Seminar Ex6 and Graphs III

Matching, Covering and More Graph Problems

Fredrik Präntare (fredrik.prantare@liu.se)

Reasoning and Learning Lab
Artificial Intelligence and Integrated Computer Systems
Department of Computer and Information Science

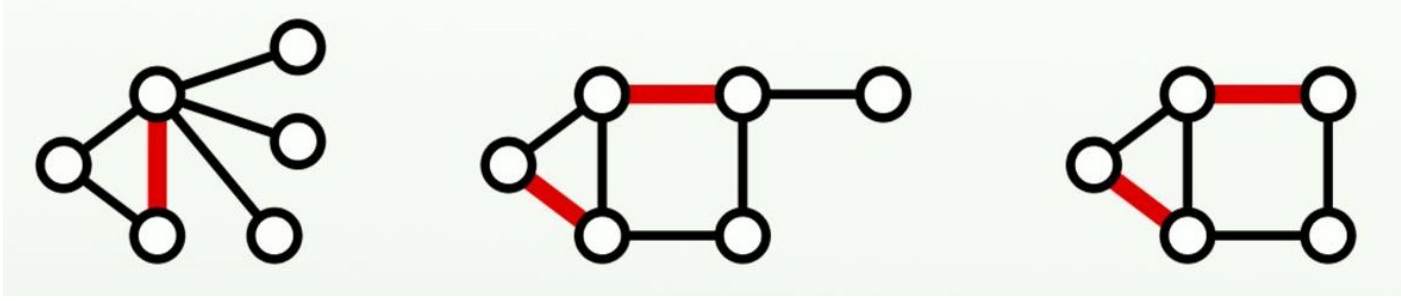
March 4, 2022

Outline

- ~~Ex6: Graphs II~~
  - ~~Full Tank?~~
  - ~~Island Hopping~~
  - ~~George~~
  - ~~Councilling~~
- Matching, Covering and Graph Problems

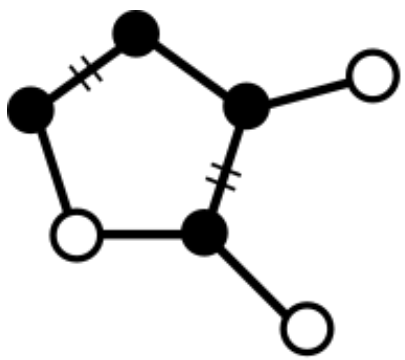
Preliminaries: Graph Matching

A *matching* in a graph is a subset of its edges without common vertices.

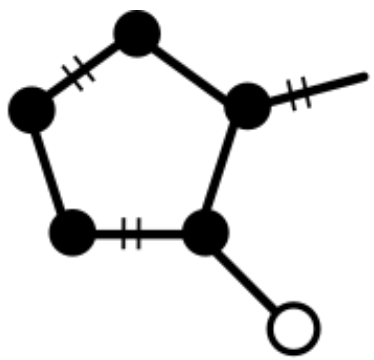


Preliminaries: Matching Types

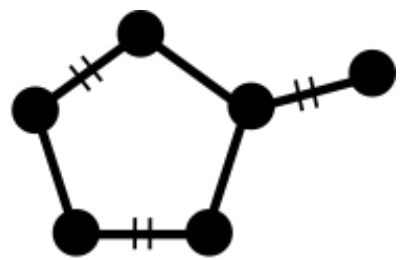
- *Maximal*: Cannot add more edges.
- *Maximum*: There are no other matchings with more edges.
- *Perfect*: All vertices are included.



