of u to "walk" to, according to the distribution $\Pr[v] = P_{u,v} = \mathsf{w}_{u,v}/\mathsf{w}_u$.

Example 3.3.1 (PageRank)**.** An important example of a random walk algorithm is PageRank, which is used (for example, by Google) to determine the relative importance of web pages, as measured (loosely) by the number of other pages linking to the page in question. Consider a graph whose vertices are all the web pages on the world wide web (or perhaps some other network), and where there is an edge from u to v if the page u contains a link to the page v (note that this is a *directed graph*). If you are at page u, you can sample a neighbouring page by choosing one of the links on page u uniformly at random, and following the link, to get to a new page. The algorithm works by first assuming that all pages have equal rank. Then a random walk is simulated, and every time the walker arrives at a vertex u, the page rank of u is increased, to reflect the fact that arriving at u is more likely the more pages there are that link to u.

Whereas random walk algorithms assume the ability to sample a neighbour of any given vertex u according to the distribution $P_{u,v} = \mathsf{w}_{u,v}/\mathsf{w}_u$, the quantum walk algorithms we discuss will assume the ability to produce a "quantum sample" given any vertex u:

$$|u,0\rangle \mapsto \sum_{v\in\Gamma(u)} \sqrt{P_{u,v}}|u,v\rangle \propto \sum_{v\in\Gamma(u)} \sqrt{\mathsf{w}_{u,v}}|u,v\rangle. \tag{7}$$

3.3.1 Edge Lists

We have just seen that a classical random walk algorithm assumes that it is algorithmically possible, via some implicit subroutine, to sample a neighbour of any u, according to the distribution $P_{u,\cdot}$ given by the u-th row of P. In practice, this usually consists of two steps: (1) sampling some i from $\{1,\ldots,d(u)\}$, where $d(u)$ is the *degree* of u; (2) computing the i-th neighbour of u.

Let us say more about (2), which is connected to a common model of accessing an input graph, the *edge list model*. In the edge list model, a graph is described by a list of neighbours for each vertex. Formally, we model this as having query access to a function $f_u : [d(u)] \to V(G)$ whose image is $\Gamma(u)$, for each $u \in V(G)$; as well as being able to query the degrees $d(u)$. Then we call $f_u(i)$ the i-th neighbour of u.

In fact, it is not necessary that the labels, i, that constitute the domain of f_u, are numbers $[d(u)]$. In general, for each u, we can have a label set $L(u)$ such that $f_u : L(u) \to V(G)$ is one-to-one and has image $\Gamma(u)$. As an example, consider a *Johnson graph*, which you have seen in the context of the quantum walk algorithm for element distinctness [dW19, Section 8.3.2]. The vertices are sets $S \subset [n]$ of size r, and the neighbours of S are those vertices S' obtained from S by removing some $j \in S$, and adding some $j' \in [n] \setminus S$. So we can take a step of the random walk by first sampling some $j \in S$ and $j' \in [n] \setminus S$, which gives us a label

$$(j, j') \in L(S) = S \times ([n] \setminus S).$$

Then we must compute $f_S(j, j') = (S \setminus \{j\}) \cup \{j'\}$.

What are the quantum versions of the operations described in (1) and (2)? Naturally, the quantum version of (1) is to generate the superposition

$$|u,0\rangle \mapsto \sum_{i\in L(u)} \sqrt{P_{u,f_u(i)}}|u,i\rangle.$$

For (2), we want to compute $f_u(i)$ from u, however, unlike in the classical world, we can't just replace u with $v = f_u(i)$, since this is not reversible. However, if $j \in L(v)$ is such that $f_v(j) = u$, then the map that acts as

$$|u, i\rangle \mapsto |v, j\rangle$$

for all $u \in V(G)$ and $i \in L(u)$ is reversible. That's because (u, i) and (v, j) are just different ways of referring to the same edge $\{u, v\}$: we can think of it as the edge coming out of u towards its i-th neighbour (which is v), or as the edge coming out of v towards its j-th neighbour (which is u).

For simplicity, we will only consider the special case where $L(u) = \Gamma(u)$ and $f_u(v) = v$, which just gives us the model we first described, where we sample a neighbour directly from $\Gamma(u)$. In this special case, the operation $|u, i\rangle \mapsto |v, j\rangle$ is just the vertex *swap* $|u, v\rangle \mapsto |v, u\rangle$, which can be thought of as swapping the vertex v into the "current vertex" register. However, in some cases, such as the Johnson graph example mentioned above, this swap is incredibly inefficient.

As an exercise to check your understanding, you may verify that the results of this section, and in particular, the proof in Section 3.7, hold if we let U (see, for example, the statements of Theorem 3.5.3, Theorem 3.5.6, Theorem 3.5.8 and Theorem 3.7.1) be the cost of the following two operations:

(1) For any $u \in V(G)$, generate the state $\sum_{i \in L(u)} \sqrt{P_{u,f_u(i)}} |u, i\rangle$

(2) The vertex transition that acts, for any $u \in V(G)$ and $i \in L(u)$, as $|u, i\rangle \mapsto |v, j\rangle$, where $v = f_u(i)$ and $j = f_v^{-1}(u)$

3.4 Analyzing Random Walk Algorithms

As will be made very explicit in the following section, in order to analyze random walk algorithms, we need to be able to bound the number of steps a walker needs to take before a certain desirable event is likely to have occurred (called a *stopping time*). For example, if $M \subset V(G)$ is a set of vertices that we are searching for (see following section), then we can define the *hitting time* to M:

> Hitting time: $\mathcal{HT}(P, M)$ = expected number of steps before a walker, starting from the stationary distribution, hits a vertex $u \in M$.

The hitting time is an expectation, but it is also the case that after $\Theta(\mathcal{HT}(P, M))$ steps, a walker has seen a marked vertex with probability $\Omega(1)$.

Exercise 3.4.1. *Show that for any constant $c \geq 1$, the probability that a walker reaches a vertex of M in the first $c\mathcal{HT}(P, M)$ steps, starting from the stationary distribution, is at least $1 - \frac{1}{c}$.*

Another important quantity is the *mixing time*, which measures how long it takes for a walker's distribution to become completely spread over the graph, according to its stationary distribution. Note that this spreading is only possible if G is connected, and so we assume that this is the case. For a connected graph G, and a parameter $\epsilon \in (0, 1)$, the *mixing time* $\tau_{\text{mix}}(\epsilon)$ is the number of steps after which a walker will be ϵ-close to the distribution π. To measure this closeness, we use the following.

Definition 3.4.1 (Total Variation Distance)**.** *For two probability distributions ρ and σ on a finite set V, their* total variation distance *is defined:*

$$\|\rho - \sigma\|_{\text{TV}} := \max_{S \subset V} |\rho(S) - \sigma(S)| = \frac{1}{2} \sum_{u \in V} |\rho(u) - \sigma(u)|.$$

Then we can formally define $\tau_{\text{mix}}(\epsilon)$ as the smallest t such that a walker starting in any distribution σ will be ϵ-close in total variation distance to π after t steps:

$$\tau_{\text{mix}}(\epsilon) := \min\left\{t \in \mathbb{Z}_{\geq 0} : \sup_{\sigma} \left\|\pi - \sigma P^t\right\|_{\text{TV}} \leq \epsilon\right\} \quad \textbf{[9]}$$

When we don't specify a value ϵ, it should be taken to be $1/4$ (this choice is somewhat arbitrary). A closely related quantity is $1/\gamma_\star$, where $\gamma_\star$ is the *spectral gap*.

Spectral Gap: $\gamma_\star := 1 - \max\{|\lambda| : \lambda < 1 \text{ is an eigenvalue of } P\}$. **[10]**

Note that since P is stochastic, all of its eigenvalues are in $[-1, 1]$, but it is possible for P to have an eigenvalue -1, in which case, $\gamma_\star = 0$. This happens precisely when G is a bipartite graph, which is a graph whose vertices can be partitioned into two disjoint sets $V_{\mathcal{A}}$ and $V_{\mathcal{B}}$ such that all edges have one endpoint in each of $V_{\mathcal{A}}$ and $V_{\mathcal{B}}$. In that case, a random walker starting from $V_{\mathcal{A}}$ will keep swapping between $V_{\mathcal{A}}$ and $V_{\mathcal{B}}$ and will never reach the distribution π, so the mixing time is not finite. For any G that is not bipartite, $\gamma_\star \in (0, 1]$.

The quantity $1/\gamma_\star$ is sometimes called the *relaxation time*, and it is within a log factor of the mixing time (see, for example, [LP17, Theorems 12.3-12.4]),

$$(1/\gamma_\star - 1)\ln\frac{1}{2\epsilon} \leq \tau_{\text{mix}}(\epsilon) \leq \frac{1}{\gamma_\star}\ln\frac{1}{\epsilon\pi_{\min}}$$

making it a reasonable proxy (here $\pi_{\min} = \min\{\pi(u) : u \in V(G)\}$). An intuitive explanation as to why $1/\gamma_\star$ steps are sufficient for mixing can be found in [dW19, Chapter 8.1].

3.5 Random Walk Search Algorithms, and their Quantum Counterparts

In a random walk search algorithm, we set up a graph so that some of its vertices encode solutions to some abstract problem we want to solve. We call those special vertices encoding solutions "marked" and we wander around the graph until we find a marked vertex. More precisely:

Fix a graph G, and a marked set $M \subset V(G)$, both of which may implicitly depend on an input x. **[11]**

Algorithm 1. Random Walk Search 1

1. Sample u from V according to π
2. Repeat $T \geq \mathcal{HT}(P, M)$ times:
   (a) Check if $u \in M$, if so, output u
   (b) Sample v from $\Gamma(u)$ according to $P_{u,\cdot}$ and set $u \leftarrow v$
3. Output ''no marked vertices''

We have chosen to let our random walk begin in the stationary distribution, π, which sometimes makes the analysis easier. In practice the initial distribution should be something that is easy to sample from. The stationary distribution is uniform in regular unweighted graphs, and this is usually an easy distribution to sample.

If we don't find a marked vertex after T steps, we conclude there are none. As we have seen, in Exercise 3.4.1, as long as $T \geq c\mathcal{HT}(P, M)$ for some constant $c > 1$, this algorithm finds a marked vertex, if one exists, with probability $\Omega(1)$.

Algorithm 1 is not really an algorithm, but rather a template for an algorithm, into which an algorithm designer must plug some graph G (which then determintes P and π), and marked set M. The complexity of an algorithm of the form Algorithm 1 depends on the upper bound T on $\mathcal{HT}(P, M)$, and the complexity of three subroutines that would need to be implemented based on the actual graph G and marked set M:

- sampling an initial vertex u according to π, whose cost we will denote by S;
- checking, for any u, if $u \in M$, whose cost we will denote by C;
- sampling, for any u, a neighbour v, whose cost we will denote by U.

Then the total cost of Algorithm 1 is:

$$\mathsf{S} + \mathcal{HT}(P, M)(\mathsf{C} + \mathsf{U}),$$

assuming $\mathcal{HT}(P, M)$ is known, so that we can set $T = \mathcal{HT}(P, M)$. If we don't know $\mathcal{HT}(P, M)$ – which is likely, as it generally depends on the implicit input – but we know some upper bound $\boldsymbol{\mathcal{HT}}$, we can set T to that, and the complexity will depend on this upper bound.

Example 3.5.1. In a *constraint satisfaction* problem, given a set of *contraints* $\{C_i : \{0,1\}^n \to \{0,1\}\}_{i\in[m]}$, you want to find a solution $z \in \{0,1\}^n$ such that $\varphi(z) := C_1(z) \dots C_m(z) = 1$ (i.e. z satisfies all constraints). If there is no additional structure, the best first step is usually to choose a random z and check if $\varphi(z) = 1$. If not, you could choose another random z, but if instead you just change a single bit of z, you will only have to recompute the constraints that depend on that bit, which might be very few of them if each constraint only depends on, say, a constant number of bits of z. This can be modelled by a random walk on the *Boolean hypercube*, which is the graph with vertex set $V = \{0,1\}^n$, and an edge between two vertices if they differ in exactly one bit. The marked set is the set of z such that $\varphi(z) = 1$.

