Quantum Computing 3/8/2022

Lecture 12: Quantum Walks

Instructor: Dieter van Melkebeek Scribe: Soyeon Choi, Anirudh Yadav

Following amplitude amplification, we discuss another quantum paradigm, quantum walks. A quantum walk (QW) is the quantum version of classical random walk, which has found a lot of algorithmic applications. We begin by discussing classical random walks and study concepts fundamental to their operation, such as Markov chains. Then we develop their quantum counterpart, quantum walks, and discuss how they try to mimic random walks in a way that takes advantage of their quantum features for speedup. This lecture will be more of a survey of the general concepts and an application of quantum walks, with detailed analysis left for next lecture.

1 Random Walk

1.1 Setup

A *graph* is a set of points called *vertices* connected by lines called *edges*. Each edge (u, v) will have a *weight* w, which is proportional to the probability of moving to vertex v when currently at vertex u. We only consider finite undirected graphs $G = (V, E)$ where V denotes the set of $N = n(V)$ vertices, and E denotes the set of edges.

We can represent the weighted graph as a symmetric function $w : V \times V \to [0, \infty)$, where each possible edge that can exist between two vertices gets mapped to a weight ranging from 0 to infinity. The edges that are in $V \times V$ but not in E have weights equal to zero (i.e., for $x \in V \times V$ and $x \notin E$, $w(x) = 0$); therefore, E has nonnegative edge weights.

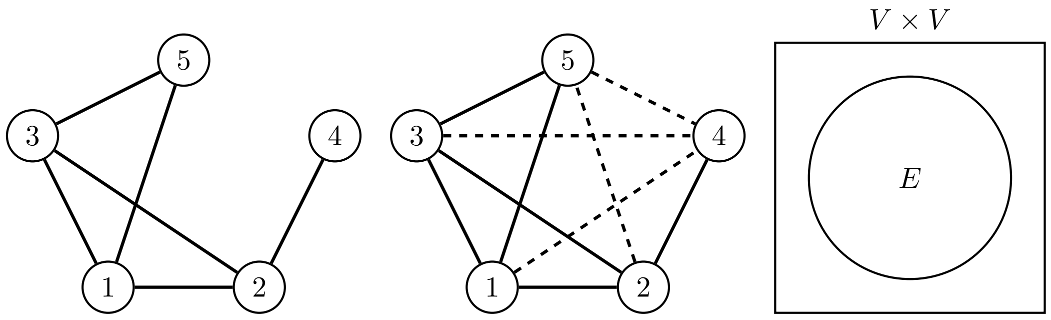


Figure 1: The leftmost graph shows the actual edge set E, the next graph shows the set $V \times V$ where the dotted lines indicate the edges not in E. The rightmost is a Venn-diagram of the edge set.

The undirected nature of G is reflected in the symmetry of w with respect to the two vertices that it connects: $w(u, v) = w(v, u)$.

The weight of a vertex $u \in V$ is defined as the sum of the weights of all incident edges on u:

$$w(u) \doteq \sum_{v \in V} w(u, v)$$

We assume that none of the vertices in G is isolated. In the unweighed setting, this means that every vertex is connected to at least one other vertex. In general, this indicates $w(u) > 0$ for each $u \in V$.

1.2 Process

A *random walk* is a process that starts by picking a starting vertex from an initial probability distribution σ of V. In each step it moves from the current vertex $u \in V$ to a neighboring vertex $v \in V$ with a probability proportional to $w(u, v)$. When the graph is unweighted, the subsequent vertex that the walk moves to is picked uniformly at random among the neighbors of the current vertex. This process is repeated for a number of steps. We are interested in how the probability distribution evolves as the number of steps increases. This process can be modeled with Markov chains, which we will cover next.

2 Markov chain

One way to model and analyze the behavior of a random walk is by viewing it as a *Markov chain.* This is a memoryless stochastic process, where the probability distribution of the next step only depends on the current state, and retains no history of past steps to arrive at where you are.

2.1 Transition matrix

The probability distribution obtained from a Markov chain can be described by a *transition matrix* T, which is an $N \times N$ matrix with components defined as follows:

$$T_{vu} \doteq \Pr[\text{move to } v \mid \text{at } u] = \frac{w(u, v)}{w(u)}$$

The transition matrix T can be used to describe the probability distribution after t steps of the random walk as $T^t\sigma$, i.e., as the product of the t-th power of T with a column vector representing the initial probability distribution σ. The column vector has one component for every vertex v, equal to the probability of being at v. The transition matrix T is symmetric ($T_{uv} = T_{vu}$ for all $u, v \in V$) iff $w(u) = w(v)$ for all $u, v \in V$, i.e., iff the weight $w(u)$ of every vertex u is the same. For unweighted graphs, T is symmetric iff the degree of every vertex is the same, i.e., the graph G is regular.

