CS367 Lecture 8 Perfect Matching
Scribe: Jaehyun Park **Date:** April 28, 2014
Edited by: Ilan Goodman **Updated:** October 16, 2015

1 Successor Matrix for Shortest Paths

So far, we studied how to compute the shortest path distances under various matrix products, but never showed how one can compute the paths corresponding to these shortest distances. These notes will focus on how to find the actual paths, by modifying the algorithms we previously studied.

The only type of matrix product we will use throughout the notes is the $(\min, \odot)$ product, defined as $(A \odot B)[i,j] = \min_k(A[i,k] \odot B[k,j])$ for some operation $\odot : \mathbb{Z} \to \mathbb{Z}$. We will only consider operations $\odot$ that makes this matrix product associative. Note that many problems we considered in previous lectures, such as Boolean matrix multiplication or all-pairs shortest paths on directed, weighted graphs, fall under this framework. Also, in these notes, we will use the notation $i \rightsquigarrow j$ to denote a path from i to j, and whether it represents the shortest path or not should be clear from the context.

To motivate the discussion, suppose that we want to compute shortest paths between *all* pairs of nodes. However, in general, the size of the output can be $\Omega(n^3)$, and as a result, many related problems become uninteresting. Thus, we will look for a *successor matrix* instead, which essentially has the same representational power as the list of all shortest paths, but using only $\tilde{O}(n^2)$ memory.

Definition A *successor matrix* is an $n \times n$ matrix S, where we define $S[i,j] = k$ such that k is the next node after i on the $i \rightsquigarrow j$ shortest path. For the degenerate case $i = j$, then $S[i,i] = i$.

This definition immediately gives a procedure to retrieve a shortest path between any given pair of nodes using the successor matrix, assuming that the path is simple[1]; start from a node, proceed to the next one that is given by the successor matrix, and repeat. This shows that finding a successor matrix is basically as useful as having the entire list of shortest paths.

Proposition 1.1. *Given S, for any i, j, we can output a shortest $i \rightsquigarrow j$ path in time linear in the number of edges of that path.*

Proof. Look at the appropriate entries in the matrix and keep following until we find the destination node. □

2 Witness Matrix

Definition A *witness matrix* for the boolean matrix product of A and B is a matrix W such that $\forall i, j$, $W[i,j] =$ some k such that $A[i,k] = B[k,j] = 1$ if such a k exists, or $= \infty$ otherwise.

Remark A witness matrix is not unique necessarily: we only need to compute one such k for each i, j, but many may exist.

Let's assume that one can compute a witness matrix for the Boolean product of two $n \times n$ matrices in $\tilde{O}(n^\omega)$ time. We will show how to compute successor matrices for:

1. Transitive closure
2. Zwick's algorithm
3. Seidel's algorithm

[1]The shortest paths problems we consider all have the property that there is always a simple shortest path.

2.1 Transitive Closure

Recall we can compute transitive closure by successive squarings of the adjacency matrix (under the Boolean product). Boolean matrix multiplication (BMM) can actually be viewed as $(\max, \times)$ restricted to $\{0, 1\}$ entries, where $\times$ is the ordinary multiplication; by negating the entries of one of the matrices, BMM can also be represented as a $(\min, \odot)$ product.

If there is a path from i to j, then the (i, j) entry of the successor matrix S should give the node next to i in some simple path $i \rightsquigarrow j$. If there is no path, then $S[i, j]$ can be arbitrary (this is consistent with the definition of successor matrices). Then we can use the following algorithm:

Algorithm 1: $\mathsf{TC}(A)$

$A^0 \leftarrow A$
$\forall i, j,\ S^0[i, j] \leftarrow j$ if $(i, j) \in E$
foreach $k \in \{1, \ldots, \log n\}$ **do**
  $A^{(k)} \leftarrow A^{(k-1)} \cdot A^{(k-1)}$
  $W^{(k)} \leftarrow$ witness matrix of above product
  $\forall i, j,\ S^{(k)} \leftarrow S^{(k-1)}[i, W^{(k)}[i, j]]$
return $A^{(\log n)}, S^{(\log n)}$

In this algorithm, $A^{(k)}[i, j] = 1$ iff there's an $i \rightsquigarrow j$ path of length $\leq 2^k$. Algorithm 1 returns the transitive closure T, along with the successor matrix S. To see its correctness, we argue that $S^{(k)}$ is a correct successor matrix for all pairs of nodes i, j such that j is reachable from i via a path of length $\leq 2^k$. If there is a path $i \rightsquigarrow j$ of length at most 2^k, then the corresponding entry of $W^{(k)}$ gives a midpoint node w so that there are paths $i \rightsquigarrow w$ and $w \rightsquigarrow j$, both of which have length at most 2^{k-1}. The successor of path $i \rightsquigarrow j$ can then be retrieved from the successor of path $i \rightsquigarrow w$, which is already computed in $S^{(k-1)}$. The base case $k = 0$ is obtained directly from the adjacency matrix. This shows that $S^{(k)}$ is indeed a correct successor matrix.

