CME 305: Discrete Mathematics and Algorithms

Instructor: Professor Aaron Sidford (sidford@stanford.edu)

February 6, 2018

Lecture 9 - Matrix Multiplication Equivalences and Spectral Graph Theory [1]

In the last lecture we introduced fast matrix multiplication, i.e. algorithms for multiplying two $n \times n$ matrices in faster than the trivial $O(n^3)$ time this takes to multiply two $n \times n$ matrices and we saw how to use these algorithms to compute the inverse of a matrix and therefore determine whether or not the determinant of a matrix is 0. In this lecture, we discuss several equivalences between matrix multiplication, i.e. we show how to both reduce problems to fast matrix multiplication and reduce fast matrix multiplication to other problems with only constant loss in running time. Moreover, we discuss boolean matrix multiplication (BMM) and discuss its connection to combinatorial problems. This completes our study of matrix multiplication in the course and we conclude the lecture by starting our coverage of different algebraic graph techniques in the area of spectral graph theory.

1 Matrix Multiplication Equivalences

In the last lecture we defined to fast matrix multiplication problem defined as follows.

Definition 1 (Fast Matrix Multiplication (FMM))**.** The *fast matrix multiplication problem* is as follows, given two square matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times n}$ compute $\mathbf{C} = \mathbf{AB}$. The best known value of ω for which this can be done in $O(n^\omega)$ is known as the *matrix multiplication constant.*.

As we discussed, the current best known bounds on ω are $\omega \in [2, 2.373]$ and it is open whether or not $\omega = 2$ or $\omega > 2$. We also saw how this allowed us to compute $\mathbf{A}^{-1}$ for $\mathbf{A} \in \mathbb{R}^{n \times n}$ in $O(n^\omega)$. Here we discuss a broader set of equivalences regarding fast matrix multiplication. First we show that not only can we use FMM to compute inverses, but given any inverse algorithm we can use that to compute FMM.

Lemma 2. *If for all $\mathbf{A} \in \mathbb{R}^{n \times n}$ we can compute $\mathbf{A}^{-1}$ in $O(n^\alpha)$ then $\omega \leq \alpha$.*

Proof. Let $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times n}$ be arbitrary and let

$$\mathbf{D} = \begin{pmatrix} \mathbf{I} & \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{B} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{pmatrix} \in \mathbb{R}^{3n \times 3n} \text{ and } \mathbf{E} = \begin{pmatrix} \mathbf{I} & -\mathbf{A} & \mathbf{AB} \\ \mathbf{0} & \mathbf{I} & -\mathbf{B} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{pmatrix}.$$

Now, direct calculation shows that $\mathbf{DE} = \mathbf{I}$ and therefore $\mathbf{E} = \mathbf{D}^{-1}$. Since we can compute $\mathbf{D}^{-1}$ in $O((3n)^\alpha) = O(n^\alpha)$ and this immediately yields $\mathbf{AB}$ we have the desired result. □

This lemma in conjunction with the analysis lecture shows that we actually could have defined ω as the value for which we can compute the inverse of a matrix in $O(n^\omega)$ and this would have been equivalent. There are multiple results showing equivalence between matrix multiplication and other results and we give a few examples below.

Theorem 3. *If any of the following can be solved in $O(n^\alpha)$ in the worst case then all can be solved in $O(n^\alpha)$ in the worst case*

- *Compute $\mathbf{C} = \mathbf{AB}$ for $\mathbf{A} \in \mathbb{R}^{n \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times n}$.*

[1] These lecture notes are a work in progress and there may be typos, awkward language, omitted proof details, etc. Moreover, content from lectures might be missing. These notes are intended to converge to a polished superset of the material covered in class so if you would like anything clarified, please do not hesitate to post to Piazza and ask.

- *Compute* $\mathbf{A}^{-1}$ *for* $\mathbf{A} \in \mathbb{R}^{n \times n}$
- *Compute a LU-factorization of* $\mathbf{A}$*, that is* $\mathbf{LU} \in \mathbb{R}^{n \times n}$ *where for some reordering of the coordinates* $\mathbf{A} = \mathbf{LU}$ *and* $\mathbf{L}$ *is lower triangular, i.e.* $\mathbf{L}_{ij} = 0$ *for all* $j > i$ *and* $\mathbf{U}$ *is upper triangular, i.e.* $\mathbf{U}_{ij} = 0$ *for all* $j < i$*.*

We have already seen the first two, the third can be shown by similar techniques that we saw for inverting a matrix.