2.2 Stationary distribution

If a Markov chain converges to some distribution, that distribution needs to be stationary, i.e., invariant under T.

There may be multiple stationary distributions, but the following distribution π always is one:

$$\pi(v) \doteq \frac{w(v)}{W} \text{ where } W \doteq \sum_{v} w(v)$$

Proof. When the current distribution is π, the weight that is sent from u to v in one step of the random walk equals $T_{vu} \cdot \pi(u)$. Rewriting this expression, we get the following:

$$T_{vu} \cdot \pi(u) = \frac{w(u,v)}{w(u)} \cdot \frac{w(u)}{W} = \frac{w(u,v)}{W}$$

Since w is symmetric in u and v (i.e., $w(u,v) = w(v,u)$), the weight sent from u to v equals the weight sent from v to u, which gives us the following result:

$$\frac{w(u,v)}{W} = \frac{w(v,u)}{W} = T_{uv} \cdot \pi(u) \qquad \square$$

Corollary 1. *Note that π is uniform iff $w(u)$ is independent of u iff T is symmetric. In particular, in the unweighted case, π is uniform iff the graph G is regular.*

As a side note, a Markov chain that has a stationary distribution σ with the property that $T_{vu} \cdot \sigma(u) = T_{vu} \cdot \sigma(v)$, is called reversible; random walks on undirected graphs are equivalent to reversible Markov chains.

3 Convergence analysis

We restrict to an important case where T is symmetric; if that is the case, we can do a spectral analysis of T to analyze the convergence of the Markov chain to a stationary distribution π. In the general case, one uses the discriminant matrix D defined as:

$$D_{uv} \doteq \sqrt{T_{vu} \cdot T_{uv}}$$

Note that D always is symmetric, and that $D = T$ iff T is symmetric.

3.1 Spectrum of T

Since T is a symmetric real matrix, all its eigenvalues are real. Moreover, the eigenvalues have absolute values at most 1.

Proof. Since T is stochastic, for any vector x

$$\|Tx\|_1 \doteq \sum_v |\sum_u T_{vu} x_u| \leq \sum_u |x_u| \sum_v T_{vu} = \sum_u |x_u| \doteq \|x\|_1 \tag{1}$$

If λ is an eigenvalue of T with corresponding eigenvector x, then $|\lambda| \|x\|_1 = \|\lambda x\|_1 = \|Tx\|_1$, which by (1) implies that $|\lambda| \leq 1$. $\square$

The eigenvalue 1 of T represents invariance, so there always is an eigenvalue of 1, namely corresponding to the invariant distribution π. The multiplicity of the eigenvalue 1 equals the number of connected components of G. This means that there is a unique stationary distribution iff G is connected. In addition, T has an eigenvalue -1 iff G is bipartite. Thus, this leads us to the following conclusion:

Corollary 2. *With the exception of the stationary distribution π, all eigenvectors of T have eigenvalues strictly less than 1 in absolute value iff G is connected and non-biparite.*

We are going to use these facts in our convergence analysis of the stationary distribution π.

3.2 Convergence analysis for symmetric T

As a symmetric matrix, T has an orthogonal basis of eigenvectors including π. Then every vector, which is the distribution at any point in time throughout the random walk, can be described as a linear combination of these eigenvectors. Note that the component of any distribution σ along π equals π. This is because the inner product

$$(\sigma, \pi) = \frac{1}{N}\sum_{u\in V}\sigma(u) = \frac{1}{N}$$

is independent of σ. It follows that $(\sigma - \pi, \pi) = 0$, so that $\sigma - \pi$ has no component along π.

