Quantum algorithms (CO 781/CS 867/QIC 823, Winter 2013)
Andrew Childs, University of Waterloo

LECTURE 17: Learning graphs

While span programs provide a powerful tool for proving upper bounds on quantum query complexity, they can be difficult to design. The model of learning graphs, introduced by Belovs, is a restricted class of span programs that are more intuitive to design and understand. This model has led to improved upper bounds for various problems, such as subgraph finding and k-distinctness.

Learning graphs and their complexity

A learning graph for an n-bit oracle is a directed, acyclic graph whose vertices are subsets of $[n]$, the set of indices of input bits. Edges of the learning graph can only connect vertices $\sigma \subset [n]$ and $\sigma \cup \{i\}$ for some $i \in [n] \setminus \sigma$. We interpret such an edge as querying index i, and we sometimes say that the edge $(\sigma, \sigma \cup \{i\})$ *loads* index i. Each edge e has an associated weight $w_e > 0$. We say that a learning graph computes $f\colon S \to \{0,1\}$ (where $S \subseteq \{0,1\}^n$) if, for all x with $f(x) = 1$, there is a path from $\varnothing$ to a 1-certificate for x (a subset of indices σ such that $f(y) = f(x)$ for all y such that $x_\sigma = y_\sigma$, where x_σ denotes the restriction of x to the indices in σ).

Associated to any learning graph for f is a complexity measure $\mathcal{C} = \sqrt{\mathcal{C}_0 \mathcal{C}_1}$, the geometric mean of the 0-complexity $\mathcal{C}_0$ and the 1-complexity $\mathcal{C}_1$. The 0-complexity is simply $\mathcal{C}_0 := \sum_e w_e$, where the sum is over all edges in the learning graph.

The definition of the 1-complexity is somewhat more involved. Associated to any x with $f(x) = 1$, we consider a *flow* in the learning graph, which assigns a value p_e to each edge so that for any vertex, the sum of all incoming flows equals the sum of all incoming flows. There is a unit flow coming from vertex $\varnothing$; this is the only source. A vertex can be a sink if and only if it contains a 1-certificate for x. The complexity of any such flow is $\sum_e p_e^2/w_e$. (Note that $w_e > 0$ for any edge in a learning graph, although we also have the possibility of omitting edges.) The complexity $\mathcal{C}_1(x)$ is the smallest complexity of any valid flow for x. Finally, we have $\mathcal{C}_1 := \max_{x \in f^{-1}(1)} \mathcal{C}_1(x)$, the largest complexity of any 1-input.

Unstructured search

Perhaps the simplest example of a learning graph is for the case of unstructured search. The learning graph simply loads an index. In other words, there is an edge of weight 1 from $\varnothing$ to $\{i\}$ for each $i \in [n]$. Clearly, we have $\mathcal{C}_0 = n$. To compute the 1-complexity, consider the input $x = e_i$ for some $i \in [n]$. For this input there is a unique 1-certificate, namely $\{i\}$. The only possible flow is the one with unit weight on the edge from $\varnothing$ to $\{i\}$. This gives $\mathcal{C}_1(e_i) = 1$ for all i, so $\mathcal{C}_1 = 1$. Therefore the complexity of this learning graph is $\mathcal{C} = \sqrt{\mathcal{C}_0 \mathcal{C}_1} = \sqrt{n}$.

It is not hard to see that the same learning graph works for the total function OR: for each x with $f(x) = 1$, we can send all flow to any particular i for which $x_i = 1$.

From learning graphs to span programs

We now show that every learning graph implies a dual adversary solution of the same complexity, so that the learning graph complexity is an upper bound on quantum query complexity, up to constant factors.

We construct vectors $|v_{x,i}\rangle$ for all $x \in S$. These vectors consist of a block for each vertex σ of the learning graph, with the coordinates within each block labeled by possible assignments of the bits in σ. Since we fix a particular index i, we can think of the blocks as labeling edges $e_{\sigma,i} := (\sigma, \sigma \cup \{i\})$. We define

$$|v_{x,i}\rangle = \begin{cases} \sum_\sigma \sqrt{w_{e_{\sigma,i}}} |\sigma, x_\sigma\rangle & \text{if } f(x) = 0 \\ \sum_\sigma \frac{p_{e_{\sigma,i}}}{\sqrt{w_{e_{\sigma,i}}}} |\sigma, x_\sigma\rangle & \text{if } f(x) = 1 \end{cases} \tag{1}$$

where the sums only run over those $\sigma \subset [n]$ for which $e_{\sigma,i}$ is an edge of the learning graph.

