CS 7880: Algorithms for Big Data (Fall'22) **Northeastern University**

Lecture 4

September 20, 2022

Instructor: Soheil Behnezhad *Scribe: Thien Nguyen*

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications.*

1 Perfect matching in regular bipartite graphs

Given an n-vertex m-edge graph $G = (V, E)$, a *matching* $M \subseteq E$ is a collection of vertex disjoint edges. A *maximum matching* is a matching of the largest possible size. Our focus in this lecture will be on *regular* bipartite graphs, i.e., all vertices have the same degrees. It is well known that any regular bipartite graph has a matching that matches every vertex. Such matchings are known as *perfect matchings*. Our goal is to show that a perfect matching can be found extremely efficiently in regular bipartite graphs. In particular, we prove the following theorem by Goel, Kapralov, and Khanna [GKK10]:

Theorem 1 ([GKK10])**.** *Provided adjacency list access to an n-vertex d-regular bipartite graph, a perfect matching can be found in $O(n \log n)$ expected time.*

Note that if $d = \Theta(n)$, then the graph G has $\Theta(n^2)$ edges. Therefore, Theorem 1 does not even read all the edges and so runs in sublinear time in the input size.

Remark. Edmonds [Edm65] was first to show that maximum matching can be solved in polynomial time. This led to the introduction of the class of polynomial time algorithms P. For bipartite graphs, Hopcroft and Karp [HK73] presented an $O(m\sqrt{n})$ time algorithm. Improving this bound had remained elusive, until a series of recent works resulted in a breakthrough "almost linear time" algorithm by Chen, Kyng, Liu, Peng, Gutenberg, and Sachdeva [CKL+22] running in $O(m^{1+o(1)})$ time. Note, however, that Theorem 1 runs even faster than this (albeit with the extra assumption that the graph is regular).

1.1 Augmenting path and augmentation graph

Augmenting path. An indispensible tool in finding large matchings is the concept of *augmenting paths*. Fix a graph G and a matching M. A path in G is an *augmenting path for M* if (1) the edges of the path alternatively belong to M, and (2) the first and the last edges of the path do not belong to M. Note that for every augmenting path P of a matching M, we can increase the size of the matching by at least 1 by taking the symmetric difference between P and M. For an example of an augmenting path, see Figure 1.

Claim 2. *A matching is maximum if and only if it has no augmenting path.*

Proof. If M is a maximum matching, it clearly admits no augmenting path given that they can be used to increase its size as described above. So let M be a non-maximum matching and let $M^\star$ be an arbitrary maximum matching in the garph. Take the subgraph $U := M \cup M^\star$. Any connected component that is not an augmenting path for M can be confirmed to have exactly the same number of edges of M and $M^\star$. Thus if $|M| < |M^\star|$, there must exist at least one augmenting path for M. □

Now that we have seen that one path to finding maximum matchings is via augmenting paths, we will now see how to do so given a bipartite graph.

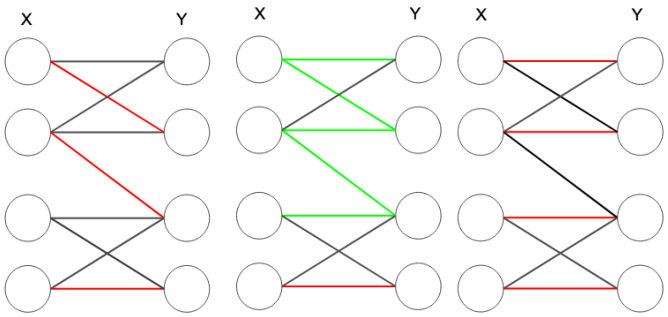

Figure 1: Augmenting path for bipartite graphs. Left: A partial match of size 3. Middle: An augmenting path for the partial match on the left. Right: The symmetric difference between the augmenting path and the partial match gives rise to a larger matching. In this case, the application of the augmenting path turns the matching into a perfect matching (therefore, also maximum).

Finding augmenting path via finding paths on augmentation graph. One way to find augmenting path for regular bipartite graphs is via a construction of an auxilary graph (as in Construction 3) for which we will refer to as an *augmentation graph*. Given a matching M, we mean that a vertex $x \notin M$ if no edge in M contains x. Given a bipartite graphs $G = (X \cup Y, E)$ with left vertices X and right vertices Y each with n vertices (e.g. Figure 1), consider the construction below given a partial match M.

