Design and Analysis of Algorithms Lecture 18

Randomized Perfect Bipartite Matching

Lecturer: Daniel A. Spielman March 22, 2012

18.1 Introduction

These are my notes for part of today's lecture. In this half, we will cover a new randomized algorithm by Ashish Goel, Michael Kapralov and Sanjeev Khanna for finding perfect matchings in regular bipartite graphs. The paper, entitled "Perfect Matchings in $O(n \log n)$ Time in Regular Bipartite Graphs", appears in the Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC), 2010, and may also be found at `http://arxiv.org/abs/0909.3346`.

Recall that a graph is regular if every vertex has the same degree. It is easy to show that every regular bipartite graph has a perfect matching. This paper presents an incredibly simple algorithm for finding one. On a graph with n vertices and m edges, the algorithm finds one in expected time $O(n \log n)$!

Yes, you read that correctly. The running time does not depend on the number of edges present in the graph. At first this sounds absurd. After all, one should have to read the entire input. In general, that is true. In this case, the running time bound holds as long as the graph is provided in the correct format. What the algorithm needs is a list of the neighbors of every vertex, and this list needs to be presented in such a way that the algorithm can choose a random neighbor of a vertex. Given these lists, the algorithm runs in the stated time.

Note that this is much faster than the maximum matching algorithm we previously derived from the Ford-Fulkerson algorithm, which takes time $O(nm)$. But, the new algorithm can actually be understood as a randomized version of Ford-Fulkerson. Keep in mind that the new algorithm only works for bipartite regular graphs, whereas the algorithm we saw before finds maximum matchings in any bipartite graph.

18.2 Perfect Matchings in Bipartite Graphs

To begin, let's see why regular bipartite graphs have perfect matchings. Let $G = (X \cup Y, E)$ be a d-regular bipartite graph with $|X| = |Y| = n$. Recall that Hall's matching theorem tells us that G contains a perfect matching if for every $A \subseteq X$, $|N(A)| \geq |A|$. We will use this theorem along with some elementary counting to prove that G has a perfect matching.

Theorem 1. *If $G = (X \cup Y, E)$ is a regular bipartite graph, then $|N(A)| \geq |A|$ for every $A \subseteq X$.*

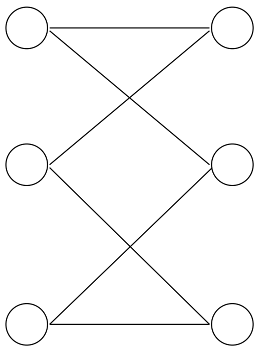
Proof. Let d be the degree of every vertex in G. As every vertex in A has degree d, there are $d\,|A|$ edges touching vertices in A. Similarly, there are $d\,|N(A)|$ edges touching vertices in $N(A)$. As every edge touching a vertex in A must also touch a vertex in $N(A)$, we have

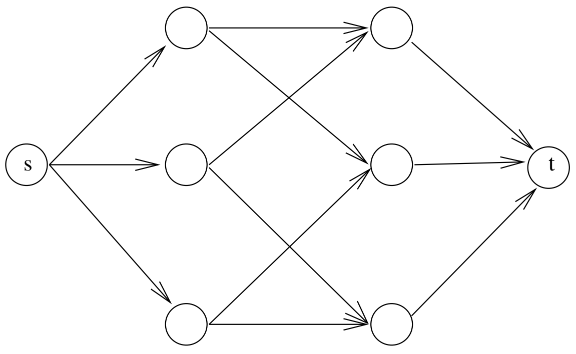
$$d\,|A| \leq d\,|N(A)|\,,$$

which implies $|A| \leq |N(A)|$. □

18.3 Review of Ford-Fulkerson

We now review how the Ford-Fulkerson algorithm can be applied to find maximum matchings in bipartite graphs. Let $G = (X \cup Y, E)$ be a bipartite graph between two vertex sets, X and Y, both of size n. To construct a maximum flow problem, we append two additional vertices, s and t, and add directed edges from s to every vertex in X and from every vertex in Y to t. We also direct every edge in E from X to Y. We now try to find a maximum flow in the new graph, taking all edge capacities to be 1.


