6.897: Advanced Data Structures Spring 2005

Lecture 8 — March 1, 2005

Prof. Erik Demaine *Scribe: Vincent Yeung*

1 Overview

In this lecture we discuss two problems for dynamic directed graphs: **transitive closure** and **all-pairs shortest paths**. We will present various results and open problems for each. Then, we will focus on one particular result due to Sankowski [6] for transitive closure, namely that of $O(n^2)$ updates and $O(1)$ queries, both worst-case times.

2 Dynamic directed graphs

Instead of the standard insert and delete operations for edges, we would like our data structures to support more powerful *bulk update* operations. These refer to the insertion/deletion of a vertex and an arbitrary set of incident edges.

2.1 Transitive closure

The dynamic transitive closure problem is essentially that of dynamic reachability. We would like to answer queries of the form, "Is there a path from v to w?" We will now present various results for dynamic transitive closure:

2.1.1 Results

- bulk update: $O(n^2)$ amortized
  query: $O(1)$ worst case [1]
- bulk update: $O(n^2)$ worst-case, but randomized
  query: $O(1)$ worst case [6] **(what we will show today)**
- **OPEN**: Can we achieve bulk update in $o(n^2)$ worst case?

Note that, in some sense, the $O(n^2)$ result is optimal if we approach this problem by storing the transitive closure matrix *explicitly* (we don't necessarily have to do this, however, to answer queries). This is because in the worst case, an update can modify $\Omega(n^2)$ edges in the transitive closure.

Next are two results for which the product of the bulk update and query times is $O(mn)$, where m is the number of edges. The first of the two results allows for some amount of trade-off between the two operations.

- bulk update: $O(m\sqrt{n}\ t)$ amortized
  query: $O(\sqrt{n}/t)$ worst case, where $t = O(\sqrt{n})$ [3]
- bulk update: $O(m + n \lg n)$ amortized
  query: $O(n)$ worst case [2]
- **OPEN**: Can we achieve full trade-off with the product of update and query times equal to $O(mn)$ or $O(n^2)$?

Finally, we give two results for special cases of the problem:

- **(acyclic graphs)**
  update: $O(n^{1.575}\ t)$
  query: $O(n^{0.575}/t)$, $t = O(n^{0.325})$ [1]
- **(decremental transitive closure)**
  update: $O(n)$ amortized
  query: $O(1)$ worst case [1]

2.2 All-pairs shortest paths

Here we present various results for the all-pairs shortest paths problem (dynamic shortest paths), which answers the question of, "What is the weight of the shortest path from v to w?"

2.2.1 Results

- bulk update: $O(n^2(\lg n + \lg^2(1 + m/n)))$ amortized
  query: $O(1)$ worst case [8]
- **OPEN**: Is it possiblie to achieve $O(n^2)$ or $o(n^2)$ update, the latter even for just undirected graphs?
- update: $O(n^{2.75})$ worst case
  query: $O(1)$ worst case [9]
- **(unweighted graph)**
  update: $O(m\sqrt{n}\ \text{polylg}\, n)$ amortized
  query: $O(n^{3/4})$ worst case [4]
- **(unweighted, undirected, and (1+ϵ)-approximate)**
  update: $O(\sqrt{m}\ nt)$ amortized
  query: $O(\sqrt{m}/t)$ worst case, $t = O(\sqrt{m})$ [5]

3 $O(n^2)$ worst case dynamic transitive closure

As promised, we now discuss the result in [6]. The key idea is to transform the problem at hand (transitive closure) into one (**disjoint cycle cover**) that is solvable via linear algebra, with the use of various matrix properties.

Preliminaries: We will show the connection between transitive closure and disjoint cycle cover. To do so, we observe that each of the following is equivalent:

- There is a path from i to j in the graph.

$\Leftrightarrow$ If after adding the edge (j, i), there is a cycle through (j, i).

$\Leftrightarrow$ If after removing edges $(*, i)$ (all incoming edges to i) and $(j, *)$ (all outgoing edges from j), and adding the edge (j, i), there is a cycle through (j, i) (or through i, or j – these are equivalent conditions).

$\Leftrightarrow$ If after adding loops (k, k) for all vertices k, and removing edges $(*, i)$ and $(j, *)$ (note that this removes (i, i) and (j, j)), there is a disjoint set of cycles covering all vertices. This is the *disjoint cycle cover* problem. (Here, all nodes other than i and j can be covered by their self-loops, so again we are looking to see if there is a cycle through j.)

$\Leftrightarrow$ If after the same modifications as immediately above, there is a permutation π on the vertices such that $(i, \pi(i))$ is an edge $\forall i$.