Example 3.5.2 (Element Distinctness)**.** The main running example throughout this chapter will be the problem of *element distinctness*, which you may recall from [dW19, Section 8.3.2]. In this problem, the input is a string $x \in [q]^n$, accessed via an oracle, and the goal is to decide if there exist distinct $i, j \in [n]$ such that $x_i = x_j$, called a *collision*. This problem, or some version of it, comes up in many different contexts. Unlike in the related problem of *collision finding* in hash functions, we do not assume that there are many collisions. In fact, we may assume without loss of generality that there are 0 collisions, or a unique collision.

We can solve this problem with a random walk as follows. Let G be the unweighted *Johnson graph* $J(n, r)$ whose vertices are:

$$V = \{v_S : S \subset [n], |S| = r\}$$

and $\{v_S, v_{S'}\} \in E$ if and only if $|S \cap S'| = r - 1$. Assume that with the vertex v_S, we store not only the set S, but also the query values of S:

$$v_S = \{(i, x_i) : i \in S\}.$$

We say a vertex is marked if it contains $(i, x_i), (j, x_j)$ such that $i \neq j$ and $x_i = x_j$. Then if we want to check if a vertex is marked, we can do so using $\mathsf{C} = 0$ queries to x (for simplicity, we will only analyze the query complexity of this example).

The Johnson graph $J(n, r)$ is a regular graph, so its stationary distribution is uniform. To uniformly sample a vertex v_S, we must uniformly sample a set S of size r, and then query each $i \in S$, costing $\mathsf{S} = r$ queries. To move to a neighbouring vertex, we need to remove an index $j \in S$, and choose a new index j' to add to S – this j' must be queried, so we have $\mathsf{U} = 1$.

How long before we can expect to have seen a marked vertex? Suppose there is a unique collision $x_a = x_b$. After t steps of this random walk, we have seen about $r + t$ elements, so we need $t \approx n$ before we can expect to have seen (a, x_a). However, to find the collision, we need to add b before we remove a. Since we expect to remove a approximately r steps after we add it, the probability we add b in that time is only about r/n. Thus, we need to encounter a approximately n/r times before we can expect to see b at the same time. This can be formalized to show:

$$\boldsymbol{\mathcal{HT}} \approx \frac{n}{r} n = \frac{n^2}{r}$$

is an upper bound on the hitting time. Thus, we can find a collision (or conclude there is none) in

$$\mathsf{S} + \boldsymbol{\mathcal{HT}}(\mathsf{C} + \mathsf{U}) = r + \frac{n^2}{r} = O(n^2/r)$$

classical queries and r space, for any r.

This random walk algorithm is severe overkill. It does not do better than a classical algorithm that simply samples a set S of size r, queries everything and checks for a collision, and repeats. That is, a classical random walk of this form does no better than brute force search for element distinctness. However, we will soon see that the situation is different when we move to quantum algorithms.

Szegedy Framework An important observation, originally due to Szegedy, is that given a classical algorithm of the form in Algorithm 1, one can immediately get a quantum algorithm that uses quadratically fewer steps of the walk.

Theorem 3.5.3 (Quantum Walk Search Framework: Szegedy Type [Sze04, AGJK20])**.** *Fix a weighted graph G and marked set $M \subset V(G)$. Let P be the transition matrix of the random walk on G, and π its stationary distribution (see Section 3.2). Suppose we can implement the following quantum subroutines:*

Setup: *Generate the state $|\pi\rangle = \sum_{u \in V(G)} \sqrt{\pi_u} |u\rangle$, in cost S*

Check: *For any $u \in V(G)$, check if $u \in M$, in cost C*

Update: **(1)** *For any $u \in V(G)$, generate the state $\sum_{v \in V(G)} \sqrt{P_{u,v}} |u, v\rangle$, in cost U*

(2) *Implement the vertex swap that acts, for all $u \in V(G)$ and $v \in \Gamma(u)$, as $|u, v\rangle \mapsto |v, u\rangle$, also in cost U*

Let $\boldsymbol{\mathcal{HT}}$ be an upper bound on $\mathcal{HT}(P, M)$ whenever $M \neq \emptyset$. Then there is a quantum algorithm that finds $u \in M$ (or determines $M = \emptyset$) with bounded error in complexity:

$$\widetilde{O}(\mathsf{S} + \sqrt{\boldsymbol{\mathcal{HT}}}(\mathsf{U} + \mathsf{C})).$$

We remark that the costs S and U have slightly different meanings than the identical notation in the cost of the classical algorithm above. Generating a superposition $\sum_u \sqrt{\pi_u} |u\rangle$ may be more difficult than sampling from the distribution π. However, in practice, these costs are often the same. In fact, usually π is uniform, and the distributions $P_{u,\cdot}$ are uniform on $\Gamma(u)$, and so the classical and quantum sampling are of approximately equal difficulty.

Example 3.5.4. Let $f : \{0,1\}^n \to \{0,1\}$. Let G be defined by $V(G) = \{0,1\}^n \cup \{v\}$ and $E(G) = \{\{v, z\} : z \in \{0,1\}^n\}$, so G is a *star graph* with v at its center and 2^n leaves. Let $M = \{z \in \{0,1\}^n : f(z) = 1\}$. Then we can show that $\mathcal{HT}(P, M) = O(2^n)$. In this way, we can see that Theorem 3.5.3 is a generalization of quantum search.

Example 3.5.5 (Element Distinctness)**.** Let everything be as in Example 3.5.2. Then there is a quantum walk algorithm for element distinctness with query complexity (neglecting polylog factors):

$$\mathsf{S} + \sqrt{\boldsymbol{\mathcal{HT}}}(\mathsf{U} + \mathsf{C}) = r + \sqrt{\frac{n^2}{r}} = r + \frac{n}{\sqrt{r}}.$$

Unlike in the classical random walk of Example 3.5.2, the second term in the complexity does not always dominate. We can minimize this expression by choosing r so that the two terms are equal: $r = n^{2/3}$, leading to a quantum algorithm for element distinctness with query complexity $\widetilde{O}(n^{2/3})$, which is optimal [AS04]. This algorithm was first described by Ambainis in [Amb07], before Szegedy's framework. In fact, Szegedy's framework is inspired by Ambainis' element distinctness algorithm.

It is quite remarkable that quantum computers can solve element distinctness using only $n^{2/3}$ queries to the input. You should be able to convince yourself that a classical algorithm requires $\Omega(n)$ queries to solve this problem.

MNRS Framework Consider a slightly different type of classical random walk search algorithm. In the following, we let $\pi_M = \sum_{u \in M} \pi_u$ be the probability that a vertex sampled according to π is in M, and let $\gamma_\star$ be the spectral gap of the random walk on G (see Section 3.4).

Algorithm 2. Random Walk Search 2

1. Sample u from V according to π
2. Repeat $T_1 \geq 2/\pi_M$ times:
   (a) Check if $u \in M$, if so, output u
   (b) Repeat $T_2 \geq \frac{1}{\gamma_\star} \ln \frac{2}{\pi_M \pi_{\min}}$ times:
      i. Sample v from $\Gamma(u)$ according to $P_{u,\cdot}$ and set $u \leftarrow v$
3. Output ''no marked vertices''

As with Algorithm 1, we begin by sampling a vertex according to the stationary distribution, π, and then we do a random walk. In constrast to Algorithm 1, we do not check if the current vertex is marked at every step. In some applications, the checking step is expensive, and so it might make sense to do it less often. Instead, Algorithm 2 walks for at least $1/\gamma_\star$ steps before checking. This ensures that every time we check, we have an independent sample from π. Let us discuss this point further in the precise language of Markov chains. We begin by sampling a vertex according to π, which gives us a random variable X_0 on $V(G)$ (what we call u in the first line of Algorithm 2). The next vertex, X_1, is sampled from the neighbourhood of X_0, so it has distribution $\pi P = \pi$, since π is the stationary distribution of P, the transition matrix of the random walk on G. In fact, if $X_0, X_1, \ldots, X_{T_1 T_2}$ is the sequence of vertices visited by Algorithm 2, then each X_i has marginal distribution π. However, these random variables are *not* independent. For example, X_{i+1} can only be a neighbour of X_i, so its distribution given that $X_i = u$ for some $u \in V$ is supported only on $\Gamma(u)$. As we take more steps from any X_i, the resulting random variable, X_{i+k}, gets less dependent on X_i (i.e. as k grows). The *mixing time* tells us after how many steps this dependence is almost gone.

Exercise 3.5.1. *For any square n, describe a connected graph on n vertices, and a marked set of size n^a for some constant $a \in (0,1)$ (for example, $a = 2/3$), such that there exists a constant $b \in (0,1)$ (for example, $b = 1/3$) such that for any constant c, the probability that you have found a marked vertex after c/π_M steps starting from a vertex sampled according to π is $O(1/n^b)$. You may use the fact that the stationary distribution is uniform in a regular graph.*

Exercise 3.5.2. *Let $(X_i)_{i=0}^{\infty}$ be the Markov chain associated with sampling X_0 according to π, and then doing a random walk on G starting from X_0.*

1. *Show that for any $i \geq 0$, and $T_2 \geq \frac{1}{\gamma_\star} \ln \frac{2}{\pi_M \pi_{\min}}$, and any $u \in V(G)$,*

$$\Pr[X_{i+T_2} \in M | X_i = u] \geq \frac{\pi_M}{2}.$$

2. *Show that for any $T_2 \geq \frac{1}{\gamma_\star} \ln \frac{2}{\pi_M \pi_{\min}}$, and $T_1 \geq \frac{2}{\pi_M}$,*

$$\Pr\left[\bigvee_{j=0}^{T_1-1} X_{jT_2} \in M\right] \geq \frac{3}{5}.$$

Hint: You may use the useful fact that $1 - x \leq e^{-x}$ for all $x \in \mathbb{R}$.

Thus, the sequence of random variables that we check, $X_0, X_{T_2-1}, X_{2T_2-1}, \ldots, X_{T_1T_2-1}$ are close to being T_1 independent samples from π. The probability that any of them is marked is π_M, so taking $T_1 \approx 1/\pi_M$ is sufficient to find a marked vertex with constant probability.

As in the case of Algorithm 1, there is a generic quantum speedup for algorithms of this type.

Theorem 3.5.6 (Quantum Walk Search Framework: MNRS Type [MNRS11])**.** *Fix a weighted graph G and marked set $M \subset V(G)$. Let P be the transition matrix of the random walk on G, and π its stationary distribution (see Section 3.2). Suppose we can implement the following quantum subroutines:*

Setup: *Generate the state $|\pi\rangle = \sum_{u \in V(G)} \sqrt{\pi_u} |u\rangle$, in cost S*

Check: *For any $u \in V(G)$, check if $u \in M$, in cost C*

Update: **(1)** *For any $u \in V(G)$, generate the state $\sum_{v \in V(G)} \sqrt{P_{u,v}} |u, v\rangle$, in cost U*

(2) *Implement the vertex swap that acts, for all $u \in V(G)$ and $v \in \Gamma(u)$, as $|u, v\rangle \mapsto |v, u\rangle$, also in cost U*

Let δ be a lower bound on $\gamma_\star$, and ε a lower bound on π_M whenever $M \neq \emptyset$. Then there is a quantum algorithm that finds $u \in M$ (or determines $M = \emptyset$) with bounded error in complexity:

$$\widetilde{O}\left(\mathsf{S} + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}\right)\right).$$

We can compare Theorem 3.5.6 to Theorem 3.5.3. In the first version of Theorem 3.5.3 [Sze04], the algorithm did not *find* a marked vertex, but only decided if one existed. An improved algorithm extended the result to finding [AGJK20], but in the meantime, the MNRS framework in its original formulation could already find a marked element. To compare the two frameworks, we note that:

$$\frac{1}{\pi_M} \leq \mathcal{HT}(P, M) \leq \frac{1}{\pi_M \gamma_\star}. \tag{13}$$

Which framework is better depends on where $\mathcal{HT}(P, M)$ falls in this range, and the relative costs of updates, U, and checking, C.