2.2 Zwick's Algorithm

In Zwick's algorithm, we look at a shortest path such that the shortest path has between $\left(\frac{3}{2}\right)^{k-1}$ and $\left(\frac{3}{2}\right)^{k}$ edges. The idea was that the middle part has length $\sim \left(\frac{3}{2}\right)^{k-1}$, so we choose s from this random sample and the tails (parts of the path) on either side are each of length $\leq \left(\frac{3}{2}\right)^{k-1}$.

In step j, we compute some $(\min, +)$ product on matrices of the following dimensions: $n \times \frac{n}{\left(\frac{3}{2}\right)^k} \star \frac{n}{\left(\frac{3}{2}\right)^k} \times n$. We want witnesses for these products.

Definition A $(\min, \odot)$-*product of* A, B is C such that $C[i, j] = \min_k A[i, k] \odot B[k, j]$.

Remark Boolean product is a $(\min, +)$ product: $\bar{A} \leftarrow$ integer $\{0, 1\}$ corresponding to boolean A and $\bar{B}[i, j] = -1$ if $B[i, j] = 1$ and $= 0$ otherwise.

Definition A *witness matrix* for $(\min, \odot)$ is W such that $\forall i, j,\ W[i, j] = \arg\min_k A[i, k] \odot B[k, j]$.

If witnesses for $(\min, +)$ on matrices with entries in $\{-M, \ldots, M\}$ can be computed in $\tilde{O}(Mn^\omega)$ time, then successor matrices for Zwick's algorithm can be found in $\tilde{O}(M^{-.68}n^{2.575})$.

Remark The purpose of these witness matrices is that rather than finding the length of the shortest path, we want to obtain the actual shortest path. It costs a little more to do this, but not appreciably more.

2.3 Seidel's Algorithm

We presented a nice algorithm by Seidel for APSP in undirected graphs. For Seidel's algorithm, however, the above approach for computing the successor matrix does not apply as Seidel's algorithm runs recursively on a new graph and it computes the distances in a special way, using the counting power of integer matrix multiplication. Nevertheless, we can show that for undirected graphs, just being given the matrix of pairwise distances is sufficient to obtain the successor matrix in $O(n^\omega)$ time.

One of the main ideas in Seidel's algorithm was that for all i, j, and a neighbor k of i, $d[i,j]$ and $d[k,j]$ differ by at most 1, where d is the shortest distance matrix. Moreover, any k that satisfies $d[k,j] = d[i,j]-1$ is a valid successor of i in path $i \rightsquigarrow j$. Now, because $d[k,j]$ can be only 1 away from $d[i,j]$, we know that $d[k,j] \equiv d[i,j]-1 \pmod 3$ implies that k is a successor. This fact can be used to compute the successor matrix, given the matrix D of distances $D[i,j] = d[i,j]$, as follows.

Compute $d(i,j)$ for all i,j in $\tilde{O}(n^\omega)$ time (e.g., via Seidel). For all $s \in \{0,1,2\}$, set

$$D^{(s)}[k,j] = \begin{cases} 1 & \text{if } D[k,j] \equiv s-1 \pmod 3 \\ 0 & \text{otherwise.} \end{cases}.$$

Let A be the adjacency matrix. Compute $A \cdot D^{(s)}$ (boolean product, takes $O(n^\omega)$ time) for each choice of s and the witness matrix $W^{(s)}$ for this product. For all i,j, let $s_{ij} \equiv d(i,j) \mod 3$ and set $S[i,j] = W^{(s_{ij})}[i,j]$. To see why this is correct, fix i and j, and consider the (i,j) entry of the product $A \cdot D^{(s_{ij})}$. The index k that contributes to this entry satisfies $A[i,k] = D^{(s_{ij})}[k,j] = 1$. By construction, this means that $D[k,j] \equiv D[i,j]-1 \pmod 3$, and from the previous observation, we can conclude that k is a correct successor of i in path $i \rightsquigarrow j$. The witness is some k such that $A[i,k] = 1$ and $d(k,j) = d(i,j)-1 \mod 3$, i.e., $W^{(s_{ij})}[i,j]$ is the successor.

3 Computing Witness Matrices

In this section, we show an algorithm for computing witness matrices associated with the matrix product $A \odot B$, so that the successor matrix can be computed as described in the previous part.

Before handling the general case, we first focus on an easier variant of the problem, in which the witnesses are unique. From there, we will show we can easily find any witness matrix.

Definition A *unique witness matrix* for a $(\min, \odot)$-product of A and B is a matrix U such that $\forall i,j$, $U[i,j] = k$ such that k is the unique column such that $(A \odot B)[i,j] = A[i,k] \odot B[k,j]$ and $= \infty$ if no unique witness exists.

Special case: unique witness for BMM. Given A, B, create A' such that $A'[i,j] = j$ if $A[i,j] = 1$ and $= 0$ otherwise. Multiply $A' \cdot B$ (integer product). If k is a unique witness for i,j, then $\sum_l A'[i,l]B[l,j] = k$.

