# CS 341 - Algorithms, Spring 2017. University of Waterloo

## Lecture 15: Bipartite matching

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

We will present an efficient algorithm, solve its "dual problem" minimum vertex cover, and show some interesting and nontrivial applications of the problem.

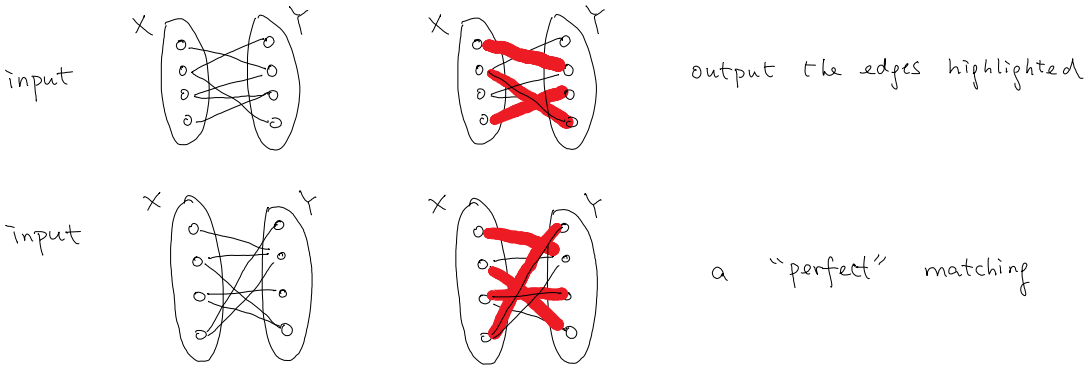
The "augmenting path method" to solve the bipartite matching problem is a key technique underlying most of the developments in combinatorial optimization, including the network flow problem that you can learn from CO 351.

---

### 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 <u>matching</u> if no two edges in $F$ share a vertex; in other words, edges in $F$ are vertex disjoint.

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

A matching is called a <u>perfect matching</u> if every vertex in the graph is matched; Obviously, a perfect matching is the best that we can hope for.

We are interested in finding a <u>maximum matching</u>, a matching of maximum cardinality.

Note that a graph may not have a perfect matching; see the first example above.

We will characterize when a graph does not have a perfect matching later.

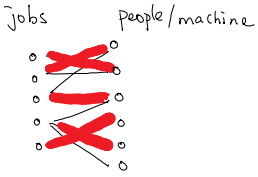
### Applications

## Applications

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

Here we first see a standard and useful application, and this is 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. We would like 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, one vertex for each person/machine.

There is an edge between a job vertex j and a person vertex p if and only if person p is capable of doing job j.

Then, it is easy to see that a matching corresponds to an assignment of jobs to people such that no person is assigned more than one job.

Let $|J|$ be the number of jobs. Then the assignment problem is possible if and only if there is a matching of $|J|$ edges.

---

## Augmenting path method

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

The most tempting approach is to go greedy: whenever there is an edge $e=uv$ with both $u$ and $v$ free, then we add $e$ to our (partial) solution and repeat.

This may not work as the following example shows:


input


greedy solution


optimal solution

How do we know what edges are in the optimal solution in the above example?

Actually, we don't know of an efficient way to tell whether an edge will be in an optimal solution or not, unlike in the shortest path problem or in the minimum spanning tree problem where we proved that a shortest edge / a minimum weight edge can be extended to an optimal solution.

Instead of making a decision and commit to it, the new approach is to find ways to improve the current solution.

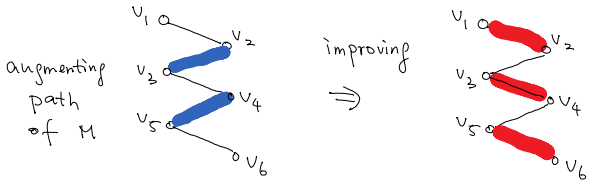
Definition (augmenting path)