To check for convergence to π, we want to bound the difference between the distribution after t steps of the random walk, and the stationary distribution π. Since we know that $\sigma - \pi$ is orthogonal to π we can expand $\sigma - \pi$ in the set of all orthogonal eigenvectors β_i of T excluding π. That is to say there exist coefficients c_i such that

$$\sigma - \pi = \sum_i c_i\beta_i, \beta_i \neq \pi, \forall\ i \Rightarrow T^t(\sigma - \pi) = \sum_i c_i T^t \beta_i$$

$$T^t(\sigma - \pi) = \sum_i c_i T^t\beta_i \implies \|T^t(\sigma - \pi)\|_2 = \|\sum_i c_i\lambda_i^t\beta_i\|_2 \leq \|\sum_i c_i\lambda_{\max}^t\beta_i\|_2$$

$$\|T^t(\sigma - \pi)\|_2 \leq \lambda_{\max}^t\|\sum_i c_i\beta_i\|_2 = \lambda_{\max}^t\|\sigma - \pi\|_2$$

Note that the norm of π is equal to 1, from which you can show that the norm of $\sigma - \pi$ is at most 1:

$$\|\pi\|_2 = 1 = \|\pi - \sigma + \sigma\|_2 = \|\pi - \sigma\|_2 + \|\sigma\|_2 \implies \|\sigma - \pi\|_2 = 1 - \|\sigma\|_2 \leq 1$$

Therefore, we have that

$$\|T^t\sigma - \pi\|_2 = \|T^t\sigma - T^t\pi\|_2 = \|T^t(\sigma - \pi)\|_2 \leq \lambda_{\max}^t \cdot \|\sigma - \pi\|_2$$

Using the above results, we arrive at the conclusion

$$\|T^t\sigma - \pi\|_2 \leq \lambda_{\max}^t \cdot \|\sigma - \pi\|_2 \leq \lambda_{\max}^t \tag{2}$$

where $\lambda_{\max}$ denotes the maximum absolute value of an eigenvalue corresponding to an eigenvector that is orthogonal to π: $\lambda_{\max} \doteq \max\{|\lambda| : \text{(eigenvector with eigenvalue } \lambda) \perp \pi\}$. (2) gives us an upper bound on how far the distribution after t steps of the Markov chain is from the stationary distribution π.

We can express this upper bound in terms of the spectral gap δ, where $\delta \doteq 1 - \lambda_{\max}$, as follows:

$$\lambda_{\max}^t = (1-\delta)^t \leq \exp(-\delta)^t = \exp(-\delta t),$$

where we used the fact that $1 + x \leq \exp(x)$ for $x = -\delta$; this follows from the convexity of the exponential function.

In order to bound the statistical distance between the two distributions, we use the Cauchy-Schwartz inequality to upper bound the 1-norm as a function of the 2-norm[1]:

$$\|x\|_1 = \sum_i |x_i| = (X, \hat{1}) \leq \|X\|_2\|\hat{1}\|_2 = \|X\|_2\sqrt{N} = \|x\|_2\sqrt{N}$$

[1] $X = [|x_1|, |x_2|, \ldots]^T$, $\hat{1} = [1, 1, 1, \ldots, 1]^T$

Hence,

$$\|T^t\sigma - \pi\|_1 \leq \sqrt{N}\|T^t\sigma - \pi\|_2 \leq \sqrt{N}\lambda_{max}^t \leq \sqrt{N}\exp(-\delta t).$$

Thus, we can guarantee that $\|T^t\sigma - \pi\|_1 \leq \eta$ by setting $t \geq \frac{1}{\delta}\log(\sqrt{N}/\eta)$.

Convergence to π is guaranteed as long as $\delta > 0$, which is the case iff G is connected and non-bipartite. The number of steps required to approach π scales as $1/\delta$. If we want to guarantee fast convergence, we need a large spectral gap δ. On the other hand, if δ is close to zero, convergence may be slow.

4 Applications of random walks

Random walks can be used to model real world phenomena; for instance, in physics they are used as simple models of Brownian motion. They also can be used to model gambling. In this section we review two of the most common applications in computer science: sampling and search. We focus a little more on search, as our study of quantum walks is focused on their applications in search.

4.1 Sampling

This is the most common use of classical random walks. If we want to sample from a complicated distribution π, we can set up a random walk with stationary distribution π, start from a simple distribution σ (e.g., a point distribution), and run the random walk for a number of steps. If the spectral gap δ is large, then a small number of steps suffice to bring us close to π. This is known as rapid mixing. If the underlying graph is simple, the process yields an efficient way to (approximately) sample from π.

The process may also be of interest in case π is simple, say uniform, but one cannot obtain a uniform sample as a primitive. In such settings expander graphs are often helpful. These are infinite families of graphs that have bounded (i.e., constant) degree and a spectral gap δ that is at least some positive constant. By the above analysis, regular expanders exhibit rapid mixing to the uniform distribution, so a small number of steps of a random walk starting from some fixed vertex suffices to obtain an almost-uniform sample.