It is easy to check that this definition satisfies the dual adversary constraints. For any $x, y \in S$ with $f(x) = 0$ and $f(y) = 1$, we have

$$\sum_{i\colon x_i \neq y_i} \langle v_{x,i} | v_{y,i} \rangle = \sum_{i\colon x_i \neq y_i} \sum_\sigma \sqrt{w_{e_\sigma,i}} \frac{p_{e_{\sigma,i}}}{\sqrt{w_{e_{\sigma,i}}}} \langle x_\sigma | y_\sigma \rangle \tag{2}$$

$$= \sum_{i\colon x_i \neq y_i} \sum_{\sigma\colon x_\sigma = y_\sigma} p_{e_{\sigma,i}}. \tag{3}$$

Now observe that the set of edges $\{e_{\sigma,i}\colon x_\sigma = y_\sigma, x_i \neq y_i\}$ forms a cut in the graph between the vertex sets $\{\sigma\colon x_\sigma = y_\sigma\}$ and $\{\sigma\colon x_\sigma \neq y_\sigma\}$ Since $\varnothing$ is in the former set and all sinks are in the latter set, the total flow through the cut must be 1.

Recall that we do not have to satisfy the constraint for $f(x) = f(y)$ since there is a construction that enforces this condition without changing the complexity provided the condition for $f(x) \neq f(y)$ is satisfied.

It remains to see that the complexity of this dual adversary solution equals the original learning graph complexity. For $b \in \{0, 1\}$, we have

$$C_b = \max_{x \in f^{-1}(b)} \sum_{i \in [n]} \| |v_{x,i}\rangle \|^2 \tag{4}$$

$$= \max_{x \in f^{-1}(b)} \sum_{i \in [n]} \sum_\sigma \begin{cases} w_{e_{\sigma,i}} & \text{if } b = 0 \\ \frac{p^2_{e_{\sigma,i}}}{w_{e_{\sigma,i}}} & \text{if } b = 1 \end{cases} \tag{5}$$

$$= \begin{cases} \mathcal{C}_0 & \text{if } b = 0 \\ \max_{x \in f^{-1}(1)} \mathcal{C}_1(x) & \text{if } b = 1 \end{cases} \tag{6}$$

$$= \mathcal{C}_b. \tag{7}$$

Therefore $\sqrt{C_0 C_1} = \sqrt{\mathcal{C}_0 \mathcal{C}_1} = \mathcal{C}$ as claimed. In particular, $\mathrm{Adv}^{\pm}(f) \leq \mathcal{C}$, so $Q(f) = O(\mathcal{C})$.

Learning graphs are simpler to design than span programs: the constraints are automatically satisfied, so one can focus on optimizing the objective value. In contrast, span programs have exponentially many constraints (in n, if f is a total function), and in general it is not obvious how to even write down a solution satisfying the constraints.

Note, however, that learning graphs are not equivalent to general span programs. For example, learning graphs (as defined above) only depend on the 1-certificates of a function, so two functions with the same 1-certificates have the same learning graph complexity. The 2-threshold function (the symmetric Boolean function that is 1 iff two or more input bits are 1) has the same certificates as element distinctness, so its learning graph complexity is $\Omega(n^{2/3})$, whereas its query complexity is $O(\sqrt{n})$. This barrier can be circumvented by modifying the learning graph model, but even such variants are apparently less powerful than general span programs.

Element distinctness

We conclude by giving another simple example of a learning graph, one for element distinctness. (This requires generalizing learning graphs to non-Boolean input alphabet, but this generalization is straightforward.) We assume for simplicity that there is a unique collision—in fact, the analysis of the learning graph works for the general case by simply fixing one particular collision when designing a flow.

