Algorithm II

10. Extending Tractability

WU Xiaokun 吴晓堃

xkun.wu [at] gmail

Coping with $\mathcal{NP}$-completeness

Q. Suppose I need to solve an $\mathcal{NP}$-complete problem. What should I do?
A. Theory says you're unlikely to find poly-time algorithm.

Must sacrifice one of three desired features.

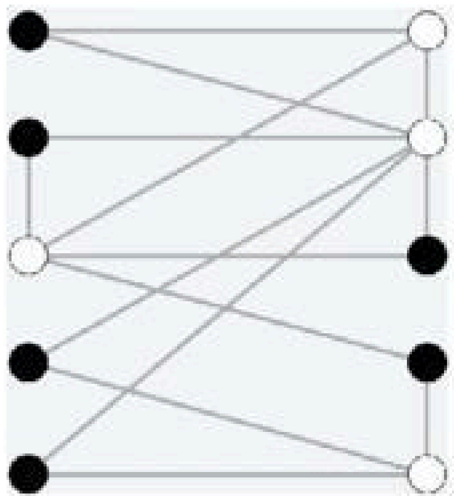
- Solve problem to optimality.
- Solve problem in polynomial time.
- Solve *arbitrary instances* of the problem.

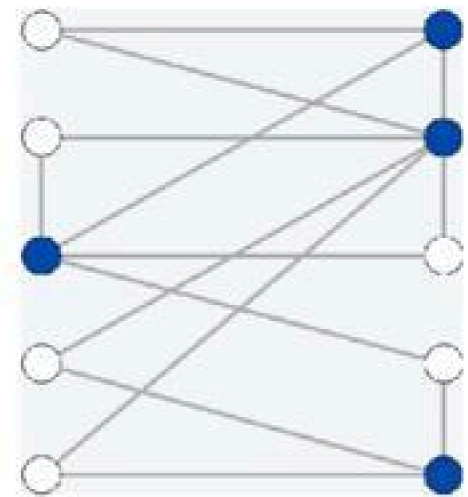
This lecture. Solve some special cases of $\mathcal{NP}$-complete problems.

Finding small vertex covers

Vertex cover

Given a graph $G = (V, E)$ and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \leq k$, and for each edge (u, v) either $u \in S$ or $v \in S$ or both?





Finding small vertex covers

Q. VERTEX-COVER is $\mathcal{NP}$-complete. But what if k is small?

Brute force. $O(kn^{k+1})$.

- Try all $C(n, k) = O(n^k)$ subsets of size k.
- Takes $O(kn)$ time to check whether a subset is a vertex cover.

Goal. Limit exponential dependency on k, say to $O(2^k kn)$.

Finding small vertex covers

Q. VERTEX-COVER is $\mathcal{NP}$-complete. But what if k is small?

Brute force. $O(kn^{k+1})$.

- Try all $C(n,k) = O(n^k)$ subsets of size k.
- Takes $O(kn)$ time to check whether a subset is a vertex cover.

Goal. Limit exponential dependency on k, say to $O(2^k kn)$.

Ex. $n = 1000, k = 10$.
Brute. $kn^{k+1} = 10^{34} \Rightarrow$ infeasible.
Better. $2^k kn = 10^7 \Rightarrow$ feasible.

Remark. If k is a constant, then the algorithm is poly-time; if k is a small constant, then it's also practical.

Finding small vertex covers

Claim. Let (u, v) be an edge of G. G has a vertex cover of size $\leq k$ iff at least one of $G - \{u\}$ and $G - \{v\}$ has a vertex cover of size $\leq k - 1$.

Pf. $\Rightarrow$ Suppose G has a vertex cover S of size $\leq k$.

- S contains either u or v (or both). Assume it contains u.
- $S - \{u\}$ is a vertex cover of $G - \{u\}$.

Pf. $\Leftarrow$ Suppose S is a vertex cover of $G - \{u\}$ of size $\leq k - 1$.

- Then $S \cup \{u\}$ is a vertex cover of G.

Finding small vertex covers

Claim. Let (u, v) be an edge of G. G has a vertex cover of size $\leq k$ iff at least one of $G - \{u\}$ and $G - \{v\}$ has a vertex cover of size $\leq k - 1$.

Pf. $\Rightarrow$ Suppose G has a vertex cover S of size $\leq k$.

