CS 388R: Randomized Algorithms Fall 2017

Lecture 12 — October 10, 2017

Prof. Eric Price *Scribe: Shuangquan Feng, Xinrui Hua*

1 Overview

In this lecture, we will explore

1. Problem of finding matchings in bipartite graphs.
2. *Hall's Theorem*: sufficient and necessary condition for the existence of perfect matchings in bipartite graphs.
3. An algorithm for finding perfect matching in d-regular bipartite graphs when $d = 2^k$ which is based on the idea of Euler tour and has a time complexity of $O(nd)$.
4. A randomized algorithm for finding perfect matchings in all d regular bipartite graphs which has an expected time complexity of $O(n \log n)$.
5. Problem of online bipartite matching.

2 Matchings in Bipartite Graphs

Definition 1 (Bipartite Graph)**.** *A graph $G = (V, E)$ is said to be 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 2 (Matching)**.** *Given an undirected graph $G = (V, E)$, a matching is a subset of edges $M \subseteq E$ that have no endpoints in common.*

Definition 3 (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 $|M| \geq |M'|$.*

Definition 4 (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.*

There are several well known algorithms for the problem of finding maximum matchings in bipartite graphs.

- **Ford Fulkerson Algorithm** We first add a source and sink node to the bipartite graph. As long as there is a path from the source (start node) to the sink (end node), we send flow along one of the paths. Then we find another path, and so on. At last we get the max flow on the resulting $s - t$ network. Given the max flow, we output the intermediate edges on which there is a unit flow as the matching edges. The time complexity of finding a path is $O(|E|)$ and the time complexity of this algorithm is $O(|V||E|)$.

- **Hungarian Algorithm** This is used in the cases where each edge of the bipartite graph has a weight or a cost associated with it. Using Hungarian Algorithm, we can find a minimum cost matching in time $O(|V|^3)$ time.
- **Hopcroft-Karp Algorithm** This algorithm is also based on the idea of augmenting paths, but in each step the algorithm tries to find several augmenting paths and thus reduces the time complexity to $O(|E|\sqrt{|V|})$.

In the lecture, we first describe Gabow Karivan's algorithm which can find perfect matching in d-regular graphs when $d = 2^k$. Then we describe Goel Kapralov Khannafor's algorithm [2], which is a randomized algorithm to find a perfect matching in regular bipartite graphs with an expected time complexity of $O(|V| \log(|V|))$.

2.1 Hall's Theorem

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

Theorem 5. *(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 $S \subseteq L$, $|N(S)| \geq |S|$ where $N(S)$ denotes the neighborhood of S.*

Proof. We first prove the necessary condition. Consider there is one subset $S \subseteq L$ that $|N(S)| < |S|$. Then there remains some vertices in S that won't be connected to distinct vertices of R, which contradicts perfect matching. Hence for all subsets $S \subseteq L$, $|N(S)| \geq |S|$.

We now prove the sufficient condition.
We assume that there is no perfect matching. 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.

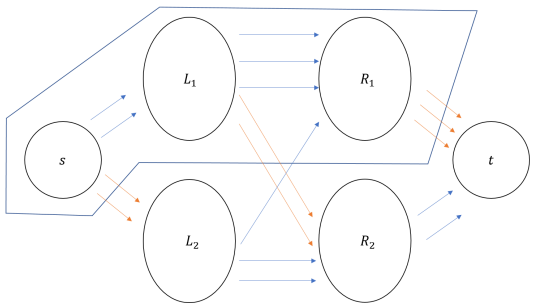

Figure 1: Hall's theorem

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.

$$\begin{aligned} |cut| &= |L_2| + |R_1| + edges(L_1 \to R_2) \\ &\geq n - |L_1| + |R_1| + |N(L_1) \cap R_2| \end{aligned}$$