Example 3.5.7 (Element Distinctness)**.** Let everything be as in Example 3.5.2. We claim that the spectral gap of $J(n, r)$ is $\delta = \Theta(1/r)$. A high-level argument for this is that after $\Theta(r)$ steps, most elements of the set have been replaced with uniform random ones, so the state is approximately uniformly distributed.

If there is at least one marked vertex, meaning there exists a collision $x_a = x_b$, then the probability that a uniform random set of size r contains a is approximately r/n, and the probability it contains

b is approximately r/n, and these are almost independent, so the probability it contains a and b (and thus, is marked) is at least $\varepsilon = \Omega(r^2/n^2)$. Thus, there is a quantum walk algorithm that decides element distinctness in query complexity (neglecting log factors):

$$\mathsf{S} + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}\right) = r + \sqrt{\frac{n^2}{r^2}}\sqrt{r} = r + \frac{n}{\sqrt{r}}.$$

This is the same complexity as we saw in Example 3.5.5, because for a Johnson graph and this choice of marked set, the hitting time is asymptotically equal to $\frac{1}{\pi_M \gamma_\star}$, and the checking cost is 0.

Exercise 3.5.3. *Modify the quantum algorithm in Example 3.5.7 so that the marked set is*

$$M = \{v_S : \exists i \in S, j \in [n] \setminus \{i\}, x_i = x_j\}.$$

Analyze your algorithm.

Looking at the two frameworks we have seen so far, a natural question is if we can give generic speedups for intermediate classical algorithms, in which the number of steps between checks is some intermediate value, $t \in \{1, \ldots, 1/\gamma_\star\}$, as in the following.

Algorithm 3. Random Walk Search 3

For parameter $t \in \{1, \ldots, 1/\gamma_\star\}$:
1. Sample u from V according to π
2. Repeat $T_1 \geq \mathcal{HT}(P^t, M)$ times:
   (a) Check if $u \in M$, if so, output u
   (b) Repeat t times:
      i. Sample v from $\Gamma(u)$ according to $P_{u,\cdot}$ and set $u \leftarrow v$
3. Output ‘‘no marked vertices’’

In fact, for any choice of t, including $t = 1/\gamma_\star$, which gives us Algorithm 2, this turns out to be a special case of Algorithm 1. A hint of this fact is already present in our choice for T_1, which we now discuss. For any Markov process transition matrix P, the matrix P^t is also stochastic, so it is also the transition matrix of a Markov process, and the following exercise is to show that if P is a random walk transition matrix on V, then so is P^t.

Exercise 3.5.4. *Show that if P is the transition matrix of a reversible Markov process (i.e. a random walk on a graph), then so is P^t. Furthermore, show that if π is the stationary distribution of P, it is also the stationary distribution of P^t.*

The random walk P^t is simply the walk that, from any u, samples a next vertex v from the distribution you would get by taking t steps in the walk on G (described by the transition matrix P). Thus, Algorithm 3 is simply Algorithm 1 with the random walk P^t: we take t steps of P (i.e. one step of P^t), and then check, and we repeat this $\mathcal{HT}(P^t, M)$ times. By Theorem 3.5.3, we know that there is a quantum algorithm that finds a marked vertex in the setting of Algorithm 3 in complexity

$$\mathsf{S} + \sqrt{\mathcal{HT}(P^t, M)}(\mathsf{U}(t) + \mathsf{C})$$

where S is the cost to generate $|\pi\rangle$ (which is also the stationary distribution of P^t), C is the cost to check membership in M, but now $\mathsf{U}(t)$ is the cost to take one step of P^t. For a classical algorithm, taking one step of P^t costs t steps of P, and quantumly, we can do quadratically better, using a technique called *quantum fast-forwarding* [AS19]. In some subtle sense, fast-forwarding allows us to take t steps of a random walk in $\sqrt{t}$ steps. Note that this is not as crazy as it sounds – a random walk on a line of length T takes $\Theta(T^2)$ steps to find the end of the line, starting from the beginning, so

even with a quadratic speedup, it's not possible to cover more than T "distance" in T steps. We use quantum fast-forwarding to show that $\mathsf{U}(t) = \Theta(\sqrt{t})$. What this means precisely depends on what we mean by "take one step of P^t." We had formally defined U, in, for example, Theorem 3.5.3, to be the cost to generate states of the form $\sum_{v \in V(G)} \sqrt{P_{u,v}}|u,v\rangle$, and to swap $|u,v\rangle \mapsto |v,u\rangle$. These operations are sufficient to implement a unitary called the *walk operator* (which is similar to the unitary in (3.2)) that is used in the quantum algorithm that implements Theorem 3.5.3. What quantum fast-forwarding accomplishes is precisely to implement the walk operator of P^t in $O(\sqrt{t})$ calls to the walk operator of P, which gives us the following:

Theorem 3.5.8 (Unified Quantum Walk Search Framework [AGJ20])**.** *Fix a weighted graph G and marked set $M \subset V(G)$. Let P be the transition matrix of the random walk on G, and π its stationary distribution (see Section 3.2). Suppose we can implement the following quantum subroutines:*

Setup: *Generate the state $|\pi\rangle = \sum_{u \in V(G)} \sqrt{\pi_u}|u\rangle$, in cost S*

Check: *For any $u \in V(G)$, check if $u \in M$, in cost C*

Update: **(1)** *For any $u \in V(G)$, generate the state $\sum_{v \in V(G)} \sqrt{P_{u,v}}|u,v\rangle$, in cost U*

(2) *Implement the vertex swap that acts, for all $u \in V(G)$ and $v \in \Gamma(u)$, as $|u,v\rangle \mapsto |v,u\rangle$, also in cost U*

Let t be any integer in $\{1, \ldots, \lceil 1/\delta \rceil\}$, where δ is a lower bound on $\gamma_\star$. Let $\boldsymbol{\mathcal{HT}}_t$ be an upper bound on $\mathcal{HT}(P^t, M)$ whenever $M \neq \emptyset$. Then there is a quantum algorithm that finds $u \in M$ (or determines $M = \emptyset$) with bounded error in complexity:

$$\widetilde{O}\left(\mathsf{S} + \sqrt{\boldsymbol{\mathcal{HT}}_t}\left(\sqrt{t}\mathsf{U} + \mathsf{C}\right)\right).$$

It is easy to see that Theorem 3.5.3 is the special case of this for $t = 1$. What is a bit more subtle is that Theorem 3.5.6 is a special case of this when $t = 1/\delta$. To see this at a high level, note that if $(X_t)_{t=0}^{\infty}$ is the Markov chain arising from a random walk $P^{1/\delta}$ starting from the distribution π, the distribution of this sequence is close to a sequence of independent random variables each distributed according to π, since $1/\delta$ is (approximately) an upper bound on the mixing time. Thus, $\mathcal{HT}(P^{1/\delta}, M)$ should be at most $1/\pi_M$.

Finally, a natural extension of these frameworks is to consider algorithms that start in some distribution other than π, such as at a particular vertex s, or some other easy to prepare distribution. In Section 3.7, we will see that we can quantize algorithms like Algorithm 1, Algorithm 2 and Algorithm 3 (actually all a special case of Algorithm 1, as we've argue) but starting in any distribution σ. However, we will not get the kind of generic squareroot speedup we've seen in this section. Instead, the quantum algorithm will have a complexity that depends on the square root of a quantity that is not always the complexity of the corresponding classical algorithm. For this we need some more graph theory.

3.6 Random Walks and Electric Networks

A weighted graph can be viewed as a network of resistors, with an edge e of weight w modeling a $1/\mathsf{w}$-ohm resistor. It turns out that the properties of this electrical network are intimately related to the behaviour of a random walker on the network. This beautiful theory is expounded in [DS84], and also covered in [LP17]. We will use some of these electric network properties to understand the running time of quantum walk algorithms.

Definition 3.6.1 (Flows and Circulations)**.** *A* flow *on G is a real-valued function $\theta : \overrightarrow{E}(G) \to \mathbb{R}$ extended to edges in both directions by $\theta(u,v) = -\theta(v,u)\ \forall (v,u) \in \overrightarrow{E}(G)$. $\forall u \in V(G)$, define $\theta(u) = \sum_{v \in \Gamma(u)} \theta(u,v)$ (flow coming out of u) If $\theta(u) = 0$, flow is* conserved *at u. If flow is conserved at every vertex, we call θ a* circulation*. If $\theta(u) > 0$, u is a* source*. If $\theta(u) < 0$, u is a* sink*. A flow with*

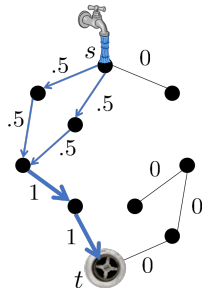

Figure 3.1: We can think of an st-flow as a unit of water entering at s, flowing through the edges of the graph, and then exiting at t. Here we specify non-zero flow across an edge by a direction, shown by an arrow, and a positive number indicating the strength of the flow. For all vertices other than s and t, the amount of incoming flow must equal the amount of outgoing flow.

unique source s and sink t is called an st-flow. *If $\theta(s) = 1$, it's a* unit st-flow. *The* energy *of a flow is $\mathcal{E}(\theta) = \sum_{(u,v)\in\vec{E}(G)} \frac{\theta(u,v)^2}{\mathsf{w}_{u,v}}$.*

We can think of a flow as water flowing through the edges of a graph as if they're pipes. When flow is conserved at a vertex, all flow coming in through some edges must leave through the other edges. For example, in an st-flow, water comes in at vertex s, flows through the edges incident to s and then through the graph until it all leaves through a sink at t (see Figure 3.1). The electric network analogy would be the *current* flowing through the network when there is a potential difference between s and t, but in that case, the current corresponds to a particular unit st-flow: the one with minimal energy.

Definition 3.6.2 (Effective Resistance between s and t)**.** *Let $s, t \in V(G)$ be distinct vertices of a graph G. The effective resistance between s and t in G is defined as the minimum energy of any unit st-flow: $\mathcal{R}_{s,t}(G) := \min\{\mathcal{E}(\theta) : \theta$ a unit st-flow$\}$.*

As the name suggests, the effective resistance is the amount of electrical resistance across the whole network between s and t. The st-flow θ with minimum energy is proportional to the current flowing through each edge if a potential difference is places between s and t. The energy is minimized by spreading the flow as much as possible. Astonishingly, these electrical network concepts are very related to random walks.

Claim 3.6.3 ([CRR+96])**.** *For any $s, t \in V(G)$, $2\mathcal{W}(G)\mathcal{R}_{s,t}(G)$ is the expected number of steps before a walker starting from s reaches t, and then returns to s – called the* commute time *from s to t.*

Exercise 3.6.1. *Is it possible for the commute time from s to t to differ from the commute time from t to s? Let $H_{s,t}(G)$ denote the hitting time from s to t, which is the expected number of steps needed for a walker starting from s to reach t. Is it possible for $H_{s,t}(G)$ to differ from $H_{t,s}(G)$?*

We can extend the definition of effective resistance in Definition 3.6.2 to starting from a distribution of vertices, and ending in a set of vertices.

Definition 3.6.4 (Effective Resistance)**.** *For any distribution σ on $V(G)$, and marked set $M \subseteq V(G)$,*

$$\mathcal{R}_{\sigma,M}(G) := \min\{\mathcal{E}(\theta) : \theta(u) = \sigma_u \forall u \in V(G) \setminus M\}.$$

Although there is no (known) general understanding of $2\mathcal{W}(G)\mathcal{R}_{\sigma,M}(G)$ as an expected stopping time, in the special case $\sigma = \pi$, we have the following.

Claim 3.6.5. $2\mathcal{W}(G)\mathcal{R}_{\pi,M}(G) = \mathcal{HT}(P, M)$.

