CS 388R: Randomized Algorithms, Fall 2021 October 14, 2021

Lecture 14: Matchings

Prof. Eric Price *Scribe: Akash Doshi, Nikos Mouzakis*

NOTE: THESE NOTES HAVE NOT BEEN EDITED OR CHECKED FOR CORRECTNESS

1 Overview

In the last lecture we discussed sampling algorithms, as well as how to use sampling to solve problems such as finding the median in a given list of numbers.

In this lecture we will introduce matchings, a few deterministic algorithms to find the matching followed by deterministic and randomized algorithms for matchings in d-regular bipartite graphs.

2 Matching in Bipartite Graphs

Consider a bipartite graph with an equal number of vertices, n, on both sides. We want to match left vertices to right vertices, with each vertex on the left matched to ≤ 1 vertex on the right. How many vertices can you match? If you can match all n vertices in L, it is called a perfect matching.

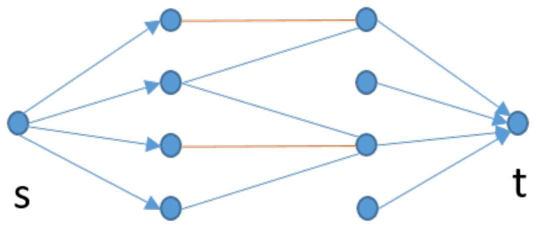

Figure 1: Bipartite Matching using the Ford-Fulkerson Algorithm

Observe in Fig. 1 how we can convert the bipartite graph into a max flow problem by the approriate addition of a source node s and sink node t. Solve the max-flow problem, using the Ford-Fulkerson algorithm [FF56]. With one BFS, you can obtain one flow in $O(m)$ time, and you have to repeat this for a maximum of n vertices, giving a total time complexity of $O(mn)$.

Other algorithms for finding perfect matchings with similar time complexities:

- Hungarian Algorithm [MU57]: Time Complexity $O(n^3)$ - One advantage of this algorithm is one can assign a cost to the edges, and find a min-cost perfect matching.
- Hopcroft-Karp Algorithm [HK73]: Time Complexity $O(m\sqrt{n})$.

Today we will be looking at algorithms to find perfect matchings on d-regular graphs i.e. each vertex has degree d. The graphs will continue to remain bipartite, unweighted with an equal number of vertices in L and R. More specifically, we will be looking at two algorithms:

- Gabow-Kariv [GK82]: Time Complexity $O(m)$. Designed only for the special case when $d = 2^K$.
- Goel - Kapralov - Khanna [GKK13]: Expected time complexity of $O(n \log n)$.

3 Hall's Theorem

Theorem 1. *Perfect matching exists in a bipartite graph* $\iff \forall\ S \in V$, $|N(S)| \geq |S|$, *where* $N(S)$ *denotes the set of vertices that are neighbours of* S*.*

Proof: $\implies$: Trivial (S has to match within $N(S)$).

$\Leftarrow$: The claim we are trying to prove is that if there is no perfect matching, $\exists S$ s.t. $|N(S)| < |S|$. Observe in Fig. 2 that if we ran [FF56], we would get some edges going backward (because we matched them) and some edges going forward (because we haven't matched them). In other words, when Ford-Fulkerson [FF56] terminates, there will be no $s - t$ path in the residual graph.

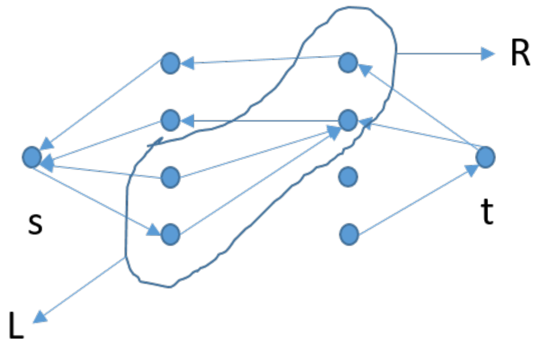

Figure 2: Proof of Halls theorem

$$\begin{aligned}
\min s - t \text{ cut} &= |\text{cut given by reachable vertices}| \\
&= (n - |L|) + |R| + |N(L) \cap \overline{R}| \\
&= (n - |L|) + |R| + |N(L)| - |R| \\
&= n + |N(L)| - |L|
\end{aligned}$$

Now we know the final expression is $< n$, because there isn't a perfect matching. Hence, we obtain that for $S = L$, $|S| > |N(S)|$, proving the theorem.

