COMPSCI 330: Design and Analysis of Algorithms **October 31, 2017**

Lecture 16 : Bipartite Matching

Lecturer: Rong Ge *Scribe: Will Wang*

1 Overview

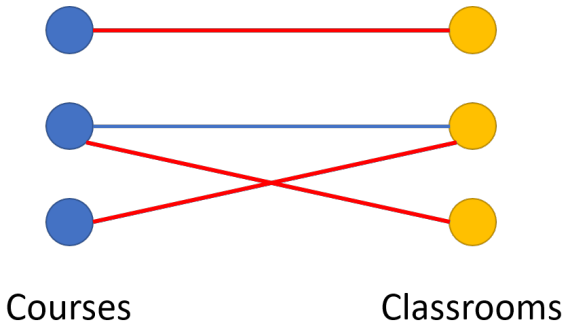
Last lecture we finish Minimum Spanning Tree (MST) problem and this lecture we introduce Bipartite Matching problem. We first introduce the concept of Bipartite Matching problem. We then show a Maximum Bipartite Matching algorithm and prove its correctness.

2 Motivation: Classroom Assignment

Consider the following motivating problem:

There are n courses and m classrooms. Because of different requirements (capacity, facility, location etc.) each course can only use a subset of classrooms. For each course, we are given the list of classrooms that it can use. **Goal:** Find a way to schedule all courses in the classrooms.(each course only need one classroom, and each classroom can only hold one course)

Modeling as Graph Problem: We could model courses and classrooms as two sets of vertices. If course i can be scheduled into classroom j, we draw an edge(i,j) between vertex i and vertex j. Then the solution should be a set of edges that do not share any vertices. (a matching)



3 Maximum Bipartite Matching

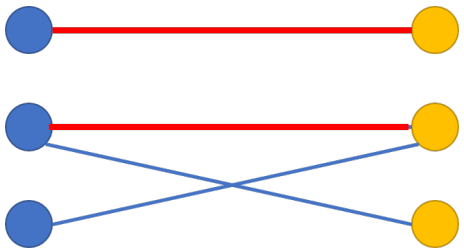
Now we present the Maximum Bipartite Matching problem. Given a bipartite graph with n vertices on one part, and m vertices on the other part. We try to find a maximum matching in the graph. Relating back to our motivating problem: matching corresponds to an assignment of courses to classrooms and maximum matching corresponds to scheduling max number of courses.

Bipartite Graph: A Bipartite Graph $G = (V_1, V_2, E)$, E is a subject of (i, j) where $i \in V_1, j \in V_2$.

Matching: A matching M is a subset of E, such that edges in M do not share vertices. The size of a matching M is just the number of edges in M.

3.1 Greedy Algorithm

Let's first consider a naive greedy algorithm. For each course, if it has a classroom that is not taken by any other course, schedule the course in that classroom.



It's easy to show that greedy algorithm is not the optimal. Consider above example, choosing blue edges could make 3 matchings. However greedy algorithm could choose red edges with 2 matchings. Clearly, greedy algorithm can produce suboptimal solution to this problem.

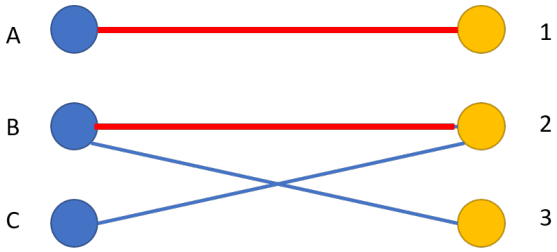
3.2 Augmenting Path Algorithm

Before introducing augmenting path algorithm, we first state several definitions. Given a Bipartite Graph G and matching M:

Matched Edge: An edge e is matched if $e \in M$ and unmatched if otherwise.

Matched Vertex: A vertex v is matched if it's connected to some $e \in M$ and unmatched if otherwise.

Augmenting Path: An augmenting path is a path from an unmatched course to an unmatched classroom, that alternates between unmatched edges and matched edges. In below example, path (C, 2), (2, B), (B, 3) is an augmenting path. By definition, we can easily derive the following, **Claim:** An augmenting path P has an odd number of edges, and it has exactly 1 more unmatched edges than matched edges.



3.2.1 XOR Operation

We denote XOR operation of set of edges by $\oplus$. If A and B are two subsets of edges, $A \oplus B$ is also a subset of edges. $e \in A \oplus B$ if $e \in A, e \notin B$ or $e \notin A, e \in B$.

Claim: If P is an augmenting path for M, then $M' = M \oplus P$ is also a matching and $|M'| = |M| + 1$.

This is not hard to see if we observe that every augmenting path P, as it is alternating and it starts and ends with a free vertex, must be odd length and must have one edge more in its subset of unmatched edges than in its subset of matched edges.

3.2.2 Basic Algorithm

The main idea of the algorithm becomes apparent at this point: we initialize matching $M = \emptyset$ and while there is an augmenting path in the graph, we find the augmenting path P and let $M = M \oplus P$.

There are two things we should be careful of. The first is how do we find augmenting path. Under case of motivating example, we first start from an unmatched course. If we are at a course vertex, we need to follow an unmatched edge (to get to a classroom). If we are at a classroom vertex and it is not matched, we are done! Otherwise, we need to follow the (only) matched edge. We could implement this using either DFS or BFS.

Secondly, to prove the correctives of this algorithm, we need to show the following theorem is true.

Theorem 1. *Given a bipartite graph G and a matching M, if there is no augmenting path with respect to this matching M, then M is a maximum matching.*

Proof. We use proof by contradiction: Start with the assumption that there is a larger matching M^* ($|M^*| > |M|$), and then we can show that this implies the existence of some augmenting path.

Look at set $Q = M \oplus M^*$. We observe the following facts. First, as both M and M are matchings, each vertex in Q can have degree not greater than 2, because it can be adjacent to at most one edge from each matching. Therefore, Q is an union of vertex-disjoint paths and cycles.

On each such path or cycle, edges of M' and M alternate. Now we have 3 cases. In a cycle, we have case 1: number of edges is same in M and M^*. In a path, we have either case 2: M^* has 1 more edge which is the an augmenting path, or case 3: M has 1 more edge.

We know that case 2 must happen because $|M^*| > |M|$. Thus, we must have an augmenting path. □

3.2.3 Implementing the algorithm

Pseudo-code to implement the algorithm under the context of motivating example is included below

Algorithm 1 Augmenting Path Algorithm

```
1: procedure FindPath(u)
2:     Mark u as visited
3:     for all edges (u, v) do                ▷ Enumerate over classrooms that course u can use
4:         if v is not visited then
5:             if v is unmatched or FindPath(matchRoom[v]) = true then   ▷ if either we found an empty classroom, or the current instructor of the classroom is able to switch to another room
6:                 matchCourse[u] = v
7:                 matchCourse[v] = u                                    ▷ I will take this room
8:                 return true                                           ▷ I have found a room for course u.
9:             end if
10:        end if
11:    end for
12:    return false          ▷ I have tried all the possible rooms, they are not empty and their instructor cannot switch to another room, so I cannot find a room for course u.
13: end procedure
14: procedure MaxMatching
15:    Initially set all nodes to be unmatched
16:    for u=1 to n do                                                   ▷ enumerate the courses
17:        Mark all vertices as unvisited                                ▷ initialize for the DFS
18:        FindPath(u)                                                   ▷ Try to schedule course u.
19:    end for
```