CMPSCI 611 Advanced Algorithms **Fall 2001**

Lecture 7: September 26, 2001

Lecturer: Micah Adler *Scribe: Rachel Smith, Yan Zeng*

7.1 Lecture Overview

- Matchings
- Intersections of Matroids

Matroids are a very powerful tool that allow us to determine if a greedy algorithm will find the optimal solution for a problem. Bipartite matching is a good place to begin looking at the intersection of matroids. In studying bipartite matchings in the last lecture, we built the machinery that we will use throughout our examination of problems involving the intersection of matroids. In this lecture, we will study the use of augmenting paths and the intersection of matroids in finding the maximum bipartite matching, and we will conclude with a reason why a greedy algorithm may not find the maximum matching in a non-bipartite graph.

7.2 Augmenting Paths

We begin with the bipartite graph given in the previous lecture.

Input: A Bipartite Graph $B = (U, V, E)$, where U, V are two sets of vertices and E a set of edges. Since B is bipartite, $(\forall (u, v) \in E).(u \in U \text{ AND } v \in V)$.

Output: A matching of the bipartite graph B of maximum size.

Definition 7.1 *Given a matching, a* **free vertex** *is a vertex that is not incident to any edge in the matching. In Figure 7.1 and 7.2 the grey vertices are free vertices.*

Definition 7.2 *Given a matching M, an* **augmenting path** *P for M is any path of alternating matching and non-matching edges, beginning and ending on a free vertex. Figure 7.2 shows an augmenting path for the matching in Figure 7.1.*

Using *augmenting paths* is a way of building matchings one edge at a time (although we will later show how to use them in any intersection of matroids problem). Like the greedy method, one edge will be added at a time. However, because this is not a matroid problem we do not have a guarantee that once an edge is accepted it will always be accepted. Therefore, we must create some method of backtracking.

Consider the following example (Figure 7.1) of a bipartite graph on which we have found a non-maximal matching. The edges in the matching are circled.

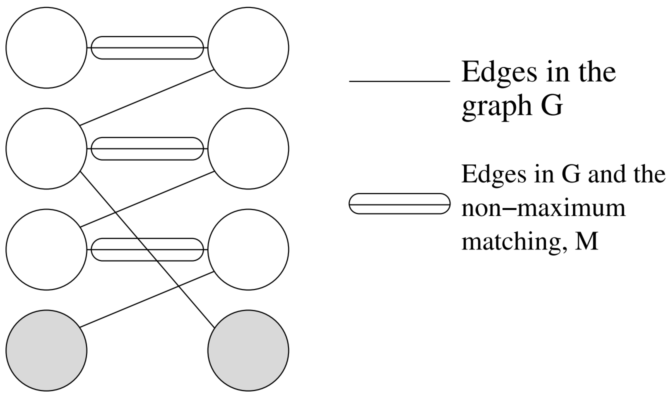

Figure 7.1: A bipartite graph G with a non-maximal matching

Now consider an augmenting path, P, for M. The edges in the augmenting path are dashed lines.

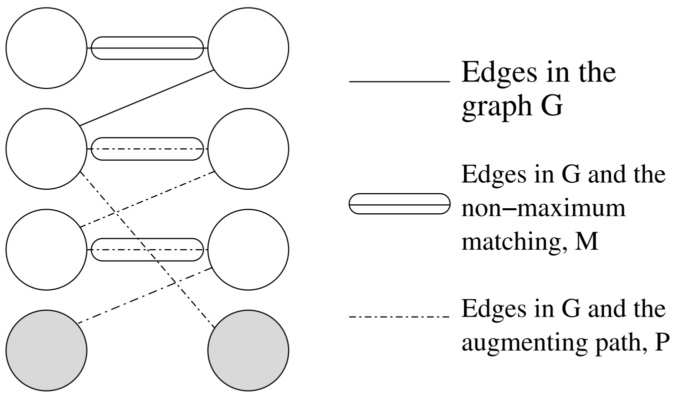

Figure 7.2: An augmenting path for the matching in Figure 7.1

Definition 7.3 *Given a matching M and an augmenting path P for M the* **symmetric difference** *between M and P is $M \cup P - M \cap P$ (i.e. all edges that exist in either M or P, but not both). If we call the symmetric difference M', we would write this as $M' = M \oplus P$.*

Figure 7.3 shows the symmetric difference between our the matching in Figure 7.1 and the augmenting path in Figure 7.2:

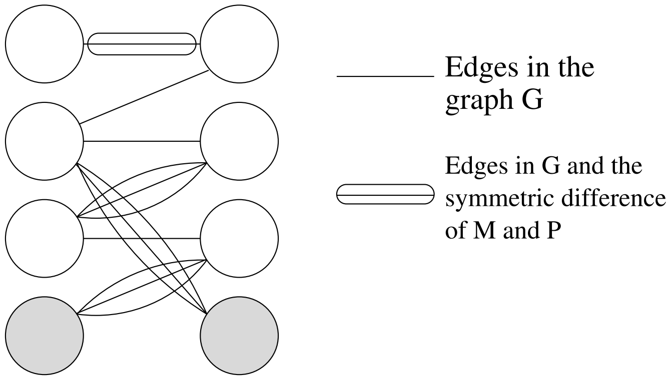

Figure 7.3: The symmetric difference between Figure 7.1 and Figure 7.2

7.2.1 Algorithm for Maximum Bipartite Matching

We can now define an algorithm for finding the maximum bipartite matching:

$M = \emptyset$
while there exists an augmenting path P
    $M \leftarrow M \oplus P$
endwhile
return(M)

7.2.1.1 Algorithm Proof of Correctness

To prove that this algorithm is correct, we need to prove:

1. At each iteration, M is a matching. (Corollary: the final output is a matching.)

   Theorem 7.4 *Given any matching M and an augmenting path P, $M' = M \oplus P$ is also a matching.*

   Proof: Consider edges $e_1, e_2 \in M'$, there are three possible cases for the membership of e_1 and e_2:

Case 1: $e_1, e_2 \in M - P$
Since M is a matching and $e_1, e_2 \in M$, they cannot share a common vertex.

Case 2: $e_1, e_2 \in P - M$
If a vertex is free, then there is at most one non-matching edge incident to it in the augmenting path. Every other vertex has two edges incident to it: one matching edge and one non-matching edge. Since e_1 and e_2 are in the augmenting path and both are non-matching (i.e. neither is in M), they cannot share a vertex.

Case 3: $e_1 \in M - P$, $e_2 \in P - M$
If e_2 is incident to a free vertex, by definition 7.1, e_2 cannot share the vertex with any edge $e_1 \in M$. If both e_2 and e_1 are incident to the same (non-free) vertex, then e_1 would have to be a member of both M and P, by definition 7.2. But $e_1 \in M - P$, so this is not possible. Therefore, e_1 and e_2 cannot share a non-free vertex.

Since no edge in M' shares a vertex with any other edge in M', M' is a matching. ∎

2. The algorithm terminates.

   Claim: $|M'| = |M| + 1$

   Since the augmenting path P starts and ends on a free vertex and we alternate edges not in M with edges in M, every even edge in P will be removed. Therefore we must add 1 edge. There is a finite number of vertices, therefore this algorithm terminates.

3. The algorithm returns a maximum matching.

   Lemma 7.5 *If a matching M is not maximum, then there exists an augmenting path for M.*

   Proof: Assume there exists a non-maximum matching M and a maximum matching M', $|M'| > |M|$. Let $E' = M \oplus M', G' = (V, E')$. The connected components of G:

   (a) (Figure 7.4a) Cycles of even length must have edges that alternate between being a part of M and M'.

   (b) (Figure 7.4b) The augmenting path does not have to finish at where it began; it could be an "alternating path."

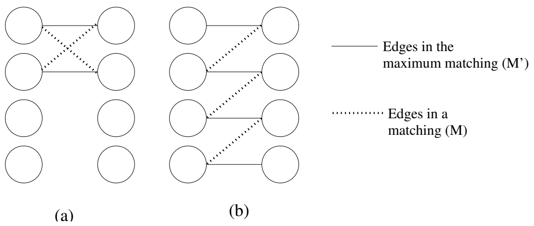

Figure 7.4: An example of cycles and a maximum matching

Consider the graphs in Figure 7.4. All of the vertices have degree ≤ 2. If $v \in V$ has degree of 2, then there exists an edge incident to v in M and a different edge incident to v in M'. We also know the connected components in G' must either be cycles or augmenting paths. The cycles must be of even length and have equal numbers of edges from M and M'. However $|M'| > |M|$, so there must exist a path with more edges in M' than in M. Since that path cannot be a cycle, it must be an augmenting path. ∎

Note that we still have not determined how to find an augmenting path nor how to find if an augmenting path exists, but merely the properties we can use once we have an augmenting path. We will now show how we can find these augmenting paths.

7.2.2 Finding An Augmenting Path

All we have to do now is find the augmenting paths. If we didn't have the alternating matching and non-matching edge requirement, we could simply do a breadth-first-search (BFS) from a free vertex to find an augmenting path.

So how do we find the augmenting path?