3.7 Quantum Walk Search: Electric Network Framework

In this section, we will present a quantum walk framework that generalizes those presented in Section 3.5, by allowing the initial distribution to be any σ. Unlike the frameworks we saw in Section 3.5, the electric network framework does not represent a generic quadratic speedup over its classical counterpart, because $\boldsymbol{\mathcal{R}} \cdot \boldsymbol{\mathcal{W}}$ (see theorem statement below) is *not* the complexity of a classical random walk algorithm. However, the results in Claim 3.6.3 and Claim 3.6.5 give us some intuition on how this complexity might relate to the complexity of a classical random walk in some special cases. For example, in the special case $\sigma = \pi$, we recover Theorem 3.5.3.

Theorem 3.7.1 (Quantum Walk Search Electric Network Framework)**.** *Fix a graph G, initial distribution σ on $V(G)$, and marked set $M \subset V(G)$. Assume we have query access to σ, meaning we can query, for any $u \in V(G)$, σ_u, in unit cost. Suppose we can implement the following quantum subroutines:*

Setup: *Generate the state $|\sigma\rangle = \sum_{u \in V(G)} \sqrt{\sigma_u}|u\rangle$, in cost* S

Check: *For any $u \in V(G)$, check if $u \in M$, in cost* C

Update: **(1)** *For any $u \in V(G)$, generate the state $\sum_{v \in V(G)} \sqrt{P_{u,v}}|u, v\rangle$, in cost* U

(2) *Implement the vertex swap that acts, for all $u \in V(G)$ and $v \in \Gamma(u)$, as $|u, v\rangle \mapsto |v, u\rangle$, also in cost* U

Let $\boldsymbol{\mathcal{W}}$ be an upper bound on $\mathcal{W}(G)$, and $\boldsymbol{\mathcal{R}}$ an upper bound on $\mathcal{R}_{\sigma,M}(G)$ whenever $M \neq \emptyset$. Then there is a quantum algorithm that finds $u \in M$ (or determines $M = \emptyset$) with bounded error in complexity:

$$\widetilde{O}(\mathsf{S} + \sqrt{\boldsymbol{\mathcal{R}} \cdot \boldsymbol{\mathcal{W}}}(\mathsf{U} + \mathsf{C})).$$

Just as we can replace P with P^t in Theorem 3.5.3 to get an algorithm that only does one check for every t walk steps (Theorem 3.5.8), we can do the same here, replacing $\boldsymbol{\mathcal{R}}$ and $\boldsymbol{\mathcal{W}}$ with upper bounds on $\mathcal{R}_{\sigma,M}$ and $\mathcal{W}$ for P^t, which should generally be smaller, and replacing $\mathsf{U} + \mathsf{C}$ with $\sqrt{t}\mathsf{U} + \mathsf{C}$.

Example 3.7.2 (Element Distinctness)**.** Let everything be as in Example 3.5.2. As usual, we can assume there is a unique collision $x_a = x_b$ (or no collision). Let σ be the uniform distribution on vertices v_S such that $a, b \notin S$ (or simply the uniform distribution if there is no collision). We can sample from a distribution very close to this by simply sampling a uniform random S of size r. More formally, for the setup, we simply generate the state:

$$|\pi\rangle = \frac{1}{\sqrt{\binom{n}{r}}} \sum_{S \subset [n]:|S|=r} |v_S\rangle = \sqrt{\frac{\binom{n-2}{r}}{\binom{n}{r}}} \underbrace{\frac{1}{\sqrt{\binom{n-2}{r}}} \sum_{S \subset [n] \setminus \{a,b\}:|S|=r} |v_S\rangle}_{|\sigma\rangle} + \underbrace{\frac{1}{\sqrt{\binom{n}{r}}} \sum_{S \subset [n]:|S|=r, a \in S \vee b \in S} |v_S\rangle}_{|\tilde{\sigma}\rangle}$$

$$= \sqrt{\left(1 - \frac{r}{n}\right)\left(1 - \frac{r}{n-1}\right)}|\sigma\rangle + |\tilde{\sigma}\rangle$$

Since we have high overlap with the desired initial state $|\sigma\rangle$, the final state upon running the quantum walk algorithm will also have high overlap with the state we would get if we perfectly generated $|\sigma\rangle$. Thus, as in Example 3.5.2, we have $\mathsf{S} = r$.

The total weight of the graph $J(n, r)$ is:

$$\boldsymbol{\mathcal{W}} = |E| = |V|d = \binom{n}{r} r(n - r)$$

since it is d-regular and unweighted (i.e. all weights are 1). To upper bound $\mathcal{R}_{\sigma,M}(G)$, we will exhibit a σ-M flow, and upper bound its energy. We need only define the flow when there is a collision pair

(so M is non-empty), and as usual, we assume it is unique: $x_a = x_b$. Define

$$M_0 = \{v_S \in V : S \subset [n] \setminus \{a, b\}\}.$$

Then σ is just the uniform distribution on M_0. Define:

$$M_1 = \{v_S \in V : S \subset [n] \setminus \{b\}, a \in S\}.$$

Finally, define M_2 to be the remaining vertices, which contain both a and b, which is just M.

We will have non-zero flow on the following edges:

$$\forall v_S \in M_0, j \in S, \quad \theta(v_S, v_{S\setminus\{j\}\cup\{a\}}) = \frac{1}{|M_0|}\frac{1}{r}$$
$$\forall v_S \in M_1, j \in S \setminus \{a\}, \quad \theta(v_S, v_{S\setminus\{j\}\cup\{b\}}) = \frac{1}{|M_1|}\frac{1}{r-1}.$$

In words: beginning with uniform incoming flow to M_0, we spread it uniformly on edges from M_0 to M_1 – those that add a – resulting in a uniform flow on M_1, which we then spread uniformly on edges from M_1 to $M_2 = M$ – those that add b without removing a. You should verify that this is indeed a σ-M flow. The total energy of this flow is:

$$\begin{aligned}\boldsymbol{\mathcal{R}} &= \sum_{v_S \in M_0, j \in S} \theta(v_S, v_{S\setminus\{j\}\cup\{a\}})^2 + \sum_{v_S \in M_1, j \in S\setminus\{a\}} \theta(v_S, v_{S\setminus\{j\}\cup\{b\}})^2 \\ &= |M_0| r \left(\frac{1}{|M_0|}\frac{1}{r}\right)^2 + |M_1|(r-1)\left(\frac{1}{|M_1|}\frac{1}{r-1}\right)^2 = \frac{1}{|M_0|r} + \frac{1}{|M_1|(r-1)} = \frac{1}{\binom{n-2}{r}r} + \frac{1}{\binom{n-1}{r-1}(r-1)}.\end{aligned}$$

We then have:

$$\boldsymbol{\mathcal{R}} \cdot \boldsymbol{\mathcal{W}} = \frac{\binom{n}{r}r(n-r)}{\binom{n-2}{r}r} + \frac{\binom{n}{r}r(n-r)}{\binom{n-1}{r-1}(r-1)} = \Theta(n + n^2/r) = \Theta(n^2/r).$$

Thus, there is a quantum walk algorithm that solves element distinctness with query complexity (neglecting polylog factors):

$$\mathsf{S} + \sqrt{\boldsymbol{\mathcal{R}} \cdot \boldsymbol{\mathcal{W}}}(\mathsf{U} + \mathsf{C}) = r + \sqrt{n^2/r} = r + \frac{n}{\sqrt{r}}.$$

This is precisely what we get for element distinctness using the Szegedy or MNRS framework.

Example 3.7.3 (Backtracking)**.** A common approach to solving constraint satisfaction problems is to search a tree of the possible solutions using an approach called *backtracking*. Consider a binary tree in which the leaves represent the settings of the n variables (or partial settings that already fix the value of the formula φ). By trying settings of some variables, and then unsetting them ("backtracking" up the tree), one can search for a satisfying assignment. Using the electric network framework, this can be sped up quadratically [Mon18]. In that case, the initial distribution σ is supported only on the root of the backtracking tree.

Theorem 3.7.1 is proven in [AGJ20]. In the remainder of this section, we will prove a weaker version of the theorem that does not apply to the task of *finding* a marked vertex, but rather, deciding between the cases $M = \emptyset$, and $M \neq \emptyset$. It may seem strange to distinguish between the finding and deciding versions of quantum walk search, since it's difficult to imagine how we might be able to tell $M \neq \emptyset$ without finding an element of M. For the framework described in Theorem 3.5.3, a distinguishing algorithm was first presented in [Sze04], and it took over ten years for an algorithm that could also find a marked vertex in the claimed complexity [AGJK20]. Similarly, a deciding version of Theorem 3.7.1 (like the one we will prove) was first presented in [Bel13], before being later extended to the finding version we have stated.

Exercise 3.7.1 (Quantum walk on a line)**.** *Consider a setting where there is an initial state, $s = v_0$, and a final state $t = v_T$, and intermediate states $v_1, \ldots, v_{T-1}$, in which for any $i < T$, you can compute the state v_{i+1} from v_i, and for any $i > 0$, you can compute the state v_{i-1} from v_i. The task is to get from the initial state, which we assume is given, to the final state, which is not given, but which we can recognize when we see it. For example, this is exactly the setting of a reversible deterministic algorithm.*

We can model this setting as a random walk on a line, $s = v_0, v_1 \ldots, v_T = t$. However, it turns out that the hitting time from s to t – the expected number of steps for a random walker starting at s to reach t – is $\Theta(T^2)$, so getting from s to t via a random walk is quadratically worse than the more obvious algorithm of moving deterministically along the line from v_0 to v_1 to v_2, until t is reached.

Show that a quantum walk on this line can find t, starting from s in $\widetilde{O}(T)$ steps of the walk.

So whereas it is very sub-optimal to simulate a classical deterministic algorithm by a random walk on a line, it is not such a terrible idea to use a quantum walk – the complexity in the above exercise has polylog(T) overhead, but this overhead is actually not necessary. It is not clear why you would want to do this, but one reason might be that you want this line to represent a small part of a more complicated graph you are walking on. Note that we could not hope to do better than $\Theta(T)$ complexity for a quantum walk on a line. This would imply a generic quantum speedup for all deterministic reversible algorithms, which is too good to be true.

3.7.1 The Quantum Walk Algorithm

The random walk algorithms that we saw in Algorithm 1, Algorithm 2 and Algorithm 3 are not literally random walks: nobody is making steps on an actual graph. Instead, we use the structure of G to design an algorithm. Similarly, the quantum algorithm that proves our framework theorem is not literally going to involve walking on the graph, but instead will use G to design a quantum algorithm, which, we will see, can be analyzed based on the properties of G. The algorithm we're going to see will be a specific form: 0-phase estimation of a product of reflections, as in Theorem 2.2.9.

The algorithm will work on the span of *edges* of G: $\text{span}\{|e\rangle : e \in E(G)\}$. However, two remarks:

1. We will actually need to work on the span of edges of some *bipartite* graph, for reasons that will become clear. We will make G bipartite without significantly changing its random walk dynamics by simply putting a new vertex $v_{\{u,u'\}}$ in the middle of each edge $\{u, u'\}$ of G, and giving the two resulting edges weight $\mathsf{w}_{u,u'}$ (see Figure 3.2). Call the resulting graph $\overline{G}$. It has bipartition $V_{\mathcal{A}} = V(G), V_{\mathcal{B}} = \{v_e : e \in E(G)\}$.

2. We have some choice in how we encode the edges of $\overline{G}$. We will use the label (u, u') for the edge between u and $v_{\{u,u'\}}$, and (u', u) for the edge between u' and $v_{\{u,u'\}}$, and both of these edges will be assigned weight $\mathsf{w}_{u,u'}$.