3.1 Connection to linear algebra

We use the standard definition of a directed graph's **adjacency matrix** A:

$$A_{ij} = \begin{cases} 1 & \text{if } (i, j) \text{ is an edge} \\ 0 & \text{otherwise} \end{cases}$$

Now we need to make the transformations discussed earlier to the graph. We begin by adding I to A to put in self-loops (k, k) on each vertex (we are assuming the graph is initially loop-free). Next, we introduce the *zap* operation, represented as $\lightning$, and define $Z = (A + I)\lightning(j, i)$ as follows:

1. Zero out column i and row j in $(A + I)$; this removes the edges $(*, i)$ and $(j, *)$.
2. Set the element in the (j, i) position to 1; this adds the edge (j, i).

At this point, we are ready to start using linear algebra. First, we employ the following definition of the determinant:

$$\det Z = \sum_{\pi} \left[\text{sign}(\pi) \underbrace{\prod_{i} Z_{i\pi(i)}}_{\substack{0 \Leftrightarrow \text{ any edge} \\ (i, \pi(i)) \text{ is missing}}} \right]$$

If there is a disjoint cycle cover, then at least one of the terms (products) in this sum must be non-zero, so the determinant should (hopefully) be nonzero, but unfortunately, the sign rules for computing determinants may potentially cancel out non-zero terms. Before we resolve this issue, though, let us first extend the determinant idea a bit further. One can find that the *adjugate* (known as the *adjoint* in some circles) of a square matrix B contains as its entries the determinants of B's n^2 "zapped" matrices (one for each possible (j, i)). Applying this on our matrix $(A + I)$, we have:

$$(\text{adj}\,(A + I))_{ij} = \det\left((A + I)\lightning(j, i)\right)$$

In other words, $\operatorname{adj}(A+I)$ contains all the terms we need to find the transitive closure (i.e. to answer reachability queries for any pair (i,j)) – if we resolve the cancellation issue above.

3.2 Avoiding cancellation

Conceptually, we want a **symbolic matrix**, which we define as:

$$(\operatorname{symb} Z)_{ij} = \begin{cases} a_{ij} & \text{if } A_{i,j} \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

In other words, every nonzero entry is represented by a unique variable. Clearly, the determinant of $\operatorname{symb} Z$ is a polynomial of degree $\leq n$ in at most n^2 variables (a_{ij}, etc.). The symbolic determinant of this matrix, stored in $(\operatorname{adj}\operatorname{symb}(A+I))_{ij}$ is equal to a zero polynomial iff there is no edge from i to j. This is because all variable are distinct, so permutation terms cannot cancel out.

But calculating the symbolic determinant takes too long. Instead, we settle for a **random matrix**, which we define as:

$$(\operatorname{rand} Z)_{ij} = \begin{cases} \begin{smallmatrix}\text{independent random number chosen} \\ \text{uniformly between 0 and } p-1\end{smallmatrix} & \text{if } A_{i,j} \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

So what we want to do is replace the ones in $(A+I)$ with random numbers (p is a large prime number). Then, if the determinant of the symbolic matrix is zero, the determinant of the random matrix will also be zero. But what about when the polynomial is not zero? The following lemma can be used to calculate the probability that the determinant of the random matrix will be zero in this case (which will make our data structure answer queries incorrectly).

Lemma 1 (Zippel-Schwartz lemma [10, 7]).

$$\Pr\{\textit{nonzero polynomial of degree } d \textit{ evaluated at uniform random inputs} \equiv 0 \pmod p\} \leq d/p$$

So, in the case that $\det\operatorname{symb} Z \neq 0$, we have $\Pr\{\det\operatorname{rand} Z = 0\} \leq n/p \leq \frac{1}{n^{c-1}}$, if we choose a prime $p \geq n^c$. We can extend this to all n^2 Z matrices that our data structure needs to handle, and we get that:

$$\Pr\{\operatorname{adj}\operatorname{rand}(A+I) \text{ has more zeros than } \operatorname{adj}\operatorname{symb}(A+I)\} \leq \frac{1}{n^{c-3}} \quad \text{(by union bound)}$$

It follows that if the data structure runs for polynomial time, we can make the probability of error $\leq \frac{1}{n^{c'}}$ for any $c' > 0$.

Rebuilding: If the data structure runs for a long time, we need to reinitialize the matrix with other random values every $n^{O(1)}$ operations. The bound remains $O(n^2)$ amortized, and can be made $O(n^2)$ worst-case by deamortizing global rebuilding as in problem 3.

3.3 Dynamic matrix inverse

Now comes the algorithmic part of this discussion. How do we maintain the matrix $\operatorname{adj} \operatorname{rand}(A+I)$ in $O(n^2)$ worst case time for bulk updates and $O(1)$ worst case time for queries? The solution is to use the following relationship between the inverse and adjugate matrices:

$$B^{-1} = \frac{\operatorname{adj} B}{\det B}, \text{ when it exists } (\det B \neq 0)$$

Thus, if during bulk udpates we can maintain the inverse and determinant of $\operatorname{rand}(A+I)$ in $O(n^2)$ time, we can answer queries in constant time. This is what we will do.