maximal



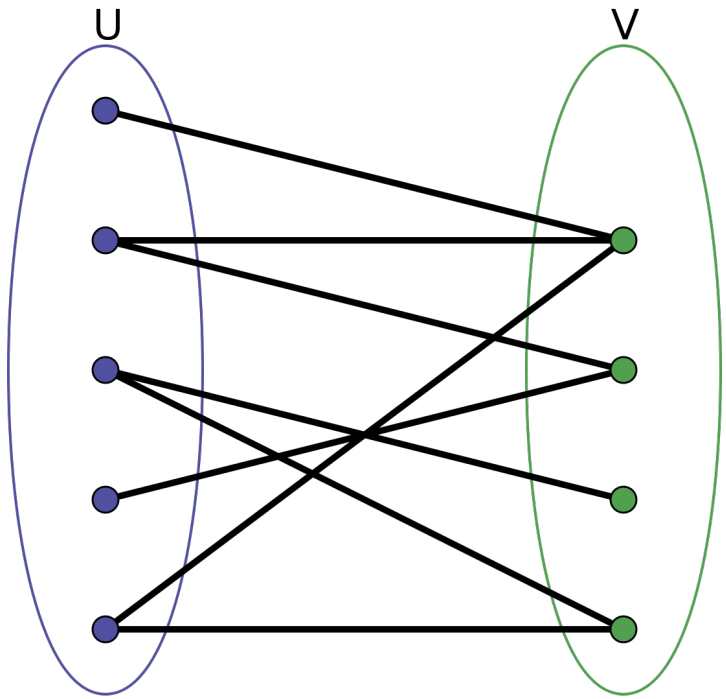
maximum



perfect

Preliminaries: Bipartite Graph

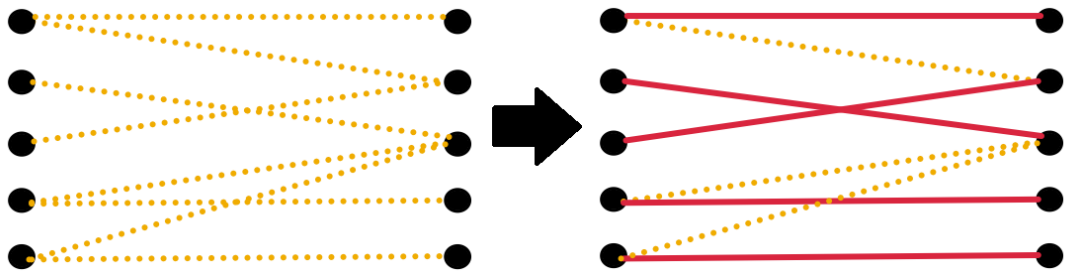
A *bipartite graph* is a graph whose vertices can be divided into two disjoint sets U and V so every edge connects a vertex in U to one in V.



Maximum Cardinality Matching

Maximum Cardinality (Bipartite) Matching (MCM/MCBM)

Input: A (bipartite) graph.
Output: A maximum matching.



Weighted MCM/MCBM

Input: A weighted (bipartite) graph.
Output: A maximum matching with highest total (sum) weight.

W.l.o.g., for bipartite we can assume $|U| = |V|$: add dummy vertices.
For cost minimization, just multiply the weights by minus one.

Some Applications

- Task assignment: Workers $\mapsto$ jobs.
- Resource allocation: Indivisible resources $\mapsto$ institutions.
- Revenue-maximizing auctions: Goods $\mapsto$ bidders (the winners).
- Target tracking: Sensors/cameras $\mapsto$ targets.

Assigning workers to jobs

Multi-sensor
multi-target tracking

How To Solve Cardinality Matching?