It can also be shown in certain computational models that compute the determinant is also equivalent to matrix multiplication. This is particularly odd as the determinant is a single number. Computing the determinant from a *LU* factorization is easy as if $\mathbf{A} = \mathbf{LU}$ then it is not too hard to see that

$$\det(\mathbf{A}) = \det(\mathbf{L}) \det(\mathbf{U}) = \left(\prod_{i \in [n]} \mathbf{L}_{ii} \right) \cdot \left(\prod_{i \in [n]} \mathbf{U}_{ii} \right).$$

However, how would we go in the reverse direction? Suppose we had an algorithm to compute the determinant that simply performed standard arithmetic operations on coordinates and the results, i.e. it just added, subtracted, multiplied, or divided two entries, stored the result somewhere and then repeated on the result. Such an algorithm is called an arithmetic circuit and the time it takes to run such an algorithm is just the number of arithmetic operations performed. Note that both our naive algorithm for matrix multiplication as well as our recursive Strassen can be viewed as algorithms of this form as well as many of the faster matrix multiplication algorithms.

Now suppose we have such a circuit computing $\det(\mathbf{A})$. Note that

$$\begin{aligned} \frac{d}{d\mathbf{A}_{ij}} \det(\mathbf{A}) &= \lim_{\alpha \to 0} \frac{\det(\mathbf{A} + \alpha \vec{1}_i \vec{1}_j^\top) - \det(\mathbf{A})}{\alpha} = \lim_{\alpha \to 0} \frac{(1 + \alpha \vec{1}_j^\top \mathbf{A}^{-1} \vec{1}_i) \det(\mathbf{A}) - \det(\mathbf{A})}{\alpha} \\ &= \left[\mathbf{A}^{-1}\right]_{ji} \cdot \det(\mathbf{A}) \end{aligned}$$

where to show $\det(\mathbf{A} + \alpha \vec{1}_i \vec{1}_j^\top) = (1 + \alpha \vec{1}_j^\top \mathbf{A}^{-1} \vec{1}_i) \det(\mathbf{A})$ we used the "matrix determinant lemma." Consequently, computing a partial derivative of $\det(\mathbf{A})$ with respect to an entry of the input matrix gives an entry of of the inverse of $\mathbf{A}$. It can actually be shown that with only constant overhead the time to evaluate an arithmetic circuit is the time to compute the partial derivative of every input variable of that circuit with respect to that output. This is done by a careful and clever application of chain rule. Consequently, by computing these partial derivatives the inverse of $\mathbf{A}$ can be computed in the same time that it would take to compute $\det(\mathbf{A})$.

Finally, it is worth noting that there are many problems we have used FMM to solve for which it is open whether or not faster algorithms for these problems imply faster algorithms for FMM. For example, checking whether a n-vertex graph has a perfect matching, checking whether the determinant of $n \times n$ matrix is non-zero, or solving a linear system in a $n \times n$ matrix it is open whether or not any of these are solveable in $O(n^\alpha)$ and yet $\alpha < \omega$.

2 Boolean Matrix Multiplication

With the equivalence between various matrix multiplication problems in mind, it is worth noting that even easier variants of FMM over $\mathbb{R}^{n \times n}$ can be shown to be equivalent to natural combinatorial problems. For example, let $\mathbf{A}, \mathbf{B} \in \{0,1\}^{n \times n}$ we interpret $\mathbf{A}_{ij} = 1$ as the variable "true" and $\mathbf{A}_{ij} = 0$ as the variable "false."

With this interpretation in mind we define the *boolean matrix multiplication (BMM) problem* as computing $\mathbf{C} = \mathbf{A} \odot \mathbf{B} \in \{0,1\}^{n \times n}$ where for all $i, j \in [n]$ we have

$$\mathbf{C}_{ij} = \vee_{k \in [n]} (\mathbf{A}_{ik} \wedge \mathbf{B}_{kj}) = \begin{cases} 1 & \exists k \in [n] \text{ with } \mathbf{A}_{ik} = \mathbf{B}_{kj} = 1 \\ 0 & \text{otherwise} \end{cases}.$$

Note that we can clearly compute $\mathbf{C}$ in $O(n^\omega)$ by simply multiplying $\mathbf{A}$ and $\mathbf{B}$ over the reals and then setting any entry larger than 1 to 1. However, all known algorithms for solving BMM in $O(n^{3-\delta})$ for $\delta > 0$, i.e. subcubic time, go through algebraic fast matrix multiplication techniques. That is they all do some sort of algebraic reduction using cancellation as we did in Strassen's algorithm. It is a long-standing open problem, to find a "combinatorial algorithm," i.e. one that is more focused on graph structure and simply combinatorial updates that solve BMM in subcubic time. In fact, there is a conjecture known as the BMM conjecture, that no such algorithm exists. This conjecture is somewhat informal as there is not a strict universal definition of combinatorial algorithm that applies. Instead this conjecture should be viewed as a constraint on all known algorithms for BMM that it would be interesting to find a counter example for and without it, one should be careful in what algorithms they leverage.