We can easily see that the quantity $|R_1| + |N(L_1) \cap R_2|$ is an upper bound for $|N(L_1)|$ since we are overcounting by assuming that L_1 has edges to each point in R_1. So we have

$$\begin{aligned} |cut| &\geq n - |L_1| + |R_1| + |N(L_1) \cap R_2| \\ &\geq n + (|(N(L_1)| - |L_1|) \end{aligned}$$

But we know that $|cut| < n$, which implies

$$|N(L_1)| < |L_1|$$

Thus L_1 is a set that contradicts the assumption. This completes the proof of the sufficient condition. □

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

Theorem 6. *Every d regular bipartite graph has a perfect matching.*

Proof. Consider any set $S \subseteq L$. The number of edges from S to $N(S)$ is exactly $|S|d$ since each vertex contributes d outgoing edges. We also have that the number of incoming edges on $N(S)$ is at most $d|N(S)|$. This is an upper bound on the number of edges from S to $N(S)$. Hence, we have

$$d|S| \leq d|N(S)| \Leftrightarrow |S| \leq |N(S)|$$

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

2.2 Matchings in d-regular bipartite graphs for d = 2^k

In this section, we describe Gabow, Karivan's algorithm for finding perfect matchings in d-regular graphs where $d = 2^k$, which has the time complexity of $O(nd)$.

Definition 7 (Euler tour)**.** *An Euler tour in an undirected graph is a cycle that uses each edge exactly once.*

An undirected graph has an Euler tour if and only if each vertex has even degree, and all of its vertices with nonzero degree belong to a single connected component.

Now for a d-regular bipartite graph with d = 2^k, we can find a matching recursively:

- $d = 1$: It is a perfect matching precisely.
- $d = 2^k$: In this case there always exist Euler tours of each single connected component of this bipartite graph. We take all the edges of the tour from L to R and get a new $\frac{d}{2}$-regular bipartite graph. Repeat this process until we get 1-regular bipartite. The time is given by:

$$\begin{aligned} T(nd) &= O(nd) + T(\frac{nd}{2}) \\ \Rightarrow T(nd) &= O(nd) \end{aligned}$$

2.3 Matchings in general d-regular bipartite graphs

In this section we describe the randomized algorithm proposed by Goel, Kapralov and Khanna for finding matchings in all d-regular bipartite graphs. This algorithm takes time $O(n \log n)$ time both in expectation as well as with high probability[2].

Intuition: A random walk of finding augmenting paths in Ford Fulkerson algorithm will be very fast if there are lots of short paths.

Algorithm: The matching is constructed by performing one augmentation at a time, and new augmenting paths are found by performing an random walk with respect to the current matching[2].

Lemma 8. *Let k be the number of unmatched vertices, Then*

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

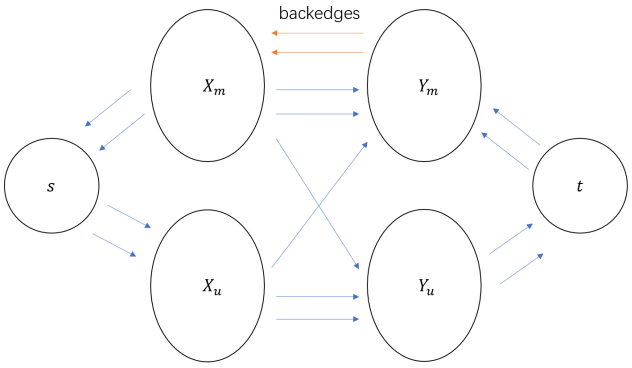

Figure 2: lemma

Proof. Let X and Y be the left and right partitions of vertices in G. M be the matched vertices. U be the unmatched vertices where $|U| = k$. Furthermore we define

X_m: matched vertices in X
Y_m: matched vertices in Y, $|X_m| = |Y_m| = |M|$.
X_u: unmatched vertices in X
Y_u: unmatched vertices in Y, $|X_u| = |Y_u| = |U|$.
$M(x) = y$ and $M(y) = x$ if (x, y) is matched where $x \in X_m$ and $y \in Y_m$.

