Chapter 16
Quantum Random Walks

Both quantum and classical random walks can be visualized as walks on graphs. The graphs may be finite or infinite, directed or undirected. First we consider the classical case.

Classical Random Walks

Classical random walks on graphs are a fundamental topic in computational theory. The idea of a walk is easy to picture. Suppose you are at a node $u \in V$, and suppose there are edges out of u to neighbors $v_1, \ldots, v_d$. In the **standard random walk**, you pick a neighbor v_i at random, that is with probability $1/d$. In a general random walk, there is a specified probability $p_{u \to v_i}$. Either way, if you choose v_i, you go there by setting $u := v_i$ and repeat. There are three main questions about a classical random walk:

1. Given a node w different from u, what is the expectation for the number of steps to reach w starting from u?
2. How many steps are expected for the walk to visit all nodes w in the graph, in case $n = |V|$ is finite?
3. If you stop the walk after a given number t of steps, what is the probability $p_t(w)$ of ending at node w? How does it behave as t gets large?

The questions can have dramatically different answers, depending on whether G is directed or undirected. To see this, first consider the undirected graph in which the vertices stand for the integers, and n is connected to $n-1$ and to $n+1$. If we start at 0, what is the expected number of steps to reach node n? Each step is a coin-flip, heads you move right, tails you move left. Hence reaching cell n means sometime having an excess of n more heads than tails. Now the standard deviation

of N-many coin-flips is proportional to $\sqrt{N}$, and it follows that the expected time to have a deviation of $+n$ is $O(n^2)$.

This result carries over to any undirected n-vertex graph. If node y is reachable at all from node x, then there is a path from x to y of length at most $n-1$. It is possible that some node u along this path may have degree $d \geq 3$ with $d-1$ of the neighbors further away from y, so that the chance of immediate progress is only $1/d$. However, this entails that the original distance from x to y was at most $n-d+1$. Thus any graph structure richer than the simple path trades against the length, and it can be shown that the $O(n^2)$ step expectation of the simple path remains the worst case to reach any given node in the same connected component as x. In particular, this yields a log-space, polynomial-time randomized algorithm to tell whether an undirected graph is connected.

For directed graphs, however, the time can be exponential. Consider directed graphs G_n with $V = \{ 0, \ldots, n-1 \}$ and edges $(i, i+1)$ and $(i, 0)$ for each i. The walk starts at $u = 0$ and has goal node $y = n-1$, which we may suppose has both out-edges going to 0. Now a "tail" sends the walk all the way back to 0, so the event of reaching y is the same as getting $n-1$ consecutive heads. The expected time for this is proportional to 2^n. Thus mazes with one-way corridors are harder to solve than the familiar kind with undirected corridors.

This difference undergirds central open questions in computational complexity. Random walks can be performed in $O(\log n)$ space, needing only $O(\log n)$ bits to store the address of the current node and provide some indices i, j for iterating over the adjacency matrix of the graph. The graph is given as read-only input that doesn't count against space, so in particular one may not mark already-visited nodes. Omer Reingold [8] completed a decades-long project of showing that the random-walk algorithm for connectivity can be "de-randomized," that is replaced by a deterministic algorithm of equivalent efficiency. This classified the undirected connectivity problem into deterministic logspace, which is called L. Directed connectivity, however, remains complete for nondeterministic logspace, which is called NL. The $\mathsf{NL} = \mathsf{L}$ problem bears some resemblance to $\mathsf{NP} = \mathsf{P}$, but has some differences, most notably that NL is known to be closed under complementation, while belief that $\mathsf{NP} \neq \mathsf{co\text{-}NP}$ is almost as strong as that in $\mathsf{NP} \neq \mathsf{P}$.

There are two main further insights in the road to quantum random walks. The first is that quite apart from how directedness can make locations difficult to reach with high probability, it is possible to cancel the probability of being in certain locations at certain times altogether. The second is like the difference between AC and DC electricity. Instead of seeing a walk as "going somewhere" like a current, it is better to view it as a dance back-and-forth on the vertices according to some eventually-realized distribution. Both insights require representing walks in terms of actions by matrices, and again we can get much initial mileage from the classical case.

Random Walks and Matrices

Classical random walks on graphs $G = (V,E)$ can be specified by matrices A whose rows and columns correspond to nodes u, v. Here A is like the adjacency matrix of G, in that $A(u,v) \neq 0$ only if there is an edge from u to v in G, but the entries on edges are probabilities. Namely, $A(u,v) = p_{u\to v}$, which denotes the probability of going next to v if the "walker" is at u. The matrix A is row-stochastic; that is, the values in each row are nonnegative and sum to 1.

It follows that A^2 is also a row-stochastic matrix, and gives the probabilities of pairs of steps at a time. That is, for any nodes u and w,

$$A^2(u,w) = \sum_v A(u,v)A(v,w) = \sum_v p_{u\to v}p_{v\to w}.$$

Since the events of the walk going from u to different nodes v are mutually exclusive and collectively exhaustive, this sum indeed gives the probability of going from u to w in 2 steps. The same goes for A^3 and paths of three steps, and so on for A^k, all $k \geq 0$.

A probability distribution D on the nodes of G is **stable** under A if for all nodes v,

$$D(v) = \sum_v D(u)A(u,v).$$

Intuitively this says that if $D(v)$ is the probability of finding a missing parachute jumper at any location v, then the probability is the same even if the jumper has had time to do a random step according to A after landing. Mathematically this says that D is an *eigenvector* of A, with eigenvalue 1; the eigenvector is on the left, giving $DA = D$.

If G is connected, finite, undirected graph that is not bipartite, there is an integer k such that for all $\ell \geq k$, and all $x, y \in V$, there is a path of exactly ℓ steps from x to y. It follows that for for any matrix A defining a random walk on G, all entries of A^k and all higher powers are nonzero. It then further follows—this is a hard theorem—that the powers of A converge pointwise to a matrix A^* that projects onto some stationary distribution. That is, for any initial distribution C, $CA^* = D$, and moreover the sequence $C_k = CA^k$ converges pointwise to D. This goes even for the distribution $C(u) = 1$, $C(v) = 0$ for all $v \neq u$, which represents our random-traveler initially on node u.