A convenient simplification is to break up the learning graph into k *stages*, which are simply subsets of the edges. To compute the complexity of a stage, we sum over only the edges in that stage. It is easy to see that there is a learning graph whose complexity is at most the sum of the complexities of the stages times the square root of the number of stages (which we will take to be constant). Let $\mathcal{C}_b^j$ denote the b-complexity of stage j. By dividing the weight of every edge in stage j by $\mathcal{C}_0^j$, we send $\mathcal{C}_0^j \to 1$ and $\mathcal{C}_1^j \to \mathcal{C}_0^j \mathcal{C}_1^j$. Then the total 0-complexity becomes $\mathcal{C}_0 = k$ and the total 1-complexity becomes

$$\mathcal{C}_1 = \sum_{j=1}^{k} \mathcal{C}_0^j \mathcal{C}_1^j \leq \left(\sum_{j=1}^{k} \sqrt{\mathcal{C}_0^j \mathcal{C}_1^j} \right)^2 \tag{8}$$

(since the 1-norm upper bounds the 2-norm), so $\mathcal{C} \leq \sqrt{k} \sum_{j=1}^{k} \sqrt{\mathcal{C}_0^j \mathcal{C}_1^j}$.

Another useful modification is to allow multiple vertices corresponding to the same subset of indices. It is straightforward to show that such learning graphs can be converted to span programs at the same cost, or to construct a new learning graph with no multiple vertices and the same or better complexity.

The learning graph for element distinctness has three stages. For the first stage, we load subsets of size $r-2$. We do this by first adding edges from $\varnothing$ to $\binom{n-i}{r-3}$ copies of vertex $\{i\}$, so that there are $\sum_{i=1}^{n} \binom{n-i}{r-3} = \binom{n}{r-2}$ singleton vertices. Then, from each of these singleton vertices, we load the remaining indices of each possible subset of size $r-2$, one index at a time. Every edge has weight 1. Then the 0-complexity of the first stage is $(r-2)\binom{n}{r-2}$.

To upper bound the 1-complexity of the first stage, we route flow only through vertices that do not contain the indices of a collision, sending equal flow of $\binom{n-2}{r-2}^{-1}$ to all subsets of size $r-2$. This gives 1-complexity of at most $(r-2)\binom{n-2}{r-2}\binom{n-2}{r-2}^{-2} = (r-2)\binom{n-2}{r-2}^{-1}$.

Overall, the complexity of the first stage is at most

$$\sqrt{(r-2)^2 \binom{n}{r-2} \binom{n-2}{r-2}^{-1}} = (r-2)\sqrt{\frac{n(n-1)}{(n-r+2)(n-r+1)}} = O(r). \tag{9}$$

The second and third stages each include all possible edges that load one additional index from the terminal vertices of the previous stage. Again every edge has unit weight. Thus, the 0-complexity is $(n-r+2)\binom{n}{r-2}$ for the second stage and $(n-r+1)\binom{n}{r-1}$ for the third stage. We send the flow through vertices that contain the collision pair (namely, that contain the first index of a collision in the second stage and the second index of a collision in the third stage). Thus, the 1-complexity is $\binom{n-2}{r-2}\binom{n-2}{r-2}^{-2} = \binom{n-2}{r-2}^{-1}$ in both the second and the third stages. This gives total complexity

$$\sqrt{(n-r+2) \binom{n}{r-2} \binom{n-2}{r-2}^{-1}} = O(\sqrt{n}) \tag{10}$$

for the second stage and

$$\sqrt{(n-r+1)\binom{n}{r-1}\binom{n-2}{r-2}^{-1}} = \sqrt{\frac{n(n-1)}{(r-1)}} = O(n/\sqrt{r}) \tag{11}$$

for the third stage.

Overall, the complexity is $O(r + \sqrt{n} + n/\sqrt{r})$. This optimized by choosing r to equate the first and last terms, giving $r = n^{2/3}$. The overall complexity is $O(n^{2/3})$, matching the optimal quantum walk search algorithm.

Other applications

The simple examples discussed above only involve problems for which the optimal query complexity was previously known using other techniques. However, several new quantum query upper bounds have been given using learning graphs. These include improved algorithms for the triangle problem (and more generally, subgraph finding, with an application to associativity testing) and the k-distinctness problem. (Note that the algorithm for k-distinctness uses a subtle modification of the learning graph framework.) Unfortunately, the details of these algorithms are beyond the scope of the course.