(a) A bipartite graph

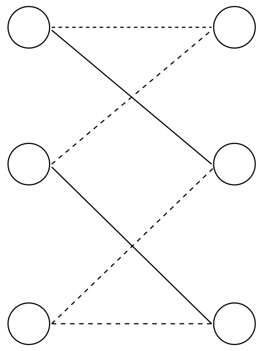


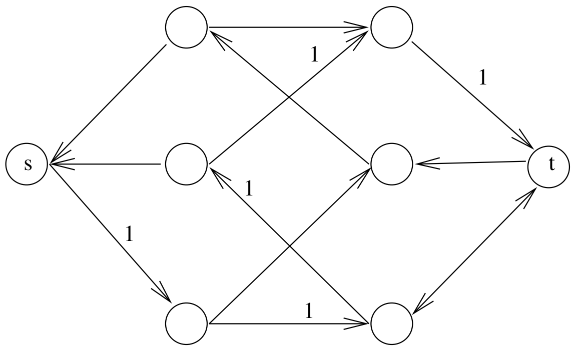
(b) The graph of the directed flow problem.

Let $M \subseteq E$ be a matching in the graph. It corresponds to a flow that sends one unit from s to x, from x to y, and from y to t for every edge $(x, y) \in M$. So, the residual graph of this flow reverses the directions of all the edges in the matching, and of all the edges between s and t and the matched vertices. Recall that the Ford-Fulkerson algorithm increases the flow by finding an augmenting path—a path from s to t in the residual graph. Such a path will never use one of the edges between s and a matched vertex, as it goes the wrong way. The same holds for the edges between t and the matched vertices. So, let's just ignore those edges.

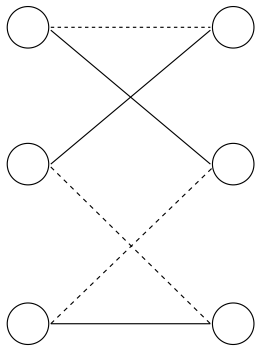
Now, let P be a path from s to t in the residual graph. When we add a flow of 1 along it, it has the effect of adding to the matching every edge in $E \cap P$ that was not in M, and removing from M every edge of $E \cap P$ that was in M, as you can see in the figure below.

Theorem 2. *Let $G = (X \cup Y, E)$ be a connected d-regular graph and let M be a matching in G of size less than n. Let H be the graph of the corresponding flow problem, let f be the flow corresponding to M, and let H_f be the residual graph. Then there is a path in H_f from every vertex to t.*


(c) The original matching



(d) The residual flow graph with a unit flow from s to t.


(e) The updated matching

Proof. Let X_B be the (possibly empty) subset of verties in X that do not have a path to t, and let Y_B be the analogous subset of vertices in Y. As every unmatched vertex in Y has a direct path to t, all of the vertices in Y_B must be matched.

For each vertex v that is matched, let $M(v)$ denote its match. Each matched vertex y has exactly one out-going edge in H_f, and it goes to $M(y)$. So, $y \in Y_B$ if and only if $M(y) \in X_B$. This tells us that $|Y_B| \leq |X_B|$.

On the other hand, all of the edges leaving an $x \in X_B$ must go to a vertex in Y_B. That is $N(X_B) \subset Y_B$. Theorem 1 now tells us that $|Y_B| \geq |X_B|$. Thus, we can conclude that $|Y_B| = |X_B|$, and that all of the edges touching vertices in X_B end at vertices in Y_B, and *vice versa*. As the graph is connected, this implies that either $Y_B = \emptyset$ or $Y_B = Y$. As we assumed that the size of the matching was less than n, it must be that Y_B, and thus X_B, is empty.

To see that there is a path from s to t, note that there must be some unmatched vertex in X, and so s has an edge poining to it in H_f. $\square$

18.4 The Randomized Algorithm

When we first saw Ford-Fulkerson, we found augmenting paths by running BFS from s. In the randomized algorithm, we will find them by taking random walks in the residual graph from s. We stop the random walk when it gets to t.

That is, we imagine a bug that begins at s. At each step, it follows a random edge leaving its present vertex. It stops when it reaches t. We let P be the path of edges that it follows. If we do this in the residual graph, we will find a path from s to t. It is possible that some vertices will appear twice on this path. But, that is easy to deal with: just remove the cycles from the path to make it a simple path.

We will show that the expected length of the path will be short. The length of the path we find will be 3 plus two times number of back-edges its follows. So, it will suffice to bound the expected number of back-edges in such a path. In particular, we will prove the following theorem.

Theorem 3. *Let $G = (X \cup Y, E)$ be a regular bipartite graph, let M be a matching containing k edges, and let G_M be the residual graph of the corresponding flow problem. The expected number of back edges appearing in a random walk from s that stops at t is at most*

$$\frac{n}{n-k} - 1.$$

This means that early on in the algorithm we take very few steps. If the present matching has fewer than $n/2$ edges, we expect to follow at most one back-edge!