4.2 Search

Another application that is perhaps less common classically, but is very important in the quantum setting, is search.

Assume we have a set V of vertices, some of which are good and others bad, and our goal is to find a good vertex. As before, we represent goodness as a Boolean function $f : V \to \{0, 1\}$, where $f(v) = 1$ indicates that v is good. One way to search for a good vector is to pick one uniformly at random, and check whether f evaluates to one. If so, we are done; if not, we retry. If the fraction ϵ of good vertices is large, we expect only need a few trials are needed until the first success, namely $1/\epsilon$. However, similar to the above sampling setting, obtaining a uniform vertex may be expensive. Instead of picking a fresh uniform sample for each trial, it may be cheaper to move to a neighboring vertex in the underlying graph G, and use that for our next trial. In such settings, searching for a good vertex using a random walk makes sense.

We distinguish between three contributions to the cost of this search process: setup, update, and check.

- ◦ Setup cost S: The cost of picking a sample from the start distribution σ, which may or may not be the stationary distribution π. We incur this cost each time we start the random walk.
- ◦ Update cost U: The cost to execute one step of the random walk. We incur this cost t times when we perform a random walk of length t.
- ◦ Check cost C: The cost to determine whether the current vertex v is good, i.e., whether $f(v) = 1$. We may or may not want to do this for every step of the random walk.

5 Quantum Walk

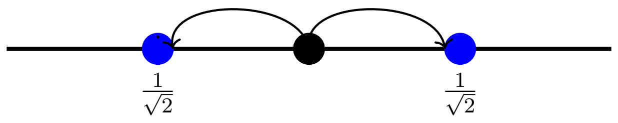
We like to setup an iterative process just like the classical walk that allows us to mimic a classical random walk, but in such a way that we move faster. Quantum walks take advantage of the quantum phenomenon of interference to speed up the convergence.

5.1 Model

For ease of exposition we consider the unweighted symmetric case. Now, if we are at vertex v, we would like to have a unitary transformation (quantum operations are unitary) that maps a vertex v to a uniform superposition of the neighbors of v.

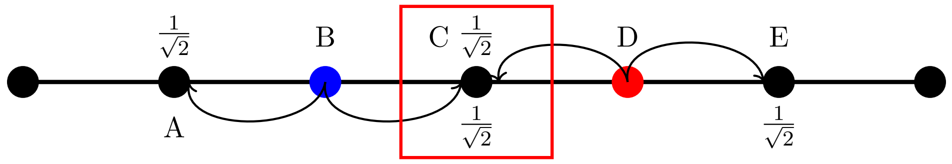
$$|v\rangle \mapsto |N_v\rangle \doteq \frac{1}{\sqrt{|N_v|}} \sum_{u \in N_v} |u\rangle$$

If, however, we apply this operation at every vertex, this is not a unitary operation. Consider a random walk on a line. The above quantum version applied to a basis state $|v\rangle$ results in the following superposition after one step:



As desired, the probability of being at the left neighbor of v equals $1/2$, as is the probability of being at the right neighbor of v. This is realized by transition amplitudes of $\frac{1}{\sqrt{2}}$ to both points.

But if we apply this operation at every vertex, the resulting transformation is not unitary, as the following figure illustrates.



Consider the basis states corresponding to the blue and the red vertex. As distinct basis states, the two states are orthogonal. After performing one step of the quantum walk on the blue state, the resulting superposition has an amplitude of $\frac{1}{\sqrt{2}}$ for both A and C:

$$|B\rangle \mapsto \frac{1}{\sqrt{2}}(|A\rangle + |C\rangle)$$

Similarly, one step of the quantum walk on the red state leads to a superposition of C and E, both with amplitude $\frac{1}{\sqrt{2}}$:

$$|D\rangle \mapsto \frac{1}{\sqrt{2}}(|C\rangle + |E\rangle)$$

These two superpositions are no longer orthogonal as they overlap in exactly one component (as marked by the red box in the graphic). In contrast, a unitary transformation maps orthogonal states to orthogonal states. Therefore, this operation is not unitary.

$$\langle B|D\rangle = 0 \mapsto \frac{1}{2}(\langle A| + \langle C|)(|C\rangle + |E\rangle) = \frac{1}{2} \neq 0$$

To fix this we can keep track of more of the history to prevent interference from happening. It suffices to remember the previous vertex to get this idea to work, i.e., the state consists of the current vertex v as well as the vertex u where we came from (or will be moving to next): $|u, v\rangle$.

