CS 341 - Algorithms, Spring 2021. University of Waterloo

Lecture 15 : Bipartite Matching

We study the augmenting path algorithm for the bipartite matching problem.

Introduction

The bipartite matching problem is an important problem both in theory and in practice.

In these two lectures, we will study efficient algorithms for solving bipartite matching and its "dual" problem minimum vertex cover, and see some interesting and non-trivial applications of these problems.

We will learn a new technique called "the augmenting path" method to solve the bipartite matching problem.

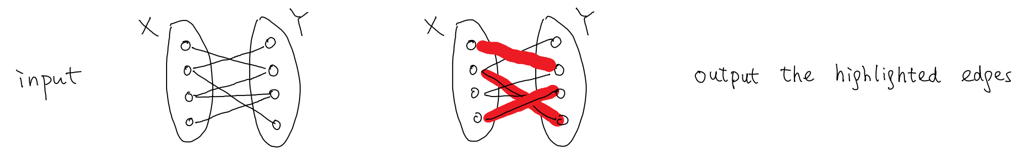
This is an important technique that underlies many algorithms for combinatorial optimization problems, including the network flow problem that you can learn from CO351.

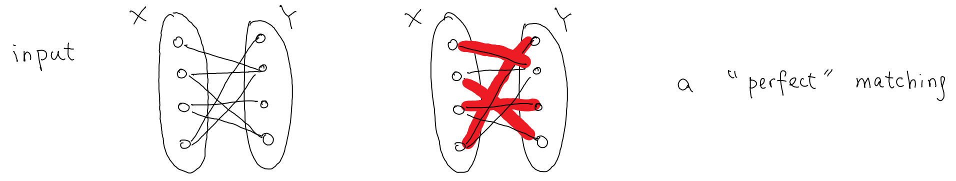
More generally, the augmenting path method can be understood as one way of solving combinatorial optimization problems using the general framework of linear programming. We won't get this far; see [DPV].

Problem

Input : A bipartite graph $G=(X,Y;E)$.

Output : A maximum cardinality subset of edges that are vertex disjoint.





A subset of edges $M \subseteq E$ is called a matching if edges in M are pairwise vertex disjoint, or in other words, no two edges in M share a vertex.

Given a matching $M \subseteq E$, we say a vertex v is matched if v is an endpoint of some edge $e \in M$; otherwise we say v is unmatched or free.

We are interested in finding a maximum matching, a matching with the maximum number of edges.

A matching is called a perfect matching if every vertex in the graph is matched.

Obviously, a perfect matching is the best that we can hope for for the maximum matching problem.

As shown in the above example, not all bipartite graphs have a perfect matching.

We will see a nice characterization of graphs that don't have a perfect matching.

Applications

There are many interesting and non-trivial applications of bipartite matching, as we will see later.

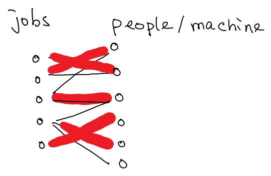
Here we first see a standard and useful application, which also explains why the bipartite matching problem is sometimes called the assignment problem.

We are given n jobs, and m people / machines.

Each person / machine is only capable of doing a subset of jobs.

Our task is to assign all the jobs to people / machines, without assigning more than one job to a person / machine.

We can model this as a bipartite matching problem.



We create one vertex for each job, and one vertex for each person / machine.

We add an edge between a job vertex j and a person vertex p iff person p is capable of doing job j. So, the graph is bipartite.

By construction, a matching corresponds to an assignment of jobs to people s.t. no one is assigned more than one job.

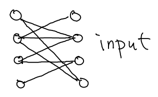
Let |J| be the number of jobs. Then the assignment problem is possible iff there is a matching of |J| edges.

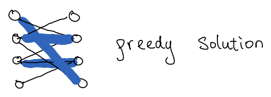
Augmenting Path Method

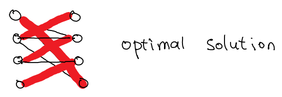
We study a new algorithmic approach to solve the bipartite matching problem.

