COS 330: Great Ideas in Theoretical Computer Science

Fall 2025

Precept 3

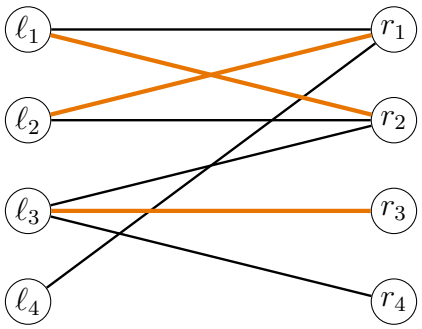
My Flow

Learn

In the Learn section of today's precept we'll see a complete self-contained analysis of the bipartite matching algorithm we saw in lecture.

First recall the statement of the problem. You are given a bipartite graph $G = (L, R, E)$, where L and R ($|L| = |R| = n$) are the left and right vertex sets, respectively, and E ($|E| = m$) is the set of edges such that each edge $(u, v) \in E$ has $u \in L$ and $v \in R$. A *matching* is a subset of edges $M \subseteq E$ such that no two edges in M share an endpoint. Find the maximum matching of G, i.e., the matching with the largest number of edges.

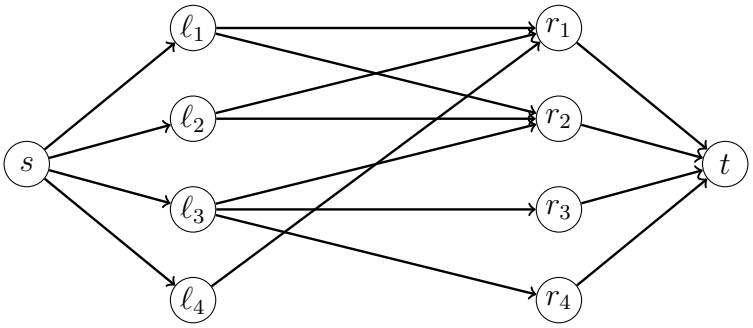
Here is an example of a bipartite graph whose maximum matchings have 3 edges, one of which is shown in orange.



We can solve this problem by constructing a flow network $G' = (V', E')$ from G such that the maximum flow in G' corresponds to the maximum matching in G. We set $V' = L \cup R \cup \{s, t\}$, where s is a new source node and t is a new sink node. The edge set E' is constructed as follows:

- For each vertex $u \in L$, create a directed edge from s to u with capacity 1.
- For each vertex $v \in R$, create a directed edge from v to t with capacity 1.
- For each edge $(u, v) \in E$, create a directed edge from u to v with capacity 1.

For the above example, the corresponding flow network is shown below.



Once we have constructed G', we can run the Ford-Fulkerson algorithm to find the maximum flow from s to t. The value of the maximum flow will be equal to the size of the maximum matching in G. The running time of this algorithm is $O(nm)$, since the value of the maximum flow is at most n.

To prove that the above works we will show two things: if f is a flow in G' with value $|f|$, then we can construct a matching M in G such that $|M| = |f|$, and if M is a matching in G with size $|M|$, then we can construct a flow f in G' such that $|f| = |M|$. Note that this implies that whatever flow the algorithm finds corresponds to a matching of the same size, and that the maximum matching corresponds to a flow of the same size, so the algorithm must find the maximum matching.

From flow to matching. Suppose f is a flow in G' with value $|f|$. We can construct a matching M in G as follows: for each edge $(u, v) \in E$ such that $f(u, v) = 1$, add (u, v) to M. In other words, we are adding to M all edges that carry flow from L to R. Consider any edge $(u, v) \in M$, where $u \in L$ and $v \in R$. The indegree of u is 1 (from s), which means at most 1 unit of flow can leave u (by the flow conservation property). Similarly, the outdegree of v is 1 (to t), which means at most 1 unit of flow can enter v. Thus, there is no other vertex $u' \in L$ such that $f(u', v) = 1$ or $f(u, u') = 1$, so no two edges in M share an endpoint.

We have established that M is a matching. To see that $|M| = |f|$, first note that the value of the flow $|f|$ is equal to the total flow leaving s. Since the capacity of each edge leaving s is 1, the total flow leaving s is equal to the number of edges (s, u) such that $f(s, u) = 1$. By the flow conservation property, for each such edge (s, u) there must be exactly one edge (u, v) such that $f(u, v) = 1$. Thus, the number of edges (s, u) such that $f(s, u) = 1$ is equal to the number of edges (u, v) such that $f(u, v) = 1$, which is equal to $|M|$. Therefore, $|M| = |f|$.

From matching to flow. Suppose M is a matching in G with size $|M|$. We can construct a flow f in G' as follows: for each edge $(u, v) \in M$, set $f(s, u) = 1$, $f(u, v) = 1$, and $f(v, t) = 1$. For all other edges $(x, y) \in E'$, set $f(x, y) = 0$. In other words, we are sending 1 unit of flow along the path $s \to u \to v \to t$ for each edge $(u, v) \in M$. By construction, the flow conservation property is satisfied at each vertex in L and R. Additionally, the value of the flow $|f|$ is equal to the number of edges in M, since each edge in M contributes 1 unit of flow from s to t. Therefore, $|f| = |M|$.

This proof strategy is similar to the one we used to prove the max-flow min-cut. One takeaway is that when you want to prove that the maximum/minimum elements of two sets have the same metric (e.g., the maximum flow value and the maximum matching size, or the minimum cut capacity and the maximum flow value), one approach is to show that each one element from one set can be transformed into one element from the other set with the same metric. In the case of max-flow min-cut, we have to take a slightly different approach, since we take a flow and construct a cut whose capacity is *at most* the flow value, and then from a cut we construct a flow whose value is *at least* the cut capacity. So it's akin to showing that for some sets A and B, $\max(A) = \min(B)$ by showing that for all $a \in A$ there exists $b \in B$ such that $a \leq b$, and for all $b \in B$ there exists

$a \in A$ such that $b \geq a$. We will see another example of this proof strategy when we study duality in linear programming (which in fact is a generalization of max-flow min-cut).

Practice

Problem 1

There is a video game with n platforms connected by m two-way bridges. A character wants to travel from platform s to platform t and back as many times as possible. However, each bridge can only be used once, once it is used it collapses and cannot be used again. Design an $O(nm)$-time algorithm to find the maximum number of one-way trips the character can make from s to t and back (so $s \to t$ counts as one trip, $s \to t \to s$ counts as two trips, etc.).

Problem 2

Suppose that given two n by n matrices A and B, we can compute their product $C = A \times B$ in $O(n^w)$ time for some w. Show that we can then compute the number of paths of length exactly k between every pair of vertices in a directed graph with n vertices in $O(n^w \log k)$ time.

Challenge

Problem 1

Suppose you are given a bipartite graph $G = (L, R, E)$. A *vertex cover* is a subset of vertices $C \subseteq L \cup R$ such that for every edge $(u, v) \in E$, at least one of u or v is in C. Show that the size of the minimum vertex cover is equal to the size of the maximum matching.