When A is the standard random walk, the limiting probability is $D(u) = deg(u)/2|E|$. Non-uniform walks A may have other limiting probabilities, but they still have the remarkable property that any initial distribution is converged pointwise to D. The relation between $\varepsilon > 0$ and the power k needed to ensure $||CA^\ell - D|| \leq \varepsilon$ for all $\ell \geq k$, where $||\cdot||$ is the max-norm, is called the **mixing time** of A, while the k for $\max_{u,v} |D(v) - A^\ell(u,v)| \leq \varepsilon$ for all $\ell \geq k$ is called the **hitting time**.

If G is bipartite, there is still a stationary D, but not all C will be carried toward it—any distribution with support confined to one of the two partitions will alternate between the partitions. When G is directed, similar behavior occurs with period 3

in a directed triangle, and so on. However, provided that for every u, v and prime p there is a path from u to v whose number of steps is not a multiple of p, the above limiting properties still hold for every random walk on G, and the notions of mixing and hitting times are still well-defined. In an undirected graph, for constant ε, the hitting time of any walk is polynomial, but in a directed graph even the standard walk may need exponential time, as the directed graphs in the last section show.

The analogy here is that the stationary distribution D is like "AC current" in that you picture a one-step dance back and forth but the overall state remains the same. This differs from the "DC" view of a traveler going on a random walk. What distinguishes the quantum case is that via the magic of quantum cancellation we can often arrange for $D(v)$ to be *zero* for many undesired locations v, and hence pump up the probability of the "traveler" being *measured* as being at a desired location u.

An Encoding Nicety

To prepare for the notion of quantum random walks, we consider the probabilities p as derived from a set C of random outcomes. In the background is a function $h(u,c) = v$ that specifies the destination node for each outcome c.

To encode the standard random walk in which the next node is chosen with equal probability among all out-neighbors v of u, we simply take $|C|$ to be the least common multiple of the out-degrees of all the vertices in the graph. Then for each vertex we assign outcomes in C to choices of neighbor evenly. This is well-defined also for classes of infinite graphs of bounded degree. Indeed the infinite path graph remains a featured example, taking $C = \{ 0, 1 \}$ and thinking of c as a "coinflip." We could extend this formalism to allow arbitrary distributions on C, but uniform suffices for the main facts and applications

Now we make a matrix A' whose rows index pairs u, c of nodes and random outcomes. We can write this pair without the comma. So we define

$$A'(uc, v) = 1 \qquad \text{if } h(u,c) = v$$

and $A'(uc, v) = 0$ otherwise. Now each row has one 1 and $n-1$ 0s. We can, however, obtain the stochastic matrix A above via

$$A(u,v) = \frac{1}{|C|} \sum_c A'(uc, v).$$

If we had a non-uniform distribution on C, we could use a weighted sum accordingly. Note also that our functional view of matrices makes this undisturbed by the possibility that V, and hence A and A', could be infinite.

We do one more notational change that already helps with the classical case by making the matrix square again. We make the same random outcome part of the column value as well, by defining:

$$B(uc, vc') = 1 \qquad \text{if } h(u,c) = v \text{ and } c' = c,$$

with $B(uc, vc') = 0$ otherwise. Then B acts like the identity on the C-coordinates, and acts like A' on the V-coordinates, that is on the nodes. Now the stochastic matrix A is given by

$$A(u,v) = \frac{1}{|C|} \sum_c B(uc, vc).$$

The sum on the right-hand side goes down the diagonal of the C-part, much like the trace operation does on an entire matrix. It is called a *partial trace* operation, and is generally important in quantum mechanics. In the classical case, all this does is get us back to our original idea of entries $A(u,v)$ being probabilities, but it will help in the quantum case where they are *amplitudes*, meaning complex numbers whose squared norms are probabilities.

Quantum Random Walks

The reason we need the added notation of C in the quantum case is that on the whole space $V(G) \otimes C$, quantum evolution is an entirely deterministic process. It is because the action on C is unknown and unseen before being implicitly "traced out" that gives the effect of a randomized walk.

Definition 16.1. A *quantum random walk* on the graph G is defined by a matrix U with analogous notation to B above, but where U is unitary, and allowing the action U_C of U on the C coordinates to be different from the identity.

Indeed, in the case $|C| = 2$, by making U_C have the action of a 2×2 Hadamard matrix H, we can also simulate the action of flipping the coin at each step. Again the action of H itself is deterministic, but because measurement involves making a choice over the entries of H, the end effect is nondeterministic. Here is an example that packs a surprise.

Let G be the path graph with seven nodes, labeled $u = -3, -2, -1, 0, 1, 2, 3$. Our state space is $V(G) \otimes \{0,1\}$. To flip a coin, we apply the unitary matrix $\mathsf{C} = \mathsf{I} \otimes \mathsf{H}$ where I is the 7-dimensional identity matrix. To effect the outcome b, we apply the 14×14 permutation matrix P that maps $(u,0)$ to $(u-1,0)$ and $(u,1)$ to $(u+1,1)$. Since we will apply this only three times to a traveler beginning at 0, it doesn't matter where $(-3,0)$ and $(3,1)$ are mapped—to preserve the permutation property they can go to $(3,0)$ and $(-3,1)$, respectively, thus making the action on $V(G)$ circular. Our walk matrix is thus $\mathsf{A} = \mathsf{PC}$. We apply A^3 to the quantum basis state η_0 that has a 1 in the coordinate for $(0,0)$ and a 0 everywhere else.

In three steps of a classical random walk on G starting at the origin, the probabilities on the nodes $(-3,-1,1,3)$ respectively are $(\frac{1}{8}, \frac{3}{8}, \frac{3}{8}, \frac{1}{8})$ according to the familar binomial distribution. (Those on the even nodes are zero since G is bipartite.) This is arrived at by summing over paths, each path being a product of three entries of the

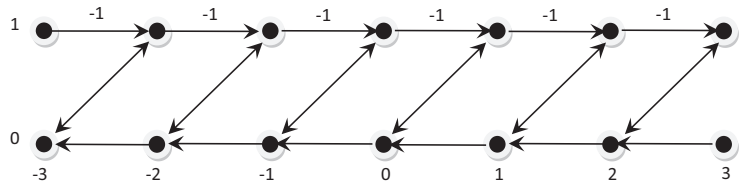

Fig. 16.1 Expanded graph G' of quantum walk on path graph G.

walk matrix. Since each nonzero entry is $\frac{1}{2}$, the middle values come about because there are three different ways to go from 0 to $+1$ in three steps, and likewise form 0 to -1.