Theorem 3 tells us that the expected number of steps we need to turn a matching of size k into one of size $k+1$ is at most

$$2 + 2\frac{n}{n-k}.$$

As we can sum expectations, we find the that expected number of steps required to go from a matching of size 0 to one of size n is at most

$$\sum_{k=0}^{n-1}(2 + 2\frac{n}{n-k}) = 2n + 2n\sum_{i=1}^{n}\frac{1}{i}.$$

The summation in this last expression is one that we frequently encounter. It is called the nth Harmonic number, and is known to approach $\gamma + \ln n$, where $\gamma = 0.577215\cdots$ is Euler's constant. For our purposes, we just require the fact that

$$\sum_{i=1}^{n}\frac{1}{i} \leq 1 + \ln n.$$

An easy proof of this comes from

$$\sum_{i=2}^{n}\frac{1}{i} \leq \int_{x=1}^{n}\frac{1}{x}dx = \ln(n).$$

Returning to the main story, we conclude that the expected number of steps in all the augmenting paths used to construct a perfect matching is at most

$$2n + 2n(1 + \ln n) = O(n \log n).$$

So, the algorithm takes expected time $O(n \log n)$.

18.5 Analysis of the random walk

We now prove Theorem 3. Assume that every vertex of G has degree d. Let X_M and Y_M denote the matched vertices in X and Y respectively. Also let X_U and Y_U denote the unmatched vertices. For any vertex $y \in Y$, let $M(y)$ be the vertex in X with which it is matched.

For each vertex v, let $b(v)$ denote the expected number of back-edges used in a random walk from v that stops when it reaches t. Our goal is to prove an upper bound on $b(s)$. We are going to do this by establishing relations between the values of $b(v)$ for various values of v, and then combining these relations. To be sure that we do not get garbage when we subtract one of these expectations from another, we should be careful to check that $b(v)$ is *finite* for all v. This follows from Theorem 2, which tells us that every vertex has a path to t. This means that if we walk for $2n$ steps from any vertex v, then there is some chance that we reach t. So, in the same way that we[1] computed the expected number of times we need to flip a biased coin before it comes up heads, we can compute an upper bound on the expected number of times we need to walk $2n$ steps before we reach t.

As the walk first moves from s to a random neighbor of s, which is a random node in X_U,

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

For $y \in Y_U$, the only edge leaving y points to t. So in this case $b(y) = 0$. On the other hand, a vertex $y \in Y_M$ has one edge leaving it, and it goes to $M(y)$. So,

$$b(y) = \begin{cases} 0 & y \in Y_U, \\ 1 + b(M(y)) & y \in Y_M. \end{cases}$$

A vertex $x \in X_U$ has d edges leaving it, all pointing to different vertices in Y. The expected number of back-edges in a walk from x will be the average of the expected number of back-edges in a walk from a random one of its neighbors. So, for $x \in X_U$,

$$b(x) = \frac{1}{d} \sum_{(x,y) \in E} b(y) \qquad \Longrightarrow \qquad db(x) = \sum_{(x,y) \in E} b(y).$$

A vertex $x \in X_M$ only has $d-1$ edges leaving, as it is attached to one back-edge. So, for such an x we have

$$b(x) = \frac{1}{d-1} \sum_{(x,y) \in E-M} b(y) \qquad \Longrightarrow \qquad (d-1)b(x) = \sum_{(x,y) \in E-M} b(y).$$

Now recall that for $(x,y) \in M$, $b(y) = 1 + b(x)$. By adding $b(x)$ to both sides, we find

$$db(x) = -1 + \sum_{(x,y) \in E} b(y).$$

By summing these equations for $b(x)$ over all $x \in X$, we obtain

$$d \sum_{x \in X} b(x) = -k + \sum_{x \in X} \sum_{(x,y) \in E} b(y)$$

As each vertex $y \in Y$ is attached to d vertices in X, this equals

$$-k + d \sum_{y \in Y} b(y),$$

[1] At least those of us who were at the lecture before Spring Break.

Plugging in our formulas for $b(y)$, we find

$$\sum_{y \in Y} b(y) = k + \sum_{x \in X_M} b(x).$$

So,

$$d \sum_{x \in X} b(x) = -k + dk + d \sum_{x \in X_M} b(x),$$

which gives

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

and

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

18.6 Conclusion

I recommend that you read the history of the paper. You will be shocked by the more complicated approaches were tried and by how long it took to obtain this result. It is a nice reminder that there remain untried, simple ideas that can result in substantial scientific progress.