Construction 3 (Augmentation Graph Construction)**.** Given an undirected bipartite graph $G = (X \cup Y, E)$ and a partial match M,

1. Create a directed graph $\tilde{G} = (\tilde{V}, \tilde{E})$ where we add a source s and sink t node to the bipartite graph i.e. $\tilde{V} := X \cup Y \cup \{s, t\}$ while leaving $\tilde{E} \leftarrow \emptyset$ for construction.
2. For each unmatched edge $(x, y) \in E - M$ with $x \in X$ and $y \in Y$:
   (a) Add an edge from x to y i.e. $\tilde{E} \leftarrow \tilde{E} \cup (x, y)$.
   (b) If $x \notin M$, add an edge from the sink to the left vertex i.e. $\tilde{E} \leftarrow \tilde{E} \cup (s, x)$.
   (c) If $y \notin M$, add an edge from right vertex to the sink i.e. $\tilde{E} \leftarrow \tilde{E} \cup (y, t)$.
3. For each edge in the partial match $(x, y) \in M$ with $x \in X$ and $y \in Y$:
   (a) Add an edge from y to x i.e. $\tilde{E} \leftarrow \tilde{E} \cup (y, x)$.
4. Return the augmentation graph $\tilde{G} = (\tilde{V}, \tilde{E})$ of G.

For our running example in Figure 1, the resulting application of Construction 3 is shown in Figure 2 (left). The main property of the above construction is the following:

Proposition 4. *For a graph G and a partial match M, let $\tilde{G}$ be the output of Construction 3. Then, any path from s to t of $\tilde{G}$ induces an augmenting path for the given partial match.*

Proof. If $\{(s, x_1), (x_1, x_2), \ldots, (x_{n-1}, x_n), (x_n, t)\}$ is a path of length n from s to t of $\tilde{G}$, then we claim that $\{(x_1, x_2), \ldots, (x_{n-1}, x_n)\}$ is an augmenting path for M. Clearly, $(x_1, x_2) \notin M$ since $x_1 \notin M$ by construction. Similarly, $x_n \notin M$ (due to the edge to t) implies $(x_{n-1}, x_n) \notin M$.

For every edge in between, we claim that they must alternate between being in M and not in M. Suppose not. WLOG, assume that (x_1, x_2) and (x_2, x_3) are not alternating. They cannot be both in M since M is a matching. Suppose they are both not in M. Then because G is bipartite, either x_1 and x_3 must be in the same group of vertices (either left or right). If they are both in the set of left vertices, $(x_2, x_3) \in \tilde{E} \iff (x_2, x_3) \in M$ by construction. If they are both in the set of right vertices, $(x_1, x_2) \in \tilde{E} \iff (x_1, x_2) \in M$ by construction. In either case, we cannot have both (x_1, x_2) and (x_2, x_3) not in M. Hence, we have established the contradiction and our claim. □

An example of Proposition 4 is presented in Figure 2 (right). Note that this yields the same augmenting path presented in Figure 1. Now, we are ready to develop a sublinear algorithm for perfect matching on (regular) bipartite graphs.

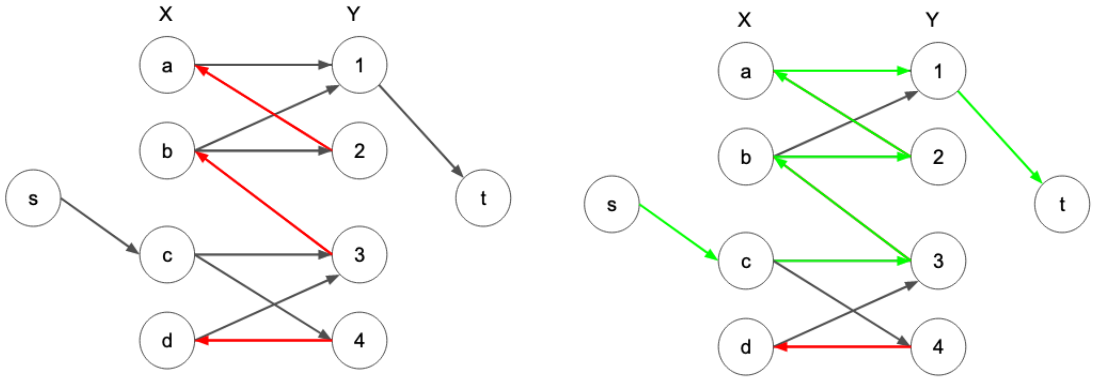

Figure 2: Left: The augmentation graph for a partial matching. Right: A path from s to t yields an augmenting path for the given partial matching.