In the quantum random walk, there is also a sum over paths, with each path being a product of three entries in the matrix A, but there are three differences. First, the entries have $\sqrt{2}$ rather than 2 in their denominators—they will be squared again when going from amplitudes to probabilities at the end. Second, the numerators can be -1 and $+1$. Third, and the unseen part under the hood, the paths being summed by Nature fork not only in the G part, but also in the C part of the space. That is to say, each coin outcome, which is represented by a column of the ordinary 2×2 Hadamard matrix,

$$H = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix},$$

has two ways of reaching that outcome, via the first or second row. When the coin outcome is 0 for "tails," both entries contribute a numerator of $+1$, but when the outcome is 1, one path contributes a $+1$, and the other -1.

Hence the walk is really taking place in a 14-node graph G' that includes the coinflips. This graph has *directed* edges from $(u,0)$ and $(u,1)$ to $(u-1,0)$ for the outcome "tails," say wrapping around to $(3,0)$ in the case $u=-3$. And for "heads" it has edges from $(u,0)$ and $(u,1)$ to $(u+1,1)$, again wrapping around, with the crucial difference that the rightward edges from $(u,1)$ (representing a previous outcome of heads) have multiplier -1. The other edges have $+1$. Now the three-step paths from $(0,0)$ in G', and their multiplier values, are:

$$\begin{aligned}
&(0,0) \to (1,1) \to (2,1) \to (3,1) : 1 \cdot -1 \cdot -1 = 1 \\
&(0,0) \to (1,1) \to (2,1) \to (1,0) : 1 \cdot -1 \cdot 1 = -1 \\
&(0,0) \to (1,1) \to (0,0) \to (1,1) : 1 \cdot 1 \cdot 1 = 1 \\
&(0,0) \to (1,1) \to (0,0) \to (-1,0) : 1 \\
&(0,0) \to (-1,0) \to (0,1) \to (1,1) : 1 \cdot 1 \cdot -1 = -1 \\
&(0,0) \to (-1,0) \to (0,1) \to (-1,0) : 1 \\
&(0,0) \to (-1,0) \to (-2,0) \to (-1,1) : 1 \\
&(0,0) \to (-1,0) \to (-2,0) \to (-3,0) : 1
\end{aligned}$$

Thus there are six, not four, different destinations. The crux is that the two paths that reach destination $(1,1)$ have multipliers of 1 and -1 and hence *cancel*, while the two paths with destination $(-1,0)$ both have multipliers of 1 and hence *amplify*. The 14-dimensional vector representing the quantum state of the outcome of the walk, given the initial vector which had a 1 in place $(0,0)$ and nothing else, becomes this when arranged in a 2×7 grid:

$$\phi_0 = \frac{1}{\sqrt{8}} \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 2 & 0 & -1 & 0 & 0 \end{pmatrix}.$$

Finally, to obtain the probabilities while "tracing out" the unseen coin, we sum the *squares* in each column. The final classical probability vector, giving the probability of finding the "traveler" at each of the original seven nodes of G after a measurement, is

$$(\frac{1}{8}, 0, \frac{5}{8}, 0, \frac{1}{8}, 0, \frac{1}{8}),$$

in marked contrast to the classical outcomes. What happened, and why the loss of symmetry?

It is at least reassuring that the bipartiteness of G showed through, giving zero probability again on the even-numbered vertices. But the leftward bias flies in the face of fairness when flipping a coin. The Hadamard matrix is used all the time to introduce quantum nondeterminism, so why does it give off-center results? The reason is that the -1 entry biases against "heads," causing cancellations that do not happen for "tails."

These cancellations can be harnessed for a rightward bias by starting the traveler at $(0,1)$ rather than $(0,0)$. That is, unknown to the traveler or any parties observing just the original graph G, we are starting the coin in an initial state of "heads" rather than "tails." In the $G \otimes C$ space this means the initial state η_1 has a 1 in the coordinate for $(0,1)$ and a 0 everywhere else. From $(0,1)$, some relevant 3-step paths are:

$$\begin{aligned}
&(0,1) \to (1,1) \to (2,1) \to (1,0) : -1 \cdot -1 \cdot 1 = 1 \\
&(0,1) \to (1,1) \to (0,0) \to (1,1) : -1 \cdot 1 \cdot 1 = -1 \\
&(0,1) \to (-1,0) \to (0,1) \to (1,1) : 1 \cdot 1 \cdot -1 = -1 \\
&(0,1) \to (-1,0) \to (0,1) \to (-1,0) : 1 \cdot 1 \cdot 1 = 1 \\
&(0,1) \to (1,1) \to (0,0) \to (-1,0) : -1 \cdot 1 \cdot 1 = -1.
\end{aligned}$$

These show a mirror-image amplification and cancellation, and give the 14-vector

$$\phi_1 = \frac{1}{\sqrt{8}} \begin{pmatrix} 0 & 0 & 1 & 0 & -2 & 0 & -1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

These amplitudes give the classical probabilities $(\frac{1}{8}, 0, \frac{1}{8}, 0, \frac{5}{8}, 0, \frac{1}{8})$ on G, now biased to the right.

You might hence think that you can cancel the biases by starting the system up with the coin in the "half-tails, half-heads" state

$$\eta_2 = \frac{1}{\sqrt{2}}(\eta_0 + \eta_1).$$

This is like saying Schrödinger's cat has half a tail, or rather the square root of half a tail. By the linearity of quantum mechanics—remember A^3 is just a matrix—the final state you get now is

$$\begin{aligned}\phi_2 &= \frac{1}{\sqrt{2}}(\phi_0 + \phi_1) \\ &= \frac{1}{4}\begin{pmatrix} 0 & 0 & 2 & 0 & -2 & 0 & 0 \\ 2 & 0 & 2 & 0 & 0 & 0 & 0 \end{pmatrix}.\end{aligned}$$

Note that we got some more cancellations—that is to say the two walks *interfered* with each other—and those were both on the right-hand side, so we have bias to the left again with probabilities $(\frac{1}{4}, 0, \frac{1}{2}, 0, \frac{1}{4}, 0, 0)$. In particular, the rightmost node is now unreached.