- S contains either u or v (or both). Assume it contains u.
- $S - \{u\}$ is a vertex cover of $G - \{u\}$.

Pf. $\Leftarrow$ Suppose S is a vertex cover of $G - \{u\}$ of size $\leq k - 1$.

- Then $S \cup \{u\}$ is a vertex cover of G.

Claim. If G has a vertex cover of size k, it has $\leq k(n - 1)$ edges.
Pf. Each vertex covers at most $n - 1$ edges.

Finding small vertex covers: algorithm

Claim. The following algorithm determines if G has a vertex cover of size $\leq k$ in $O(2^k kn)$ time.

`Vertex-Cover`(G, k)

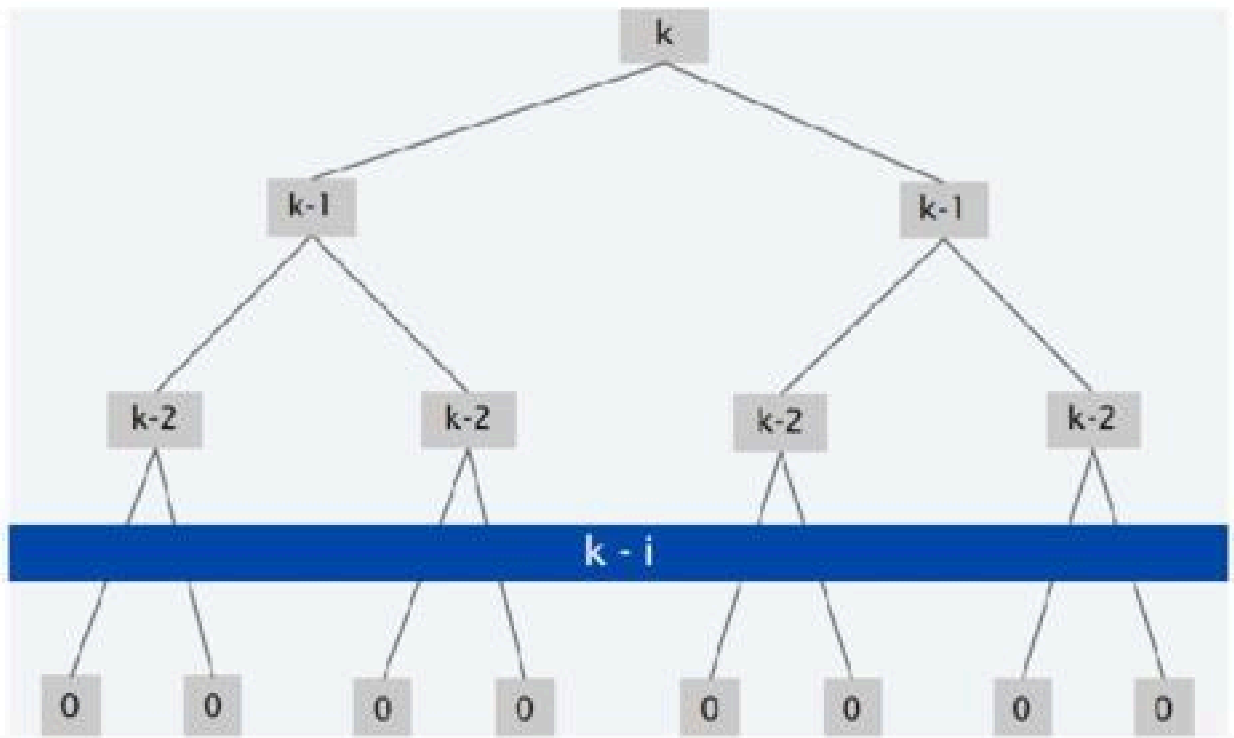
1. if (G contains no edges) return true;
2. if (G contains $\geq kn$ edges) return false;
3. let (u, v) be any edge of G;
    1. a = `Vertex-Cover`$(G - \{u\}, k - 1)$;
    2. b = `Vertex-Cover`$(G - \{v\}, k - 1)$;
4. return (a or b);

Pf.

- Correctness follows from previous two claims.
- There are $\leq 2^{k+1}$ nodes in the recursion tree; each invocation takes $O(kn)$ time.