1.2 The final algorithm

With Proposition 4 and Construction 3, we now consider the remaining question of finding paths from s to t. To do so, we run a random walk starting at s. Recall that we work under the adjacency list model, where we can query a random neighbor of a vertex in constant time. Therefore a random walk of length d can be implemented in $O(d)$ time. We formalize the algorithm below.

Algorithm 1 Sublinear perfect matching on regular bipartite graph

1. **Input**: A d-regular bipartite graph $G = (V, E)$.
2. Initialize $M \leftarrow \emptyset$.
3. For n steps:
   (a) Given the augmentation graph $\tilde{G}$ as in Construction 3 using M, start a random walk from $s \in \tilde{G}$ until we reach t (see Subroutine 5) and obtain a path P from s to t.
   (b) The path P induces an augmenting path P_A.
   (c) Take the symmetric difference of P_A and M to obtain a larger matching and set that to be M for the next iteration.
4. **Return**: M.

Note that, naively, step 3a in Algorithm 1 will take $O(nd)$ time if we explicitly construct the augmentation graph $\tilde{G}$. However, we do not need to fully construct to perform the random walk on $\tilde{G}$. Since we have the full rules for constructing the edges in $\tilde{G}$, we can perform the random walk as in Subroutine 5.

Subroutine 5 (Random walk on $\tilde{G}$)**.** Given M, we can perform a random walk on the augmentation graph $\tilde{G}$ of M without explicitly constructing $\tilde{G}$ as follows:

1. Start from s and pick a random vertex $x \in X$ that is not in M.
2. **Repeat**:
   (a) (We are at a vertex $x \in X$) Go to a random neighbor $y \in Y$ of x.
   (b) If $y \in M$:

i. Go to y's match in M i.e. $M(y) \in X$.

(c) Else (i.e. $y \notin M$):

i. We can only go to t.

ii. **Return** the path.

Note that for Step 2a of Subroutine 5, any Y neighbor of x would be progress towards t if $x \notin M$. On the other hand, if $x \in M$, we make progress towards t if we pick a vertex $y \notin M$. At this step, we can avoid backtracking to x's match $M(x)$ by randomly go to a random neighbor of X in $Y - \{M(x)\}$.

1.3 Analysis of Algorithm 1

Correctness. When Algorithm 1 terminates, we must have found n augmenting paths and so find a perfect matching of size n.

Running time. Now, we show that Algorithm 1 is a sublinear algorithm for computing the perfect matching of a regular bipartite graph given the adjacency list model. First, we show an important Lemma.

As discussed, the running time of each step of the algorithm is linear in the length of the random walk from s to t. The following lemma is the key to bounding this length. We say an edge is a *backward edge* if it is directed from Y to X (note that all such edges are the ones in M).

Lemma 6. *Let k be the size of the matching M we have so far. Then the expected number of backward edges traversed by the random walk on the augmentation graph $\tilde{G}$ of M is at most $\frac{n}{n-k} - 1$.*

Note that with this Lemma, we can easily conclude the proof of Theorem 1:

$$\begin{aligned}
\mathbb{E}\left[\text{running time}\right] &\leq \sum_{k=0}^{n-1} \mathbb{E}\left[\text{running time for step } k\right] \\
&\leq \sum_{k=0}^{n-1} O(\frac{n}{n-k} - 1) \leq O(n) \sum_{k=0}^{n-1} \frac{1}{n-k} \\
&= O(n) \sum_{k=1}^{n} \frac{1}{k} \leq O(n \log n).
\end{aligned}$$

We now present the proof of Lemma 6.

Proof of Lemma 6. For any vertex v, define $b(v)$ to be the expected number of backward edges seen if we start the random walk from v. Let us use X_M and Y_M to denote the vertices matched by M in X and Y respectively. Similarly, we use X_U and Y_U to denote the unmatched vertices of X and Y. Additionally, for any vertex v matched in M, we use $M(v)$ to denote the vertex that v is matched to.