We want to learn $b(v) := E[\#$ backedges in random walk from v to t $]$
Then $E[$path length from s to $t] = b(s) \times 2 + 3$, because if there are no backedges on the path then the shortest path is $s \to X_{ui} \to Y_{uj} \to t$ of length 3. When a backedge is present, it means the path contains $X \to Y_m \to X$ so the path length increases by 2.

Our goal is to prove $b(s) \leq \frac{n}{k}$ so that $E[$path length from s to $t] = b(s) \times 2 + 3 = O\left(\frac{n}{k}\right)$.

By the definitions above we have:

1. if $y \in Y$

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

If $y \in Y_u$ then the path goes directly to t. Otherwise random walk has to follow the backedge to $M(y)$.

2. if $x \in X$

- if $x \in X_u$

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

- if $x \in X_m$

$$\begin{aligned} b(x) =& \frac{1}{d-1} \left(\sum_{y \in N(x)} b(y) - b(M(x)) \right) \\ \implies (d-1)b(x) &= \sum_{y \in N(x)} b(y) - b(M(x)) \end{aligned}$$

Obverse that $b(M(x)) = 1 + b(x)$, because in the search path $M(x)$ always have to follow the backedge and come to x. So we have

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

Combining above two cases we have

$$d \cdot b(x) = \begin{cases} \sum_{y \in N(x)} b(y) & \text{if } x \in X_u \\ \sum_{y \in N(x)} b(y) - 1 & \text{if } x \in X_m \end{cases}$$

From the equations of $b(x)$ and $b(y)$ we can now calculate

$$\begin{aligned} d\sum_{x\in X} b(x) &= \sum_{X\in X_u} d\cdot b(x) + \sum_{X\in X_m} d\cdot b(x) \\ &= -|M| + d\sum_{y\in Y} b(y) \\ &= -|M| + d\left(|M| + \sum_{X\in X_m} b(x)\right) \\ \implies d\sum_{X\in X_u} b(x) &= (d-1)|M| \end{aligned}$$

The random walk starts from s and randomly pick a $x \in X_u$, so

$$E[b(s)] = \frac{1}{|U|}\sum_{X\in X_u} b(x) = \frac{|M|}{|U|}\frac{d-1}{d} = \frac{(n-k)(d-1)}{kd} < \frac{n}{k}$$

Then $E[\text{path length from } s \text{ to } t] = b(s)\times 2 + 3 = O(n/k)$. □

And from the lemma, the total amount of time we need is $O(n\sum_{i=1}^{n} 1/i) = O(nH_n) = O(n\log n)$.

3 Online bipartite matching

3.1 Introduction

The online bipartite matching concerns the problems in which vertices come online in a sequence. The algorithm must immediately assign an irrevocable edge for the vertex without knowing future sequence. Also the bipartite is not necessarily regular and we want the matching to be as good as possible.

We can think of a situation where there are n merchants(right vertices). Customers(left vertices) come one by one and each one is interests in some specific merchants. Without knowing future customers' preferences, we must immediately assign the customer to one merchant that he or she is interested in. Each merchant can only talk with one customer.

A $(1-1/e)$-approx algorithm was given by Karp, Vazirani, and Vazirani [3]. If the true perfect matching is OPT, this algorithm has expected $(1-1/e)OPT$ matched edges.

3.2 Deterministic greedy algorithm

Naively for each customer we can assign the first available merchant. Since each edge can block at most two matching edges, this deterministic greedy algorithm is at least 1/2-approx.

However, no deterministic algorithm can do better than 1/2 since one can easily design adversary inputs. For example in Figure 3, if we choose the red edge for a customer among two edges, the adversary input can let the red edge block other customers' edges(dotted line). Thus the deterministic algorithm can achieve no better than 1/2-approx.