In the remainder of this section we will discuss several natural graph problems are somewhat equivalent to solving BMM in subcubic time. In light of the BMM conjecture and the difficulty in getting fast practical FMM algorithms, one way to interpret these results is that should you encounter them, you likely want to use more structure if you want to solve them in subcubic time theoretically or heuristics if you want to solve them fast empirically.

So the first problem is computing something the transitive closure of a directed graph, which we encountered earlier in the course.

Definition 4 (Transitive Closure)**.** For directed graph $G = (V, E)$ its *transitive closure* is graph $G^* = (V, E^*)$ where $(i, j) \in E^*$ if and only if i can reach j in G.

This is a natural combinatorial problem and one might hope for fast combinatorial algorithms like we saw for shortest path or maximum flow. However, we can easily show this is impossible assuming the BMM conjecture.

Lemma 5. *Given an algorithm to compute transitive closure of n-node graphs in $O(n^\alpha)$ BMM can be computed in $O(n^\alpha)$.*

Proof. Suppose we have $\mathbf{A}, \mathbf{B} \in \{0,1\}^{n \times n}$ and we wish to compute $\mathbf{C} = \mathbf{A} \odot \mathbf{B}$. To do this, create a graph with vertices $a_1, ..., a_n$, $b_1, ..., b_n$, and $v_1, ..., v_n$. Now for all $i, j \in [n]$ if $\mathbf{A}_{ij} = 1$ add an edge from a_i to v_j and if $\mathbf{B}_{ij} = 1$ and an edge from v_i to b_j. Now it can be checked that the only way a_i can reach b_j is if there is a v_k with $(a_i, v_k) \in E$ and $(v_k, b_j) \in E$, i.e. $\mathbf{A}_{ik} = \mathbf{B}_{kj} = 1$. Consequently, $(a_i, b_j) \in E^*$ if and only if $\mathbf{C}_{ij} = 1$. Consequently, by computing the transitive closure of G, which we can do in $O((3n)^\alpha) = O(n^\alpha)$ time we can compute $\mathbf{C}$. □

We can also show that we can compute transitive closure using BMM.

Lemma 6. *Given an algorithm to compute BMM in $O(n^\alpha)$ we can compute transitive closure of a n-node graph in $O(n^\alpha)$.*

Proof. Given a graph $G = (V, E)$ let $\mathbf{A} \in \{0,1\}^{V \times V}$ be its adjacency matrix, i.e. for all $i, j \in V$ let

$$\mathbf{A}_{ij} = \begin{cases} 1 & (i, j) \in E \\ 0 & \text{otherwise} \end{cases}.$$

Now, to use BMM to compute the transitive closure lets look at power of $\mathbf{A}$ under BMM. What is $\mathbf{A}^{\odot k} \stackrel{\text{def}}{=} [\mathbf{A} \odot \mathbf{A} \odot \cdots \odot \mathbf{A}]$ where the product happens k times. We have that $\mathbf{A}^{\odot k}_{ij} = 1$ if and only if there exists $k_1, ..., k_c$ such that

$$\mathbf{A}_{ik_1} = \mathbf{A}_{k_1k_2} = \mathbf{A}_{k_2k_3} = \cdots = \mathbf{A}_{k_{c-1}k_c}\mathbf{A}_{k_cj} = 1\,.$$

Consequently, $\mathbf{A}^{\odot k}_{ij} = 1$ if and only if there is a path of length exactly k from i to j in G. Furthermore, this means that $(\mathbf{I} + \mathbf{A})^{\odot k}_{ij} = 1$ if and only if there is a path of length at most k in G as adding the identity can be viewed as allowing self loops or the path to stay at a vertex for a step. Since if two vertices can reach each other they can reach each other by a path of length at most $n-1$ we have that if we let $\mathbf{A}^*$ denote the result of raising $\mathbf{A}$ to a high enough boolean power that powering it again does not change it then the the adjacency matrix of the transitive closure of the graph induced by G is given by

$$(\mathbf{I} + \mathbf{A})^* - \mathbf{I} = (\mathbf{I} + \mathbf{A})^{\odot n} - \mathbf{I}$$

Consequently, we can compute the transitive closure in $O(n^\alpha \log n)$ by repeatedly squaring $\mathbf{A}$. However, we wish to compute $\mathbf{A}^*$ in $O(n^\alpha)$, so how do we remove this log factor?