Lemma 2. *d-regular bipartite graphs always have perfect matchings.*

Proof: For any $S \in$ left, there are $d|S|$ edges from S to $N(S)$, and $d|N(S)|$ edges from $N(N(S))$ to $N(S)$. Then, observe that $N(N(S)) \supseteq S$. Hence $d|S| \leq d|N(S)|$, which implies $|S| \leq |N(S)|$, proving the criteria of Hall's theorem, and establishing the existence of a perfect matching.

4 Perfect Matching in d-regular bipartite graphs

This algorithm is from [GK82]. Given that $d = 2^K$:

- Construct Eulerian tour (DFS type algorithm) at each connected component.
- Orient edges by whether path goes $\leftarrow$ or $\rightarrow$ on them.
- Recurse on $\rightarrow$ edges ($d/2$- regular graph with $m/2$ edges).

Running time: $O(m + m/2 + m/4 + m/8 \ldots) = O(m)$.

Let us now consider some possible randomized algorithms for matching:

- Randomly pick an edge and add it to the matching. Drawback: In a complete graph, it can take $O(n^2)$ time to find the last edge to complete the matching.
- Go to one of the unmatched vertices and pick a random edge from there. Drawback: As shown in Fig. 3, if I pick the edges shown initially, I cannot find a matching.

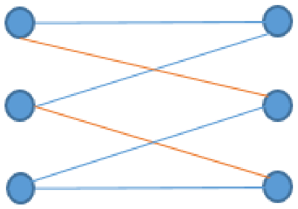

Figure 3: No perfect matching possible.

5 O(n log(n)) Randomized matching for d-regular bipartite graphs

Another idea for a randomized algorithm for matching, due to [GKK13], is to use a randomized version of the Ford-Fullkerson algorithm on the $s-t$ flow construction. Specifically, instead of using DFS or BFS to find an augmenting path, we will use a random walk. The intuition is that if there are lots of short paths, a random walk will find an augmenting path very fast.

More specifically, we can quantify this with the following lemma:

Lemma 3. *In a d-regular bipartite graph on n vertices on each side, if the residual graph has k unmatched vertices, then the expectation of the length T_k of the random walk until an augmenting path is found, is bounded by $E[T_k] \leq O(\frac{n}{k})$.*

Using this lemma we can bound the expectation of the total number of steps by $E[\sum T_k] = \sum \frac{n}{k} = O(nlog(n))$.

Proof. We consider an intermediate stage of the algorithm, with k unmatched vertices on each side.

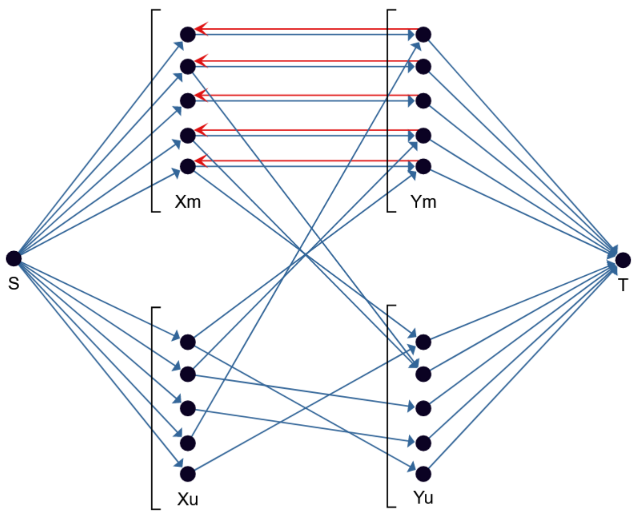

Figure 4: Example of residual network during the algorithm. Back-edges and forward edges are in red and blue, respectively.

The sets X_m, Y_m denote the sets of matched vertices on each side of the bipartite graph. X_u, Y_u denote the sets of unmatched vertices. By assumption, $|X_u| = |Y_u| = k$. Moreover, $M(x) = y \wedge M(y) = x$ if $x \in X_m, y \in Y_m$ are matched.