Then we can define:

$$H_{\overline{G}} := \text{span}\{|u, u'\rangle : \{u, u'\} \in E(G)\} \equiv \text{span}\{|e\rangle : e \in E(\overline{G})\}. \tag{14}$$

We will now describe an algorithm that works for *any* bipartite graph $\overline{G}$, forgetting about its special structure for the most part (this is done to fully appreciate certain intuition that I think is useful). It will be useful to suppose that the edges of $\overline{G}$ have been assigned some arbitrary direction, by letting $\overrightarrow{E}(\overline{G})$ be a set that includes exactly one of (u, v) or (v, u) for each $\{u, v\} \in E(\overline{G})$. Since $\overline{G}$ is bipartite, a natural choice is to include $(u, v) \in V_{\mathcal{A}} \times V_{\mathcal{B}}$. We will use this choice, but other orientations might also make sense for a particular graph, and the choice does not matter technically. Then we can let

$$\{|e_{u,v}\rangle : (u, v) \in \overrightarrow{E}(\overline{G})\}$$

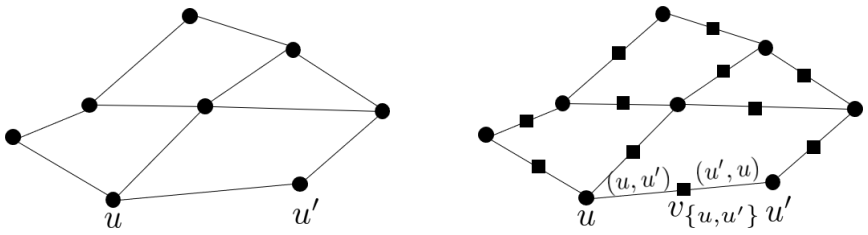

Figure 3.2: An example of a gaph G (left), and the modified $\overline{G}$ (right) with the new vertices shown as squares.

be the orthonormal basis of $H_{\overline{G}}$ we get by letting $|e_{u,v}\rangle = |\ell\rangle$ where ℓ is our chosen label for the edge $\{u, v\}$. For example, because of the way we have defined $\overline{G}$ from G, we can orient all edges from $V(G)$ outwards, so $\overrightarrow{E}(\overline{G}) = \{(u, v_{\{u,u'\}}) : \{u, u'\} \in E(G)\}$. Then $|e_{u,v_{\{u,u'\}}}\rangle := |u, u'\rangle$, and $|e_{u',v_{\{u,u'\}}}\rangle := |u', u\rangle$, and so

$$H_{\overline{G}} = \text{span}\{|e_{u,v}\rangle : (u, v) \in \overrightarrow{E}(\overline{G})\}$$

coincides with the definition of $H_{\overline{G}}$ above. Then, for any $(u, v) \in \overrightarrow{E}(\overline{G})$, $(v, u) \notin \overrightarrow{E}(\overline{G})$, so the notation $|e_{v,u}\rangle$ is not yet assigned. We will assign it as follows:

$$|e_{v,u}\rangle := -|e_{u,v}\rangle. \tag{3.1}$$

This lets us intuitively switch the direction of an edge by negating it. Although direction of edges shouldn't matter in an undirected graph, direction of *flow* is important, and we have already seen that $\theta(u, v) = -\theta(v, u)$ for any flow θ.

We will modify the graph one final time. Let $\overline{G}'$ be like $\overline{G}$ but with a new vertex v_0 (see also Figure 3.3):

$$V(\overline{G}') = V(\overline{G}) \cup \{v_0\} \qquad \overrightarrow{E}(\overline{G}') = \overrightarrow{E}(\overline{G}) \cup \{(u, v_0) : u \in \text{supp}(\sigma) \cup M\}$$

$$\mathsf{w}_{u,v_0} = \begin{cases} \sigma_u \mathsf{w}_0 & \forall u \in \text{supp}(\sigma) \\ \mathsf{w}_M & \forall u \in M, \end{cases}$$

where $\text{supp}(\sigma) = \{u : \sigma_u \neq 0\}$.

We have connected v_0 to every vertex in M by an edge of weight w_M, a weight to be set later, and to every vertex u in $\text{supp}(\sigma)$ by an edge of weight $\sigma_u \mathsf{w}_0$ for some scaling factor w_0 to be set later. The graph $\overline{G}'$ might not be bipartite, but that doesn't matter. The final space that the algorithm will actually work in is:

$$H = H_{\overline{G}'} = H_{\overline{G}} \oplus \text{span}\{|u, v_0\rangle : u \in M \cup \text{supp}(\sigma)\},$$

where we have implicity made the choice that $|e_{u,v_0}\rangle = |u, v_0\rangle$ and $|e_{v_0,u}\rangle = -|u, v_0\rangle$.

In the remainder of the section, it will be convenient to assume that

$$\text{supp}(\sigma) \subseteq V_{\mathcal{A}} \text{ and } M \subseteq V_{\mathcal{B}}.$$

Our construction of $\overline{G}$ ensures the first assumption, but does not satisfy the second. However, we can satisfy the second by simply redefining M as those $v_{\{u,u'\}}$ such that either u or u' is marked.

Define *star states* for the graph $\overline{G}'$ as follows:

$$\forall u \in V(\overline{G}),\ |\psi_\star^{\overline{G}'}(u)\rangle := \sum_{v \in \Gamma_{\overline{G}'}(u)} \sqrt{\mathsf{w}_{u,v}} |e_{u,v}\rangle.$$

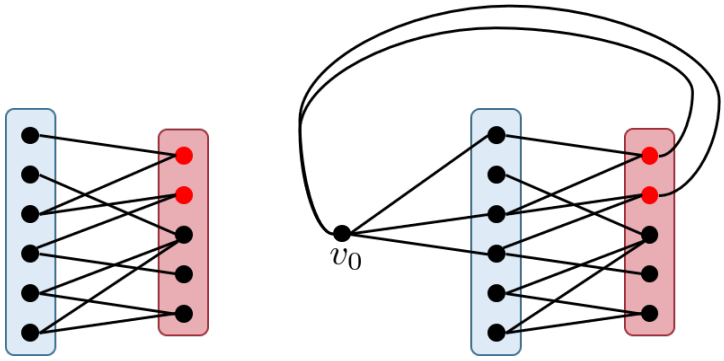

Figure 3.3: A bipartite graph G (left, not necessarily of the form $\overline{G}$); and its modified version G' (right), where we have added a vertex v_0 with edges going to the support of σ, which we assume to be in the blue part of the bipartition ($V_{\mathcal{A}}$), and M, shown in red, which we assume to be in the red part of the bipartition ($V_{\mathcal{B}}$). The edges going from v_0 to M all have some weight w_{M}, whereas the edges from v_0 to the support of σ have weights proportional to σ. Note that v_0 is not part of $V_{\mathcal{A}}$, nor $V_{\mathcal{B}}$, and G' is not necessarily bipartite.

Then for $u \in V(\overline{G}) \setminus (M \cup \mathrm{supp}(\sigma))$, $\Gamma_{\overline{G}'}(u) = \Gamma_{\overline{G}}(u)$, so

$$|\psi_\star^{\overline{G}'}(u)\rangle = \sum_{v \in \Gamma_{\overline{G}}(u)} \sqrt{\mathsf{w}_{u,v}} |e_{u,v}\rangle =: |\psi_\star^{\overline{G}}(u)\rangle.$$

For $u \in M$,

$$|\psi_\star^{\overline{G}'}(u)\rangle = \underbrace{\sum_{v \in \Gamma_{\overline{G}}(u)} \sqrt{\mathsf{w}_{u,v}} |e_{u,v}\rangle}_{=:|\psi_\star^{\overline{G}}(u)\rangle} + \sqrt{\mathsf{w}_{\mathsf{M}}} |u, v_0\rangle$$

and finally, for $u \in \mathrm{supp}(\sigma)$,

$$|\psi_\star^{\overline{G}'}(u)\rangle = \underbrace{\sum_{v \in \Gamma_{\overline{G}}(u)} \sqrt{\mathsf{w}_{u,v}} |e_{u,v}\rangle}_{=:|\psi_\star^{\overline{G}}(u)\rangle} + \sqrt{\sigma_u \mathsf{w}_0} |u, v_0\rangle.$$

[17]

The star states are generally not normalized, but they are proportional to states $\sum_{v \in \Gamma_{\overline{G}}(u)} \sqrt{\overline{P}'_{u,v}} |e_{u,v}\rangle$, which are similar to those we assume we can generate as part of our update (see statement of Theorem 3.7.1). We use these states to define the two spaces that will be parameters of the phase estimation algorithm.

Let $\Psi_{\mathcal{A}} := \{|\psi_\star^{\overline{G}'}(u)\rangle : u \in V_{\mathcal{A}}\}$, and $\mathcal{A} = \mathrm{span}\{\Psi_{\mathcal{A}}\} \subset H$.
Let $\Psi_{\mathcal{B}} := \{|\psi_\star^{\overline{G}'}(u)\rangle : u \in V_{\mathcal{B}}\}$, and $\mathcal{B} = \mathrm{span}\{\Psi_{\mathcal{B}}\} \subset H$. **[18]**

Note that the states in $\Psi_{\mathcal{A}}$ are pairwise orthogonal, as are the states in $\Psi_{\mathcal{B}}$. This is crucial in allowing us to implement the reflections around $\mathcal{A}$ and $\mathcal{B}$ respectively in Lemma 3.7.4 below, and is only true because each of $V_{\mathcal{A}}$ and $V_{\mathcal{B}}$ is an independent set[1] (i.e. $\overline{G}$ is bipartite). This is why we need to work with a bipartite graph. Referring to Definition 2.2.3, the final parameter we need for a phase estimation algorithm is an initial state, and we will use

[1] An independent set of vertices is a set of vertices such that no two vertices in the set share an edge.

$|\psi_0\rangle := \sum_{u \in V_{\mathcal{A}}} \sqrt{\sigma_u}|u, v_0\rangle$, which is in $\mathcal{B}^{\perp}$, as needed.

Note that $V_{\mathcal{A}}$ contains the full support of σ, by assumption.

Then the algorithm will simply be a 0-Phase Estimation Algorithm (Theorem 2.2.9), in which we do phase estimation on the unitary

$$U = (2\Pi_{\mathcal{A}} - I)(2\Pi_{\mathcal{B}} - I), \tag{3.2}$$

with initial state $|\psi_0\rangle$, and check if we measure a 0 in the phase register.

3.7.2 Analysis of the Algorithm

Referring to Theorem 2.2.9, we can see that we need to analyze the complexity of generating $|\psi_0\rangle$ and implementing U; and we need to exhibit a positive witness whenever $M \neq \emptyset$, and a negative witness whenever $M = \emptyset$, and use the properties of these witnesses to set parameters c_+ and $\mathcal{C}_-$.

Note that the complexity of generating

$$|\psi_0\rangle = \sum_{u \in V_{\mathcal{A}}} \sqrt{\sigma_u}|u, v_0\rangle = \left(\sum_{u \in V(G)} \sqrt{\sigma_u}|u\rangle \right) |v_0\rangle \tag{3.3}$$

is just S, by its definition in the theorem statement. We now show that we can implement U using the update and checking subroutines referred to in the theorem statement.