We can finally fix the bias by making the second random walk occur with a quarter-turn *phase* displacement. This means starting up in the state

$$\eta_3 = \frac{1}{\sqrt{2}}(\eta_0 + i\eta_1).$$

This state is like Schrödinger's cat with half a tail and an imaginary head. Again by linearity the final state is $\phi_3 = (\phi_0 + i\phi_1)/sqrt2$, so that

$$\phi_3 = \frac{1}{4}\begin{pmatrix} 0 & 0 & 1+i & 0 & -2i & 0 & 1-i \\ 1+i & 0 & 2 & 0 & -1+i & 0 & 0 \end{pmatrix}.$$

Taking the squared norms and adding each column gives the probabilities

$$\frac{1}{16}(0+2, 0, 2+4, 0, 4+2, 0, 2+0) = (\frac{1}{8}, \frac{3}{8}, \frac{3}{8}, \frac{1}{8})$$

again, at last modeling the classical random walk.

A more-robust wat to fix the bias is to use a fully-balanced matrix other than Hadamard for the quantum coin-flip action. A suitable unitary matrix is

$$\mathsf{J} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & i \\ i & 1 \end{bmatrix}$$

If we take higher powers A^k, whether $\mathsf{A} = \mathsf{P}(\mathsf{I} \otimes \mathsf{H})$ or $\mathsf{A} = \mathsf{P}(\mathsf{I} \otimes \mathsf{J})$ or whatever, the circular connectivity of G guarantees that the uniform classical distribution giving probability $\frac{1}{7}$ to each node of G is approached. It is stable *as* the induced classical distribution.

The Big Factor

If instead we extend the graph G beyond seven nodes, we come immediately to the surprise of greatest import. Let G have $n = 9$ nodes, labeled -4 to 4, and define G' as before including wrapping at the endpoints. Extend P to be 18×18 accordingly, keep H as the action on the coin space, and consider walks of length 4. Labeling the 16 basic walks of length 4 by HHHH through TTTT gives us a shortcut to compute the destination (m,a) and multiplier $b \in \{\ 1,-1\ \}$ for each walk w:

- m is the number of H in w minus the number of T.
- a is 0 if w ends in T, 1 if w ends in H.
- b is -1 if HH occurs an odd number of times as a substring of w, 1 if even.

For example, HHTT, HTHT, and THHT all come back to $(0,0)$, and their respective multipliers are -1, 1, and -1. Importantly, this implies there is a cancellation at the origin, leaving -1 there. Similar happens with HTTH, THTH, and TTHH ending at $(0,1)$, leaving 1, while HTTT, THTT, and TTHT all hit $(-2,0)$ with weight 1, reinforcing each other to leave 3 there. The full 2×9 vector for the quantum state after the walk is:

$$\phi_0 = \frac{1}{4}\begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 \\ 1 & 0 & 3 & 0 & -1 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

For initial state $(0,1)$ we have the same rules, except with Hw in place of w. This yields

$$\phi_1 = \frac{1}{4}\begin{pmatrix} 0 & 0 & 1 & 0 & -1 & 0 & 3 & 0 & 1 \\ 1 & 0 & 1 & 0 & -1 & 0 & -1 & 0 & 0 \end{pmatrix}.$$

Again $(\phi_0 + i\phi_1)/sqrt2$ results from superposing the initial states with a 90-degree phase shift on the latter. Taking the squared norms of its entries gives

$$\frac{1}{32}\begin{pmatrix} 0 & 0 & 2 & 0 & 2 & 0 & 10 & 0 & 2 \\ 2 & 0 & 10 & 0 & 2 & 0 & 2 & 0 & 0 \end{pmatrix}.$$

Summing the columns finally yields the classical probabilities

$$(\frac{1}{16}, 0, \frac{3}{8}, 0, \frac{1}{8}, 0, \frac{3}{8}, 0, \frac{1}{16}).$$

This is the surprise: the five nonzero numerators differ from $(\frac{1}{16}, \frac{1}{4}, \frac{3}{8}, \frac{1}{4}, \frac{1}{16})$ which is the classical walk's binomial distribution. The quantum coin-flip distribution is flatter in the middle, with more weight dispersed to the edges.

As n increases, this phenomenon becomes more pronounced: the locations near the origin cumulatively have low probability, while most of the probability is on nodes at distance proportional to n. This phenomenon persists under various quantum coin matrices. The general reason is that the many paths under the classical "H-T" indexing that end close to the origin tend to cancel themselves out, while the fewer classical paths that travel do enough mutual reinforcement that the squared norms compensate for the overall count.

The effect is that unlike the classical distribution, which for n steps has standard deviation proportional to $\sqrt{n}$, the quantum distributions have standard deviation fully proportional to n. In fact reasonable schemes make them approach the uniform distribution on $[-\frac{n}{\sqrt{2}}, \frac{n}{\sqrt{2}}]$, whose standard deviation is $\sqrt{\frac{1}{6}}n > 0.4n$, which is a pretty big factor of n. Thus the quantum traveler does a lot of boldly going where it hasn't been before.

Search in Big Graphs

We have seen that at least on the path graphs, a quantum random walk does a good and fast job of spreading amplitude fairly evenly among the nodes, rather than lumping it near the origin as with a classical random walk. When the graph G is bushier and has shorter distances than the path graph, we can hope to accomplish such spreading in fewer steps. Then if we are looking for a node or nodes with special properties, we can regard the evened-out amplitude as the springboard for a Grover search. If the graph's distances relative to its size are small, then we can even tolerate the size becoming super-polynomial—provided the structure remains regular enough that the action of a coin with d basic outcomes can be applied efficiently at any node of degree d.

At this point in the last main section of our final main chapter, we finally skirt full details. However, we can leverage the foregoing ideas well enough to explain a "meta-theorem" that underlies how quantum random walks serve as an algorithmic toolkit. For motivation we discuss the following problem, whose solution by Andris Ambainis [2] is most credited for commanding attention to quantum walks.

Element distinctness: Given a function $f : [n] \to [n]$, test whether the elements $f(x)$ are all distinct, i.e. f is 1-to-1.

The best-known classical method is to sort the objects according to $f(x)$, then traverse the sorted sequence to see if any entry is repeated. If we consider evaluations $f(x)$ and comparisons to take unit time, then this takes time proportional to $n \log n$.

