CS 341 - Algorithms, Winter 2016. University of Waterloo

Lecture 16: Bipartite vertex cover

We study the "dual" problem of maximum bipartite matching: the minimum vertex cover problem in bipartite graphs.

We will prove a min-max theorem relating matchings and vertex covers in bipartite graphs. This will help us understand concisely when a graph has no perfect matching for instance.

We then discuss further applications of maximum bipartite matching.

Vertex cover in bipartite graphs

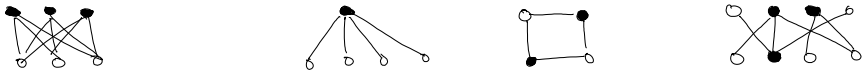
Given a graph $G=(V,E)$, a subset of vertices $S \subseteq V$ is called a <u>vertex cover</u> if for every edge $uv \in E$, $\{u,v\} \cap S \neq \phi$. In other words, S is a vertex cover if for every edge $uv \in E$, at least one of the two endpoints is in S, i.e. S intersects every edge.

Obviously, the whole vertex set is a vertex cover, but we are interested in a vertex cover of minimum cardinality.

Input: A bipartite graph $G=(V,E)$

Output: A vertex cover of minimum cardinality.

Some examples:



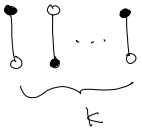
You could imagine that this problem (in general graphs) is useful in computer networks, say by using the minimum number of computers to monitor all the links in the network.

Min-max theorem

The vertex cover problem seems to have nothing to do with the matching problem, but they are actually "dual" of each other, in a sense two sides of a coin.

To start to see their connection, first we observe that if there is a matching of size k, then any vertex cover must have at least k vertices.

The reason is simple: since the k matching edges are vertex disjoint, we must use at least one vertex to cover each such edge and these vertices



must be distinct, and so any vertex cover must have at least k vertices.

Therefore, the size of a maximum matching is a lower bound on the size of a minimum vertex cover.

Surprisingly, this is a tight lower bound, i.e. there is always a vertex cover of that size.

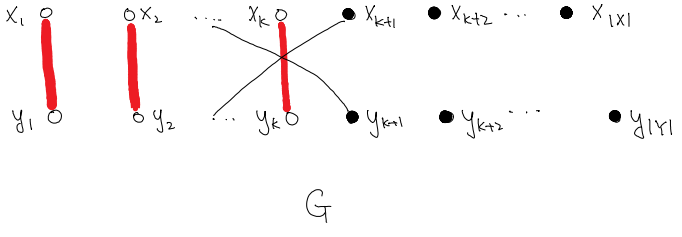
Theorem (König) For a bipartite graph, the size of a maximum matching is equal to the size of a minimum vertex cover.

proof We will give an algorithmic proof using the augmenting path algorithm.

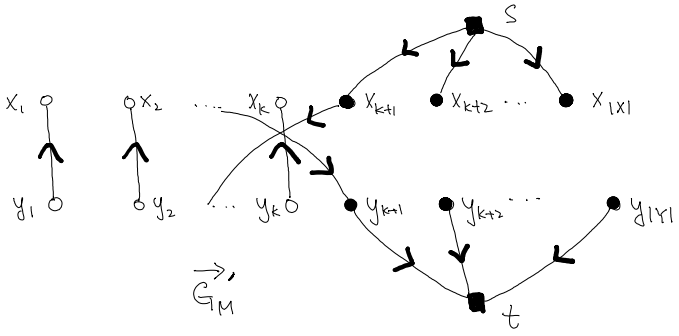
We will prove that given the current matching M of size k, if we couldn't find an augmenting path of M, then we can find a vertex cover $C \subseteq V$ of size k.

This would imply that M is a maximum matching and C is a minimum vertex cover, because we know that there can't be a matching larger (because of C) and can't be a vertex cover smaller (because of M).

Following this plan, let's consider the iteration when there is no augmenting path of M.



G



$\vec{G}'_M$

Let the vertices on one side of the bipartite graph be $x_1, x_2, \ldots, x_{|X|}$, and the vertices on the other side be $y_1, y_2, \ldots, y_{|Y|}$.

Let the current matching M be $x_1y_1, x_2y_2, \ldots, x_ky_k$, having k edges.