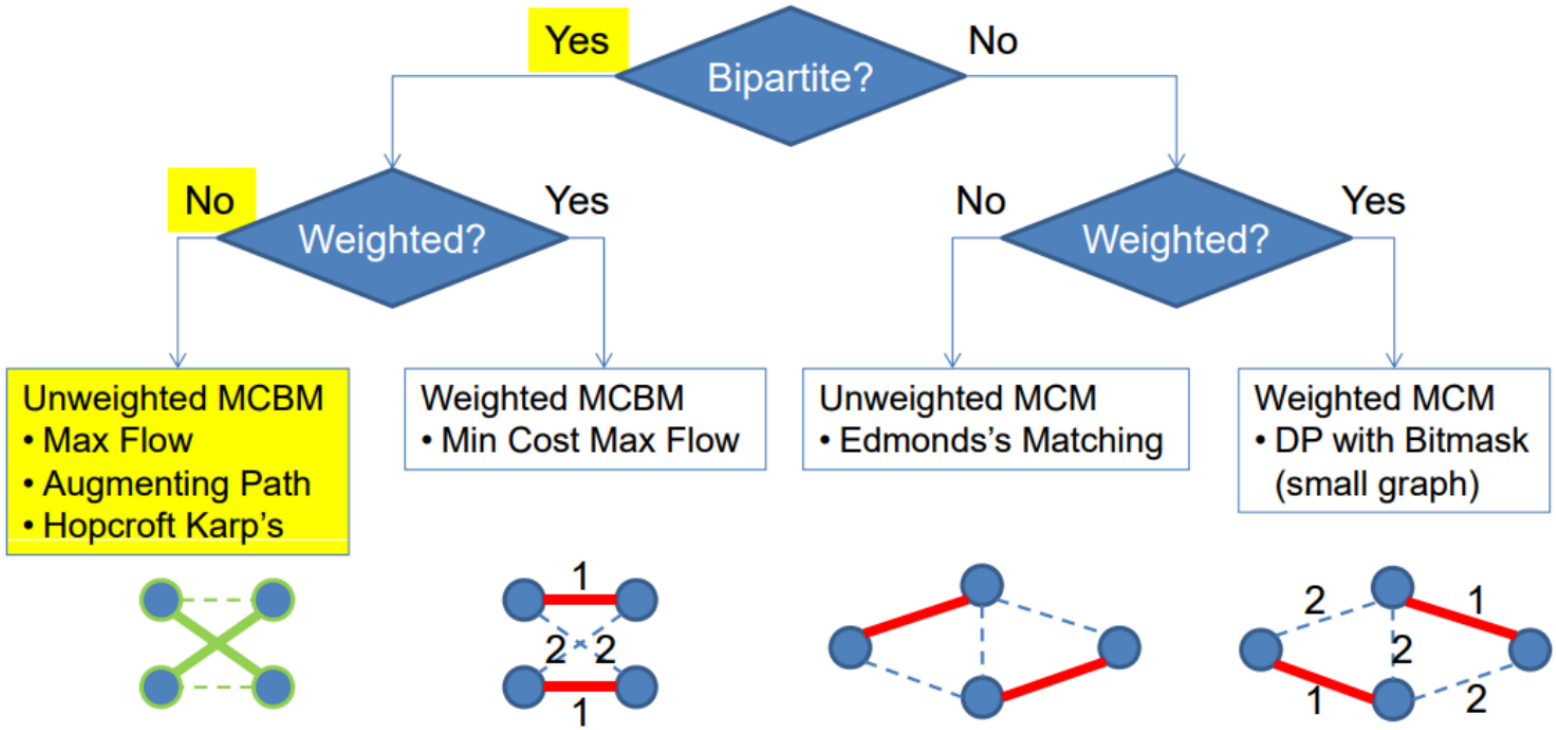
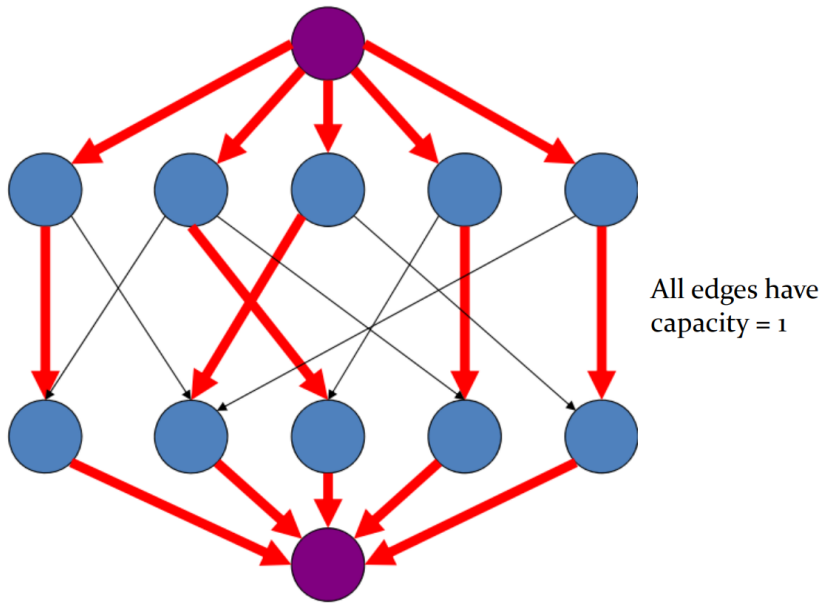