5.2 Two different phases of a quantum walk

Each step of a quantum walk consists of two phases: a coin flip phase and a deterministic phase.

- **Coin flip phase**: We first apply a unitary on $|u\rangle$ that is controlled by $|v\rangle$ and replaces $|u\rangle$ by a superposition that corresponds to a random neighbor u' of v. We call this the *coin flip operator* C_v. Intuitively, in this step we decide where to move next.
- **Deterministic phase**: In the second phase we perform the actual move by swapping the two components:
$$S\,|u, v\rangle = |v, u\rangle$$
Now u is the current vertex, and v the previous one. This phase is deterministic.

6 Choices for coin flip

There are multiple choices for the coin flip operation. The quest is for choices that lead to improvements over the classical setting. Here are two natural choices.

- **Hadamard type:** For this type the amplitude of every neighbor $u' \in N_v$ has the same absolute value. This is a true quantum realization of selecting a neighbor at random. In the case of a walk on a line, the Hadamard transform is an obvious choice. It leads to interesting results in which the walk after t steps seems to spread more or less uniformly up to a distance of $\Theta(t)$ from the start vertex, whereas a classical walk remains concentrated within a distance $\Omega(\sqrt{t})$ of the start vertex. For higher degrees, one can use generalizations of the Hadamard transform like the Fourier transform over larger groups, which involves complex amplitudes. However, that line of research has not been very fruitful to date.
- **Grover type:** Given that we know which vertex we came from, we can consider two different (real) transition amplitudes: one amplitude a for going back to the current component u (the vertex we came from) and another amplitude b for every other vertex $u' \in N_v$. In fact, treating going back differently sometimes also makes sense in classical random walk, but is nonstandard and requires extending the state space (which is the standard for quantum

walks). As we will see, this choice of coin flip is closely related to the Grover iterate, and has led to several interesting speedup results for general random walks. This is the coin flip operator we consider.

6.1 Analysis of the Grover coin flip

Let us denote the Grover type coin flip transformation as C_v. The defining operation of C_v when it acts on the state $|u, v\rangle$, where the current vertex v has the neighbouring vertex set N_v can be expressed by the following equation:

$$C_v \otimes I\, |u, v\rangle = a\, |u, v\rangle + b \sum_{u' \in N_n \setminus \{u\}} |u', v\rangle$$

Where the operator C_v is described by the following $N \times N$ matrix acting on the neighboring nodes N_v of v, where $N \doteq |N_v|$:

$$C_v \doteq \begin{bmatrix} a & b & b & \cdots & b \\ b & a & b & \cdots & b \\ & & & \cdots & \\ b & b & b & \cdots & a \end{bmatrix}$$

The requirement that C_v be unitary is equivalent to the following two conditions:

$$a^2 + (N-1)b^2 = 1 \quad \text{and} \quad 2ab + (N-2)b^2 = 0.$$

The first condition expresses that every column of the matrix needs to have a two-norm equal to 1. The second condition expresses that the inner product of two distinct columns of the matrix is zero. The system of equations has the following two solutions, one of them being

$$(a, b) = \pm(1, 0)$$

which is just the diagonal matrix and uninteresting, and

$$(a, b) = \pm\left(\frac{2}{N} - 1, \frac{2}{N}\right)$$

We analyze the latter solution further, specifically the eigenstructure of the resulting matrix C_v.

- The uniform superposition $|N_v\rangle = \sum_{u \in N_v} \frac{1}{\sqrt{N}} |u\rangle$ over N_v is an eigenvector with a eigenvalue $a + (N-1)b = 1$. This follows because,

$$\begin{bmatrix} a & b & b & \cdots & b \\ b & a & b & \cdots & b \\ & & & \cdots & \\ b & b & b & \cdots & a \end{bmatrix} \frac{1}{\sqrt{N}} \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \end{bmatrix} = \frac{1}{\sqrt{N}} \begin{bmatrix} a+b+b+b+\ldots \\ b+a+b+b+\ldots \\ b+b+a+b+\ldots \\ \vdots \end{bmatrix} = \frac{1}{\sqrt{N}}(a+(N-1)b) \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \end{bmatrix} = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \end{bmatrix}$$

- As $C_v - (a-b)I$ has rank 1, $a - b = -1$ is an eigenvalue with multiplicity $N - 1$.

Thus, C_v has one eigenvalue 1, and $N-1$ eigenvalues -1. This means that C_v is a reflection over the eigenvector with eigenvalue 1, which is $|N_v\rangle$. We therefore sometimes refer to it as the reflection coin flip.