Then the free/unmatched vertices are $x_{k+1}, \ldots, x_{|X|}$ and $y_{k+1}, \ldots, y_{|Y|}$, colored back in the picture.

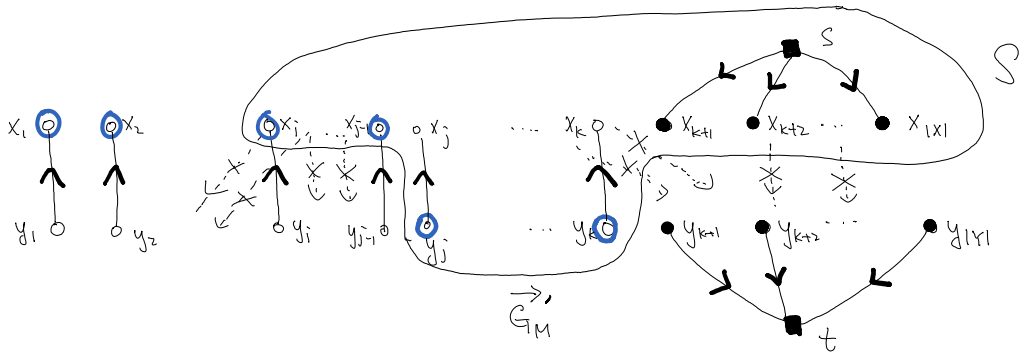
To search for an augmenting path of M, we consider the directed graph $\vec{G}'_M$ on the right, in which there is a super-source s with directed edge sx_i $\forall\ k+1 \le i \le |X|$, and there is a super-sink t with directed edges y_it for all $k+1 \le i \le |Y|$.

All the edges in $\vec{G}'_M$ pointed downward (from X to Y), except the edges in the matching pointed upward (from Y to X). Not all the edges are shown in the picture.

Now, suppose there is no augmenting path of M.

By last lecture, it is equivalent to that there is no directed path from s to t in $\vec{G}'_M$.

From what we have learned about BFS/DFS, let's call the set of vertices reachable from s be S, then we know that there are no directed edges with its tail in S and its head in $V-S$ (otherwise we could reach some vertices outside S).



First, we make sense of this picture.

Since s cannot reach t, no vertices in $\{y_{k+1}, \ldots, y_{|Y|}\}$ are in S, as otherwise s can reach t.

Next, if $y_\ell \in S$ for some ℓ, then we also have $x_\ell \in S$, because $y_\ell x_\ell$ is a directed edge but there cannot be any outgoing edges from S.

So, in the picture, $\{y_j, y_{j+1}, \ldots, y_k\}$ is in S, and so $\{x_j, x_{j+1}, \ldots, x_k\}$ is also in S.

We could have some vertices $\{x_i, \ldots, x_{j-1}\}$ in S while $\{y_i, \ldots, y_{j-1}\}$ not in S, no contradiction here.

The following claim will finish the proof, by showing that there is a vertex cover of size k.

Claim $\{x_1, x_2, \ldots, x_{j-1}\} \cup \{y_j, \ldots, y_k\}$ is a vertex cover of G. (See the highlighted blue vertices.)

Proof First, we focus on the edges not in the matching, i.e. the edges in $E-M$.

Since S has no outgoing edges, there are no directed edges from $\{x_i, \ldots, x_{|X|}\}$ to $Y - \{y_j, \ldots, y_k\}$, and this means that no edges in $E-M$ are between $\{x_i, \ldots, x_{|X|}\}$ and $Y - \{y_j, \ldots, y_k\}$, and thus all edges in $E-M$ from $\{x_i, \ldots, x_{|X|}\}$ must go to $\{y_j, \ldots, y_k\}$.

The remaining edges in $E-M$ must have one endpoint in $\{x_1, \ldots, x_{i-1}\}$, and that's why we put them in the vertex cover, so that all edges in $E-M$ are covered by $\{x_1, \ldots, x_{i-1}\} \cup \{y_j, \ldots, y_k\}$.

Finally, we need to cover the matching edges $x_i y_i, \ldots, x_{j-1} y_{j-1}$, and the vertices $\{x_i, \ldots, x_{j-1}\}$ would do (the vertices $\{y_i, \ldots, y_{j-1}\}$ would also do).

