CS 388R: Randomized Algorithms Fall 2015

Lecture 15 — October 28, 2015

Prof. Eric Price *Scribe: Kunal Lad, Abhishek Sinha*

1 Overview

In the last lecture, we looked at the all pairs shortest path problem. We saw the following:

1. An $O(mm(n)\log(n)) = O(n^{\omega+\epsilon})$ algorithm which finds the shortest distance matrix for a graph G by recursively finding the shortest distance matrix for the graph G^2.
2. The problem of determining the successor matrix for tripartite graphs.
3. $O(n^{\omega})$ time algorithm for finding successor matrix in in a simple tripartite graph when there is a unique successor for any 2 vertices.
4. An $O(mm(n)\log^2(n)) = O(n^{\omega+\epsilon})$ randomized algorithm for the case when there are an unknown number of successor vertices.

In this lecture, we will see

1. How to find the successor matrix for general graphs by reducing the problem to the tripartite graph case.
2. Problem of finding perfect matchings in bipartite graphs.
3. Sufficient and necessary condition for the existence of perfect matchings in bipartite graphs.
4. An algorithm for finding perfect matching for d-regular graphs when which is based on the idea of Eulerian Tours and has a time complexity of $O(nd)$.
5. A Las Vegas algorithm for finding perfect matchings in general d regular graphs $d = 2^k$ which has an expected time complexity of $O(n \log n)$.

2 Calculating Successor Matrix for General Graphs

We now wish to find a successor matrix P for the all pairs shortest path problem in general graphs. The successor matrix is defined in the following way,

$$P_{ij} = k \text{ if } k \in N(i) \text{ and } k \text{ lies on a shortest path from } i \text{ to } j \tag{1}$$

$N(i)$ denotes the neighborhood of i. In plain words, this means that k should be a neighbor of i and it should lie on some shortest path from i to j.

We saw in the last lecture that finding the successor matrix for tripartite graphs is equivalent to the problem of finding witnesses for the product of the boolean matrices A and B where A is the adjacency matrix for the left half and B is the adjacency matrix for the right half. A witness for a non-zero entry $(AB)_{ij}$ of the product AB is an integer $k \in [n]$ such that $A_{ik} = 1$ and $B_{kj} = 1$. Intuitively speaking, k is a *witness* or *proof* of the fact that $(AB)_{ij} = 1$.

To find the successor matrix in general graphs, we thus need to construct the boolean matrices A and B. The initial idea that was discussed in class was to construct a pair of boolean matrices for each possible distance between 2 vertices and to find the successor matrix for each case separately. More specifically, we iterate over all possible shortest distances l and in each step find the successors for vertices separated by that distance by running the witness algorithm on the matrices A and $D^{(l-1)}$ where $D^{(l-1)}$ is the matrix whose entries are 1 for a pairs of vertices having a shortest distance of $l-1$ and 0 otherwise.

Lemma 1. *Let $D^{(l-1)}$ be defined as the matrix such that ${D^{(l-1)}}_{ij} = 1 \Leftrightarrow D_{ij} = l-1$. Let i and j be 2 vertices such that $D_{ij} = l$. Then $P_{ij} = k$ if and only if it is one of the witnesses for ${AD^{(l-1)}}_{ij}$.*

Proof. First we prove the implies direction. By the definition of P, we have that $P_{ij} = k$ implies $A_{ik} = 1$ and k lies on a shortest path from i to j. Since $D_{ij} = l$, we have that $D_{kj} = l-1$ which is the same as ${D^{(l-1)}}_{ij} = 1$. Thus $A_{ik} = 1$ and ${AD^{(l-1)}}_{ij}$ which is the same as k being a witness for ${AD^{(l-1)}}_{ij}$. The converse can also be proven in a similar way. $\square$

This naive algorithm will have a time complexity of $O(n^{1+\omega+\epsilon})$. To improve the time complexity, we made the observation that for any 2 vertices i and j, if $D_{ij} = l$ then the shortest distance of any neighbor of i from $j \in \{l-1, l, l+1\}$. Formally, we state the following lemma.