We can now analyze the number of expected backward edges. We are interested in computing $b(s)$ for which we can only go to the unmatched vertices in X afterwards. That gives

$$b(s) = \frac{1}{n-k} \sum_{x \in X_U} b(x). \tag{1}$$

We now seek to compute $b(x)$ for $x \in X_U$. Fixing $x \in X_U$, since we can go to any of the d neighbors of x, each neighbor y of x contributes $\frac{1}{d} b(y)$ to $b(x)$. Hence, that gives us

$$b(x) = \frac{1}{d} \sum_{y:(x,y) \in E} b(y), \qquad \forall x \in X_U. \tag{2}$$

The sum over the neighbors of x seems annoying to deal with. Let's see if we can massage it to some nicer quantities. We seek to relate it to $x \in X_M$, for which we have $b(x) = \frac{1}{d-1}\sum_{(x,y)\in E-M} b(y)$ due to $(x, M(x))$ being the backward edge. Moving the $d-1$ term and adding $b(x)$ to both sides, we get

$$db(x) = b(x) + \sum_{(x,y)\in E-M} b(y), \qquad \forall x \in X_M. \tag{3}$$

Note that $\forall y \in Y_U, b(y) = 0$ because we must go to the sink after y. Now, $\forall y \in Y_M$, since we must go back to y's match in X (which let us denote by $M(y)$), we must have that

$$b(y) = b(M(y)) + 1 \iff b(M(x)) = b(x) + 1, \qquad \forall x \in X_M. \tag{4}$$

Hence, we can further reduce (3) to $\forall x \in X_M$:

$$\begin{aligned} db(x) &= b(x) + \sum_{(x,y)\in E-M} b(y) = b(M(x)) - 1 + \sum_{(x,y)\in E-M} b(y) \\ &= -1 + \sum_{(x,y)\in E} b(y). \end{aligned} \tag{5}$$

We have expressed $b(x)$ in terms of $b(y)$ for both X_M and X_U. Combining them and summing over all of $x \in X$, we get:

$$\begin{aligned} d\sum_{x\in X} b(x) &= d\sum_{x\in X_U} b(x) + d\sum_{x\in X_M} b(x) \\ &= \sum_{x\in X_U}\sum_{(x,y)\in E} b(y) + \sum_{x\in X_M}\left(-1 + \sum_{(x,y)\in E} b(y)\right) && \text{(using (2) and (5))} \\ &= \sum_{x\in X_U}\sum_{(x,y)\in E} b(y) - k + \sum_{x\in X_M}\sum_{(x,y)\in E} b(y) \\ &= \left(\sum_{x\in X}\sum_{(x,y)\in E} b(y)\right) - k \\ &= -k + d\sum_{y\in Y} b(y) && \text{(since the graph is } d\text{-regular)} \\ &= -k + d\sum_{y\in Y_M} b(y) && \text{(since } b(y) = 0 \text{ for } y \in Y_U) \\ &= -k + d\sum_{x\in X_M} 1 + b(x) && \text{(due to (4))} \\ &= -k + dk + d\sum_{x\in X_M} b(x) \end{aligned}$$

Moving the terms and since $X - X_M = X_U$, we get

$$d\sum_{x\in X_U} b(x) = -k + dk = (d-1)k.$$

Finally, plugging this back to (1), we get our desired bound that

$$b(s) = \frac{1}{n-k}\sum_{x\in X_U} b(x) = \frac{(d-1)k}{d\,(n-k)} \le \frac{k}{n-k} = \frac{n}{n-k} - 1. \qquad \square$$

Remark. A careful reader may have noticed that in the proof of Lemma 6, we are implicitly using the fact that the $b(v)$ values are all finite (as otherwise we cannot simply add/subtract them). Hence, to fully formalize things it also has to be proven that from any vertex v there is at least one path that reaches s, which implies $b(v)$ is finite. This is easy to prove so long as the graph is connected, an assumption that comes without loss of generality (why?).

Exercise 7. A k-edge coloring of a graph is an assignment of numbers $\{1, \ldots, k\}$ to its edges such that any two edges sharing an endpoint receive different colors. Show that by using the algorithm of Theorem 1 as a subroutine, any n-vertex bipartite graph of maximum degree Δ (not necessarily regular) can be edge-colored using Δ colors in $O\left(m \log^2 n\right)$ expected time.

References

[CKL+22] Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. *arXiv preprint arXiv:2203.00671*, 2022. 1

[Edm65] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965. 1

[GKK10] Ashish Goel, Michael Kapralov, and Sanjeev Khanna. Perfect matchings in o (n log n) time in regular bipartite graphs. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, pages 39–46, 2010. 1

[HK73] John E Hopcroft and Richard M Karp. An nˆ5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231, 1973. 1