A natural first approach is to go greedy: whenever there is an edge $uv \in E$ with both endpoints free, then we add the edge uv to our partial solution and repeat until there are no such edges.

This may not find a maximum matching as the following example shows.







How do we know what edges are in an optimal solution?

Actually, we don't know of an easy way to tell whether an edge belongs to some maximum matching or not.

This is unlike in the shortest path problem or in the minimum spanning tree problem, where we can prove that a shortest edge or a minimum weight edge can always be extended to an optimal solution.

Instead of making a greedy decision and commit to it, the new approach is to find efficient ways to improve the current solution if it is not optimal.

So, the augmenting path method can be understood as a local search algorithm.

By the way, in case you wonder, there are no known dynamic programming algorithms for bipartite matching.

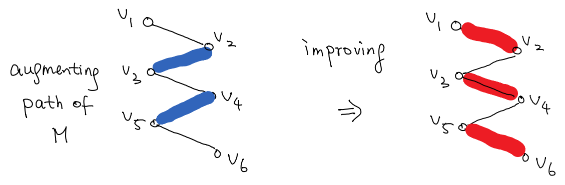
Definition (Augmenting Path)

A path $v_1, v_2, \dots, v_{2k}$ is an augmenting path of a matching M if

① v_1 and v_{2k} are free / unmatched.

② $v_{2i-1} v_{2i} \notin M \quad \forall\ 1 \le i \le k$.

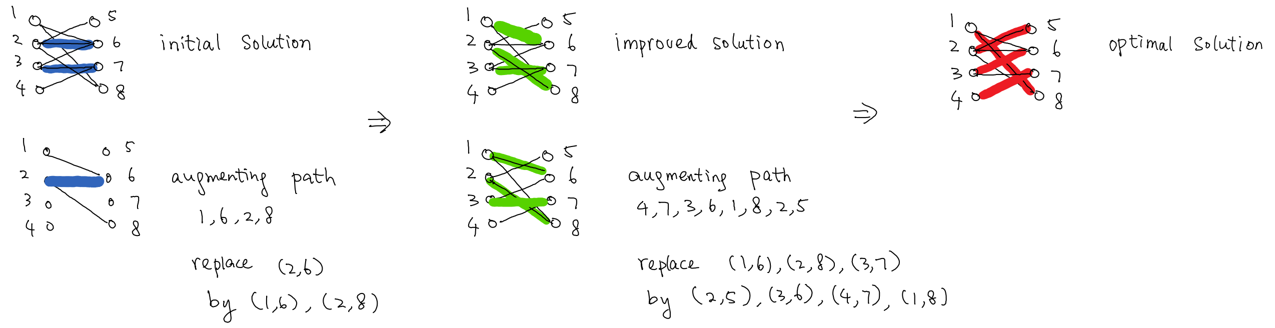
③ $v_{2i} v_{2i+1} \in M \quad \forall\ 1 \le i \le k-1$.



If we have found an augmenting path, we can use it to improve the matching size by one, by removing the edges $v_{2i} v_{2i+1}$ $\forall\ 1 \le i \le k-1$ (the even edges) from the matching M and and adding the edges $v_{2i-1} v_{2i}$ $\forall\ 1 \le i \le k$ (the odd edges) to the matching M.

Note that an edge with both endpoints free is an augmenting path of length one.

Consider the following example.



Perhaps surprisingly, it turns out that finding an augmenting path is all one need to do.

<u>Proposition</u> M is a maximum matching if and only if there is no augmenting path of M.

<u>Proof</u> (⇒) This direction is easy. We have basically done it.

The contrapositive is: if there is an augmenting path of M, then M is not a maximum matching.

Let $M' := M - \{ v_{2i} v_{2i+1} \mid 1 \le i \le k-1 \} + \{ v_{2i-1} v_{2i} \mid 1 \le i \le k \}$, where $v_1, \dots, v_{2k}$ is an augmenting path.

Then $|M'| = |M| + 1$.

Note that M' is still a matching since v_1 and v_{2k} were free in M and now have degree one in M', while $v_2, \dots, v_{2k-1}$ still has only one incident edge in M' (just that their partner is changed).