Lemma 2. *Let i and j be two vertices such that $D_{ij} = l$. Then for any $k \in N(i)$, $D_{kj} \in \{l-1, l, l+1\}$.*

Proof. First we see that $D_{kj} \not< l-1$. If this were the case then going from i to k to j will make the distance of the path less than l which is a contradiction. Now we see that $D_{kj} \not> l+1$. If this were the case, then we could go from k to i to j for a total path length of $l+1$ between k and j which is a contradiction. $\square$

Hence, rather that looking at the actual distance of the neighbor of i to j, we can only look at the distance modulo 3. We define the matrices $M^{(0)}$, $M^{(1)}$ and $M^{(2)}$ where ${M^{(k)}}_{ij} = 1 \Leftrightarrow D_{ij} \equiv k \mod 3$. Thus, to find P_{ij} for 2 vertices having shortest distance l we only need to find a witness for ${AM^{l-1 \mod 3}}_{ij}$. Formally, we prove the following lemma.

Lemma 3. *Let i and j be two vertices such that $D_{ij} = l$. Then $P_{ij} = k$ if and only if k is one of the witnesses for ${AM^{l-1 \mod 3}}_{ij}$.*

Proof. If $P_{ij} = k$ then $A_{ik} = 1$ and $D_{kj} = l-1$ which implies that $D_{kj} \equiv (l-1 \mod 3) \mod 3$. Thus k is one of the witnesses for ${AM^{l-1 \mod 3}}_{ij}$. Conversely, let k be one of the witnesses for ${AM^{l-1 \mod 3}}_{ij}$. By Lemma 2, we have that $D_{kj} \in \{l-1, l, l+1\}$. But D_{kj} can not be l or $l+1$ since they are not $l-1 \mod 3$. Hence, $D_{kj} = l-1$. This by Lemma 1, we have that $P_{ij} = k$. $\square$

We can thus run the witness algorithm with the matrices A and $M^{(k)}$ over all values of k from $0, 1, 2$. To find the successor P_{ij} where $D_{ij} = l$, we just look at the (i, j) entry of the witness matrix returned by run of the witness algorithm with matrices A and $M^{l-1 \mod 3}$. Thus, in total we will have to run the witness algorithm for 3 (constant) pairs of matrices and hence the time complexity is the same as the time complexity of the witness algorithm which is $O(n^{\omega+\epsilon})$.

3 Matchings in Bipartite Graphs

Definition 4 (Bipartite Graph)**.** *A graph $G = (V, E)$ is said to bipartite if the vertex set V can be partitioned into 2 disjoint sets L and R so that any edge has one vertex in L and the other in R.*

Definition 5 (Matching)**.** *Given, an undirected graph $G = (V, E)$, a matching is a subset of edges $M \subseteq E$ that have no endpoint in common.*

Definition 6 (Maximum Matching)**.** *Given, an undirected graph $G = (V, E)$, a maximum matching M is a matching of maximum size. Thus for any other matching M', we have that $|M| \geq |M'|$.*

The problem of finding maximum matchings in bipartite graphs is a well studied problem. We describe some of the commonly known techniques for the same.

- **Reduction to Max Flow** Given an undirected bipartite graph $G = (V, E)$ we construct a network $G' = (V', E'), s, t, c$ as follows (construction is along identical lines as [TR1])

  - $V' = V \cup s, t$ where s and t are the 2 new vertices that we add.
  - E' contains a directed edge (s, u) for every $u \in L$, a directed edge (u, v) for every $e \in E$ where $u \in L$ and $v \in R$ and directed edge (v, t) for every $v \in R$.
  - The capacity c is defined as $c(e) = 1$ for every e in E'.

  We compute a max flow on the above network ensuring that every edge has an integral flow i.e. either 0 or 1. This is always possible since the original capacities are integral. Given the maximum flow, we return the maximum matching as the following set $(u, v) \in E$ such that$f(u, v) = 1$. It is easy to see that the size of the returned matching is the same as the size of the maximum flow. The proof of why this is a maximum matching follows from the fact that for any matching of size k in the bipartite graph, there exists a flow of value k in the network G' and vice versa.

  The maximum flow can be computed using several algorithms, the most popular one being the *Ford Fulkerson* algorithm. Ford Fulkerson by iteratively building larger $s - t$ flows by finding an augmenting path between s and t. After constructing an augmenting path, we push a flow w along it where w is the minimum of the capacity of all edges along the path. We also modify the graph by adding a reverse or back edge of capacity w for every edge in the augmenting path. Intuitively, by adding these reverse edges, we are allowing a new augmenting path to push back some of the flow added in the current step. The time complexity of the Ford Fulkerson Algorithm is $O(mc)$ where c is the total outgoing capacity from the source. In the case of a bipartite graph, we can see that this is $O(|V||E|)$.