Lemma 3.7.4. *Assuming we can query σ and the vertex weights w_u (which are proportional to π) in unit time, $U = (2\Pi_{\mathcal{A}} - I)(2\Pi_{\mathcal{B}} - I)$ can be implemented in complexity $O(\mathsf{U} + \mathsf{C})$.*

Proof Sketch: To implement $(2\Pi_{\mathcal{A}} - I)$, it's enough to be able to generate normalizations of the states in $\Psi_{\mathcal{A}}$, by implementing a unitary $U_{\mathcal{A}}$ that acts, for $u \in V_{\mathcal{A}}$, as

$$|u, 0\rangle \mapsto \frac{|\psi_\star^{\overline{G}'}(u)\rangle}{\left\| |\psi_\star^{\overline{G}'}(u)\rangle \right\|},$$

which is only possible because the states in $\Psi_{\mathcal{A}}$ are pairwise orthogonal. Then

$$(2\Pi_{\mathcal{A}} - I) = U_{\mathcal{A}} \left(2 \sum_{u \in V_{\mathcal{A}}} |u, 0\rangle\langle u, 0| - I \right) U_{\mathcal{A}}^{\dagger}.$$

We describe how to implement $U_{\mathcal{A}}$. Recall that $V_{\mathcal{A}} = V(G)$, with neighbours in $V_{\mathcal{B}}$ of the form $v_{\{u,u'\}}$ such that $\{u, u'\} \in E(G)$. Because of the way we have chosen to encode edges $\{u, v_{\{u,u'\}}\}$, whose weight is $\mathsf{w}_{u,u'}$, we have $|e_{u,v_{\{u,u'\}}}\rangle = |u, u'\rangle$. Thus, for all $u \in V_{\mathcal{A}}$, we have:

$$|\psi_\star^{\overline{G}'}(u)\rangle = \sum_{u' \in \Gamma_G(u)} \sqrt{\mathsf{w}_{u,u'}}|u, u'\rangle + \sqrt{\sigma_u \mathsf{w}_0}|u, v_0\rangle.$$

(Recall that we are assuming $M \subseteq V_{\mathcal{B}}$.) The first term is proportional to the state $\sum_{u' \in \Gamma_G(u)} \sqrt{P_{u,u'}}|u, u'\rangle$ that we assume we can generate in cost U. Thus, we can generate the required state by querying σ_u and w_u, doing a single-qubit rotation to get a state proportional to $\sqrt{\mathsf{w}_u}|0\rangle + \sqrt{\sigma_u \mathsf{w}_0}|1\rangle$, uncomputing σ_u and w_u, and then mapping the $|0\rangle$ part of the state to the appropriate first term, and the $|1\rangle$ part to $|v_0\rangle$.

Implementing $(2\Pi_{\mathcal{B}} - I)$ is done similarly. To implement a similar map $U_{\mathcal{B}}$, we need to generate states of the form

$$|\psi_\star^{\overline{G}}(v)\rangle + \delta_{v,M}\sqrt{\mathsf{w}_{\mathsf{M}}}|v, v_0\rangle,$$

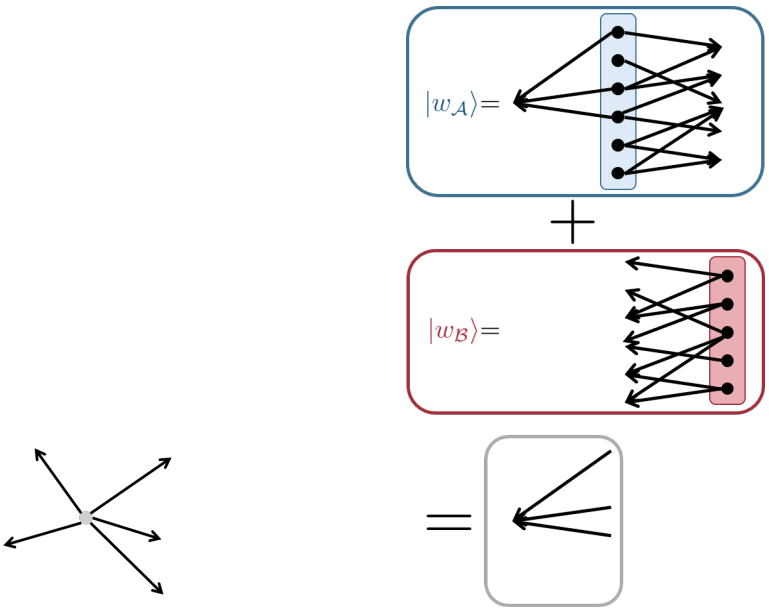

Figure 3.4: Left: Think of a star state as a star graph with edges coming out of some center vertex. Right: If $|w_{\mathcal{A}}\rangle$ is the sum of all star states for $u \in V_{\mathcal{A}}$ (shown in blue), so it's the sum of all edges coming *out of* vertices in $V_{\mathcal{A}}$; and $|w_{\mathcal{B}}\rangle$ is the sum of all star states for $u \in V_{\mathcal{B}}$, so it's the sum of all edges coming *out of* vertices in $V_{\mathcal{B}}$, then when we add these together, all edges between $V_{\mathcal{A}}$ and $V_{\mathcal{B}}$ will appear in both directions, which will cancel, leaving only the edges going into v_0, which is the only vertex not part of $V_{\mathcal{A}}$ or $V_{\mathcal{B}}$. When $M = \emptyset$, this gives a state proportional to $|\psi_0\rangle$.

for $v \in V_{\mathcal{B}}$, where $\delta_{v,M} = 1$ if v is marked and 0 otherwise. Recall that $V_{\mathcal{B}} = \{v_{\{u,u'\}} : \{u, u'\} \in E(G)\}$, and $v_{\{u,u'\}} \in M$ if and only if u or u' is marked, which can be checked in cost $O(\mathsf{C})$. In Exercise 3.7.2, you will show how to generate $|\psi_\star^{\overline{G}}(v)\rangle$ for $v \in V_{\mathcal{B}}$ in cost $O(\mathsf{U})$, giving a total cost $O(\mathsf{U} + \mathsf{C})$ for implementing $U_{\mathcal{B}}$. □

Exercise 3.7.2. *Show how to implement a unitary $U_{\mathcal{B}}$ that acts, for $v \in V_{\mathcal{B}}$, as*

$$|v\rangle \mapsto |\psi_\star^{\overline{G}}(v)\rangle / \left\| |\psi_\star^{\overline{G}}(v)\rangle \right\|$$

in cost $O(\mathsf{U})$. Hint: You may assume $|v_{\{u,u'\}}\rangle$ is encoded as $|u, u'\rangle$, where $u < u'$.

Next, we exhibit a negative witness (see Definition 2.2.6) whenever $M = \emptyset$. This shows that when $M = \emptyset$, $|\psi_0\rangle \in \mathcal{A} + \mathcal{B} = (\mathcal{A}^\perp \cap \mathcal{B}^\perp)^\perp$, and therefore, since $|\psi_0\rangle \in \mathcal{B}^\perp$, it is orthogonal to the $(+1)$-eigenspace of U, which is just the direct sum of $\mathcal{A} \cap \mathcal{B}$ and $\mathcal{A}^\perp \cap \mathcal{B}^\perp = (\mathcal{A} + \mathcal{B})^\perp$.

Lemma 3.7.5 (Negative Witness)**.** *Suppose $M = \emptyset$. Then there is a negative witness $|w_{\mathcal{A}}\rangle, |w_{\mathcal{B}}\rangle$ with $\||w_{\mathcal{A}}\rangle\|^2 \leq \frac{1}{\mathsf{w}_0}\mathcal{W}(\overline{G}) + 1 =: \tilde{\mathcal{C}}_-$.*

Proof. We first explain our approach in words and pictures. We are going to let $|w'_{\mathcal{A}}\rangle \in \mathcal{A}$ be the sum of all star states $|\psi_\star^{\overline{G}'}(u)\rangle$ for $u \in V_{\mathcal{A}}$, and similarly for $|w'_{\mathcal{B}}\rangle \in \mathcal{B}$. Then $|w'_{\mathcal{A}}\rangle + |w'_{\mathcal{B}}\rangle$ will be the sum of all star states for $u \in V(\overline{G}') \setminus \{v_0\}$. Picture a star state as a vertex with a bunch of edges coming out of it in an outward orientation (see Figure 3.4). If we add all star states together, each edge $\{u, v\}$ not touching v_0 occurs twice: once coming out of u in $|\psi_\star(u)\rangle$ as $|e_{u,v}\rangle$, and once in the opposite direction, coming out of v in $|\psi_\star(u)\rangle$ as $|e_{v,u}\rangle$. These two edges are in opposite direction, so they cancel out: $|e_{u,v}\rangle + |e_{v,u}\rangle = 0$ (see (3.1)). Thus, all that remains are the edges adjacent to v_0, pointing towards v_0 (see Figure 3.4).

Mathematically, this looks like:

$$\begin{aligned}
&\underbrace{\sum_{u\in V_{\mathcal{A}}}\sum_{v\in\Gamma_{\overline{G}'}(u)}\sqrt{\mathsf{w}_{u,v}}|e_{u,v}\rangle}_{|w'_{\mathcal{A}}\rangle}+\underbrace{\sum_{v\in V_{\mathcal{B}}}\sum_{u\in\Gamma_{\overline{G}'}(v)}\sqrt{\mathsf{w}_{u,v}}|e_{v,u}\rangle}_{|w'_{\mathcal{B}}\rangle}\\
&=\sum_{\substack{u\in V_{\mathcal{A}}\\ v\in\Gamma_{\overline{G}}(u)}}\sqrt{\mathsf{w}_{u,v}}|e_{u,v}\rangle+\sum_{u\in\Gamma(v_0)\cap V_{\mathcal{A}}}\sqrt{\mathsf{w}_{u,v_0}}|e_{u,v_0}\rangle+\sum_{\substack{v\in V_{\mathcal{B}}\\ u\in\Gamma_{\overline{G}}(v)}}\sqrt{\mathsf{w}_{u,v}}|e_{v,u}\rangle+\sum_{v\in\Gamma(v_0)\cap V_{\mathcal{B}}}\sqrt{\mathsf{w}_{v_0,v}}|e_{v,v_0}\rangle\\
&=\sum_{\{u,v\}\in E(\overline{G})}\sqrt{\mathsf{w}_{u,v}}(\underbrace{|e_{u,v}\rangle+|e_{v,u}\rangle}_{=0})+\sum_{u\in\Gamma(v_0)}\sqrt{\mathsf{w}_{u,v_0}}|e_{u,v_0}\rangle.
\end{aligned}$$

If $M=\emptyset$, then $\Gamma(v_0)=\mathrm{supp}(\sigma)$ and this is just

$$\sum_{u\in\mathrm{supp}(\sigma)}\sqrt{\mathsf{w}_{u,v_0}}|e_{u,v_0}\rangle=\sum_{u\in\mathrm{supp}(\sigma)}\sqrt{\sigma_u\mathsf{w}_0}|u,v_0\rangle=\sqrt{\mathsf{w}_0}\underbrace{\sum_{u\in\mathrm{supp}(\sigma)}\sqrt{\sigma_u}|u,v_0\rangle}_{=|\psi_0\rangle},\tag{3.4}$$

so we have shown that $|\psi_0\rangle\in\mathcal{A}+\mathcal{B}$. To get a negative witness (see Definition 2.2.6), we just set $|w_{\mathcal{A}}\rangle=\frac{1}{\sqrt{\mathsf{w}_0}}|w'_{\mathcal{A}}\rangle$ and $|w_{\mathcal{B}}\rangle=\frac{1}{\sqrt{\mathsf{w}_0}}|w'_{\mathcal{B}}\rangle$. We complete the proof by noticing that (see Exercise 3.7.3):

$$\||w_{\mathcal{A}}\rangle\|^2\le\frac{1}{\mathsf{w}_0}\mathcal{W}(\overline{G})+1.$$ □

Exercise 3.7.3. *Show that $|w_{\mathcal{A}}\rangle$ (defined in the proof of Lemma 3.7.5) satsifies $\||w_{\mathcal{A}}\rangle\|^2\le\frac{1}{\mathsf{w}_0}\mathcal{W}(\overline{G})+1$.*

Finally, we exhibit a positive witness (see Definition 2.2.4) whenever $M\neq\emptyset$, which is precisely a component of $|\psi_0\rangle$ in the $(+1)$-eigenspace of U.

Lemma 3.7.6 (Positive Witness)**.** *Fix $\mathsf{w}_{\mathsf{M}}\ge 1$. If $M\neq\emptyset$, then there exists a positive witness $|w\rangle$ such that*

$$\frac{|\langle w|\psi_0\rangle|^2}{\||w\rangle\|^2}\ge\frac{1}{\mathsf{w}_0(\mathcal{R}_{\sigma,M}(\overline{G})+1)+1}=:\frac{1}{c_+},$$

where Λ_0 is the orthogonal projector onto the $(+1)$-eigenspace of U.