Finding small vertex covers: recursion tree

$$T(n,k) = \begin{cases} c & \text{if } k = 0 \\ cn & \text{if } k = 1 \Rightarrow T(n,k) \le 2^k ckn \\ 2T(n,k-1) + ckn & \text{if } k > 1 \end{cases}$$



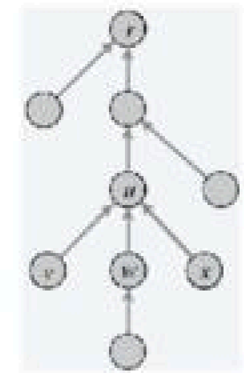
Solving NP-hard problems on trees

Independent set on trees

Independent set on trees. Given a *tree*, find a max-cardinality subset of nodes such that no two are adjacent.

Fact. A tree has at least one node that is a leaf (degree = 1).

Key observation. If node v is a leaf, there exists a max-cardinality independent set containing v.



Pf. [exchange argument]

- Consider a max-cardinality independent set S.
- If $v \in S$, we're done.
- Otherwise, let (u, v) denote the lone edge incident to v.
  - if $u \notin S$ and $v \notin S$, then $S \cup \{v\}$ is independent $\Rightarrow$ S not maximum
  - if $u \in S$ and $v \notin S$, then $S \cup \{v\} - \{u\}$ is independent

IS on trees: greedy

Theorem. The greedy algorithm finds a max-cardinality independent set in forests (and hence trees).

INDEPENDENT-SET-IN-A-FOREST(F)

1. $S = \emptyset$;
2. WHILE (F has at least 1 edge)
   1. Let v be a leaf node and let (u, v) be the lone edge incident to v;
   2. $S = S \cup \{v\}$;
   3. $F = F - \{u, v\}$;
3. RETURN $S \cup \text{nodes remaining in } F$;

Remark. Can implement in $O(n)$ time by maintaining nodes of degree 1 in postorder.

Demo: greedy for IS on trees

Weighted independent set on trees

Weighted independent set on trees. Given a tree and node weights $w_v \geq 0$, find an independent set S that maximizes $\sum_{v \in S} w_v$.

Observation. If (u, v) is an edge such that v is a leaf node, then either OPT includes u or OPT includes all leaf nodes incident to u.

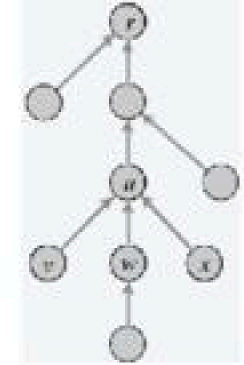
Dynamic-programming solution. Root tree at some node, say r.

- $OPT_{in}(u)$ = max-weight IS in subtree rooted at u, *including* u.
- $OPT_{out}(u)$ = max-weight IS in subtree rooted at u, *excluding* u.
- Goal: $\max\{OPT_{in}(r), OPT_{out}(r)\}$.

Bellman equation.

$$OPT_{in}(u) = w_u + \sum_{v \in children(u)} OPT_{out}(v)$$

$$OPT_{out}(u) = \sum_{v \in children(u)} \max\{OPT_{in}(v), OPT_{out}(v)\}$$



Weighted IS on trees: DP

Theorem. The DP algorithm computes max weight of an independent set in a tree in $O(n)$ time.

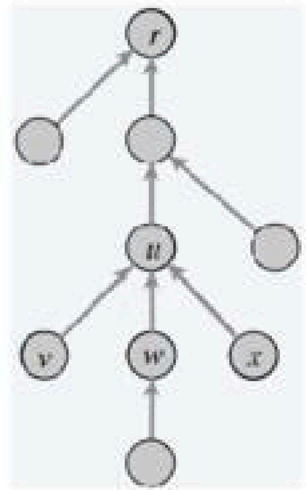
- note: can also find independent set itself (not just value)

`WEIGHTED-INDEPENDENT-SET-IN-A-TREE` (T)

1. Root the tree T at any node r;
2. $S = \emptyset$;
3. FOREACH (node u of T in postorder/topological order)
   1. IF (u is a leaf node)
      1. $M_{in}[u] = w_u$; $M_{out}[u] = 0$;
   2. ELSE
      1. $M_{in}[u] = w_u + \sum_{v \in children(u)} M_{out}[v]$;
      2. $M_{out}[u] = \sum_{v \in children(u)} \max\{M_{in}[v], M_{out}[v]\}$;
4. RETURN $\max\{M_{in}[r], M_{out}[r]\}$;

NP-hard problems on trees: intuition

Independent set on trees. Tractable because we can find a node that *breaks the communication* among the subproblems in different subtrees.