If we wish to apply Grover search, then we are searching for a *colliding pair* (x, y), $y \neq x$, such that $f(x) = f(y)$. We can implement a Grover oracle for this test easily enough, but the problem is that there are $\binom{n}{2}$ = order-n^2 pairs to consider. Thus the square-root efficiency of Grover search will merely cancel the exponent, leaving $O(n)$ time, which is no real savings considering that the encodings of x, y and values of f really use $O(\log n)$ bits each.

The idea is to make a bigger target for the Grover search. Let $r > 2$ and consider subsets R of r-many elements. Call R a "hit" if f fails to be 1-to-1 on R. It might appear that testing this involves recursion, but (i) we will expend r quantum steps to prepare a superposition of quantum states that include the values $f(u_i)$ for every r-tuple $(u_1, \ldots, u_r) \in R$, in a way that the hit check for every R is recorded, and even

apart from this, (ii) if we count only *queries*—that is linearly superposed evaluations of f—then the test incurs no extra cost. Thus the preparation time S for the walk is reckoned as proportional to r.

However, now we have order-n^r many subsets, which seems to worsen the issue we had with Grover search on pairs. This is where quantum walks allow us to exploit three compensating factors:

1. The subsets have a greater density of "hits": any $r-2$ elements added to a colliding pair make a hit. Hence the hit density is at least
$$\frac{\binom{n-2}{r-2}}{\binom{n}{r}} = \frac{r(r-1)}{n(n-1)} \approx \left(\frac{r}{n}\right)^2.$$
In general we write E for the reciprocal of this number.
2. Rather than make a Grover oracle that sits over the entire search space, we can make a quantum coin that needs to work only over the d neighbors of a given node.
3. If we make a degree-d graph that is bushy enough, we can diffuse amplitude nearly-uniformly over the whole graph in a relatively small number of steps.

The walk steps in the last point represent an additional time factor compared to Grover search, but the other two points reduce the work in the iterations. This exposes the issues for search in big graphs. Now we are ready to outline the implementation.

In thinking about the element-distinctness example for motivation in what follows, note that the vertices of the graph G are not the individual elements x, y but rather the (unordered) r-tuples of such elements, corresponding to sets R. The adjacency relation of G in this case is between R and R' that share $r-1$ elements, so that R' is obtained by swapping one element for another. This defines the so-called **Johnson graph** $J_{n,r}$. Although all our examples are on Johnson graphs, the formalism in the next section applies more generally.

General Quantum Walk For Graph Search

The first idea in formulating quantum walks generically is that the coin space need not be coded as $\{ 1, \ldots, d \}$ but can use a separate copy of the node space. Then the expanded graph G' becomes the **edge graph** of G. We still reference nodes $X, Y, Z \ldots$ in our notation; the capital letters come from thinking of G as a big graph as above. The previous node X of a walk now at node Y is preserved as with the previous coin state in the current state (X, Y). Execution of a step to choose a next node Z is achieved by changing (X, Y) to (Y, Z). This can be done by treating the first coordinate as the "coin space" and replacing X by a random Z to make (Z, Y), then either permuting coordinates to make (Y, Z) explicitly, or leaving (Z, Y) as-is and being sure to treat the other coordinate with Y as the coin space next.

The second idea is that the goal of the search is to concentrate amplitude on one or more of the "hit" nodes. This is the reverse of the notion of a walk starting from that node or nodes, which would diffuse out to uniform probability. Reversal, however, is "no problem" for quantum computation. Hence we can start up in a stationary distribution of the classical walk on G, which by the first idea will extend naturally to the quantum walk since G' uses a copy of the nodes of G. For a d-regular graph, the uniform distribution is stationary.

Of course we cannot expect that a generic walk will run in reverse to every possible start node, since that would not be invertible. The third idea is to combine the diffusion step with a Grover-type sign flip upon detecting that the current node is a hit. This will drive amplitude onto the hit nodes. Accordingly we define a unitary operator U on our doubled-up grpah space by

$$\mathsf{U}[XY] = -1 \text{ if } X \text{ or } Y \text{ is a hit}, \quad \mathsf{U}[XY] = 1 \text{ otherwise.}$$

Now let $p_{X,Y}$ give the probability of going from X to Y in the standard classical walk on G, and $p^*_{Y,X}$ the probability in the reverse walk. For a stationary distribution π of the forward walk on $V(G)$, $p^*_{Y,X}$ obeys the equation $\pi_Y p^*_{Y,X} = \pi_X p_{X,Y}$. The walk is **reversible** if in addition $p^*X,Y = p_{X,Y}$. On a regular graph π is uniform distribution, so a reversible walk gives $p_{Y,X} = p^*_{Y,X} = p_{X,Y}$ and hence is also **symmetric**. But we can define the following "right" and "left" **reflection operators** even for a general walk:

$$\mathsf{V}_\mathsf{R}[XY,XZ] = 2\sqrt{p_{X,Y}p_{X,Z}} - \delta_{Y,Z}$$
$$\mathsf{V}_\mathsf{L}[XZ,YZ] = 2\sqrt{p^*_{Z,X}p^*_{Y,Z}} - \delta_{X,Y}.$$

Here $\delta_{Y,Z}$ is the Dirac delta functon giving 1 if $Y = Z$ and 0 otherwise. If $W \neq X$ then $\mathsf{V}_\mathsf{R}[XY,WZ] = 0$, and similarly $\mathsf{V}_\mathsf{L}[XZ,YW] = 0$.