Hence, $\{x_1, \ldots, x_{j-1}\} \cup \{y_j, \ldots, y_k\}$ is a vertex cover of size k. □

This proves that both the matching and the vertex cover are optimal, and hence the min-max theorem. □

The following algorithm is immediate from the algorithmic proof

Algorithm (bipartite vertex cover)

- Use an efficient algorithm to find a maximum matching M.
- Construct the directed graph $\vec{G}_M'$.
- Do a BFS/DFS to identify all vertices reachable from s.
- Return $\{x_1, \ldots, x_{j-1}\} \cup \{y_j, \ldots, y_k\}$ as shown in the picture.

Time complexity is dominated by the first step, since the remaining steps only take $O(n+m)$ time.

Using the augmenting path algorithm gives a $O(mn)$-time algorithm for minimum vertex cover in bipartite graphs. Using the $O(m\sqrt{n})$-time algorithm for bipartite matching (that we haven't discussed) would lead to an algorithm for vertex cover in the same time complexity.

Good characterization

König's theorem is a nice example of a graph-theoretic characterization.

To see this, imagine that you work for a company and your boss asks you to find a maximum matching to assign the jobs to the employees.

If your algorithm finds a perfect matching, then your boss is happy.

But suppose there is no perfect matching in the graph, how could you convince yous boss that it is indeed the case?

Your boss may just think that you are incompetent and other smarter people could do a better assignment.

Well, with the augmenting path algorithm in the last lecture, you could try to explain that there is no augmenting path for your matching and so it is maximum.

If you have a good boss, then he/she may understand it or at least take your words for it.

For those who are not so lucky, a more convincing way to show your boss a vertex cover of size $< \frac{n}{2}$, and explain that if there is a perfect matching such a vertex cover could not exist.

This kind of min-max theorems is some of the most elegant results in combinatorial optimization, giving succint proofs for both the YES cases (exists a large matching) and the NO cases (no large matching, exists a small vertex cover).

Remark: To put the above discussion into perspective, consider the dynamic programming algorithms.

Even though we could solve the problems (such as edit-distance) in polynomial time, we don't have such a nice succint characterization for the NO cases.

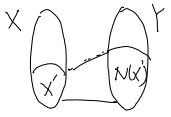
If your boss asks why you couldn't find a way to have edit-distance k-1 when your algorithm says the optimal is k, it would be pretty difficult to explain (e.g. we search for all possible states using this recurrence and there is no better solution).

The "proof" is still short in the sence that it is of polynomial size, but it is not as elegant as the proofs provided by the min-max theorems.

Hall's theorem characterizes when a bipartite graph $G=(X,Y;E)$ has a perfect matching or not.

Without loss of generality, we assume that $|X|=|Y|$, otherwise there is no perfect matching

Theorem (Hall) A bipartite graph $G=(X,Y;E)$ has a perfect matching if and only if for every subset $X' \subseteq X$, we have $|N(X')| \geq |X'|$, i.e. any subset X' must have at least $|X'|$ neighbors. (where $N(X')$ is the neighbor set of X')



Problem: Derive Hall's theorem using König's theorem.

Hall's theorem is an important result and has many applications in graph theory.

We show one corollary and may use this corollary in homework.

Corollary A d-regular bipartite graph $G=(X,Y;E)$ has a perfect matching.

proof First, observe that $|X|=|Y|$ in a d-regular bipartite graph.

There are $d|X|$ edges in the graph.

Any subset of vertices of size $\leq |X|-1$ can cover at most $d(|X|-1)$ edges.

So, any vertex cover needs at least $|X|$ vertices.

By König's theorem, there is a matching of size at least $|X|$, hence a perfect matching. □

Applications

There are many non-trivial and interesting applications of graph matching and network flows (which can be shown to be equivalent).

We don't have time, and will just show one interesting application.

We will do it in two steps.

Capacitated job assignment

This is just a slight generalization of the job assignment problem in the beginning of last lecture.

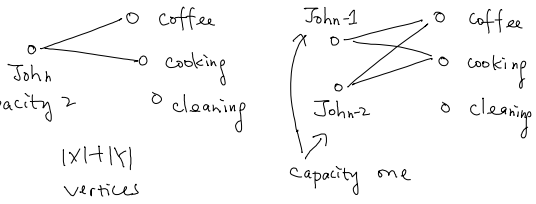
The same problem as before, but now each person/machine i has a different capacity c_i, representing how many jobs that person/machine i could handle (instead of having capacity one for each person).