So, M' is a larger matching than M, and thus M is not maximum.

(⇐) The other direction is nontrivial and is very important for the correctness of the algorithm.

The contrapositive is: if M is not a maximum matching, then there is an augmenting path of M.

Let M^* be a maximum matching. Then $|M^*| > |M|$ by our assumption.

We will focus only on the edges in $M \cup M^*$ (the graph may have other edges but we don't need them).

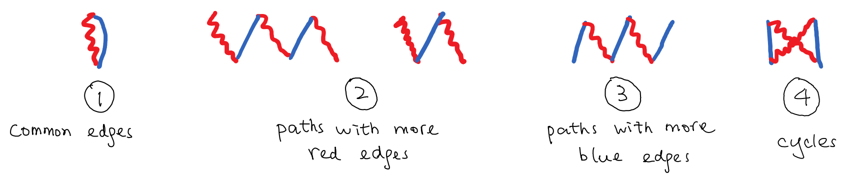
We claim that there must be an augmenting path of M in $M \cup M^*$

Since both M and M^* are matchings, the subgraph $M \cup M^*$ is of maximum degree two.

So, the connected components in $M \cup M^*$ are either a path or a cycle.

Call the edges in M blue (drawn —), and the edges in M^* red (drawn ∼∼∼).

The union $M \cup M^*$ looks like this.



Note that if there exists a component of type ② in $M \cup M^*$, then we have found an augmenting path of M because the two endpoints of the path are free in M and the edges in the path are alternating.

So, it remains to argue that a component of type ② must exist, which follows from the assumption that $|M^*| > |M|$, as in all other types the number of blue edges is at least as many as the number of red edges.

To conclude, if M is not maximum, then there must exist an augmenting path of M. $\square$

The proposition suggests the following "local search" algorithm for finding a maximum matching.

<u>Algorithm</u> (bipartite matching)

```
$M = \emptyset$    // start from an empty matching
While there is an augmenting path $P = v_1, v_2, \dots, v_{2k}$ of $M$ do
    $M \leftarrow M - \{v_{2i}v_{2i+1} \mid 1 \le i \le k-1\} + \{v_{2i-1}v_{2i} \mid 1 \le i \le k\}$
Return $M$.
```

<u>Correctness</u> follows from the proposition, which says that if there is no augmenting path, then the matching is maximum.

<u>Time complexity</u>: Let $T(m,n)$ be the time complexity to find an augmenting path of M if it exists, or report that no such paths exist, in a graph with n vertices and m edges.

Every time we have found an augmenting path, we can use it to increase the matching size by one.

Since the matching size is at most $n/2$, there are at most $n/2$ iterations in the while loop.

Therefore, the time complexity of the algorithm is $O(n \cdot T(n,m))$.

In the next section, we will show that an augmenting path can be found in $O(m+n)$ time, and thus the bipartite matching problem can be solved in $O(n \cdot m)$ time.

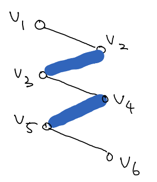
<u>Faster algorithm</u>: There is a faster augmenting path algorithm by Edmonds and Karp, which solves the bipartite matching problem in $O(m\sqrt{n})$ time.

Finding Augmenting Paths

The bipartite graph structure allows us to design a simple algorithm to find an augmenting path.

First we need to find a free vertex v_1 on the left.

If there is a neighbor v_2 of v_1 which is unmatched, then we are done because we have found an augmenting path of length one ("free" edge v_1v_2).

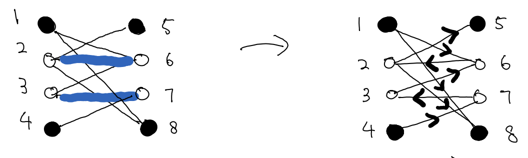


Otherwise, v_2 is matched, and to extend it to an augmenting path, we have no choice but to follow the matching edge of v_2 to go back to the left, call the matching edge v_2v_3.