We can define these operators more primitively by coding the walk directly. Let Y_0 be some basis state in the coin space—usually it is taken to be the node coded by the all-zero string but this is not necessary. Take P_R to be any unitary operator such that for all X and Y,

$$\mathsf{P}_\mathsf{R}[XY,XY_0] = \sqrt{p_{X,Y}}.$$

For $Z \neq Y_0$, $\mathsf{P}_\mathsf{R}[XY,XZ]$ may be arbitrary, but $\mathsf{P}_\mathsf{R}[XY,WZ] = 0$ whenever $W \neq X$. Among concrete possibilities for P_R, we could let Y_0 be the state coded by the all-1 string instead and control on it, so that the action on $Z \neq Y_0$ is the identity, or define $\mathsf{P}_\mathsf{R}[XY,XZ] = \sqrt{p_{X,Y\oplus Y_0\oplus Z}}$. It turns out not to matter. Define the projector and reflection about Y_0 by

$$\Pi_0[XY,WZ] = \begin{cases} 1 & \text{if } W = X \wedge Y = Z = Y_0 \\ 0 & \text{otherwise;} \end{cases}$$

$$\begin{aligned} \mathsf{J}_0[XY,WZ] &= 2\Pi_0[XY,WZ] - \delta_{XY,WZ} \\ &= \begin{cases} 1 & \text{if } X = W \wedge Y = Z = Y_0, \\ -1 & \text{if } X = W \wedge B = D \neq Y_0, \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

Lemma 16.1. $\mathsf{V_R} = \mathsf{P_R J_0 P_R^t}$. *Hence* $\mathsf{V_R}$ *is unitary, and similarly for* $\mathsf{V_L}$.

Proof. For any W,X,Y,Z, with lines without W multiplied by a tacit $\delta X,W$,

$$\begin{aligned} (\mathsf{P_R J_0 P_R^t})[XY,WZ] &= \sum_{A,B} \sum_{C,D} \mathsf{P_R}[XY,AB]\mathsf{J_0}[AB,CD]\mathsf{P_R^t}[CD,WZ] \\ &= \sum_{B,D} \mathsf{P_R}[XY,XB]\mathsf{J_0}[XB,XD]\mathsf{P_R^t}[XD,XZ] \\ &= \sum_{B} \mathsf{P_R}[XY,XB]\mathsf{J_0}[XB,XB]\mathsf{P_R}[XZ,XB] \\ &= \sum_{B} \mathsf{P_R}[XY,XB]\mathsf{P_R}[XZ,XB] \begin{cases} 1 & \text{if } B = Y_0 \\ -1 & \text{if } B \neq Y_0 \end{cases} \\ &= 2\mathsf{P_R}[XY,XY_0]\mathsf{P_R}[XZ,XY_0] - \sum_{B} \mathsf{P_R}[XY,XB]\mathsf{P_R}[XZ,XB] \\ &= 2\sqrt{p_{X,Y}p_{X,Z}} - \delta_{Y,Z} \quad \text{(since } \mathsf{P_R} \text{ is unitary)} \\ &= V_R[XY,WZ]. \end{aligned}$$

Since our U is also similar to the U in Chapter 15, the following blends Grover search with the quantum walk.

Definition 16.2. The **generic quantum walk** derived from the classical walk $P = (p_{X,Y})$ on the graph G is the walk on $V(G) \otimes V(G)$ defined by iterating the step operation

$$\mathsf{W_P} = \mathsf{V_L V_R},$$

and the **generic search algorithm** iterates

$$\mathsf{M_P} = \mathsf{V_L V_R U}.$$

There are allowable variations. Provided G is not bipartite, it is OK to omit a check for Y being a hit in the definition of U. That is, if we define

$$\begin{aligned} \mathsf{U_L}[XY] &= -1 \text{ if } X \text{ is a hit}, \quad \mathsf{U_L}[XY] = 1 \text{ otherwise;} \\ \mathsf{U_R}[XY] &= -1 \text{ if } Y \text{ is a hit}, \quad \mathsf{U_R}[XY] = 1 \text{ otherwise.} \end{aligned}$$

then it suffices to iterate $\mathsf{W_P U_L}$ without checking whether we found a hit in the Y coordinate. Or perhaps more elegantly, we can iterate $\mathsf{V_L U_L V_R U_R}$. We do not know

whether the concrete effects on implementations have been all worked out in the recent literature, likewise with particular choices for $\mathsf{P_R}$ and the analogous $\mathsf{P_L}$, but asymptotically there is no difference. All of them, however, involve similar left-right alternation, in line with the above analogy to AC electricity.

The "generic" walk idea allows extending the nodes of G with other data. That is, if we have a string D_X of data associated to a node X, and want to use states $\mathbf{A}(XD_X)$ in place of $\mathbf{A}(X)$ so as to encode the data in extra indices, we can re-create all the above encoding of operators with XD_X in place of X. Everything goes through as before, technically because we will have $p_{XD_X,YD_Y} = 0$ whenever YD_Y does not match the data update when going from node X with D_X to node Y.

These considerations also factor into the initial state of the walk. Let X_0 be the node coded by the all-zero index, Y_0 the same but on the right-hand side of indexing XY, and D_0 the data associated to Y_0. One possibility for the initial state $\mathbf{A}_0$ of the walk is defined by

$$\mathbf{A}_0(XY_0) = \sqrt{\pi_X},$$

and $\mathbf{A}_0(XY) = 0$ for $Y \neq Y_0$. This can be interpreted either as starting at the all-zero node with the coin in a stationary-superposed state, or having the coin initialized to "zero" with the walk initially in the classical stationary superposed state. On a d-regular graph G we always have $\pi_X = 1/d$. Now if we throw in the data, this technically means initializing the state

$$\mathbf{A}_0(XD_XY_0D_0) = \sqrt{\frac{1}{d}},$$

with $\mathbf{A}(XDYD') = 0$ whenever $D \neq D_X$, $Y \neq Y_0$, or $D' \neq D_0$. Since getting uniform superpositions is relatively easy, and the adjacency relation of G is usually simple, the difficulty in preparing $\mathbf{A}_0$ actually resides mainly in determining the associated data. The same applies to the update cost—the time taken to implement the concrete $\mathsf{V_R}$ and $\mathsf{V_L}$ via extended versions of $\mathsf{P_R}$ and $\mathsf{P_L}$ is mainly for producing the data associated with the node Y traversed from X.

Finally, in the concrete version of the "flip" U, there is the cost of checking the associated data to see if the current node is a "hit." In order for these steps and checks to be done in superposition, they must be coded for accomplishment by linear transformations. Thus far this is done by jockeying indices. Our point is not so much to argue that our notation is more intuitive or less cumbersome than standard notation—such as $\sum_X |XY_0\rangle\langle XY_0|$ for the projector onto $\mathbb{C}^{V(G)} \otimes |Y_0\rangle$, or $|X\rangle|D_X\rangle$ to carry along the associated data—but rather to explore lower-level details and suggest possible alternate encodings.