In the remainder of this proof we sketch how to improve the algorithm. First, we compute all strongly connected components (SCC) in G. If a vertex can reach any vertex in a SCC it can reach all of them and if a vertex in a SCC can reach another vertex then all vertices in the SCC can reach that vertex. Consequently, if we compute the transitive closure of the contracted graph we can compute the transitive closure in the original graph at just an additive $O(n^2)$ cost.

Next, since all SCCs are contracted, the resulting graph has no SCCs and is a DAG. Thus we can perform a topological sort on the vertices in $O(n^2)$, as we saw in the homework, and then by reordering the vertices ensure that if the vertices are $V = [n]$ and i can reach j then $i \leq j$. Note that the adjacency matrix $\mathbf{A}$ for this graph therefor is upper triangular and so is $\mathbf{I} + \mathbf{A}$. Now let us write $\mathbf{I} + \mathbf{A}$ in block form as

$$\mathbf{I} + \mathbf{A} = \begin{bmatrix} \mathbf{U}_1 & \mathbf{B} \\ \mathbf{0} & \mathbf{U}_2 \end{bmatrix}.$$

Now, it is not too hard to see that

$$(\mathbf{I} + \mathbf{A})^* = \begin{bmatrix} \mathbf{U}_1^* & \mathbf{U}_1^*\mathbf{B}\mathbf{U}_2^* \\ \mathbf{0} & \mathbf{U}_2* \end{bmatrix}.$$

To see this, let S_1 and S_2 be a partition of the vertices so that $\mathbf{U}_1$ corresponds to edges from S_1 to S_1, $\mathbf{B}$ corresponds to edges from S_1 to S_2, and $\mathbf{U}_2$ corresponds to edges from S_2 to S_2. Now whether or not a vertex $i \in S_1$ can reach a vertex in $j \in S_1$ is given by $[\mathbf{U}_1]^*_{ij}$ and similarly whether or not a vertex $i \in S_2$ can reach a vertex in $j \in S_2$ is given by $[\mathbf{U}_2]^*_{ij}$. Finally, whether or not a vertex $i \in S_1$ can reach a vertex $j \in S_2$ is determined by whether or not i can reach some vertex $k_1 \in S_1$ for which there is an edge from k_1 to k_2 in S_2 for which k_2 can reach j, this is precisely $[\mathbf{U}_1^*\mathbf{B}\mathbf{U}_2^*]_{ij}$. Now, recursively computing these matrices and using BMM to compute $\mathbf{U}_1^*\mathbf{B}\mathbf{U}_2^*$ gives that we can solve the problem in $T(n)$ time where $T(n) = O(n^\alpha) + 2T(\frac{n}{2}) = O(n^\alpha)$. □

We conclude with one more example of BMM equivalences, that of determining whether or not an undirected graph contains a triangel.

Lemma 7. *If BMM can be solved in $O(n^\alpha)$ then using this algorithm one can compute if undirected $G = (V, E)$ has a triangle in $O(n^\alpha)$, i.e. if there exists some $\{i, j\}, \{j, k\}, \{k, i\} \in E$.*

Proof. If $\mathbf{A}$ is the adjacency matrix of G then $\left[\mathbf{A}^{\odot 3}\right]_{ii} = 1$ if and only if there is a path a length three from i to i, i.e. G contains a triangle. Since $\mathbf{A}^{\odot 3}$ can be computed in $O(n^\alpha)$ the result follows. □

Interestingly, the reverse was also shown approximately recently:

Theorem 8. *If detecting whether or not an n-node undirected graph contains a triangle can be computed in $O(n^{3-\delta})$ time for some $\delta > 0$ then using this BMM can be solved in $O(n^{3-\delta'})$ for some $\delta' > 0$.*

Note that $\delta$and δ' need not be the same here. Consequently, this algorithm shows that finding a worst case combinatorial algorithm for detecting whether or not a graph contains a triangle cannot be done with a combinatorial algorithm in subcubic time without breaking the BMM. This is suggestive of what worst case theoretically fast triangle detection algorithms need to do.

There is an active area of research trying to further clarify the complexity of these problems and more references can be provided upon request.