- **Hungarian Algorithm** This is used in those cases where each edge of the bipartite graph has a weight or a cost associated with it. As an example, we may want to match students to rooms and each student may have a certain maximum cost s/he is willing to pay for the room. Using the Hungarian Algorithm, we can find a minimum cost matching in time $O(|V|^3)$ time.
- **Edmond-Karp Algorithm** This algorithm is also based on the idea of augmenting paths and has a time complexity is $O(|E|\sqrt(|V|)$.

Thus, we can see that for dense graphs none of these algorithms are asymptotically better that $O(|V|^{2.5})$. Later on in the lecture, we describe a randomized algorithm for finding a maximum matching in regular bipartite graphs (which is a perfect matching) which has an expected time complexity of $O(|V|\log(|V|))$.

3.1 Perfect Matching in Bipartite Graphs

Definition 7 (Perfect Matching)**.** *Given, an bipartite graph $G = (V, E)$, with the bipartition $V = L \cup R$ where $|L| = |R| = n$, a perfect matching is a maximum matching of size n.*

We now prove *Hall's Theorem* which gives both sufficient and necessary conditions for the existence of a perfect matching in a bipartite graph.

Theorem 8. *(Hall's Theorem) A bipartite graph $G = (V, E)$, with the bipartition $V = L \cup R$ where $|L| = |R| = n$, has a perfect matching if and only if for every subset $A \subseteq L$, $|N(A)| \geq |A|$ where $N(A)$ denotes the neighborhood of A.*

Proof. We first prove the necessary condition. Consider any subset $A \subseteq L$. In the perfect matching, each vertex in A will be connected to a distinct vertex of R. Hence $|N(A)| \geq |A|$.

We now prove the sufficient condition. We present the proof along identical lines as [TR2] . We prove it by contrapositive i.e. given the fact that there does not exist a perfect matching, we try to construct a set $A \subset L$ such that $|N(A)| < |A|$. We analyze a maximum (integral) flow in the network G' corresponding to the bipartite graph G which by assumption must have a value less than n. Hence, by the max-flow min-cut theorem an $s - t$ min-cut (S, S^c) of the graph also has a capacity less than n. Let $L_1 = S \cap L$, $R_1 = S \cap R$, $L_2 = S^c \cap L$ and $R_2 = S^c \cap R$. Since all edges have unit capacity and we are looking at integral flows, the capacity of the cut will simply be the number of edges going from S to S^c. Hence.

$$\begin{aligned} \mathit{capacity}(S) &= |L_2| + |R_1| + \mathit{edges}(L_1, R_2) \\ &= n - |L_1| + |R_1| + \mathit{edges}(L_1, R_2) \end{aligned}$$

But we know that $\mathit{capacity}(S) \leq n - 1$. Hence

$$n - |L_1| + |R_1| + \mathit{edges}(L_1, R_2) \leq n - 1 \tag{2}$$

This implies that

$$1 + |R_1| + \mathit{edges}(L_1, R_2) \leq |L_1| \tag{3}$$

We can easily see that the quantity $|R_1| + edges(L_1, R_2)$ is an upper bound for $|N(L_1)|$ since we are overcounting by assuming that each edge from L_1 to R_2 has a different end point in R_2. Hence we have the result that

$$1 + |N(L_1)| \leq |L_1| \Leftrightarrow |N(L_1)| < |L_1| \tag{4}$$

Thus L_1 is a set that we were looking for and this completes the proof of the sufficient condition. □

Using Hall's Theorem, we now show that every d regular bipartite graph has a perfect matching.

Theorem 9. *Every d regular bipartite graph has a perfect matching.*

Proof. Consider any set $A \subseteq L$. We try to count the number of edges from A to $N(A)$ in 2 different ways. This number is exactly equal to $|A|d$ since each vertex A contributes d outgoing edges. We also have that the number of incoming edges on $N(A)$ is at max $dN(A)$. This is an upper bound on the number of edges from A to $N(A)$ since all the incoming edges on the set $N(A)$ need not be outgoing from A. Hence, we have that

$$d|A| \leq d|N(A)| \Leftrightarrow |A| \leq |N(A)| \tag{5}$$

Hence, by Hall's theorem, the graph must have a perfect matching. □

3.2 Matchings in d-regular Graphs for d = 2^k

In the next section, we describe an algorithm for finding perfect matchings in d regular graphs where $d = 2^k$.

Definition 10 (Euler Tour)**.** *An Euler tour in an undirected graph is defined as a tour that traverses each edge of the graph exactly once.*

Neccessary and Sufficient Condition: An undirected graph has an Euler Tour iff every vertex has even degree.