In presenting algorithms for some specific problems, this is the point where we need to skimp on full details in order to highlight the generic intuition. At least it comes in the last section of the last chapter of our text.

Quantum Walk Search Algorithm and Complexity Toolkit

The last factor for efficient quantum walks is that the underlying graph G be sufficiently "bushy" relative to its size N. The formal notion is that G be an **expander**, meaning that there is an appreciably large value $h(G)$ such that for all sets T of at most $N/2$ nodes, there are at least $h(G)|T|$-many edges going to nodes outside T. This entails that there are at least $h(G)|T|/d$ different nodes outside T that can be reached in one step, but it is separately significant to have many edges that can diffuse amplitude into these neighboring nodes. An important lower bound on $h(G)$ is provided by the difference between the largest eigenvalue and the second largest absolute value of the adjacency matrix of G, which is denoted by $\Delta(G)$ and called the *eigenvalue gap*. The bound is

$$\frac{1}{2}\Delta(G) \leq h(G).$$

For a d-regular graph we write $D = d/\Delta(G)$, which is actually the reciprocal of the eigenvalue gap (commonly written δ) of the stochastic matrix of the underlying standard classical walk on G.

From the discussions here and in the last section, we have isolated five important parameters that affect the performance of concrete implementations of the generic quantum walk:

1. E: the reciprocal of the density of "hit" nodes in the graph G;
2. D: the degree divided by the eigenvalue gap of G;
3. S: the setup cost for preparing the initial quantum state, including associating to each graph node any additional data such as function values;
4. U: the update cost of executing each step of the quantum walk;
5. C: the cost of checking locally to see whether a node is a hit, if not already marked during setup.

The E and D parameters have been expressed in a way that requires no further use of the size n or degree d of the graph, though they and d are implicitly functions of n. The following "meta-theorem" expresses how quantum random walks can be used as an algorithm toolkit for search problems.

Theorem 16.1. *For any search problem on a uniform family of undirected graphs G_n with parameters $E(n)$ and $D(n)$, and any additive cost measure (such as time or the number of function queries), one can design a quantum walk with setup, update, and checking phases, such that if the respective costs of these phases are bounded by $S(n)$, $U(n)$, and $C(n)$ respectively, then the overall cost to achieve correctness probability at least $3/4$ is bounded above by a constant times*

$$S(n) + \sqrt{E(n)}(U(n)\sqrt{D(n)} + C(n)). \tag{16.1}$$

The proof given by Magniesz et al. [7] has two stages, one involving an extra factor of $\log E(n)$, and then a recursion to eliminate this factor. Also for reasons stated above, we omit it here.

The way to apply this theorem is to find appropriate graphs on which to model the search problem at different problem sizes n, get the $D(n)$ and $E(n)$ bounds, and design additional features associated to the nodes (if needed) to balance out $S(n)$, $U(n)$, and $C(n)$. Here is how this theorem plays out in some examples.

Grover search. Here G_n is the complete graph on n vertices. Since G_n has degree $n-1$ and the next-highest eigenvalue is known to be 1, the gap is $n-2$. Hence $D(n) = (n-1)/(n-2) \approx 1$. Suppose at least k of the n nodes are hits. Then $E(n) \leq n/k$. The setup takes one step of parallel Hadamard gates, the update takes one query step, and the checking is reflected in one flip step. Hence the time is $O(\sqrt{n/k})$. This is $O(\sqrt{n})$ if all we know is that there is at least one hit. Note that the quantum walk architecture for controlling the search automatically guarantees faster time if there are many hit nodes.

Element distinctness. Above we have given a parameter $r(n)$ and its effect on hit density as motivation, but we haven't defined particular graphs G_n to use. Several kinds of graphs works, but the Johnson graphs $J_{n,r}$ are the original and most popular choice. Recall that $J_{n,r}$ has a node for each $R \subset [n]$ of size r, and edges connecting R, R' when they have $r-1$ elements in common; note that the complete graph equals $J_{n,1}$. The degree of $J_{n,r}$ is $r(n-r)$ since every edge involves deleting one element u from R and swapping in one element v not in R. The second eigenvalue is $r(n-r)-n$, so the gap is n provided $r \geq 2$. Thus $D(n) = \frac{r(n-r)}{n} = r - r^2/n < r$. From the above hit density we have $E(n) = (n/r)^2$. The goal is to choose r as a function of n to balance and minimize the total cost.

The quantum algorithm needs to initialize not only a uniform superposition over nodes $R = (u_1, \ldots, u_r)$, but also the values $f(u_1), \ldots, f(u_r)$ for its elements—and to sort the latter locally. This requires r linearly-superposed queries to r, and $O(r)$ time if we ignore log factors, as we are already doing for the binary encoding of the u_i. This gives setup cost $S(n) = O(r)$. The update needs to remove the value $f(u_i)$ and add a value $f(v)$ when u_i is swapped out for v in the walk, which takes 2 queries. Again since we are ignoring the $\log n$ size of encodings, this gives $U(n) = O(1)$ for time and queries both. From the formula (16.1) we obtain overall cost of order

$$r + \frac{n}{r}(2\sqrt{r} + 0) = O(r + \frac{n}{\sqrt{r}}).$$

This is balanced with $r = n^{2/3}$ giving time $O(n^{2/3})$. It is known that the same order of queries are necessary, so this bound is asymptotically tight.

Checking Subgraph Triangle Incidence. Given an m-node subgraph H of an n-node graph G, does H have an edge of a triangle in G? Given a fixed edge (u,v) in H, we could do a $\sqrt{n}$-time Grover search for w such that (u,w) and (v,w) are also edges. But iterating this through possibly order-m^2 edges in H is clearly prohibitive.

Instead for each w, we do a quantum random walk to find (u,v). We take $r = m^{2/3}$, and define a subset R of the nodes of H to be a hit if it contains a suitable edge (u,v). Using the Johnson graph $J_{m,r}$ and data consisting of whether $(u,v),(u,w),(v,w)$ are all edges, all the parameters are the same as for element distinctness, so the time is $O(m^{2/3})$. This becomes the checking time for the Grover search, so the overall time is $O(n^{1/2}m^{2/3})$. We can amplify both the check and the Grover success probability to be at least $5/6$, so as to yield $2/3$ on the whole.