Graphs of bounded tree width. Elegant generalization of trees that:

- Captures a rich class of graphs that arise in practice.
- Enables decomposition into independent pieces.

Circular arc coverings

Wavelength-division multiplexing

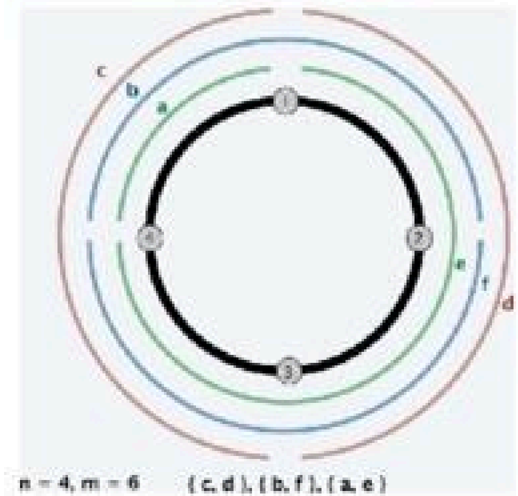
Wavelength-division multiplexing (WDM). Allows m communication streams (arcs) to share a portion of a fiber optic cable, provided they are transmitted using different wavelengths.

Ring topology. Special case is when network is a **cycle** on n nodes.

Bad news. $\mathcal{NP}$-complete, even on rings.

Brute force. Can determine if k colors suffice in $O(k^m)$ time by trying all k-colorings.

Goal. $O(f(k)) \cdot poly(m, n)$ on rings.



Review: interval coloring

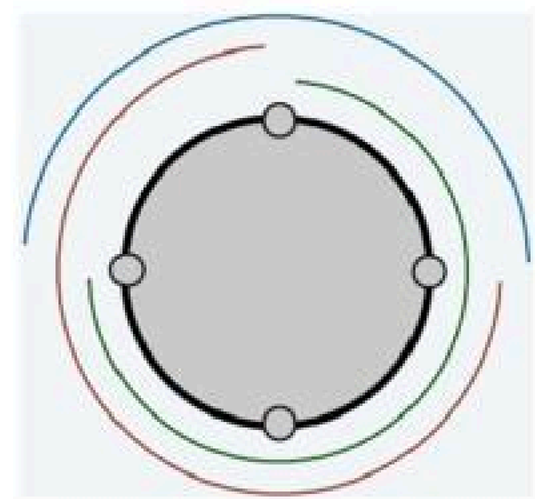
Interval coloring (partitioning). Greedy algorithm finds coloring such that number of colors *equals* depth of schedule.

- **Depth**. Maximum number that pass over any single point on the time-line.

c	c		d	d		f	f		j	j
b	b	b	b	b		g	g		i	i
a	a		e	e	e	e	h	h	h	h

Circular arc coloring.

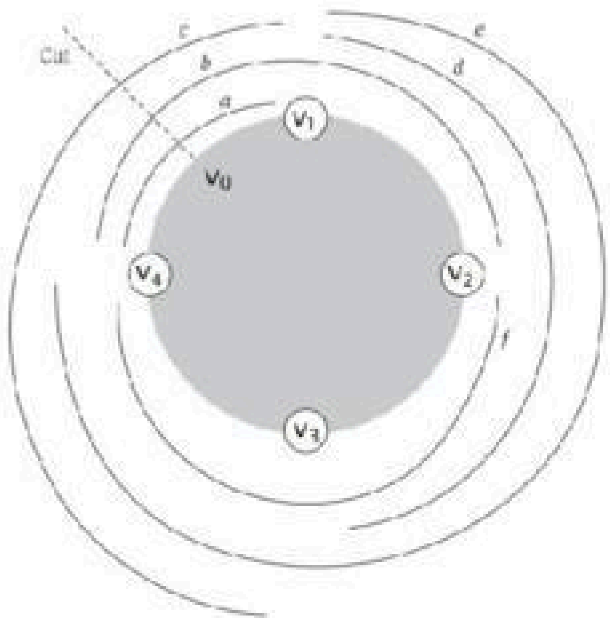
- Weak duality: number of colors $\geq$ depth.
- Strong duality does not hold.