3 Introduction to Spectral Graph Theory

For the rest of this lecture we will switch gears and begin our investigation into *spectral graph theory*. Traditionally spectral graph theory concerns the study of spectral properties (i.e. eigenvalues and eigenvectors) of the adjacency matrix associated with a graph. More recently the field of spectral graph theory has expanded to more broadly the theory connecting linear algebraic properties (i.e. eigenvalues, eigenvector, linear system solutions, etc.) of matrices associated with graphs and combinatorial properties of graphs (i.e. distances, cuts, flows, etc.). This expansion has come with numerous algorithmic advances by the research communities that work on designing efficient graph algorithms.

3.1 Adjacency Matrices

Traditionally, the study of spectral graph theory has been concerned with the adjacency matrix of undirected graphs. Formally, for a simple undirected, weighted graph $G = (V, E, w)$ with positive edge weights $w \in \mathbb{R}^E_{>0}$ the (weighted) adjacency matrix is defined as $\mathbf{A}(G) \in \mathbb{R}^{V \times V}$ where for all $u, v \in V$ we define

$$\mathbf{A}(G)_{uv} \stackrel{\text{def}}{=} \begin{cases} w_{\{u,v\}} & \{u, v\} \in E \\ 0 & \text{otherwise} \end{cases}.$$

Now a classic question in spectral graph theory is what do the eigenvalues and eigenvectors of $\mathbf{A}(G)$ say about G. There is work that extends these connections to directed graphs and even hypergraphs, however in this class we will focus on simple, undirected graphs with positive edge weights.

Note, that there is a natural bijection between positive symmetric matrices with 0 as the diagonal entries and simple, undirected graphs with positive edge weights. In other words, note that if $\mathbf{M} = \mathbf{M}^\top$ with $\mathbf{M}_{ii} = 0$ and $\mathbf{M}_{ij} \geq 0$ then $\mathbf{M}$ is the adjacency matrix of some graph. Consequently, the field of spectral graph theory can also be viewed as the study of this natural class of matrices.

3.2 Laplacian Matrices

In this class, rather than focus on the adjacency matrices we will focus on a closely related matrix known as the Laplacian matrix. We will do this for several reasons that we will see later in the class. To define the Laplacian matrix of a graph we need a little more notation. First, for graph $G = (V, E)$ and $v \in V$ let $N_G(v)$ denote the *neighbors* of v, i.e.

$$N_G(v) \stackrel{\text{def}}{=} \{u \in V : \exists (v, u) \in E\},$$

the set of vertices reachable from v by a single edge. Furthermore, let $\deg_G(v)$ denote the *(weighted) degree* of v, i.e.

$$\deg_G(v) \stackrel{\text{def}}{=} \sum_{u \in N(v)} w_{\{u,v\}} \,,$$

the sum of the weight of the edges incident to v. Finally, we define the *(degree) matrix*, $\mathbf{D}(G) \in \mathbb{R}^{V \times V}$ to be the diagonal matrix associated with the weighted degrees, i.e. for all $u, v \in V$ we have

$$\mathbf{D}(G)_{uv} \stackrel{\text{def}}{=} \begin{cases} \deg(u) & u = v \\ 0 & \text{otherwise} \end{cases} .$$

Finally, we define the *(combinatorial graph) Laplacian* associated with G as

$$\mathcal{L}(G) \stackrel{\text{def}}{=} \mathbf{D}(G) - \mathbf{A}(G) \,.$$

So when is a matrix $\mathbf{M}$ a Laplacian of some graph? Note that we require that $\mathbf{M}$ be symmetric, i.e. $\mathbf{M} = \mathbf{M}^\top$, with non-positive off-diagonal matrices, i.e. $\mathbf{M}_{ij} \leq 0$ for all $i \neq j$. Matrices with this last property are known as *Z-matrices*. Furthermore note that this is all we know about the off-diagonal entries since they are the negation of the entries of an adjacency matrix. What are the diagonal entries? Note that

$$\mathcal{L}(G)_{ii} = \mathbf{D}(G)_{ii} = \sum_{j \in N(i)} w_{\{i,j\}} = \sum_{j \neq i} -\mathbf{A}(G)_{ij}$$

as if $\mathbf{A}(G)_{ij}$ is 0 if $\{i, j\} \notin E$ and is $w_{\{i,j\}}$ otherwise. However, this constraint holding for all $i \in V$ is the same as $\mathcal{L}(G)\vec{1} = \vec{0}$ where $\vec{1}$ is the all ones vector and $\vec{0}$ is the all zero vector. Consequently, we see that Laplacians also induce a natural bijection between simple undirected graphs with positive edge weights and symmetric Z-matrices where the all ones vector is in the kernel. Similarly, this means that we can view spectral theory here as either studying simple, undirected, positively weighted graphs or this fundamental class of matrices.

In the next lecture we will take a closer look at properties of these matrices.