Proof. A positive witness $|w\rangle$ is in $\mathcal{A}^\perp\cap\mathcal{B}^\perp$, so it is orthogonal to all star states. Let us think about what is orthogonal to all star states using our graphical interpretation (see Figure 3.4). If $|w\rangle$ includes an edge $|e_{u,v}\rangle$, which we can visualize as an arrow going from u to v, then this overlaps the $|e_{u,v}\rangle$ term in $|\psi_\star(u)\rangle$, contributing 1 to $\langle w|\psi_\star(u)\rangle$ (assuming for simplicity that all weights are 1). We can ensure orthogonality by letting $|w\rangle$ also include $|e_{v',u}\rangle$ for some other v' in $\Gamma(u)$, which contributes a further $\langle e_{v',u}|e_{u,v'}\rangle=-1$ (see (3.1)). Thus, we can ensure orthogonality by ensuring that when $|w\rangle$ has an edge coming out of a vertex u (to some v), it also has an edge going into u (from some v') – in other words, a *cycle* is orthogonal to all star states, because whenever it has an incoming edge to a vertex, it also has an outgoing edge (see Figure 3.5).

More generally, linear combinations of cycles, which are precisely *circulations* (see Definition 3.6.1), are orthogonal to star states: for all the amplitude (flow) that leaves a vertex u, the same amount should come into u, ensuring orthogonality with $|\psi_\star(u)\rangle$. So to make a positive witness, we will make a circulation that overlaps $|\psi_0\rangle$. We will construct this circulation by starting with a σ-M flow, which is a flow (see Definition 3.6.1) θ that has $\sigma_u=\theta(u)$ for all $u\notin M$.

Assuming $M\neq\emptyset$ (and $\overline{G}$ is connected) there exists a σ-M flow on $\overline{G}$. Let θ be the σ-M flow with smallest possible energy, $\mathcal{R}_{\sigma,M}(\overline{G})$ (see Definition 3.6.1). Note that this flow is just used for the analysis, it is not necessary for the algorithm to be able to compute it at any point. By definition, θ is a real-valued function on $\{(u,v):\{u,v\}\in E(\overline{G})\}$. We extend it to also be defined on the edges (u,v_0) as follows:

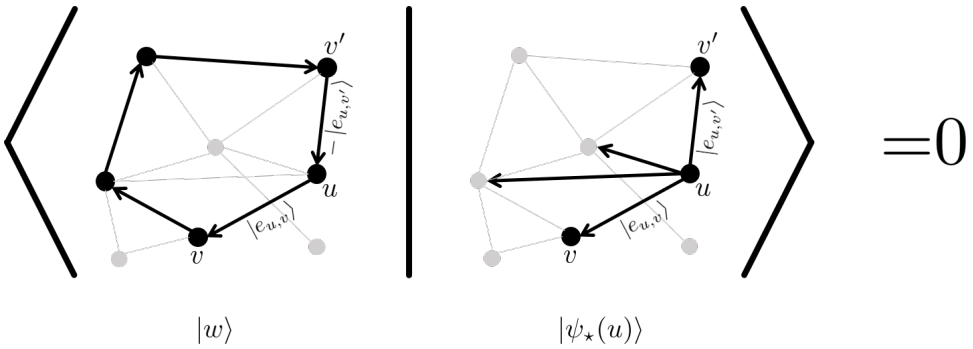

Figure 3.5: A cycle is orthogonal to all star states, so it's in $\mathcal{A}^\perp \cap \mathcal{B}^\perp$.

For $u \in \mathrm{supp}(\sigma)$, let $\theta(v_0, u) = -\theta(u, v_0) = \sigma_u$
For $u \in M$, let $\theta(u, v_0) = -\theta(v_0, u) = \sum_{v \in \Gamma_{\overline{G}}(u)} \theta(v, u) =: \theta(u)$.

[20]

In other words, we have extended θ from a σ-M flow on $\overline{G}$, to a circulation on $\overline{G}'$ by sending all excess flow on vertices in M to v_0 (this necessarily totals 1), and then sending that flow out to the support of σ in strength according to σ, ensuring that all vertices have the same amount of flow coming in as out. For example, if σ is just the distribution supported on a single vertex s, and $M = \{t\}$, then θ is an st-flow like the one in Figure 3.1, and we can extend it to a circulation on the graph G' (in which s and t are both connected to an additional vertex v_0) by sending the flow coming out of t through v_0 and back into s. From this, we define:

$$|w\rangle = \sum_{(u,v) \in \vec{E}(\overline{G}')} \frac{\theta(u,v)}{\sqrt{\mathsf{w}_{u,v}}} |e_{u,v}\rangle. \tag{3.5}$$

Then for all $(u, v) \in \vec{E}(\overline{G}')$, $\langle e_{u,v}|w\rangle = \frac{\theta(u,v)}{\sqrt{\mathsf{w}_{u,v}}}$, but similarly,

$$\langle e_{v,u}|w\rangle = -\langle e_{u,v}|w\rangle = -\frac{\theta(u,v)}{\sqrt{\mathsf{w}_{u,v}}} = \frac{\theta(v,u)}{\sqrt{\mathsf{w}_{v,u}}}.$$

The $1/\sqrt{\mathsf{w}_{u,v}}$ is to ensure that for any $u \in V(\overline{G})$,

$$\langle w|\psi_\star^{\overline{G}'}(u)\rangle = \sum_{v \in \Gamma_{\overline{G}'}(u)} \left(\frac{\theta(u,v)}{\sqrt{\mathsf{w}_{u,v}}} \langle e_{u,v}| \right) \sqrt{\mathsf{w}_{u,v}} |e_{u,v}\rangle = \sum_{v \in \Gamma_{\overline{G}'}(u)} \theta(u,v) = -\theta(u). \tag{3.6}$$

In Exercise 3.7.4, you will show that $|w\rangle \in \mathcal{A}^\perp \cap \mathcal{B}^\perp$, which is equivalent to proving that θ is indeed a circulation on $\overline{G}'$, since it is exactly proving that $\theta(u) = 0$ for all u. We also have (see (3.3)):

$$\langle w|\psi_0\rangle = \sum_{u \in V_\mathcal{A}} \sqrt{\sigma_u} \langle w|e_{u,v_0}\rangle = \sum_{u \in V_\mathcal{A}} \sqrt{\sigma_u} \frac{\theta(u,v_0)}{\sqrt{\mathsf{w}_{u,v_0}}} = \sum_{u \in V_\mathcal{A}} \sqrt{\sigma_u} \frac{-\sigma_u}{\sqrt{\mathsf{w}_0 \sigma_u}} = -\frac{1}{\sqrt{\mathsf{w}_0}} \tag{3.7}$$

which means that $|w\rangle$ is a positive witness, as desired. To complete the proof, we note that

$$\begin{aligned}
\||w\rangle\|^2 &= \sum_{(u,v)\in\overrightarrow{E}(\overline{G}')} \frac{\theta(u,v)^2}{\mathsf{w}_{u,v}} \\
&= \sum_{(u,v)\in\overrightarrow{E}(\overline{G})} \frac{\theta(u,v)^2}{\mathsf{w}_{u,v}} + \sum_{u\in\mathrm{supp}(\sigma)} \frac{\theta(u,v_0)^2}{\mathsf{w}_{u,v_0}} + \sum_{u\in M} \frac{\theta(u,v_0)^2}{\mathsf{w}_{u,v_0}} \\
&= \mathcal{R}_{\sigma,M}(\overline{G}) + \sum_{u\in\mathrm{supp}(\sigma)} \frac{\sigma_u^2}{\mathsf{w}_0\sigma_u} + \sum_{u\in M} \frac{\theta(u)^2}{\mathsf{w}_{\mathsf{M}}} \\
&\leq \mathcal{R}_{\sigma,M}(\overline{G}) + \frac{1}{\mathsf{w}_0} + 1.
\end{aligned} \tag{3.8}$$

Above we used the fact that θ has minimal energy $\mathcal{R}_{\sigma,M}(\overline{G})$ as a σ-M flow in $\overline{G}$ (see Definition 3.6.1), and $\mathsf{w}_{\mathsf{M}} \geq 1$. We also used the fact that $\sum_{u\in M}\theta(u)^2 \leq 1$. We can only assume that because θ has minimal energy. □

Exercise 3.7.4. *Let $|w\rangle$ be as in the proof of Lemma 3.7.6. Show that $|w\rangle \in \mathcal{A}^\perp \cap \mathcal{B}^\perp$.*

Since we need c_+ to be a constant, we let

$\mathsf{w}_0 = \frac{1}{\mathcal{R}_{\sigma,M}(\overline{G})+1}$ so that $c_+ = 2$.
Then $\tilde{\mathcal{C}}_- = \frac{1}{\mathsf{w}_0}\mathcal{W}(\overline{G}) + 1 = \mathcal{R}_{\sigma,M}(\overline{G})\mathcal{W}(\overline{G}) + \mathcal{W}(\overline{G}) + 1 \leq 3\mathcal{R}_{\sigma,M}(\overline{G})\mathcal{W}(\overline{G})$.

Thus, by Theorem 2.2.9, our algorithm detects the presence of a marked vertex with bounded error in complexity:

$$O(\mathsf{S} + \sqrt{\mathcal{C}_-}(\mathsf{U} + \mathsf{C}))$$

as long as $\mathcal{C}_-$ is a known upper bound on $3\mathcal{R}_{\sigma,M}(\overline{G})\mathcal{W}(\overline{G})$. However, we wanted the complexity in terms of the original graph G. To complete the proof we merely show that modifying G to $\overline{G}$ has minimal impact on $\mathcal{W}$ and $\mathcal{R}$.

Exercise 3.7.5. *Let $\overline{G}$ be constructed from G as described in Section 3.7.1. Then $\mathcal{W}(\overline{G}) = 2\mathcal{W}(G)$, and for any distribution σ on G and marked set $M \subset V(G)$, if we let $M' = \{v_{\{u,u'\}} \in V(\overline{G}) : u \in M \text{ or } u' \in M\}$, then $\mathcal{R}_{\sigma,M'}(\overline{G}) \leq 2\mathcal{R}_{\sigma,M}(G)$.*

Thus, we have

$$\tilde{\mathcal{C}}_- \leq 3(2\mathcal{R}_{\sigma,M}(G))(2\mathcal{W}(G)) \leq 12\boldsymbol{\mathcal{W}}\cdot\boldsymbol{\mathcal{R}} =: \mathcal{C}_-$$

Applying Theorem 2.2.9 with $c_+ = \frac{1}{2}$ and $\mathcal{C}_-$ as above thus gives an algorithm for deciding between the following two cases:

Positive Case: If $M = \emptyset$ then there is a positive witness $|w\rangle$ with $\frac{\||w\rangle\|^2}{|\langle w|\psi_0\rangle|^2} \leq c_+$ (Lemma 3.7.6).

Negative Case: If $M \neq \emptyset$ then there is a negative witness $|w_{\mathcal{A}}\rangle, |w_{\mathcal{B}}\rangle$ with $\||w_{\mathcal{A}}\rangle\|^2 \leq \mathcal{C}_-$ (Lemma 3.7.5).

The cost of this algorithm is

$$O(\mathsf{S} + \sqrt{\mathcal{C}_-}(\mathsf{U} + \mathsf{C})) = O(\mathsf{S} + \sqrt{\boldsymbol{\mathcal{W}}\cdot\boldsymbol{\mathcal{R}}}(\mathsf{U} + \mathsf{C})).$$

3.7.3 Further Directions

Our algorithm was based on a product of two reflections, $U = (2\Pi_{\mathcal{A}} - I)(2\Pi_{\mathcal{B}} - I)$, that depends on a choice of graph G, initial distribution σ, and marked set M. We built an algorithm from U by doing

phase estimation, as in Theorem 2.2.9, which involves calling (controlled) U a total of $O(\sqrt{\boldsymbol{\mathcal{R}} \cdot \boldsymbol{\mathcal{W}}})$ times. This is not the only way to build an algorithm for deciding if $M = \emptyset$ using $O(\sqrt{\boldsymbol{\mathcal{R}} \cdot \boldsymbol{\mathcal{W}}})$ calls to U. In fact, the simple algorithm that applies U^T to $|\sigma\rangle$ for some uniform $T \leq \sqrt{\boldsymbol{\mathcal{R}} \cdot \boldsymbol{\mathcal{W}}}$ has success probability $\Omega(1/\log(\boldsymbol{\mathcal{R}} \cdot \boldsymbol{\mathcal{W}}))$, not only for *deciding* if $M = \emptyset$, but for the more difficult task of *finding* an element $u \in M$ if one exists [AGJ20]. This algorithm, while much simpler than the one shown here, is much more difficult to analyze.