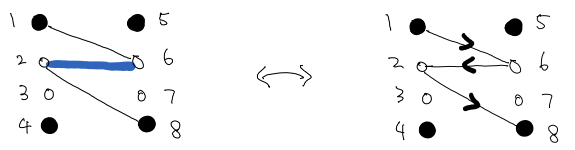
Then, we repeat the above step: if v_3 has an unmatched neighbor v_4, then we found an augmenting path; otherwise we follow the matching edge v_4v_5 to go back to the left.

<u>Idea</u>: Observe that every time we go to the right, either we are done or we follow a matching edge that takes us back to the left, because of the bipartite graph structure.

The important idea is to encode the color information on the edges by directions, such that each unmatched edge points from the left to right and each matched edge points from right to left.



G $\qquad$ $\vec{G}_M$



augmenting path of M in G $\qquad$ directed path in $\vec{G}_M$

Given a bipartite graph $G=(X,Y;E)$ and a matching $M \subseteq E$, we construct a directed graph $\vec{G}_M$ on the same vertex set and the same edge set, while the direction of every edge in M is from X to Y and the direction of every edge in $E-M$ is from Y to X.

By construction, we have the following correspondence between augmenting paths in G and directed paths in $\vec{G}_M$.

<u>Claim</u> There is an augmenting path of M in $G=(X,Y;E)$ if and only if there is a directed path in $\vec{G}_M$ from a free/unmatched vertex in X (on the left) to a free/unmatched vertex in Y (on the right).

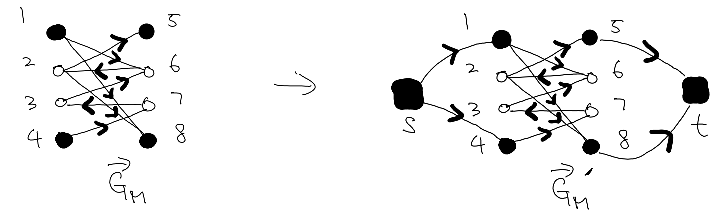
The claim allows us to reduce the problem of finding an augmenting path of M in G to a "reachability" problem in $\vec{G}_M$, for which we know how to solve by BFS/DFS.

A direct implementation is to start a BFS/DFS on each free vertex on the left and check if it can reach a free vertex on the right, but this could take $\Omega(n(n+m))$ time.

There is a simple trick to solve this "multiple-source multiple-sink" problem in $O(n+m)$ time.

We just add a super-source vertex on the left, with directed edges to the free vertices on the left, and we add a super-sink vertex on the right, with directed edges from the free vertices on the right.

and we add a super-sink vertex on the right, with directed edges from the free vertices on the right.



By construction, we reduce the augmenting path problem in G to the s-t reachability problem in $\vec{G}'_M$.

Claim There is a directed path from a free vertex on the left to a free vertex on the right in $\vec{G}_M$ if and only if there is a directed path from s and t in $\vec{G}'_M$.

Corollary There is an augmenting path of M in G if and only if there is a directed path from s to t in $\vec{G}'_M$.

Algorithm (augmenting path)

Input: a bipartite graph $G=(X,Y;E)$, and a matching $M \subseteq E$.

Output: an augmenting path P of M in G, or report that no such paths exist.

1. Construct the directed graph $\vec{G}'_M$ as described above.
2. Use BFS/DFS to determine if there exists a directed path P in $\vec{G}'_M$ from s to t.

   If yes, return P (forget the directions and also remove the edge from s and the edge to t).

   If no, return "No".

Time Complexity: It should be clear that it is $O(m+n)$.

Remark: In actual implementation, we can work directly on the directed graph and it is easy to update, so that we don't need to construct $\vec{G}'_M$ in each iteration. We leave the details to the reader.

Puzzle/Challenge: Can you solve the maximum matching problem in general (non-bipartite) graphs?

Go through the bipartite matching algorithm and see which step breaks.

Edmonds, a former C&O professor, designed a famous "blossom" algorithm to solve maximum matching.

Tutte, another former C&O professor, also done important work in the non-bipartite matching problem.