Strategy: $C \leftarrow A' \cdot B$ and for all i,j check if $A[i,C[i,j]] = B[C[i,j],j] = 1$. If so, $U[i,j] \leftarrow C[i,j]$, otherwise $U[i,j] \leftarrow \infty$.

Theorem 3.1. *If one can compute the* $(\min, \odot)$*-product of* $n \times n$ *matrices in* $T(n)$ *time, then computing* U *can be done in* $O(T(n) \log n)$ *time.*

Notation: for $S \subseteq [n]$, $A[\cdot, S]$ is the submatrix of A composed of the columns in S.

Proof of theorem 3.1. For all b from 1 to $\log n$ define $S_b = \{j | b\text{th bit of } j \text{ is } 1\}$. Let $C \leftarrow A \odot B$. Compute $(\min, \odot)$: $C_b \leftarrow A(\cdot, S_b) \odot B(S_b, \cdot)$. For all i,j,b, if $C[i,j] = C_b[i,j]$, then set the bth bit of $U[i,j]$ to 1, otherwise set it to 0. At the end, check the result $\forall i,j$, if $A[i,U[i,j]] \odot B[U[i,j],j] \neq C[i,j]$, then $U[i,j] \leftarrow \infty$ (if the witness is not unique we could get garbage).

This procedure computes the $(\min, \odot)$ product $\log n$ times, so the total running time is $O(T(n) \log n)$. It is also easy to see why this algorithm is correct: fix i and j that have a unique witness k. Then at every b, the bth bit of $W[i,j]$ will be set correctly. □

Finally, we show how to compute the more general witness matrix, given an algorithm for computing unique witnesses (get W from the ability to compute U). To do this, we do random samplings and use the algorithm above. First, we claim the following:

Lemma 3.2. *Let s be some value between 0 and $\log n$. Let (i,j) have c witnesses (in the $(\min,\odot)$-product of A and B) such that $\frac{n}{2^{s+1}} \le c \le \frac{n}{2^s}$. Let S be a random sample of $\{1,\dots,n\}$ of size 2^s. Then S contains a* unique *witness for (i,j) with probability $\ge \frac{1}{2e}$.*

If lemma 3.2 holds, for all $\log n$ values of s, repeat the following $d\log n$ times (for some constant $d > 3$): pick a random sample $S \subseteq [n]$ with $|S| = 2^s$, find $U \leftarrow$ unique witness matrix for $A(\cdot,S) \odot B(S,\cdot)$, and for all i,j, if $U[i,j] < \infty$, set $W[i,j] \leftarrow U[i,j]$.

For any (i,j) and the s such that the number of witnesses for (i,j) is in $\left(\frac{n}{2^{s+1}}, \frac{n}{2^s}\right)$, the probability that (i,j) doesn't have a unique witness in any of the $d\log n$ samples is $\le \left(1-\frac{1}{2e}\right)^{d\log n} \sim \frac{1}{n^d}$. By a union bound, the probability all (i,j) get a witness is $\ge 1 - \frac{1}{n^{d-2}}$.

Proof of lemma 3.2. Let W be the c witnesses for (i,j). Since there are 2^s elements in S and we have probability $\frac{c}{n}$ of hitting an element in W and we want to hit one element but not the others:

$$\begin{aligned} Pr[|W \cap S| = 1] &\sim \left(\frac{c}{n}\right) \cdot 2^s \left(1-\frac{c}{n}\right)^{2^s-1} \\ &\ge 2^s \cdot \frac{1}{2^{s+1}} \cdot \left(1-\frac{1}{2^s}\right)^{2^s-1} \\ &\ge \frac{1}{2e}. \end{aligned}$$

□

As stated between lemma 3.2 and its proof, this fact can be used to design a randomized algorithm that outputs a correct witness matrix. We do not know the number of witness for each individual (i,j) pair, but we know that it has to be contained in the intervals $\left[\frac{n}{2^{s+1}}, \frac{n}{2^s}\right]$ for some $s = 0,1,\dots,\log n$. The idea is to loop over all possible values of s, and determine the witnesses for all the (i,j) index pairs for which the number of witnesses fall into the right interval. Formally, the algorithm can be written as below.

Algorithm 2: WitnessMatrix(A, B)

```
C ← A ⊙ B
repeat
    for s = 0, 1, ..., log n do
        S ← random subset of {1, ..., n} of size 2^s
        Attempt to find the unique witness matrix W' of the matrix product A[·, S] ⊙ B[S, ·]
        for i, j = 1, ..., n do
            if A[i, W'[i, j]] ⊙ B[W'[i, j], j] = C[i, j] then
                W[i, j] ← W'[i, j]
until all elements of W are determined;
return W
```

For every outermost iteration, each $W[i,j]$ is filled with a correct witness with some constant probability, and once it is determined, it never changes. There are n^2 entries to be determined, so the expected number of outermost iterations is $O(\log n)$. We conclude by this final result.

Theorem 3.3. *If the $(\min,\odot)$-product is computable in $T(n)$ time, then finding the witness matrix takes $O(T(n)\log^3 n)$ time.*