Overview
Bipartite graphs

Remember cut property of minimum spanning trees?
“Cut”: split vertices into two sets, look at edges from one to other
“Bipartite”: all edges are part of a single cut
= can color vertices red and blue so each edge has both colors

Relations between two different kinds of things

Structures where all cycles have even length
Easy linear-time algorithm for testing bipartiteness

Find any spanning tree (BFS tree, DFS tree, etc)
Color vertices at even distance from root red, odd distance blue

If all edges have both colors, graph is bipartite
If not, bad edge + tree path = odd cycle, graph is not bipartite
**Matchings**

A matching is a set of edges that don't touch each other (no two matched edges share a vertex)

Like the double bonds in this benzopyrene:

Maximum matching (a matching with as many edges as possible) can be found in polynomial time, in any graph.

We will only prove the (easier) case of bipartite graphs.
Independent sets

An independent set is a set of vertices that don’t touch each other (no two are endpoints of the same edge)

In bipartite graphs, both sets of vertices are independent, but there might be other larger independent sets

Eight queens puzzle: place 8 queens on chessboard, no two attacking each other

Independent set in graph of possible queen moves

Maximum independent sets are NP-hard to find in arbitrary graphs

But they can be found using matching in bipartite graphs!
Kőnig’s theorem [Dénes Kőnig, 1931]

Let $M = \#$ edges in maximum matching, $I = \#$ vertices in max independent set. Then in bipartite graphs, $M + I = n$.

Easy direction: Independent set can only include one vertex from each matched edge, so $I \leq n - M$ (true for all graphs).

Hard direction: We can find an independent set with all unmatched vertices + one vertex from each matched edge, so $I \geq n - M$.

Proof: Later in lecture, based on algorithm for finding matchings.
Rectangle partitions
Geometric application of bipartite independent sets

Input: polygon with axis-parallel sides

Goal: slice into as few rectangles as possible

If we only slice horizontally or only vertically ⇒ too many pieces

We need a more careful algorithm
Why slice polygons into rectangles?

Decomposition of lithography masks into simple shapes for VLSI DNA microarray design

Convolution operations in image processing

Compression of bitmap images

Radiation therapy planning

Robot self-assembly planning

(as listed in Eppstein [2009])
What’s really going on here?

Polygon corners are convex \((90^\circ \text{ interior})\) or concave \((270^\circ)\)

We need to slice horizontally or vertically at concave corners to make rectangular pieces

We can use fewer slices whenever we can slice two corners with a single cut

But we lose the improvement when two slices touch or cross each other or when a slice reaches only one concave corner

New goal: Find many non-touching two-concave-corner slices
Bipartite graph of 2-vertex slices that touch or cross
Independent sets and the rectangles they produce

Red independent set: 3 vertices
3 two-concave-corner slices
3 slices at other concave corners
7 rectangles

Blue independent set: 3 vertices
3 two-concave-vertex slices
3 slices at other concave corners
7 rectangles

Max independent set: 4 vertices
4 two-concave-corner slices
1 slice at other concave corner
6 rectangles
Finding bipartite maximum matchings
**Alternating paths**

Suppose you have a matching that is **not** maximum
And you find a path starting and ending at unmatched vertices, alternating between matched and unmatched edges

Then you can get a **bigger** matching by changing unmatched edges to matched and vice versa on the path
Non-maximum matchings have alternating paths

Overlay any non-maximum matching with any maximum matching

The parts of the graph that are not matched the same way in both matchings form alternating paths and cycles

Cycles don’t change \# matched vertices
so there must be at least one path
Algorithm for maximum matching (main idea)

Start with an empty matching (one that has no edges in it)

Repeat:
Find an alternating path
Change unmatched edges on the path
to matched edges and vice versa

When no more paths can be found, return your current matching
How to find a (short) alternating path

Variation of breadth-first search

Initialize queue with all unmatched red vertices

For each vertex in queue:
- If red, loop over unmatched edges and add unreached neighbors to queue
- If blue and matched, add matched neighbor to queue
- If blue and unmatched, found end of an alternating path!

Time: $O(m)$ per path, $O(mn)$ to find maximum matching

Define the parent of a vertex in the same way as BFS: the vertex who added it to the queue

Result is an alternating path forest
Proof of König’s theorem

Recall what we still need to prove: In a bipartite graph, we can always find an independent set containing all unmatched vertices and one vertex from each matched edge

- Run an alternating path search from the maximum matching
- Resulting forest includes all unmatched red vertices, some matched edges, and none of the unmatched blue vertices (if it included an unmatched blue vertex, we would have found an alternating path and a bigger matching)
- Independent set: red vertices in the forest and blue vertices that are not in the forest
- It’s independent because any edge would allow us to grow the forest bigger
Hopcroft–Karp–Karzanov algorithm (sketch)

Speed up matching by finding many shortest alternating paths every time we do an alternating path search

Repeat:

- Do an alternating path search, but keep going after finding the first alternating path, to find all unmatched blue vertices at the same level in the forest
- List for each vertex its possible predecessors in the forest (all neighbors in the previous level, not just its parent)
- Backtrack through these lists to find as many disjoint shortest alternating paths as you can before getting stuck

[Hopcroft and Karp 1973; Karzanov 1973]
Hopcroft–Karp–Karzanov analysis

Each alternating path search + backtracking takes time $O(m)$.

The path lengths increase by two in each search
So after $\sqrt{n}$ searches they are $\geq 2\sqrt{n}$.

Number of remaining paths still need to find is $\leq n/(\text{path length})$
So the algorithm stops after an additional $O(\sqrt{n})$ searches

Total time: $O(m\sqrt{n})$
Non-bipartite matching

Algorithms

Usually, bipartite and non-bipartite matching problems have same complexity, but much greater complication for non-bipartite

Non-bipartite version of Hopcroft–Karp–Karzanov is by Silvio Micali and Vijay Vazirani (now a UCI professor) from 1980

An application

Kidney transplant donors and recipients enter the system together (usually the donor is a friend or relative of the patient) but often don’t have compatible immune systems

Form a graph whose vertices are donor-recipients pairs and whose edges represent double compatibility (donor of one pair is compatible with recipients of the other and vice versa)

Set up a double transplant operation with four patients (two donors and two recipients) for each matched edge
Morals of the story

Bipartiteness is easy to test, natural in many matching problems, and makes those problems simpler.

In bipartite graphs, matching and independent sets are equivalent, but in other graphs independent sets are hard.

Application to optimal rectangle partition

Alternating paths for finding bipartite matchings, and Hopcroft–Karp–Karzanov for finding them quickly.
References