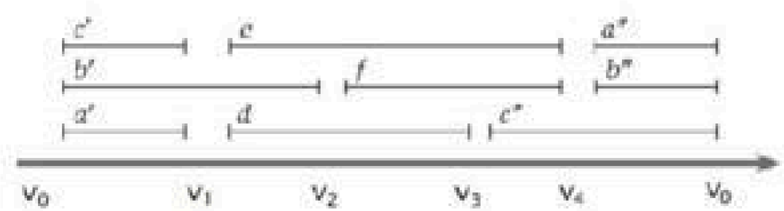


(Almost) transforming coloring

Circular arc coloring. Given a set of n arcs with depth $d \leq k$, can the arcs be colored with k colors?

Equivalent problem. Cut the network between nodes v_1 and v_n. The arcs can be colored with k colors iff the intervals can be colored with k colors in such a way that "sliced" arcs have the same color.

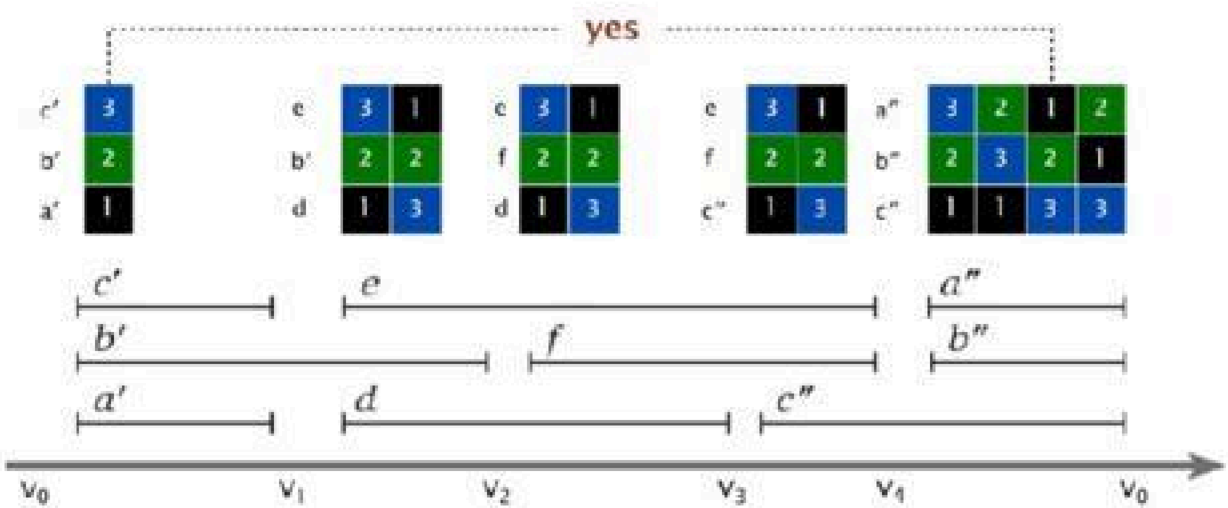




Circular arc coloring: DP

Dynamic programming algorithm.

- Assign distinct color to each interval which begins at cut node v_0.
- At each node v_i, some intervals may finish, and others may begin.
  - Enumerate all k-colorings of the intervals through v_i that are consistent with the colorings of the intervals through v_{i-1}.
- The arcs are k-colorable iff some coloring of intervals ending at cut node v_0 is consistent with original coloring of the same intervals.



Circular arc coloring: running time

Running time. $O(k! \cdot n)$.

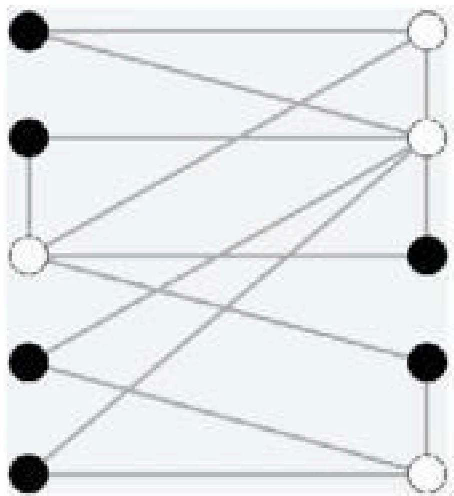
- The algorithm has n phases.
- Bottleneck in each phase is enumerating all consistent colorings.
- There are at most k intervals through v_i, so there are at most $k!$ colorings to consider.

