CS 3510 Algorithms Last updated: October 16, 2023

Lecture 13: Matchings

Lecturer: Sahil Singla *Scribe(s): Joseph Gulian, Allen Chang*

1 Matching Problem

In matching problems we would like to choose a subset $M \subseteq E(G)$ such that each vertex is incident to at most one edge in M. This may be formalized with the following: for a graph $G = (V, E)$, a matching is an $M \subseteq E$ such that $\forall u \in V(G)$, $|N_e(u) \cap M| \leq 1$ (where for the purposes of notation N_e denotes the neighboring edges of u).

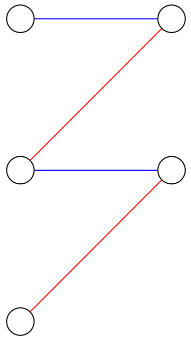

Figure 1: Two Potential Matchings

In the above example the blue edges and red edges are both matchings for the graph. Note that neither of these matchings are perfect, i.e. they "hit" every vertex. We introduce several definitions:

- These matchings are not perfect, because they leave at least one unmatched vertex (that is, a vertex not incident to an edge in M.
- Both the blue and the red matchings are maximum, because out of all possible matchings of G, they have the maximum cardinality (i.e., most people matched).
- Both the blue and the red matchings are maximal, because neither of the matchings can be made larger by adding more edges while keeping the edges already in the matching. Observe, for instance, that the matching $M = \{A, B\}$ where A, B are the top-most nodes, would not be a maximal matching, since we can add another edge into the matching.

You could think of an example where we pair students to work on a project, but students only want to work with their friends, so we need to find a matching such that nobody is in two groups and everybody who is matched is with their friend. Other applications could include matching rooms to classes, organ donor matching, or assigning medical students to schools[1].

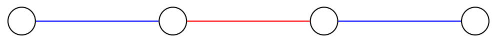

Figure 2: Maximum Matching

We'll be looking for finding a largest or maximum matching in a graph because if you think of the class partnering example, while we could make everyone work individually (a valid matching), we would like if the most people possible were matched together (the maximum matching). In the above graph, the red matching is a valid matching (note that it is maximal), but the blue matching is the maximum matching.

Our goal for this class will be to achieve a polynomial time algorithm to find a maximum matching, although we won't worry about the power within the polynomial. For instance if we get a polynomial of the form $\mathcal{O}(n^{1000})$, we'll be just as happy as if we get $\mathcal{O}(n^2)$. This comes from the idea that there's a big difference between polynomial time algorithms and exponetial time algorithms, so the first step in finding a good algorithm could be finding any polynomial time algorithm and then optimizing from there.

A matching M that has edges to all vertices is called a perfect matching. This may not exist, and is less interesting to us than the maximum matching problem where we have a matching M s.t. for any other matching M', $|M| \geq |M'|$. Looking at the graph above with the maximum matching, you may also notice that if you have a perfect matching, that will be the maximum matching on the graph.

2 A Greedy Approach

Greedy algorithms performed well on many graph problems, so let us try to find a greedy approach. Maybe if we go over all the edges one by one, we select an edge e into the matching if neither of the vertices it connects are matched. A simple counter example for this case is just the second figure; if we selected the red edge first, we would not find the maximum matching.

There's an interesting remark here that if you use the greedy approach, the matching will be at least half the size of the maximum matching. To give a simple sketch, if you match two nodes which could be matched separately, the worst you're doing is taking 1 match instead of two, and this could be formalized.

Instead of looking at a greedy approach if we take note that what worked in other cases like MST, we found some property of the minimum spanning tree (the cut lemma), and this allowed us to create a good algorithm. Then instead of first trying to develop a greedy approach, let's try to develop an idea about our solution and then use that to develop an algorithm.

[1] we're not covering stable matching, but it may be a topic of interest

3 Augmenting Paths

Suppose we have a subroutine which given a matching will either tell you it is the optimal matching or trying to increase it's size by one. In other words, the goal is to improve the sub-optimal matching by one. We'll achieve this through use of augmenting paths. An M-augmenting path for a given matching M is a path in G, s.t.

1. The path alternates between matched and unmatched edges.
2. The first and last vertices are unmatched in M.

Intuitively, why is this augmenting? Imagine that we have a path $P = v_1, v_2, \dots, v_n$ where $v_1 v_2 \notin M$ and $v_{n-1} v_n \notin M$. For all even i, we have $v_i v_{i+1} \in M$ (this is our original matching). To augment it, for each $v_i \in P$, we switch the edge in M that is incident to v_i. To see this, consider the following example, where the red edges denote my current matching M:

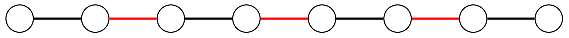

Figure 3: An M-Augmenting Path

To augment it, we switch the endpoint for each vertex that is adjacent to an edge in the M-augmenting path:

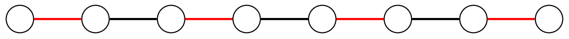

Figure 4: M after Augmentation

and we now have a larger matching. Now we have improved because we will have four edges instead of three edges.