Then, again, we want to assign all the jobs to people, so that no person will be overloaded according to the capacity.

Do you see how to solve this problem?

We don't need to solve it again. There is a simple reduction to the original problem.

Say John has capacity 2. We can simply create 2 vertices for John, say John-1 and John-2, connecting to the same subset of jobs as John did.



Then, it is clear that there is an assignment of all jobs satisfying the capacities if and only if there is a matching in the new graph with all jobs matched.

This reduction may not be efficient, as the number of vertices in the new graph could become $|X||Y|$, as each person could have a capacity as high as $|X|$.

But it is still polynomial in terms of the input instance, and this is good enough for our purpose.

There is a direct way to work on the more compact representation (like what we did for the Dijkstra's algorithm as a simulation of BFS in the big graph in L10), but we won't do it.

Baseball/basketball league winner

Suppose you are a fan of the Toronto Blue Jays or Toronto Raptors or Waterloo Warriors.

They are playing in the regular season. They are doing okay but not at the top of the table.

You wonder whether it is still (mathematically) possible for them to be the top at the end of the season.

Input: The current standing, and the remaining schedule.

Output: whether it is possible or not that your team can win the league.

The feature of the baseball/basketball league is that there is only win or lose for each game, i.e. there is no draw like football.

Suppose your team has currently won W games.

First, it is obvious that you assume that they will win all remaining games, for a total of W^* games.

Then, you want to know whether there is a scenario that no other teams will win more than W^* games.

Let the remaining teams be $\{1, \ldots, n\}$ and currently team i has won w_i games.

You have the remaining schedule, showing that there are still g_{ij} games between team i and team j.

How do you determine if there is still a scenario that your team can still win the league?

It is not difficult to reduce it to the capacitated assignment problem once you look at it the right way.

We want to "assign" the wins in such a way that no team would win more than $w^* - w_i - 1$ games.

Then this is exactly the capacitated assignment problem.

To be concrete, say Toronto Raptors has won 43 games, with 25 games remaining, and so they can win up to 68 games in the best case scenario.

Then, you look at the remaining teams, say Chicago Bulls has won 50 games, and so you want that they can win at most 17 games, and you set this capacity for each team.

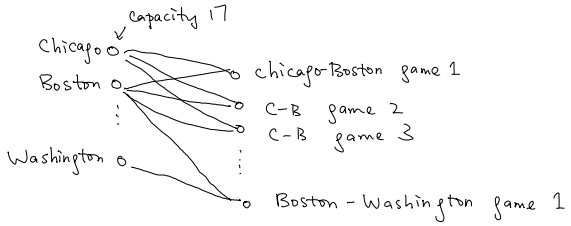
Now, you want to "fix" the remaining games in such a way that no other teams will win more than 67 games.

Each game will result in one win, and you want to "assign" the wins so that they don't go over the capacities.

Say there are still three games left between Chicago Bulls and Boston Celtics, then you create three games (think of them as three jobs in the assignment problem) connecting to them.

Then, Toronto can still win the league if and only if there is a way to assign all the games to the remaining teams so that the capacities are not violated.

This can be solved by bipartite matching!



Remark: It is NP-hard to determine if you favourite football team can still win the league (i.e. winner takes 3 points, each team takes 1 point in a draw).

Other applications: We can solve the network flow problem by the techniques used in bipartite matching.

Then we can find the maximum number of edge-disjoint paths between two vertices s and t, which is an important application both in theory and in practice.

You can take a look at [KT, DPV, CLRS] for many applications of the network flow problem.

Extensions: The maximum matching problem can be solved in general graphs

Even the maximum weighted version, where every edge has a weight, can be solved in polynomial time, a groundbreaking result by Edmonds.

This can be used, through a clever reduction, to solve the Chinese postman problem that we mentioned in the beginning of the course.

Duality Why vertex cover is called the "dual" problem of matching? How we come up of it?

It turns out that these is a systematic way to define the dual problem of an optimization problem, through the use of linear programming.

Most combinatorial optimization problems can be solved in the general framework of linear programming, one of the most (if not the most) powerful algorithmic framework for polytime computations.

Unfortunately we won't talk about linear programming in this course; see [KT, DPV, CLRS] for more, and/or take some courses in the C&O department.