Finding a Triangle. Now we call an m-node subset R of $V(G)$ a hit if the induced subgraph H includes an edge of a triangle in G. The E and D will hence be the same as for element-distinctness with "m" replacing "r": $E(n) = (n/m)^2$, $D(n) \leq m$. We may use the previous item to obtain checking cost $C(n) = O(n^{1/2}m^{2/3})$. The one thing that is different is that the setup and update costs are higher. The setup needs to encode the entire adjacency matrix of H, and when the update swaps in a vertex v' and swaps out a v, it needs to update the $m-1$ adjacencies of v' while erasing those of v. Thus we have $S(n) = O(m^2)$ and $U(n) = O(m)$. The formula for the overall cost becomes:

$$O(m^2) + \frac{n}{m}(O(m\sqrt{m} + n^{1/2}m^{2/3})).$$

This time, the setup cost drops out of the equation—the balancing is between the update and checking cost, and is achieved when $m^{3/2} = n^{1/2}m^{2/3}$, that is when $m^{5/6} = n^{1/2}$, so $m = n^{3/5}$. This results in the overall time

$$O(n^{6/5} + n^{2/5}n^{9/10}) = O(n^{13/10}).$$

The best known classical algorithm for finding a triangle takes the adjacency matrix A and forms $A^2 + A$; it hence runs in time $O(n^{\omega})$ where the exponent ω of matrix multiplication is known to be at most 2.372. Hence if the quantum time were just a little lower, say $O(n^{1.18})$ instead of $O(n^{1.3})$, then the speedup over the best known classical time would exceed the generic quadratic improvement from Grover search—and indeed would exceed the quadratic improvement between general quantum and classical random walks. Thus the question of further quantum improvements may be tied to the possibility of advances in classical matrix multiplication. No quantum lower bound higher than linear is known for this problem, while no lower bound higher than 2 is known for ω.

Summary

The key to understanding a classical random walk in a graph is that it is defined locally. That is, for every node u, and every neighbor v one can walk to in one step, define $A(u,v)$ to be the probability of choosing to walk to v. Then $A^2(u,w)$ is the probability of reaching node w in exactly *two* steps given that you started at u, and this carries over to any power: $A^k(u,v)$ gives the probability of ending at v in exactly

k steps, given that you started at u. Thus random walks in graphs are just linear algebra.

The nice thing about quantum random walks is that they too are just linear algebra, except the entries are complex numbers whose squared absolute values become the probabilities. The use of matrix multiplication and summing over paths is the same, except that what gets summed are possibly-complex amplitudes rather than probabilities. The difference—maybe not so nice—is that these amplitudes can cancel, thus giving *zero* probability for certain movements from u to w that would be possible in the classical case. This enables piling higher probabilities on other movements in ways that cannot be directly emulated classically. But again the key is that the definition is local for each "node" which is just a basis state, and gives you a matrix.

We have explained quantum coins with a "hidden-variables" mentality, and it is dangerous to combine that with the idea of "local." However, this view has not tried to obscure non-local phenomena such as the way certain superpositions, such as the two coin states without the phase shift, can render certain far locations unreachable. We have really tried to emphasize the role of linear algebra, and combinatorial elements such as the enlarged graph G' and the counting of HH substrings. It may seem strange to picture that Nature tracks the parity of substring counts, and formalisms we haven't touched such as calculating in the Fourier-transformed space of the walk branches avoid an exponential amount of work while at least giving good approximations, but this is evidently the effect of what Nature does.

Quantum walks have yielded improved and optimal algorithms for certain decision problems. Beyond that, they exemplify the idea that quantum computation is about creating rich quantum states, one that cannot be readily simulated by classical means, by which novel solutions can be obtained.

Problems

16.1. Verify the probabilities obtained for the 4-step walk on the graph with nodes labeled -4 to $+4$. What happens when this walk is started in the state $(\eta_0+\eta_1)/\sqrt{2}$, that is without the phase shift on the "initial heads" part of the walk?

16.2. Work out the amplitudes and probabilities for the 3-step walk with the J matrix in place of the Hadamard action on the coin space.

16.3. Can you devise a combinatorial rule for figuring the destination and amplitude of basic paths under the J matrix, in terms of the binary code of the path, analogous to the counting of HH substrings for the H matrix?

16.4. Can you use the combinatorial rule for the H matrix to prove that the amplitude of $(0,0)$ for an n-step walk (n even) on the infinite path graph is bounded by $1/\sqrt{2n}+\varepsilon$, for suitable $\varepsilon > 0$?

16.5. Show by direct calculation that the matrix V_R is unitary.

16.6. Work out the analogous decomposition of the matrix $\mathsf{V_L}$, using an arbitrary fixed basis state X_0 of the "left" node space. What happens if X_0 is not a basis state?

16.7. In the algorithm for element-distinctness, suppose we use recursion to check whether a set of r nodes has two non-distinct function values and thus constitutes a "hit." Do you get the same $O(n^{2/3})$ running time, or something less?

16.8. Sketch how to implement the element-distinctness algorithm on the *Hamming graphs* $H_{n,r}$, whose vertices are r-tuples from $[n]$, and whose edges connect r-tuples that differ in one place. Are all of $D(n)$, $E(n)$, $S(n)$, $U(n)$,a nd $C(n)$ asymptotically the same as before?

16.9. Sketch a quantum search algorithm that given three $n \times n$ matrices A, B, C finds i, j such that $\sum_k A_{ik} B_{kj} \neq C_{ij}$ if such a pair exists, else outputs "accept." You may consider arithmetic operations to be unit time, and need only count steps that query matrix entries. The goal is cost $O(n^{5/3})$.

16.10. Suppose you can make superposed black-box queries to a binary operation $\circ$ on $[n]$ whose values lie in $[k]$. Sketch a quantum algorithm to find $a, b, c \in [n]$ such that $(a \circ b) \circ c$ is *not* equal to $a \circ (b \circ c)$, if any such "non-associative triple" exists. If $k = O(1)$, what is the running time of the algorithm? Note that the data associated to a Johnson-graph node $(u_1, \ldots, u_r)$ can preserve values of $\circ$ on arguments in $[k]$ as well as the u_i.

Notes

Most of this chapter is based on the survey papers by Kempe [6] and Santha [9], also with reference to Magniesz et al. [7]. The last two problems also come from [9], while the problem about element distinctness on the Hamming graphs is based on [3].