Now for a d-regular graph with d = 2^k we can find a matching by following recursive algorithm:

- $d = 1$: Then it is a perfect matching precisely.
- $d = 2$: In this case graph corresponds to a cycle. Choosing an orientation of the cycle gives us a matching.
- $d = 2^k$: In this case we can get a matching by following procedure:
  - Walk along the edges and find an Eulerian Tour of G in O(m) time.
  - Orient the edges by the direction used in the walk.
  - Consider all forward edges, these form a regular graph with degree $d/2 = 2^{k-1}$. Thus running time is given by:

$$\begin{aligned} T(m) &= O(m) + T(m/2) \\ &= O(m) \end{aligned}$$

3.3 Matchings in d-regular Bipartite Graphs

In this section we look at a randomized algorithm proposed by Goel Kapralov and Khanna for finding matchings d-regular bipartite graphs where d may not be a perfect power of 2.

Intuition: There are large number of Bipartite Matchings on d-regular graphs.

Basic Idea: Basic idea of the algorithm is to use random walks to find a random walk from an unmatched vertex in one partition to an unmatched vertex in another partition. Use this to construct an augmenting path in Ford-Fulkerson algorithm.

Note: This algorithm assumes that we have G in adjacency array format so that we can sample edges for random walk in expected constant time.

Lemma 11. *Let k be the number of unmatched vertices after we have found a partial matching. Then:*

$$E[\textit{Time for random walk from s to t}] = O(n/k)$$

Proof. Let X and Y be the partitions of given graph G and let M be the partial matching of vertices in X and Y. We define the following wrt to M:

X_m : Set of matched vertices in X.
Y_m : Set of matched vertices in Y.
X_u : Set of matched unvertices in X.
Y_u : Set of matched unvertices in Y.
$M(x) = y$ and $M(y) = x$ if $x \in X$ is matched to $y \in Y$ under M

Let $b(v) = E[$#Back edges in random walk starting at v, ending at t$]$

Our goal is to prove $b(s) \leq n/k$

By above definition we have the following:

1. If $y \in Y$

$$\begin{aligned} b(y) &= 0 && \text{if } y \in Y_u \\ &= 1 + b(M(y)) && \text{if } y \in Y_m \end{aligned}$$

2. If $x \in X$

   - if $x \in X_u$ then

$$\begin{aligned} b(x) &= 1/d \sum_{y \in N(x)} b(y) \\ \implies db(x) &= \sum_{y \in N(x)} b(y) \end{aligned}$$

- if $x \in X_m$ then

$$
\begin{aligned}
b(x) &= 1/(d-1) \sum_{y \in \{N(x) - M(x)\}} b(y) \\
\implies (d-1)b(x) &= \sum_{y \in \{N(x) - M(x)\}} b(y) \\
\implies (d-1)b(x) &= -b(M(x)) + \sum_{y \in \{N(x)\}} b(y) \\
\implies (d-1)b(x) &= -(1 + b(x)) + \sum_{y \in \{N(x)\}} b(y) \\
\implies db(x) &= -1 + \sum_{y \in \{N(x)\}} b(y)
\end{aligned}
$$

Thus from 2 we get:

$$
\begin{aligned}
d \sum_{x \in X} b(x) &= -(n-k) + \sum_{(x,y) \in E} b(y) \\
&= -(n-k) + d \sum_{y} b(y) \\
&= -(n-k) + d(|M| + \sum_{x \in X_m} b(x))
\end{aligned}
$$

$$
d \sum_{x \in X_u} b(x) = (d-1)(n-k)
$$

$$
\begin{aligned}
b(s) &= \frac{1}{|u|} \sum_{x \in X_u} b(x) \\
&= \frac{1}{k} \frac{d-1}{d} (n-k) \\
&\leq \frac{n}{k}
\end{aligned}
$$

$\square$

The above lemma implies:

$$
E[\text{Running Time to find a matching}] \lesssim \sum_{1}^{n} (n/k) = nH_n \lesssim n \log n
$$

References

[TR1] Trevisan, Luca. Section 14.1, Combinatorial Optimization: Exact and Approximate Algorithms. Standford University (2011)

[TR2] Trevisan, Luca. Section 14.2, Combinatorial Optimization: Exact and Approximate Algorithms. Standford University (2011)

[MR] Rajeev Motwani, Prabhakar Raghavan Randomized Algorithms. *Cambridge University Press*, 0-521-47465-5, 1995.