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

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

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

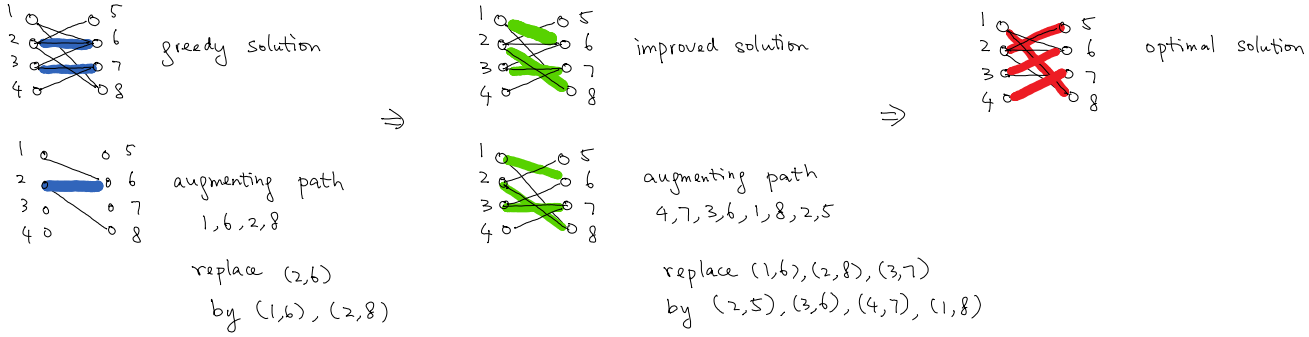
③ $v_{2i}\, v_{2i+1} \in M \quad \forall \quad 1 \le i \le k-1$.




If we have found an augmenting path, we can use it to improve the matching size by one, by replacing the edges $v_{2i}\, v_{2i+1}$ (even edges) with the edges $v_{2i-1}\, v_{2i}$ (odd edges).

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

Consider the above example again.




It turns out that finding augmenting paths is all we need to do.

Lemma M is a maximum matching if and only if there is no augmenting path of M.

proof (⇒) This direction is easy. We have essentially done it.

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

This should be clear from the above discussion. We replace the edges $v_{2i}\, v_{2i+1}$ by the edges $v_{2i-1}\, v_{2i}$. The number of edges is increased by one.

It is still a matching since $v_1$ and $v_{2k}$ were free before, and $v_2, \ldots, v_{2k-1}$ still has only one incident edge in the matching.

So, we find a larger matching and M is not maximum.

(⇐) The other direction is non-trivial and is very important for the algorithm.

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

(Why we are always proving the contrapositive?! Anyway...)

Let $M^*$ be a maximum matching, so that $|M^*| > |M|$.

Focus on the union $M \cup M^*$. (The graph may have other edges but we don't need them.)

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

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

The union looks like this:


$M$ and $M^*$ may share some common edges (case 1). We ignore those edges and just focus on their difference. We know that the difference is non-empty, since $|M^*| > |M|$.

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

Then, all the connected components are either a path (case 2 or 3) or a cycle (case 4).

Now, note that if case 2 happens, 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 case 2 must happen.

This follows from the fact that $|M^*| > |M|$: in all other cases, we have the number of blue edges (in $M$) is at least as many as the number of red edges (in $M^*$).

So, given $|M^*| > |M|$, case 2 must happen (as otherwise we conclude that $|M| \geq |M^*|$, contradiction) and there is an augmenting path of $M$.

To conclude, if $M$ is not maximum, then there is an augmenting path of $M$. □

The lemma suggests the following algorithm for finding a maximum matching.

Algorithm (bipartite matching)

$M = \phi$ // start from an empty matching

while there is an augmenting path $P$ of $M$

use it to improve the current matching size by one as described above

(if we have introduced enough mathematical notation, we can then write $M \leftarrow M \oplus P$)

return $M$.

Correctness follows from the lemma, which says that if we can't find an augmenting path of $M$, then $M$ is a maximum matching (and so we are done).

Time complexity Let $T(m,n)$ be the time complexity to find an augmenting path of $M$ if it exists and report none exists.

Every time we find an augmenting path, we increase the matching size by one.

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

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

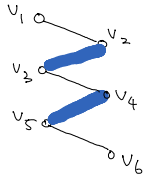
Of course, the algorithm is not complete yet, since we haven't discussed how to find an augmenting path yet, but it will turn out that we can do it in $O(m+n)$ time, and so bipartite matching can be solved in $O(mn)$ time.

---

## Finding an augmenting path

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

We need to start from a free vertex $v_1$ on the left.