As $|N_v\rangle$ is uniform over N_v, the coin flip operator C_v has a close resemblance to the second of the two unitaries in one step of Grover, namely a reflection about the uniform superposition. This is why it is sometimes also referred to as the Grover coin flip. Our analysis of the resulting quantum walk in the next lecture will give us yet another view on the quadratic speedup that Grover provides.

7 Random Walk Search Complexity Analysis

We focus on applications of random walks and their quantum variants for search, assuming that we start from a sample of the stationary distribution π, i.e., we assume $\sigma = \pi$. To simplify the derivations, we assume that π is uniform, so the transition matrix T is symmetric and we do not need to refer to the discriminant matrix D. Nonetheless the results we obtain generalize to the setting of arbitrary T, albeit involving D in the analysis.

7.1 Hitting time

The hitting time describes how many steps of the random walk are needed to find a good vertex. We formally define it as the expected number of steps until the first success. As the hitting time depends on both the underlying walk (captured by the transition matrix T) as well as which vertices are good (captured by the predicate f), we write the hitting time as $H(T, f)$.

Exercise 1. *An alternate definition found may be the minimum number of steps t such that the probability of visiting a good vertex within the first t steps is at least 50%. Both definitions differ by a factor at most 2. We leave this as an exercise.*

The hitting time can be related in the following manner to the spectral gap δ of T and the weight $\epsilon \doteq \Pr_{v\sim\pi}[f(v) = 1]$ of the good vertices:

$$\frac{1}{\epsilon} \leq H(T, f) \leq O\left(\frac{1}{\delta\epsilon}\right)$$

The lower bound follows because, without any knowledge of set of good vertices beyond its weight ϵ, the best one can do is to use a fresh sample from π in every step. In that case, we run a Bernoulli experiment with success probability ϵ, which takes an expected $1/\epsilon$ steps until the first success. Although we do not directly prove the upper bound here, the intuition is that after an unsuccessful attempt, the vertex is distributed according to π restricted to the bad vertices, and $\Omega(1/\delta)$ steps of the random walk are enough to bring that distribution close to π again. Thus, an expected $1/\epsilon$ phases of $O(1/\delta)$ steps each suffice to hit a good vertex, which is $O(1/(\delta\epsilon))$ steps in total.

7.2 Expected cost of various classical approaches

We consider three approaches and express their expected costs as a function of the three cost components defined in section 4.2.

- **Without a walk**: Every time we pick a new vertex from π. Since we are only setting up one sample and checking the vertices for one iteration, the cost is $\mathrm{S} + \mathrm{C}$. We repeat this for $1/\epsilon$ stages to get a good vertex with at least 50% probability; therefore, the cost is,

$$O\left(\frac{1}{\epsilon}(\mathrm{S}+\mathrm{C})\right)$$

- **Checking each step**: Performing a random walk and checking at each step whether the vertex is good. The setup stage occurs once in the beginning, then we update the distribution and check repeatedly until we arrive at a good vertex. Therefore, cost function becomes: $\mathrm{S} + (\mathrm{U}+\mathrm{C}) + (\mathrm{U}+\mathrm{C}) + \dots$, if we continue the updating and checking process for about hitting time $H(T, f)$ stages the cost is,

$$O(\mathrm{S} + H(T, f)(\mathrm{U}+\mathrm{C}))$$

  This is cheaper if S is large compared to $\mathrm{U} + \mathrm{C}$.

- **Checking each τ-th step**: Performing a random walk but only checking at every τ-th step whether the current vertex is good. As above the setup cost is incurred only once in the beginning and then we check after precisely τ updates or walks therefore the cost functions looks like $\mathrm{S} + (\tau \cdot \mathrm{U} + \mathrm{C}) + (\tau \cdot \mathrm{U} + \mathrm{C}) + \dots$, as before if we run the update and check stages for $H(T^\tau, f)$ number of times the final cost is,

$$O(\mathrm{S} + H(T^\tau, f)(\tau \cdot \mathrm{U} + \mathrm{C})),$$

  where we used the fact that the hitting time for τ steps of the random walk equals $H(T^\tau, f)$. This may be cheaper in case the checking cost C is high compared to the update cost U.

