Quantum algorithms (CO 781, Winter 2008)
Prof. Andrew Childs, University of Waterloo

LECTURE 16: Quantum walk algorithm for element distinctness

In this lecture we will discuss the algorithm that cemented the importance of quantum walk as a tool for quantum query algorithms: Ambainis's algorithm for the element distinctness problem. The key new conceptual idea of this algorithm is to consider walks that store information obtained from many queries at each vertex, but that do not require many queries to update this information for an adjacent vertex.

Element distinctness In the *element distinctness problem*, we are given a black-box function $f : \{1,\dots,n\} \to S$, where S is some finite set. The goal is to determine whether there are two distinct inputs $x, y \in \{1,\dots,n\}$ such that $f(x) = f(y)$.

It is clear that a classical algorithm must make $\Omega(n)$ queries to solve the problem, since deciding whether there is such a pair is at least as hard as unstructured search (suppose we give the additional promise promise that if there is a pair, it will be with $x = 1$ for which $f(1) = 1$; then we must search for a $y \in \{2,\dots,n\}$ for which $f(y) = 1$). By the same argument, there is a quantum lower bound for element distinctness of $\Omega(\sqrt{n})$.

There is a simple quantum algorithm that uses Grover's algorithm recursively to improve upon the trivial running time of $O(n)$. To see how this algorithm works, first consider the following subroutine. Query f in ℓ randomly chosen places, and check whether one of these ℓ place if half of a pair of inputs that map to the same value by performing a Grover search on the remaining $n - \ell$ inputs. The initial setup takes ℓ queries, and the Grover search takes $O(\sqrt{n-\ell}) = O(\sqrt{n})$ queries, for a total of $\ell + O(\sqrt{n})$. This subroutine fails most of the time, since it is likely that the random choice of ℓ inputs will be unlucky, but it succeeds with probability at least ℓ/n. To boost the success probability, we can use amplitude amplification,[*] which takes $O(\sqrt{n/\ell})$ steps to boost the success probability to a constant. Overall, we can obtain success probability $\Omega(1)$ using

$$(\ell + \sqrt{n})\sqrt{n/\ell} = \sqrt{n\ell} + n/\sqrt{\ell} \tag{1}$$

queries. To optimize the query complexity, we set the two terms to be equal, giving $\ell = \sqrt{n}$ and hence a query complexity of $O(n^{3/4})$. (Note that an analysis of the *running time* of this algorithm would include extra logarithmic factors, since the inner use of Grover's algorithm must check whether an element against ℓ queried function values, which can be done in time $O(\log \ell)$ provided S is ordered and we initially sort the queried values.)

So far, we have a quantum upper bound of $O(n^{3/4})$, and a quantum lower bound of $\Omega(n^{1/2})$. It turns out that both of these can be improved. On the lower bound side, Aaronson and Shi proved an $\Omega(n^{1/3})$ lower bound for the closely-related *collision problem*, in which the goal is to

[*]Amplitude amplification can be viewed in the context of quantum walk search as follows. Suppose we have a (classical or quantum) procedure that produces a correct answer with probability p (i.e., with an amplitude of magnitude $\sqrt{p}$ in the quantum case). From this procedure we can define a two-state Markov chain that, at each step, moves from the state where the answer is not known to the state where the answer is known with probability p, and then remains there. This walk has the transition matrix

$$P' = \begin{pmatrix} 1-p & 0 \\ p & 1 \end{pmatrix},$$

so $P_M = 1 - p$, giving a quantum hitting time of $O(1/\sqrt{1 - \|P_M\|}) = O(1/\sqrt{p})$.

distinguish one-to-one from two-to-one functions. This implies an $\Omega(n^{2/3})$ lower bound for element distinctness by the following reduction. Suppose we randomly choose $\sqrt{n}$ inputs of the collision problem function and run the element distinctness algorithm on them. If the function is two-to-one, then there is some pair of elements in this set mapping to the same value with high probability (by the birthday problem), which the element distinctness algorithm will detect. Hence a k-query element distinctness algorithm implies an $O(\sqrt{k})$-query collision algorithm; or equivalently, a k-query collision lower bound implies an $\Omega(k^2)$ element distinctness lower bound.

Now the question remains, can we close the gap between the $O(n^{3/4})$ upper bound and this $\Omega(n^{2/3})$ lower bound? Ambainis's quantum walk algorithm does exactly this.

Quantum walk algorithm The idea of Ambainis's quantum walk algorithm is to quantize a walk on the Johnson graph $J(n,m)$, where m is chosen appropriately. This graph has $\binom{n}{m}$ vertices corresponding to subsets of $\{1,2,\dots,n\}$ of size m, and two vertices are connected by an edge if the subsets differ in exactly one element. Notice that this is a regular graph of degree $m(n-m)$.