image source: Steven Halim

A Straightforward Max Flow Solution for MCBM



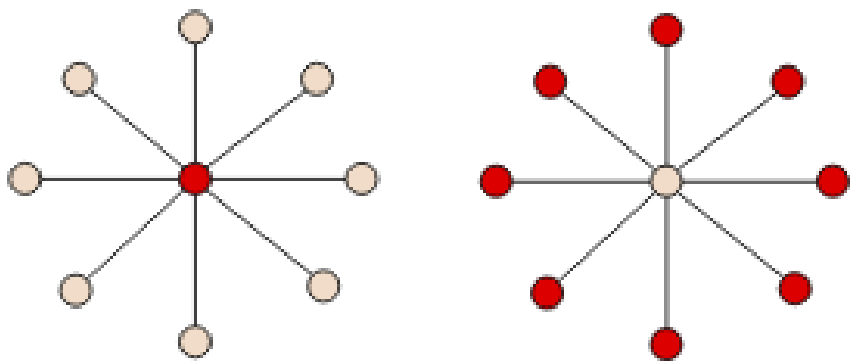
With Ford–Fulkerson + DFS, runs in $\mathcal{O}(nf_{max}) = \mathcal{O}(n^2)$, where $n = \max(|U|, |V|)$.

Weighted case: Same idea, but solve with min-cost max-flow Edmond–Karp/Dinic instead.

Independent Set

- *Indpendent set* (IS): A set of vertices for which none are adjacent.
- *Maximal IS*: IS that we cannot add vertices to.
- *Maximum IS*: An IS with a maximum number of vertices.
- *Maximum-weight IS*: IS with maximum total (sum) weight. (Vertices have weights.)

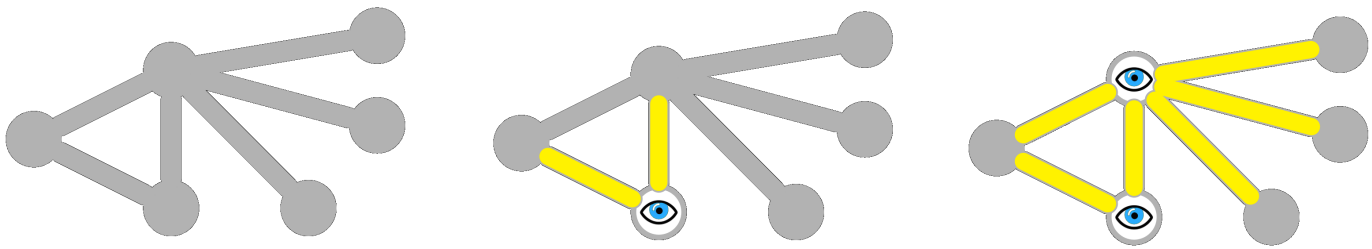
(Optimization version of finding maximum IS is NP-hard—brute force runs in $\mathcal{O}(n^2 2^n)$.)



Vertex Cover

- *Vertex cover*: A set of vertices that includes at least one endpoint of every edge.
- *Minimum vertex cover*: A vertex cover of smallest possible size.

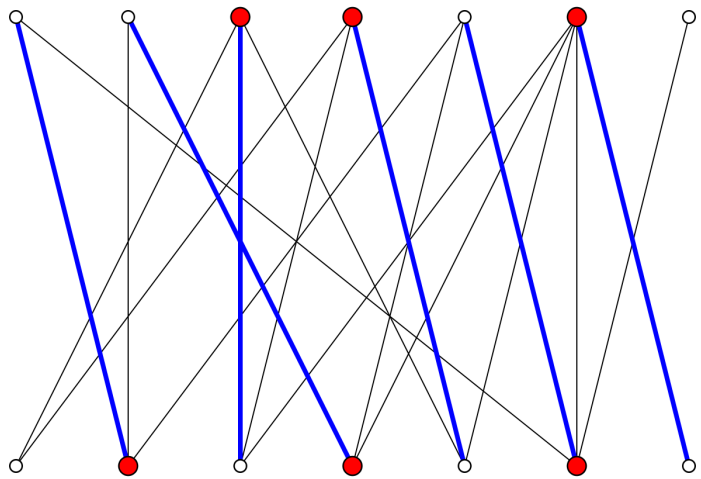
(Optimization version is NP-hard, but is fixed-parameter tractable w.r.t. to the size of the cover, can e.g., be solved in $2^k n^{\mathcal{O}(1)}$.)