Approach: When a bulk update to the vertex i occurs, row i and column i of A (really $\operatorname{rand}(A+I)$, but we will just say A for the rest of the analysis for simplicity) may be modified. Here, we will only consider modifications to column i. We represent this modification by the Δ vector: $A'_{ji} \leftarrow A_{ji} + \Delta_j,\ \forall j$. We also define the row vector $e_i^T = \begin{pmatrix} 0 & \dots & 0 & \underbrace{1}_{i^{th}\ column} & 0 & \dots & 0 \end{pmatrix}$.

Then,

$$\Delta e_i^T = \begin{pmatrix} \Delta_1 \\ \vdots \\ \Delta_n \end{pmatrix} \begin{pmatrix} 0 & \dots & 0 & 1 & 0 & \dots & 0 \end{pmatrix} = \begin{pmatrix} 0 & \dots & 0 & \Delta_1 & 0 & \dots & 0 \\ & & & \vdots & & & \\ 0 & \dots & 0 & \Delta_n & 0 & \dots & 0 \end{pmatrix}$$

Thus, the new matrix is $A' = A + \Delta e_i^T$.

The key idea is to write $A' = AB$, where $B = I + A^{-1}\Delta e_i^T = I + (A^{-1}\Delta)e_i^T = I + be_i^T$. Here $b = A^{-1}\Delta$ is a column vector $\begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$ that is a linear combination of the columns of A^{-1} and takes $O(n^2)$ time to compute. Hence, we can write B in the form:

$$B = \begin{pmatrix} 1 & & & b_1 & & & \\ & \ddots & & b_2 & & & \\ & & 1 & \vdots & & & \\ & & & 1 + b_i & & & \\ & & & \vdots & 1 & & \\ & & & b_{n-1} & & \ddots & \\ & & & b_n & & & 1 \end{pmatrix}$$

We know $A'^{-1} = B^{-1}A^{-1}$, so we need to find B^{-1}, which we can do with $\det B$ and $\operatorname{adj} B$, both of which can be computed in $O(n^2)$ with the following formulas:

$$B^{-1} = \begin{pmatrix} 1 & & & -\frac{b_1}{1+b_i} & & \\ & \ddots & & -\frac{b_2}{1+b_i} & & \\ & & 1 & \vdots & & \\ & & & \frac{1}{1+b_i} & & \\ & & & \vdots & 1 & \\ & & & -\frac{b_{n-1}}{1+b_i} & & \ddots \\ & & & -\frac{b_n}{1+b_i} & & 1 \end{pmatrix}$$

$$\det B = 1 + b_i$$

We also see that B^{-1} has $O(n)$ nonzeros. This is good because it means we can get $A'^{-1} = B^{-1}A^{-1}$ in $O(n^2)$ time (as opposed to the standard $O(n^3)$ time needed for matrix multiplication with the obvious algorithm). The new determinant can also be quickly computed since $\det A' = \det B \det A$.

There's a caveat, though. What if A is singular? We can show that this happens with very low probability (indeed, as low as we want to make it) because remember in the analysis A was really $\text{rand}(A+I)$, and from the probability analysis in the previous section, we know that if $\det \text{symb}(A+I) \neq 0$, then $\det \text{rand}(A+I) \neq 0$ with high probability, and we know that $\det \text{symb}(A+I)$ is never zero.

References

[1] Camil Demetrescu, Giuseppe F. Italiano. *Fully Dynamic Transitive Closure: Breaking Through the O(n^2) Barrier*. FOCS 2000: 381-389

[2] Liam Roditty, Uri Zwick. *A fully dynamic reachability algorithm for directed graphs with an almost linear update time*. STOC 2004: 184-191

[3] Liam Roditty, Uri Zwick. *Improved Dynamic Reachability Algorithms for Directed Graphs*. FOCS 2002: 679-

[4] Liam Roditty, Uri Zwick. *On Dynamic Shortest Paths Problems*. ESA 2004: 580-591

[5] Liam Roditty, Uri Zwick. *Dynamic Approximate All-Pairs Shortest Paths in Undirected Graphs*. FOCS 2004: 499-508

[6] Piotr Sankowski. *Dynamic Transitive Closure via Dynamic Matrix Inverse*. FOCS 2004: 509-517.

[7] Jacob T. Schwartz. *Fast Probabilistic Algorithms for Verification of Polynomial Identities. J* ACM 27(4): 701-717 (1980)

[8] Mikkel Thorup. *Fully-Dynamic All-Pairs Shortest Paths: Faster and Allowing Negative Cycles*. SWAT 2004: 384-396

[9] Mikkel Thorup. *Worst-case update times for fully-dynamic all-pairs shortest paths*. STOC 2005

[10] Richard Zippel. *Probabilistic algorithms for sparse polynomials*. EUROSAM 1979: 216-226