DISCLAIMER: These notes are not necessarily an accurate representation of what I said during the class. They are mostly what I intend to say, and have not been carefully edited.

Focus today is a special case of maximum flow: bipartite matching, and its associated ‘cut’ version: minimum vertex cover.

1 Bipartite Matching

A bipartite matching instance has two sets $A$ and $B$, with some allowed pairings $ab$ with $a \in A$ and $b \in B$. These pairings correspond to edges in this graph, which only go between $A$ and $B$. We want to match the maximum number of pairs without using a vertex twice.

To turn this into a maximum flow instance, we add a supersource $s$, a supersink $t$, and add the edges

1. $s \to a$ with capacity 1.
2. $b \to t$ with capacity 1.
3. $a \to b$ for each edge $ab$ with capacity $\infty$.

The goal is to have each matched edge $ab$ correspond to a path

$$s \to a \to b \to t.$$ 

The first two constraints mean that each $a$ and $b$ can be used at most once. The third constraint gives the allowed edges: its capacities can also work with 1, but this version is easier for the mincut conversion.

For matching, note that the total amount of capacity out of $s$, and hence the total amount of flow is at most $n$. This means on this type of graphs, the Ford-Fulkerson algorithm finishes in $O(n)$ iterations, for a total runtime of $O(n^2m)$ if we use the Bellman-Ford algorithm to check for $s \to t$ reachability in residual graphs, or $O(nm)$ if we use faster reachability checks such as DFS or BFS.

2 Applications of Matching in Scheduling

For an example of an application of the matching problem, consider we have $n_1$ jobs $A$ that can be processed on $n_2$ machines, $B$. Each machine can handle one job at a time, and each job takes a certain time on a machine.
There are two versions of this type of scheduling problem: minimizing total time, and minimizing the maximum completion time. The first version requires modifying the matching algorithm with some other routines. The second one on the other hand can be solved by checking if graphs have perfect matchings:

1. We binary search on the maximum completion time that we can allow.

2. To check if a completion time \( x \) is feasible, we add all edges with times \( \leq x \) to a bipartite graph on vertex sets \( A \) and \( B \), and check if the graph has a perfect matching.

Each of these checks take \( O(n_1 n_2) \) time, and since there are a total of \( n_1 n_2 \) edges, the total running time is \( O(n_1 n_2 \log(n_1 n_2)) \).

### 3 Perfect Matchings and Hall’s Theorem

A perfect matching is a matching in which every vertex in \( A \) and \( B \) are used. Clearly for this to be possible, we must have \(|A| = |B|\).

We prove Hall’s theorem:

**Theorem 3.1.** A bipartite graph has a perfect matching if and only if there does not exist a subset of \( k \) vertices \( S_A \) such that it’s incident to \(< k \) vertices in \( B \).

This set of incident vertices is the set of vertices in \( B \) that can be paired with some vertex in \( S_A \).

Clearly if such a set exists, we can’t match everyone in \( S_A \).

To show that if no matching is possible, we must have such a set, we utilize the max-flow min-cut theorem. Recall:

**Theorem 3.2 (Max-flow Min-cut).** The maximum \( s \rightarrow t \) flow in a graph \( G \) equals to the minimum capacity of a cut that separates \( t \) from \( s \).

Suppose such a flow is not possible, then we have a cut \( S \) of size \( \leq |A| - 1 \). It contains \( s \), and we let its intersection with \( A \) and \( B \) be \( S_A \) and \( S_B \) respectively.

Then the other side of the cut consists of:

- \( t \)
- \( A \setminus S_A \)
- \( B \setminus S_B \)

We first show that there cannot be any edges from \( S_A \) to \( B \setminus S_B \); such an edge would leave the cut, and have infinite capacity. The size of the maximum flow is at most \( n \), so such cuts cannot be the minimum cut.

Then the edges leaving the cut are the capacity 1 edges leaving \( s \) and entering \( t \). They consist of:
• One edge per vertex in $A \setminus S_A$.
• One edge per vertex in $S_B$.

So we have

$$|A| - |S_A| + |S_B| \leq |A| - 1,$$

or

$$|S_B| \leq |S_A| - 1,$$

which gives the set that we wanted.

Note that $S_B$ may be more than the set of vertices incident to $S_A$, but that would only give fewer than $|S_B|$ vertices incident to something in $S_A$.

### 4 Konig’s Theorem

In cases where the bipartite graph does not have a perfect matching, Hall’s theorem can generalize to Konig’s theorem.

**Theorem 4.1.** Let $|\text{max matching}|$ and $|\text{max independent set}|$ be the cardinalities of the maximum matching and maximum independent sets respectively. We have:

$$|\text{max matching}| = |A| + |B| - |\text{max independent set}|.$$

Before we go into details of how to prove it, we discuss one ‘application’ of it in the form of a puzzle. Given a subset of the chessboard, place the maximum of knights s.t. no two pieces can attack each other. (A knight can attack another piece if the horizontal/vertical difference in coordinates form the set $\{1, 2\}$.)

To solve this puzzle, observe that this is finding the independent set in the graph where all cells of the chess board are vertices, and two cells are connected by an edge if and only if a knight can ‘jump’ between them. Furthermore, this graph is actually bipartite: each time a knight moves, it goes to a grid of the opposite color. So directly invoking Konig’s theorem gives that we can find the maximum number of knights that we can place by just running maximum bipartite matching (or the equivalent maxflow) on this graph.

The proof of Konig’s theorem is similar to Hall’s theorem. Consider an $s \to t$ cut of finite value in this graph, let the $S$ side of it be:

$$\{s\} \cup S_A \cup S_B.$$

Similarly let

$$T_A = A \setminus S_A$$
$$T_B = B \setminus S_B$$
The fact that there are no infinite capacity edges in this cut means that there are no edges from $S_A$ to $T_B$. In other words, $S_A \cup T_B$ is an independent set.

On the other hand, the cut also contains all edges from $s$ to $T_A$, and all edges from $S_B$ to $t$. Since each of these edges have capacity 1, the total value of the cut is

$$|T_A| + |S_B| = |A| + |B| - |S_A \cup T_B|.$$ 

So we established that we can find an independent set of size at least the size of the maximum matching.

$$|A| + |B| - |\text{max independent set}|.$$ 

We can also take this construction in the reverse direction to show that any independent set corresponds to a $s \to t$ cut in this graph. However, there is a much more direct approach: note that for an edge $e$, its endpoints cannot both be in the independent set. For a set of $M$ edges in a matching, each of those edges certify that at most one of its endpoints can be chosen. This means that the max size of an independent set is

$$\leq |A| + |B| - |M|,$$

which gives the upper bound.