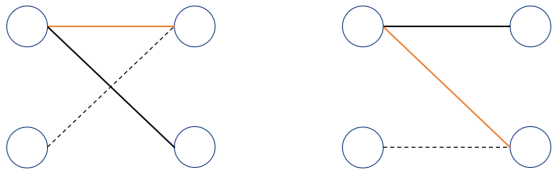

Figure 3: Adversary input for deterministic algorithm

3.3 Intuitive randomized algorithm

Now we want to introduce randomness and hope for a better approximation. An intuitive randomized algorithm is: for each left vertex, we randomly pick one available edge.

In the above example Figure 3, we now have $1/2$ chance to find a size 2 matching and $1/2$ chance to find a size 1 matching. In expectation the size of the matching is $3/2$ and this randomized algorithm gives $3/4$-approx.

But unfortunately there are still adversary inputs, as shown in Figure 4. For each first $n/2$ left vertices, the algorithm will choose an edge forward to lower $n/2$ right vertices with high probability. So the first $n/2$ correct edges are not likely to be chosen. Thus the expected size of matching given by randomized algorithm is small. Furthermore, we can prove the algorithm is only $(1/2 + O(1/\sqrt{n}))$-approx.

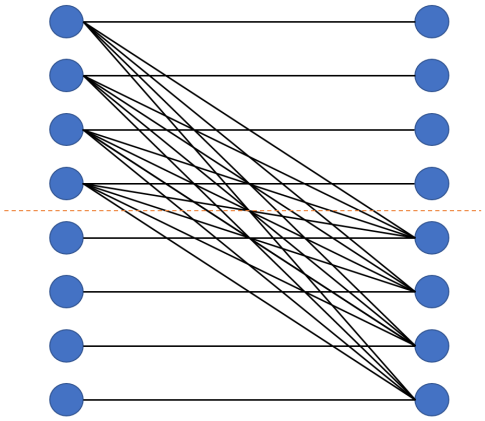

Figure 4: Adversary input for naive randomized algorithm

Lemma 9. *The intuitive randomized algorithm is a* $(\frac{1}{2} + O(\frac{1}{\sqrt{n}}))$*-approx algorithm.*

Proof. Suppose the randomized algorithm is $(1/2 + \epsilon)$-approx. In expectation there are $(n/2 + \epsilon n)$

matches. In the worst case, all lower $n/2$ right vertices are chosen, so there are expected $\geq \epsilon n$ of first $n/2$ right vertices to be chosen.

As the left $n/2$ top vertices come in sequence from top to bottom, the last one of top $n/2$ has the highest probability of choosing the correct edge. Because most of $n/2$ right bottom vertices are already matched. For this last node there are in expectation at least $1 + \epsilon n$ available choices(since there are $\geq \epsilon n$ of first left $n/2$ vertices chose correct vertices as mentioned above). So $\mathbb{P}$[choose the correct vertex] $\leq 1/(\epsilon n + 1)$.

$$E[\#\text{correct vertices in first n/2}] \leq \frac{n}{2}\frac{1}{\epsilon n} = \frac{1}{2\epsilon}$$

Because there are at least ϵn correct edges in the first $n/2$ choices, we need

$$\frac{1}{2\epsilon} \geq \epsilon n \implies \epsilon = O(\frac{1}{\sqrt{n}})$$

Thus the algorithm is $(1/2 + 1/\sqrt{n})$-approx.

□

3.4 Ranking algorithm

The algorithm above is susceptible to adversary input mostly because right vertices with larger degrees are more likely to be chosen by the left vertices. In fact we implicitly ranked the right vertices by their degrees. To correct this algorithm we can give merchants random priorities and assign each customer to the highest priority merchant available.

References

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

[2] Goel, Ashish, Michael Kapralov, and Sanjeev Khanna. Perfect Matchings in O(n logn) Time in Regular Bipartite Graphs. SIAM Journal on Computing 42.3 (2013): 1392-1404.

[3] Karp, Richard M., Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. *Proceedings of the twenty-second annual ACM symposium on Theory of computing*. ACM, 1990.