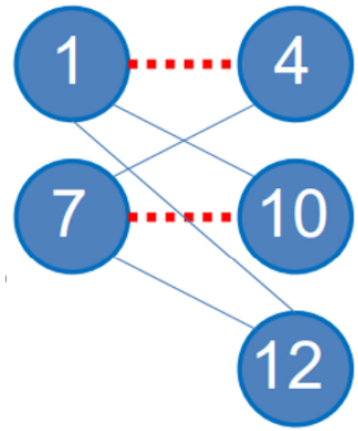
König's Theorem

Theorem 1 (König's Theorem) *In any bipartite graph, the number of edges in a maximum matching is equal to the number of vertices in a minimum vertex cover.*

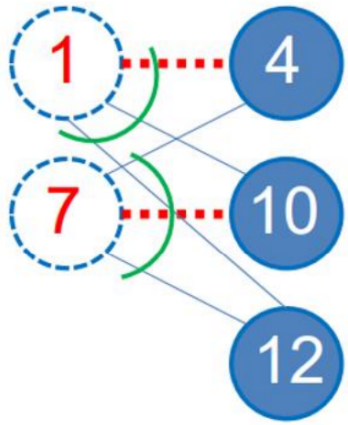
Theorem 2 *In a bipartite graph, the* ***complement*** *of a maximum independent set is a minimum vertex cover.*



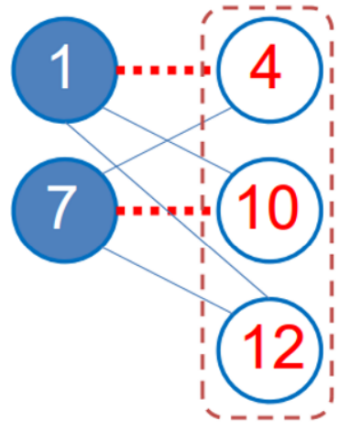
König's Theorem Applied



Maximum Cardinality Bipartite Matching



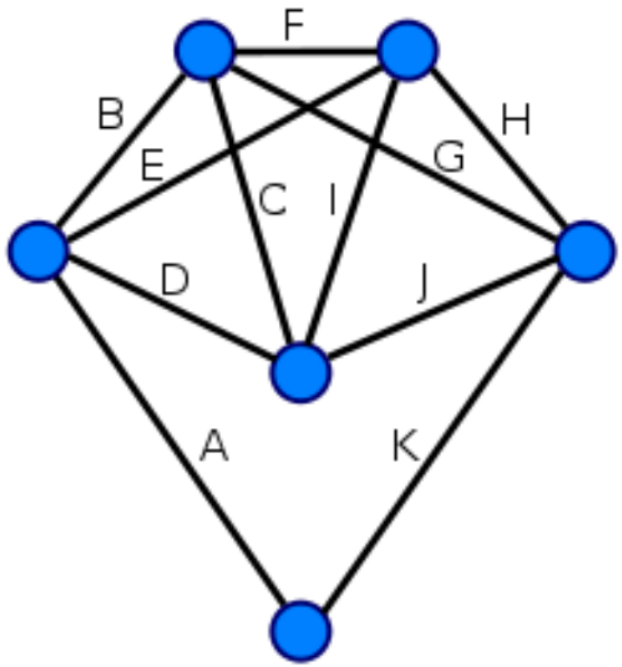
Minimum Vertex Cover (König's Theorem)

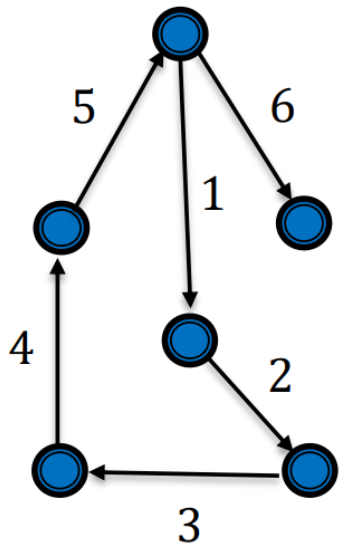


Maximum Independent Set

Eulerian Path/Cycle

A *Eulerian path* is a path in a graph that visits every edge exactly once.





A *Eulerian cycle* also returns to the starting vertex.

Finding Eulerian Cycles

Hierholzer's algorithm:

- **Check if a Eulerian cycle exists:** Each vertex needs to have equal in degree and out degree ($\implies$ even degree), and all vertices with non-zero degree are part of the same connected component.
- Start from any vertex v. Follow a path of edges from it until returning to v. Cannot get stuck due to "in degree = out degree". Add the path to the tour.
- As long as there exists a vertex v that belongs to the current tour but that has adjacent edges not part of the tour, start another path from v, following unused edges until returning to v, and join the tour formed in this way to the previous tour.

Naive version runs in $\mathcal{O}(|E| + |V|)$. Careful implementation in $\mathcal{O}(|E|)$.

www.liu.se