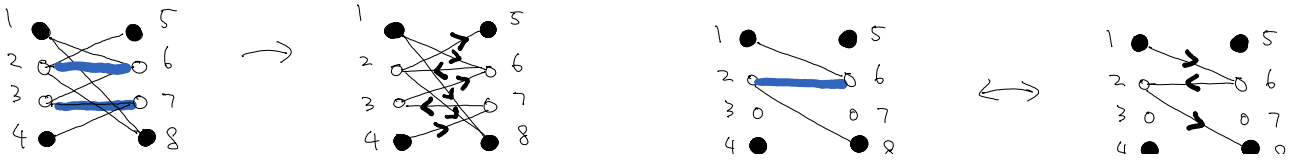

If there is a neighbor $v_2$ of $v_1$ which is unmatched, then we are happy because we have found an augmenting path of length one (a "free edge").

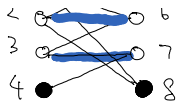
Otherwise, $v_2$ is matched, and to find an augmenting path, we have no choice but to follow the matching edge of $v_2$ to go back to the left, say $v_2v_3$ is the matching edge.

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

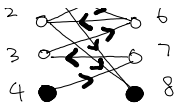
Notice that every time we go to the right, either we are done or have to follow the matching edge to go back to the left.

The important idea here is that we can encode this information using a directed graph.

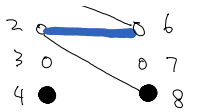




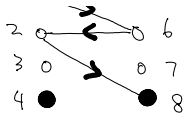

G




$\vec{G}_M$




augmenting path of M in G




directed path in $\vec{G}_M$

Given a bipartite graph $G$ and a matching $M$, we construct a directed graph $\vec{G}_M$ by having all the edges in $M$ to point from right to left, while having all other edges in $E-M$ to point from left to right.

Now, there is a one-to-one correspondence between augmenting paths in $G$ and directed paths in $\vec{G}_M$.

Claim There is an augmenting path of $M$ in $G$ if and only if there is a directed path in $\vec{G}_M$ from a free vertex on the left to a free vertex on the right.

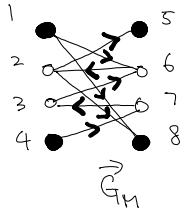
proof follows from the construction. See the picture above; black vertices are free vertices.

The claim allows us to reduce the problem of finding an augmenting path in an undirected graph to a reachability problem in a directed graph.

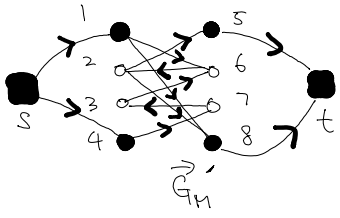
A direct implementation is to do a DFS/BFS from each free vertex to see if it can reach a free vertex on the right. That could take $\Theta(mn)$ time.

A simple trick can solve this reachability 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 add a super-sink vertex on the right, with directed edges from free vertices on the right to the super-sink vertex.




$\vec{G}_M$




$\vec{G}'_M$

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 $s$ to $t$ in $\vec{G}'_M$.

proof follows from the construction. □

Corollary There is an augmenting path of $M$ in $G$ if and only if there is a directed

path from $s$ to $t$ in $\vec{G}'_M$.

Proof Combine the two claims in this section. □

From the corollary, we reduce the problem of finding an augmenting path to finding an $s$-$t$ path in a directed graph.

We know how to solve the latter problem using BFS/DFS in $O(m+n)$ time, so we solve the augmenting path problem in $O(m+n)$ time.

Algorithm (augmenting path)

Input: bipartite graph $G=(V,E)$, a matching $M \subseteq E$

Output: an augmenting path of $M$ in $G$, or report none exists.

- Construct the directed graph $\vec{G}'_M$ as described above.
- Use BFS/DFS to determine if there exists a path $P$ from $s$ to $t$ in $\vec{G}'_M$:
  return $P$ (forget the directions and delete the edges from $s$ and to $t$) if it exists,
  return none if it doesn't exist.

In implementation, we can directly work with the directed graph and it is easy to update. There is no need to "construct" the graph in every iteration.

We leave these details to the reader.

**Puzzle:** Can we solve the maximum matching problem in general (non-bipartite) graphs?

There is a sophisticated algorithm by Edmonds (a former CSO professor) to solve the problem in general graphs (the famous "blossom" algorithm).

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

**Challenge:** Can you figure out an algorithm for general graphs?

**Remark:** It is known that if every time we use a shortest path from $s$ to $t$, which can be done by BFS in $O(n+m)$ time, along with some simple modifications, the bipartite matching problem can be solved in $O(m\sqrt{n})$ time. We don't have time

to do this unfortunately. I'm sure you can learn it from C&O.