- Start at a free vertex

- Is there an alternating path to another free vertex?
- Use searching to find a path
- We must modify the search to make it find an alternating matching/non-matching path
- To modify the search, make the edges in G' directed.
- Perform depth-first(or breadth-first) search, making the start nodes the vertices of only one side of the bipartite graph.

7.2.3 Running Time for Maximum Bipartite Matching

Now that we have a complete algorithm for maximum bipartite matching, what is the running time?

- Number of augmenting paths: We add one edge per augmenting path, therefore we will need as many augmenting paths as the size of the maximum bipartite matching. Since we cannot have more edges in the maximum matching than vertices in one set, the time is: $O(min(|U|, |V|))$
- Time to find each augmenting path: We need to start a BFS from each node. By saving our search results we only need to visit each vertex once. The time for each path is: $O(|E|)$

Total running time = $O(|E| * min(|U|, |V|))$.

7.3 Extension to Generic Intersection of Matroid Problems

Extension to any subset system that is intersection of (E, I) and (E, J) that are matroids.

(E, I): E = the edges of the graph and I = acyclic subsets of E

(E, J): E is as above and J = all edges such that at most one incoming edge per vertex, except the root vertex which has no incoming edges.

The idea here is to find a sequence of elements $e_1, e_2, e_3, \ldots, e_{odd}$
Assume we start with an element m such that $m \in I \cap J$

Find an e_1 such that $m + e_1 \in I$ and $m + e_1 \notin J$	`an edge from some free vertex`
Select an e_2 such that $m + e_1 - e_2 \in I \cap J$	`add a matching edge`
Select an e_3 such that $m + e_1 - e_2 + e_3 \in I$ and $m + e_1 - e_2 + e_3 \notin J$	`add the next non-matching edge`
...and so on until ...	`...and so on until ...`
$m + e_1 - e_2 + e_3 \ldots - e_{even} + e_{odd} \in I \cap J$	`add the last edge to a free vertex`

If this sequence exists, then for any intersection of matroids we can find it using BFS. This alternating path *does* exist (we do not provide a proof of this), and we can apply it to a lot of applications. One of the applications is a directed version of the Minimum-Spanning Tree problem.

7.3.1 Application of Intersection of Matroids

7.3.1.1 Branching for a Directed Spanning Tree

Problem: Find a branching for a directed spanning tree (also known as an arborescence).

Definition: A branching is a subset of edges of the graph, such that:

1. the corresponding undirected graph is a tree, and
2. there is some root vertex such that all edges in the branching point away from the root vertex.

Or: This is an equivalent statement of the requirements:

1. the corresponding undirected graph is a tree, and
2. each vertex has exactly one incoming edge, except the root vertex.

Input: A directed, acyclic graph $G = (V, E)$ with a root vertex r:

Output: If there is a valid branching rooted at r, then return it.

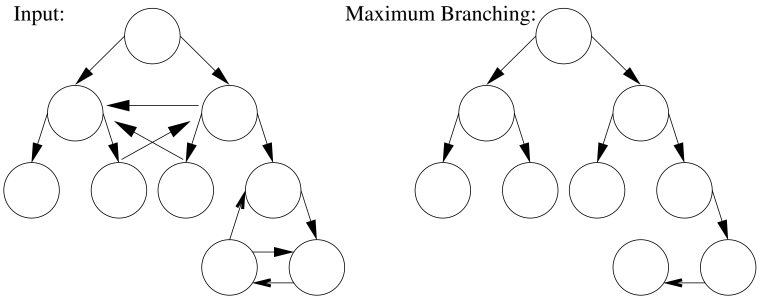

Figure 7.6: An example of a directed, acyclic graph and a branching of the graph

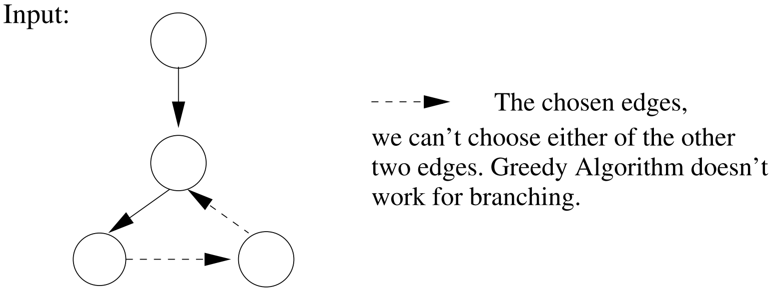

Figure 7.7: An example of when Greedy doesn't work

There is a faster algorithm that runs in time: $O(|E| + |V|log|V|)$.

References

1 HERTZ, M. and SIEH, J., Scribe Notes for CS611 Advanced Algorithms Lecture 6, *UMass Amherst Computer Science, 2000*