Let $b(v)$ denote the expected number of back-edges in the random walk beginning from node v, until a node in Y_u is reached. Then the expected length of the random walk from S to T is $E[T_k] = 2b(S) + 2 = 2(\frac{1}{k}\sum_{x \in X_u} b(x)) + 3$, since the first step of the random walk can only pick nodes in X_u.

By the definition of $b(v)$, we have the following equations:

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

$$b(x) = \begin{cases} \frac{1}{d}\sum_{y \in N(x)} b(y) & \text{if } x \in X_u \\ \frac{1}{d-1}\sum_{y \in N(x) \setminus M(x)} b(y) & \text{if } x \in X_m \end{cases} \tag{2}$$

This is because from an unmatched node in X_u, the random walk can visit any of its neighbors next, while for an already matched node, it can visit every one of it's neighbors except the one that it is already matched with.

Rewriting (2), we get for $x \in X_u, d * b(x) = \sum_{y \in N(x)} b(y)$, and for $x \in X_m, (d-1) * b(x) = \sum_{y \in N(x)} b(y) - b(M(x)) = \sum_{y \in N(x)} b(y) - 1 - b(x) \iff d * b(x) + 1 = \sum_{y \in N(x)} b(y)$. More succinctly, $\forall x, d * b(x) + \mathbb{1}_{x \in X_m} = \sum_{y \in N(x)} b(y)$.

Summing for every x, we get $d * \sum_{x \in X} b(x) + |M| = d * \sum_{y \in N(x)} b(y) = d * (|M| + \sum_{x \in X_m} b(x)) \implies \sum_{x \in X_u} b(x) = \frac{(d-1)*|M|}{d}$. Hence, $\frac{1}{k}\sum_{x \in X_u} b(x) = \frac{(d-1)*(n-k)}{d*k} \leq \frac{n}{k}$, which means $E[T_k] \leq 2 * \frac{n}{k} + 3 \in O(n/k)$. □

6 Online bipartite matching

In the online version of this problem, the vertices arrive in an unknown sequence, and the algorithm must match each arriving vertex irrevocably the moment it arrives. The adversary can change the sequence depending on the choices of the algorithm so far.

We can show that any deterministic algorithm can only achieve up to a 1/2-approximation ratio compared to the optimal matching.

Lemma 4. *Any deterministc algorithm for online bipartite matching can achieve at most* 1/2-*approximation ratio.*

Proof sketch. The adversary first provides vertex A as in Figure 5:

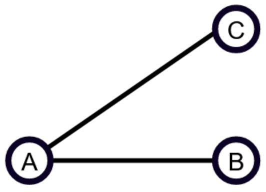

Figure 5: Initial vertices in sequence

Then, depending on whether the algorithm chose to match A with B or C, the adversary provides vertex D as in Figure 6.

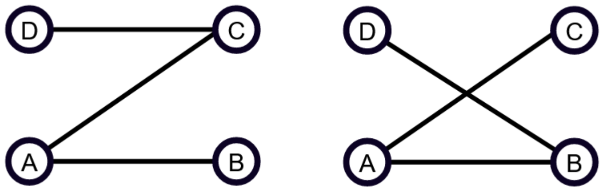

Figure 6: Next vertex in sequence

In both of these case the algorithm only manages to output a matching of size 1, while the optimal matching is of size 2. ☐

In the next class we are going to discuss how to use randomization to get past this bound.

References

[FF56] Ford Jr, L. R., & Fulkerson, D. R. Solving the transportation problem. *Management Science*, 3(1): 24–32, 1956

[MU57] Munkres, J. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1): 32–38, 1957

[HK73] Hopcroft, J. E., & Karp, R. M. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4): 225–231, 1973

[GK82] Gabow, H. N., & Kariv, O. Algorithms for edge coloring bipartite graphs and multigraphs. *SIAM journal on Computing*, 11(1): 117–129, 1982

[GKK13] Goel, A., Kapralov, M., & Khanna, S. Perfect Matchings in $O(n \log n)$ Time in Regular Bipartite Graphs. *SIAM Journal on Computing*, 42(3): 1392–1404, 2013