We will soon prove a lemma: M is maximum matching if and only if there is no augmenting path. This says two things: when we have no augmenting path, we have found the maximum matching, and when we have not found the maximum matching, there exists an augmenting path. Then we have realized that finding a maximum matching reduces to finding an augmenting path.

3.1 Proof of Lemma

($\Leftarrow$) We will show the contrapositive. Assume that M is not the maximum matching; we will show that there will always exist an augmenting path. Let $M^\star$ be the maximum matching, so $|M| < |M^\star|$, we want to show that M has an augmenting math.

Consider the symmetric difference of M and $M^\star$: $M \triangle M^\star = (M|M^\star) \cup (M^\star|M)$. For intuition, we color the edges in M red and the edges in $M^\star$ blue.

Now, observe that the edges of $M \triangle M^\star$ form cycles or paths[2] In a cycle, there must be the same number of blue and red edges. In a path, this may not be the case. Since the

[2]Exercise: why is this true? Hint: what happens if the vertices have more than three incident edges in the symmetric difference? Furthermore, why must all cycles of $M \triangle M^\star$ always have even length?

number of blue edges is strictly greater than the number of red edges by assumption, i.e. $|M^\star| > |M|$, there must exist a path that starts and ends at blue edges. This, by definition, is an M-augmenting path, and we are done.

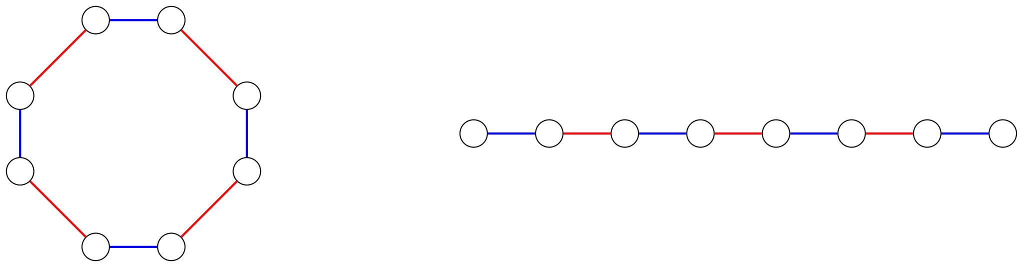

Figure 5: Situations for Matchings

($\Rightarrow$) We would like to show the contrapositive; that if there is an M-augmenting path, M is not a maximum matching. We showed this earlier, so we are done.

The takeaway here is that to find a max-matching in polynomial time, it suffices to find a augmenting path in polynomial time. This is actually a challenging problem for general graphs, so we're simplify the problem a bit by only using bipartite graphs. A graph is bipartite if there exists a partition of vertices A, B, such that all edges are of the form a, b where $a \in A, b \in B$.

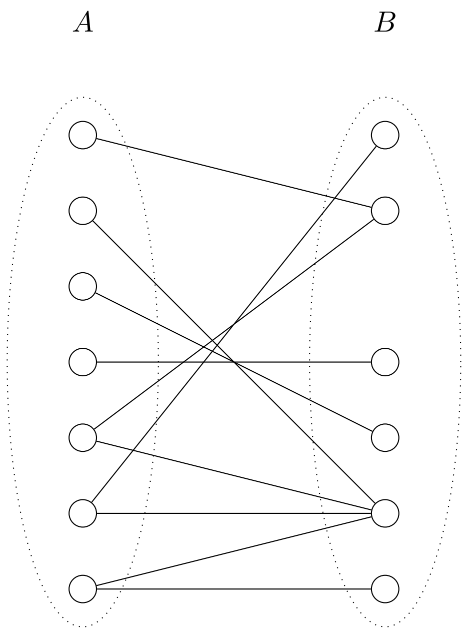

Figure 6: A Bipartite Graph

We'll observe the following things about augmenting paths.

1. Augmenting paths have odd lengths (see Figure 5 above)

2. Augmenting paths start and end at unmatched vertices (by definition)

Now, we would like to find an M-augmenting path in a bipartite graph. Observe that since an M-augmenting path must have odd length, it must start and end in different partitions of G. WLOG, we assume that such a path must start and end in A; thus, we would like to find such a path that begins in an unsaturated vertex $a \in A$ and ends in an unsaturated vertex $b \in B$.

Now, we are almost done. Observe that it suffices to find a directed path from an unmatched vertex in A to an unmatched vertex to B. This is easy to do via BFS in $\mathcal{O}(m)$ time, where m is the number of edges. Since we may want to augment at most n times (the number of vertices), then this algorithm finds a maximum matching in $\mathcal{O}(mn)$ time, and we are done.

So, now we know how to find matchings in Bipartite graphs; there also exist algorithms for finding matchings in general graphs, but these are too challenging to cover in this class.