At each vertex of the graph, we store the values of the function at the inputs in the subset it describes. In other words, representing the subset $\{x_1,x_2,\dots,x_m\} \subset \{1,2,\dots,n\}$ by the state

$$|x_1,x_2,\dots,x_m\rangle \tag{2}$$

(say, with $x_1 < x_2 < \cdots < x_m$ so that the list is unique), we represent the corresponding vertex by

$$|x_1,x_2,\dots,x_m,f(x_1),f(x_2),\dots,f(x_m)\rangle. \tag{3}$$

To prepare such states, we must query the black-box function. In particular, to prepare an initial superposition over vertices of this graph takes m queries. However, we can move from one vertex to an adjacent vertex using only two queries: to replace x_i by x_j, we use one query to erase $f(x_i)$, and amother to compute $f(x_j)$.

In this search problem, the marked vertices are those corresponding to subsets containing some $x \neq y$ with $f(x) = f(y)$. Notice that, given the stored function values, we can check whether we are at a marked vertex with no additional queries. The total number of marked vertices (in the case where the elements are not all distinct) is at least $\binom{n-2}{m-2}$, so the fraction of marked vertices is

$$\begin{aligned}\epsilon &\geq \frac{\binom{n-2}{m-2}}{\binom{n}{m}} &(4)\\ &= \frac{(n-2)!}{(m-2)!(n-m)!}\cdot\frac{m!(n-m)!}{n!} &(5)\\ &= \frac{m(m-1)}{n(n-1)}. &(6)\end{aligned}$$

To analyze the walk, we also need the eigenvalues of the Johnson graph. The Johnson graph is an example of a *distance-regular graph*, a kind of graph with a highly regular structure that has a nice algebraic description. Indeed, the eigenvalues of this graph can be computed exactly (see for example the notes by Chris Godsil on association schemes for the details of this calculation). The eigenvalues of the adjacency matrix are given by

$$m(n-m) - i(n+1-i) \qquad\qquad i = 0,1,\dots,m \tag{7}$$

(with certain multiplicities that we do not need to know). The largest eigenvalue is of course the degree, $m(n-m)$, corresponding to $i = 0$. The gap between the largest and second largest eigenvalue (with $i = 1$) is n. (The most negative eigenvalue, with $i = m$, is $m(n - m) - m(n + 1 - m) = -m$, so the gaps to absolute values of negative eigenvalues are all much smaller.) Normalizing the eigenvalues by the degree, we have a spectral gap of

$$\delta = \frac{n}{m(n-m)}. \tag{8}$$

Finally, how many queries does this algorithm use? Taking into account the initial m queries used to prepare the starting state and the 2 queries per step of the walk, we have a total number of queries

$$m + 2 \cdot O\left(\frac{1}{\sqrt{\delta\epsilon}}\right) = m + O\left(\sqrt{\frac{m(n-m)}{n}}\sqrt{\frac{n(n-1)}{m(m-1)}}\right) \tag{9}$$

$$= m + O\left(\sqrt{\frac{(n-m)(n-1)}{m-1}}\right) \tag{10}$$

$$= m + O\left(\frac{n}{\sqrt{m}}\right). \tag{11}$$

Again we can set the two terms equal to optimize the performance. We have $m^{3/2} = O(n)$, so we should take $m = \Theta(n^{2/3})$. Then the total number of queries is $O(n^{2/3})$, which matches the lower bound, and hence is optimal.

Note that for the classical random walk search algorithm that we have quantized, the corresponding query complexity is $m + O(n^2/m)$, which is optimized by $m = n$. This gives no improvement over querying every input, as we knew must be the case.

Quantum walk search algorithms with auxiliary data Algorithms based on similar ideas turn out to be useful for a wide variety of problems, including deciding whether a graph contains a triangle (or satisfies some other monotone graph property), checking matrix multiplication, and testing whether a group is abelian. In general, as in the element distinctness case, we may need to store some data at each vertex, and we need to take into account the operations on this data when analyzing the walk.

Suppose we have a setup cost S, a cost U to update the state after one step of the walk, and a cost C to check whether a vertex is marked. For example, in Ambainis's algorithm for element distinctness, we had

$$S = m \qquad \text{to query } m \text{ positions} \tag{12}$$

$$U = 2 \qquad \text{to remove one of the items and add another} \tag{13}$$

$$C = 0 \qquad \text{since the function values for the subset are stored.} \tag{14}$$

In general, there is an algorithm to solve such a problem with total cost

$$S + \sqrt{\frac{1}{\delta\epsilon}}(U + C). \tag{15}$$

It turns out that for some problems, when the checking cost C is much larger than the update cost U, it is advantageous to take many steps of the walk on the unmarked graph before performing a phase flip on the marked sites. This is actually how Ambainis's algorithm originally worked, though for element distinctness it is not actually necessary. Using this idea, one can give a general quantum walk search algorithm with total cost

$$S + \sqrt{\frac{1}{\epsilon}} \left(\frac{1}{\sqrt{\delta}} U + C \right). \tag{16}$$

In fact, it is also possible to modify the general algorithm so that it finds a marked item when one exists.