  A particular setting of interest is $\tau = \Theta(1/\delta)$. Based on the intuition behind the upper bound $H(T, f) = O(1/(\delta\epsilon)$ that $\Omega(1/\delta)$ steps of the random walk after an unsuccessful trial bring us (close to) π again, we have that $H(T^\tau, f) = O(1/\epsilon)$ for this choice of τ, so the expected cost is

$$O\left(\mathrm{S} + \frac{1}{\epsilon}\left(\frac{1}{\delta}\mathrm{U} + \mathrm{C}\right)\right)$$

7.3 Quantum Walks Search Speed up

We end this section by stating the speedups that can be realized for search by the quantum walks based on the reflection / Grover coin flip. For comparison, let us first restate the costs for search based on a classical random walk:

Runtime	Classical random walk	Quantum random walk
Checking each step	$O(\mathrm{S} + H(T, f)(\mathrm{U}+\mathrm{C}))$	$\tilde{O}(\mathrm{S} + \sqrt{H(T, f)}(\mathrm{U}+\mathrm{C}))$
Checking each τ-th step	$O(\mathrm{S} + H(T^\tau, g)(\tau \cdot \mathrm{U} + \mathrm{C}))$	$\tilde{O}(\mathrm{S} + \sqrt{H(T^\tau, g)}(\sqrt{\tau} \cdot \mathrm{U} + \mathrm{C}))$
Checking each τ-th step for $\tau = \Theta(1/\delta)$	$O(\mathrm{S} + \frac{1}{\epsilon}(\frac{1}{\delta}\mathrm{U} + \mathrm{C}))$	$\tilde{O}(\mathrm{S} + \frac{1}{\sqrt{\epsilon}}(\frac{1}{\sqrt{\delta}}\mathrm{U} + \mathrm{C}))$

When checking each step, since classically the number of iterations for achieving at least 50% probability of success was $H(T,f)$, the same result only requires $\sqrt{H(T,f)}$ steps using quantum walk. Similarly, when checking each τ-th step, in each iteration, classically we required τ stages to end up at the $\tau-$th stage vertex; however, using the Grover coin flip for quantum walk we would need only $\sqrt{\tau}$ updates for roughly the same outcome. This arises from the fast forwarding lemma, which will be covered in the next lecture (Lecture 13, Lemma 2). Note the square roots on the hitting time, δ, and ϵ, resulting in the potential for a square-root speedup. Two caveats:

- The costs S, U, and C refer to somewhat different processes in the classical vs. the quantum setting.
- The costs in the quantum setting have additional polylog factors, represented by the use of the symbol $\tilde{O}$ instead of O. The additional polylog factors are not needed when the goal is to merely detect the existence of a good vertex.

8 Collision Problem

The collision problem takes as input a black-box for a function $h : \{0,1\}^n \to R$ for some range R and the goal is to find two distinct inputs x_1, x_2 such that $h(x_1) = h(x_2)$ or report that no such pair of inputs exists. The decision version of this problem is known as *element distinctness.* Quantum walk search yields an optimal query complexity up to constant or poly-log factors. The collision problem was one of the main drives behind research into quantum walk search.

8.1 First Attempt

The first idea to tackle such a problem would be to use the Grover search algorithm. We would pick l values uniformly at random and check if there is a collision among them. If no collisions are found among the initial group, we simply use the Grover search algorithm to find a collision with any of those l values. Assuming there is a collision, in the worst case there is only a single collision. For that collision to be found, at least one of the two values involved in the collision must be in the l values chosen. Therefore, the probability of success in this case is $\binom{N-1}{l-1}/\binom{N}{l} = \frac{l}{N}$. As the probability of success is at least $\frac{l}{N}$, we repeat the process $\frac{N}{l}$ times to get a high confidence. As the number of queries for one iteration is $l + \sqrt{N}$, the overall number of queries is $O\left(\frac{N}{l}(l+\sqrt{N})\right)$, which isn't any better than $O(N)$.

8.2 Second Attempt

For the second attempt we might think that we can improve our previous algorithm by utilizing amplitude amplification. For this we use the same algorithm as above except we use amplitude amplification to boost the success of finding a collision between one of the l values and the other values. Using this method we are able to decrease the number of repetitions from $\frac{N}{l}$ to $\sqrt{\frac{N}{l}}$. This reduces the overall number of queries to $O\left(\sqrt{\frac{N}{l}}(l+\sqrt{N})\right)$. As the function $g(l) = \sqrt{\frac{N}{l}}(l+\sqrt{N})$ goes to ∞ as $l \to 0$ or $l \to \infty$, we can optimize the function for l. If a function is the sum of a decreasing and increasing term, finding the intersection point gives the optimum value up to a factor of two, which is good enough as we don't care about constants. The factor of two follows because