We saw in Theorem 3.5.3 that it is pssible to find a marked vertex using only $\sqrt{\boldsymbol{\mathcal{H}}\boldsymbol{\mathcal{T}}}$ updates (steps of the walk), and this turns out to be optimal for general graphs. However, this algorithm also uses $\sqrt{\boldsymbol{\mathcal{H}}\boldsymbol{\mathcal{T}}}$ checks, which is not optimal – we saw in Theorem 3.5.6 that $\frac{1}{\sqrt{\varepsilon}}$ checks are sufficient (it turns out also necessary), but in this case, we need $\frac{1}{\sqrt{\varepsilon\delta}}$ updates, which might be larger than $\sqrt{\boldsymbol{\mathcal{H}}\boldsymbol{\mathcal{T}}}$. An interesting question is whether we can achieve the optimal number of updates and optimal number of checks at the same time. It turns out that we can, at least in the special case of a single marked vertex [DH17], which we can find in cost

$$\mathsf{S} + \sqrt{\boldsymbol{\mathcal{H}}\boldsymbol{\mathcal{T}}}\mathsf{U} + \frac{1}{\sqrt{\varepsilon}}\mathsf{C}.$$

Sometimes some steps of a quantum walk cost more than others. Meaning the cost U actually varies depending where you are in the graph. In classical random walks, if some steps are expensive and some are very cheap, this cost should average out. This is much less obvious for quantum walks, but it turns out that something like this also happens [Jef22]. A special case of this that's useful to know about is something called *variable-time quantum search* [Amb10]. Suppose you want to find some $i \in \{0,1\}^N$ such that $f(i) = 1$, and suppose that computing $f(i)$ costs T_i. Standard Grover's algorithm would find such an i in complexity $O(\sqrt{N}\max_i T_i)$, whereas a more clever quantum algorithm can find such an i in cost

$$\widetilde{O}\left(\sqrt{\sum_{i\in[N]} T_i^2}\right) = \widetilde{O}\left(\sqrt{N}\sqrt{\frac{1}{N}\sum_{i\in[N]} T_i^2}\right).$$

This can be a significant improvement if the costs vary a lot.

As mentioned, sometimes the terminology "discrete-time quantum walk" is used to refer to algorithms based on *any* product of reflections, whether or not they are based on an underlying graph. As we will see in the next section, this includes a model called *span programs*, which are a completely general way of expressing quantum algorithms.

3.8 Continuous-Time Quantum Walks and Exponential Speedups

A continuous-time quantum walk is obtained by implementing the unitary $U(t) = e^{itL}$ whose *Hamiltonian*, L, is the Laplacian of a graph G, defined, in the case of an unweighted graph:

$$L_{u,v} = \begin{cases} 1 & \text{if } \{u,v\} \in E(G) \\ -d(u) & \text{if } u = v \\ 0 & \text{else,} \end{cases}$$

where $d(u) = |\Gamma(u)|$ is the degree of u. If G is a regular graph, meaning every vertex has the same degree d, then $L = d(I - P)$, where P is the transition matrix of the random walk on P. We will not say much about continuous-time quantum walks, but they are important for context, since they were the first model of quantum walk considered.

Perhaps the most important example of a continuous-time quantum walk is the quantum walk for the *welded trees* problem, which we will define shortly. This is important because while the quantum algorithm has complexity $\text{poly}(n)$, there is a classical lower bound of $2^{\Omega(n)}$, so this is a rare example of an exponential quantum speedup over classical algorithms.

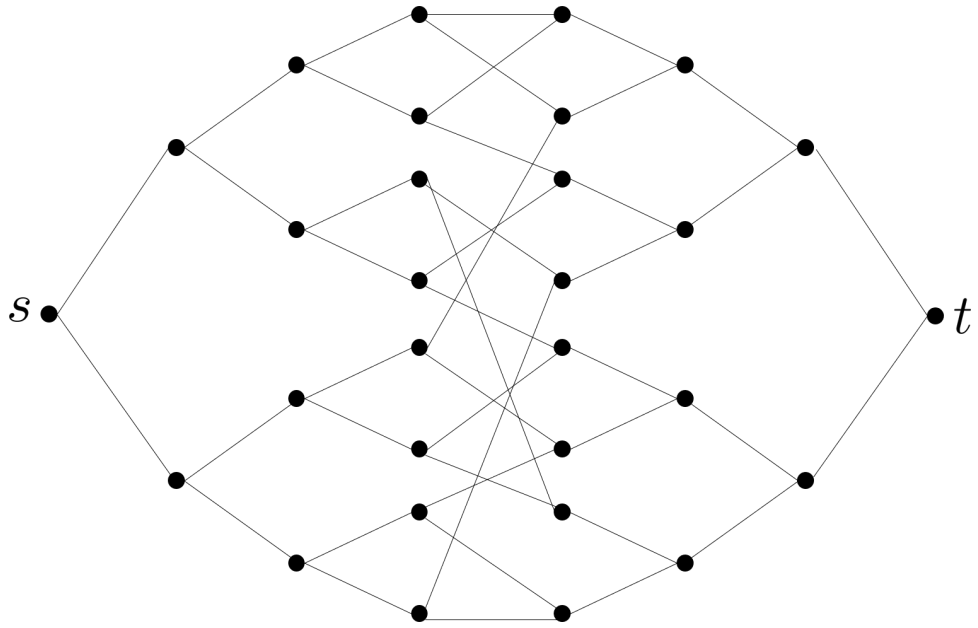

Figure 3.6: A welded trees graph.

A *welded trees graph*, G, has $V(G) \subset \{0,1\}^{2n}$ with $|V(G)| = 2^{n+2} - 2$. Note that this means that only a 2^{-n+2} fraction of strings acually label vertices of G, so even guessing a vertex of G is hard, but we are promised that $s = 0^{2n}$ is in $V(G)$. The vertices of G are connected as follows. The vertex s is the root of a full binary tree with 2^n leaves. There is a second disjoint full binary tree with 2^n leaves, whose root we will call t. The leaves of these two trees are connected by a pair of perfect matchings (see Figure 3.6), where a perfect matching between two sets of vertices of size 2^n is just a set of 2^n edges from one set to the other in which each vertex is incident to exactly one edge. Note that this means that every vertex in G has degree 3, except for s and t, which each have degree 2.

Problem: WeldedTrees$_n$

Input: An oracle $\mathcal{O}_G$ for some welded trees graph G that acts, for any $u \in V(G)$, as $|u\rangle|0\rangle \mapsto |u\rangle|v_1, v_2, v_3\rangle$, where $v_1, v_2, v_3 \in \{0,1\}^{2n} \cup \perp$ are the neighbours of u, in lexicographic order (if u has degree 2, then $v_3 = \perp$).

Output: The $2n$-bit string labeling t.

This problem is set up to force a classical algorithm to solve it by a random walk, starting from s. The walk must start from s, since even just finding another vertex takes exponential time. Although a walker starting from s could quickly find the middle by never backtracking (even if the walker moves randomly, she will get to the middle quickly because there are twice as many edges leading towards the middle as away), once in the middle, the walker quickly gets lost. This is the intuition behind a highly non-trivial proof that a classical algorithm for this problem requires $2^{\Omega(n)}$ queries.

Somewhat astonishingly, a quantum walker can traverse this graph in linear time [CCD+03]. An excellent exposition can be found in [Chi21, Section 16].

While this algorithm was first presented as a continuous-time quantum walk, it is also possible to get a poly(n)-time discrete-time quantum walk, but not in the specific frameworks we have seen [JZ23].

Bibliography

[AGJ20] Simon Apers, András Gilyén, and Stacey Jeffery. A unified framework of quantum walk search. In *Proceedings of the 38th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 6:1–6:13, 2020. arXiv: 1912.04233 11, 14, 23

[AGJK20] Andris Ambainis, András Gilyén, Stacey Jeffery, and Martins Kokainis. Quadratic speedup for finding marked vertices by quantum walks. In *Proceedings of the 52nd ACM Symposium on the Theory of Computing (STOC)*, page 412–424, 2020. arXiv: 1903.07493 7, 9, 14

[Amb07] Andris Ambainis. Quantum walk algorithm for element distinctness. *SIAM Journal on Computing*, 37(1):210–239, 2007. Earlier version in FOCS'04. arXiv: quant-ph/0311001 8

[Amb10] Andris Ambainis. Quantum search with variable times. *Theory of Computing Systems*, 47:786–807, 2010. arXiv: quant-ph/0609168 23

[AS04] Scott Aaronson and Yaoyun Shi. Quantum lower bounds for the collision and the element distinctness problems. *Journal of the ACM*, 51(4):595–605, 2004. 8

[AS19] Simon Apers and Alain Sarlette. Quantum fast-forwarding: Markov chains and graph property testing. *Quantum Information and Computation*, 19(3&4):181–213, 2019. arXiv: 1804.02321 10

[Bel13] Aleksandrs Belovs. Quantum walks and electric networks. arXiv: 1302.3143, 2013. 14

[CCD+03] Andrew M. Childs, Richard Cleve, Enrico Deotto, Edward Farhi, Sam Gutmann, and Daniel A. Spielman. Exponential algorithmic speedup by a quantum walk. In *Proceedings of the 35th ACM Symposium on the Theory of Computing (STOC)*, pages 59–68, 2003. arXiv: quant-ph/0209131 24

[Chi21] Andrew M. Childs. Lecture notes on quantum algorithms. Available at https://www.cs.umd.edu/~amchilds/qa/, 2021. 24

[CRR+96] Ashok K. Chandra, Prabhakar Raghavan, Walter L. Ruzzo, Roman Smolensky, and Prasoon Tiwari. The electrical resistance of a graph captures its commute and cover times. *Computational Complexity*, 6(4):312–340, 1996. 12

[DH17] Cătălin Dohotaru and Peter Høyer. Controlled quantum amplification. In *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 18:1–18:13, 2017. 23

[DS84] Peter G. Doyle and J. Laurie Snell. *Random walks and electric networks.* Mathematical Association of America, 1984. arXiv: math/0001057 11

[Jef22] Stacey Jeffery. Quantum subroutine composition. arXiv: 2209.14146, 2022. 23

[JZ23] Stacey Jeffery and Sebastian Zur. Multidimensional quantum walks and application to k-distinctness. In *Proceedings of the 55th ACM Symposium on the Theory of Computing (STOC)*, pages 1125–1130, 2023. arXiv: 2208.13492 24

[LP17] David A. Levin and Yuval Peres. *Markov chains and mixing times.* AMS, Providence, RI, USA, 2nd edition, 2017. 5, 11

[MNRS11] Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. Search via quantum walk. *SIAM Journal on Computing*, 40(1):142–164, 2011. Earlier version in STOC'07. arXiv: quant-ph/0608026 9

[Mon18] Ashley Montanaro. Quantum-walk speedup of backtracking algorithms. *Theory of Comput-*

ing, 14(15):1–24, 2018. arXiv: 1509.02374 14

[Sze04] Mario Szegedy. Quantum speed-up of Markov chain based algorithms. In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 32–41, 2004. arXiv: quant-ph/0401053 7, 9, 14

[dW19] Ronald de Wolf. Quantum computing lecture notes. arXiv: 1907.09415v5, 2019. 3, 5, 6