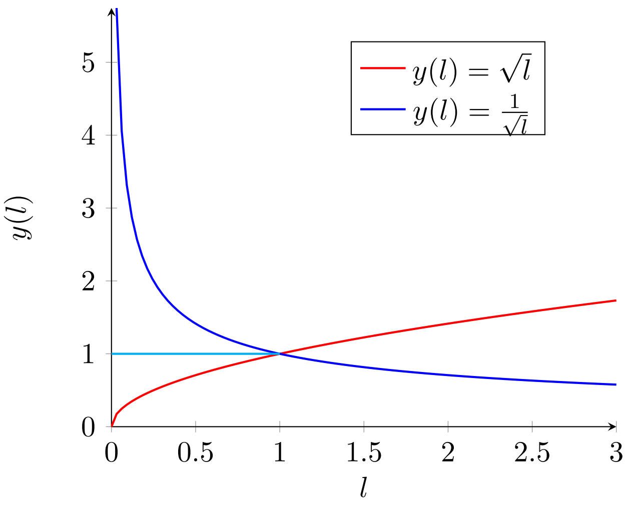


Figure 2: Plot of the intersection of the functions $\sqrt{Nl}$ and $N/\sqrt{l}$ for the case when $N = 1$.

before the intersection point, the decreasing term is larger than the value μ at the intersection point, and after the intersection point the increasing term is larger than μ, so the minimum cannot be smaller than μ, whereas the value at the intersection point of the sum of the two terms equals 2μ. See Figure 2 for an illustration when $N = 1$.

$$\sqrt{Nl} = \frac{N}{\sqrt{l}} \Rightarrow l = \sqrt{N}$$

Alternatively we can also differentiate the function to obtain the optimum value of the function $g(l)$,

$$\frac{dg}{dl} = \frac{\sqrt{N}}{2\sqrt{l}} - \frac{N}{2l^{3/2}} = 0 \Rightarrow l = \sqrt{N} \Rightarrow g(\sqrt{N}) = 2N^{3/4}$$

So, when the number of queries is optimized with $l = \sqrt{N}$, we get $O(N^{3/4})$ (the factor of 2 was dropped as we don't care about constants). While this is an improvement, we can still do better.

8.3 Using Quantum Walk Search

For quantum walk search, we begin the same way as the first two attempts by randomly selecting l values and checking if there is a collision among them. The difference now is that if no collision has been found among the l values, we simply swap one of the l values for a value not among the l values and check if the new value collides with any of the other $l-1$ values. This can be viewed as a random walk on a graph for which subsets of size l among the N values serve as the vertices and edges connecting subsets that are only one swap apart. This type of graph is known as a Johnson graph $J(N, l)$.

A Johnson graph $J(N, l)$ is a regular graph (all vertices have the same degree) with a degree of $l(N-l)$. As the graph is undirected and regular, the stationary distribution π is uniform. $J(N, l)$ also has a spectral gap of $\delta = \frac{N}{l(N-l)}$, which is $\Theta(\frac{1}{l})$ for $l \ll N$. Assuming that there is at least one collision, within this graph there must be at least $\binom{N-2}{l-2}$ good subsets (ones in which a collision is found). Using this we get:

$$\epsilon \geq \binom{N-2}{l-2} \Big/ \binom{N}{l} = \Theta\left(\frac{l^2}{N^2}\right)$$
$$H = O\left(\frac{1}{\delta\epsilon}\right) = O\left(\frac{N^2}{l}\right)$$

The query cost for the setup is $\mathrm{S} = l$ as we need to query f for each of the selected l values. The query cost for the update is $\mathrm{U} = 2$ as we need to clear the original value in a reversible way (that is, query f and XOR the result with the stored value) and query f for the new value. Finally, the query cost for the check is $\mathrm{C} = 0$ as we already have the outputs of f for the l values. This results in a total decision cost of $O(\mathrm{S} + \sqrt{H}(\mathrm{U} + \mathrm{C})) = O(l + \frac{N}{\sqrt{l}})$ queries. This cost is optimized for $l = N^{2/3}$. To see this, we use the same strategy as before where we set the increasing term equal to the decreasing term.

$$l = \frac{N}{\sqrt{l}}$$
$$l^{3/2} = N$$
$$l = N^{2/3}$$

Using the optimum value l, we get a tight upper bound of $O(N^{2/3})$ on the number of queries. For search, rather than decision, a $\tilde{O}$ cost is incurred.