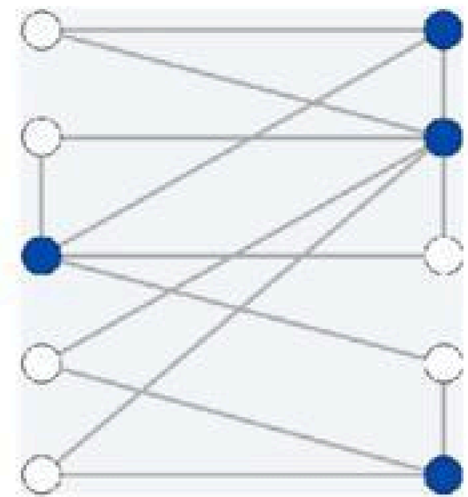
Remark. This algorithm is practical for small values of k (say $k = 10$) even if the number of nodes n (or paths) is large.

Vertex cover in bipartite graphs

Vertex cover

Given a graph $G = (V, E)$ and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \leq k$, and for each edge (u, v) either $u \in S$ or $v \in S$ or both?

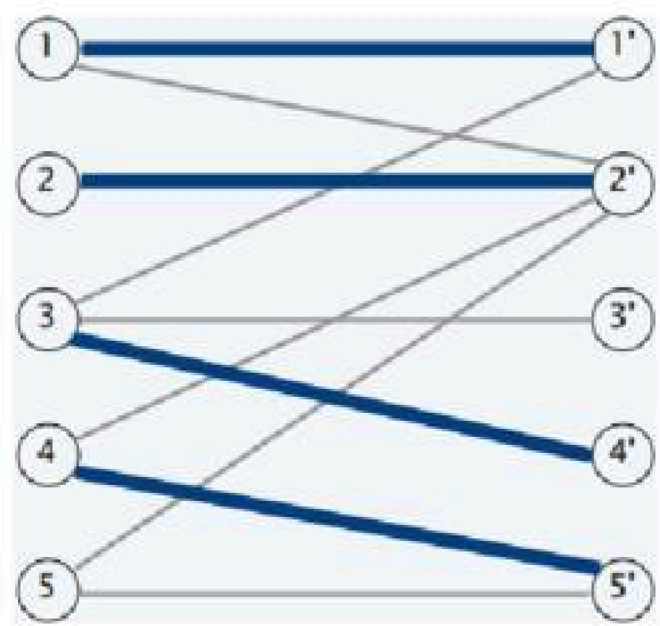




Vertex cover and matching

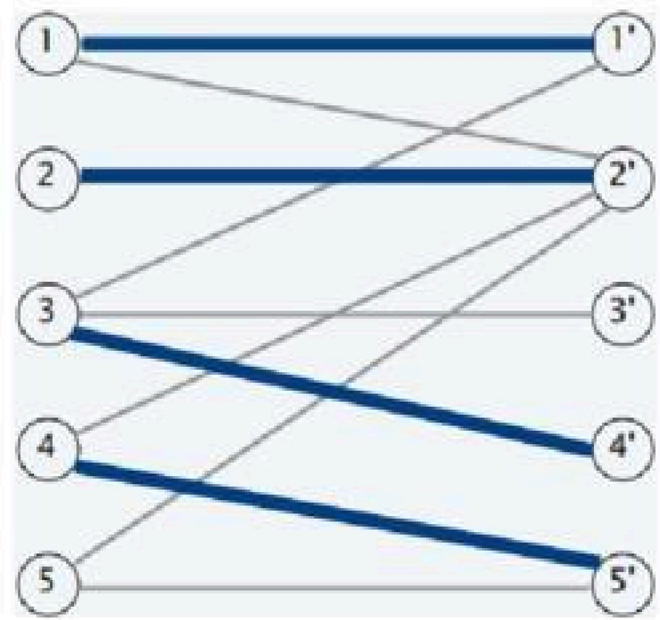
Weak duality. Let M be a matching, and let S be a vertex cover. Then, $|M| \leq |S|$.

Pf. Each vertex can cover at most one edge in any matching.



König-Egerváry Theorem

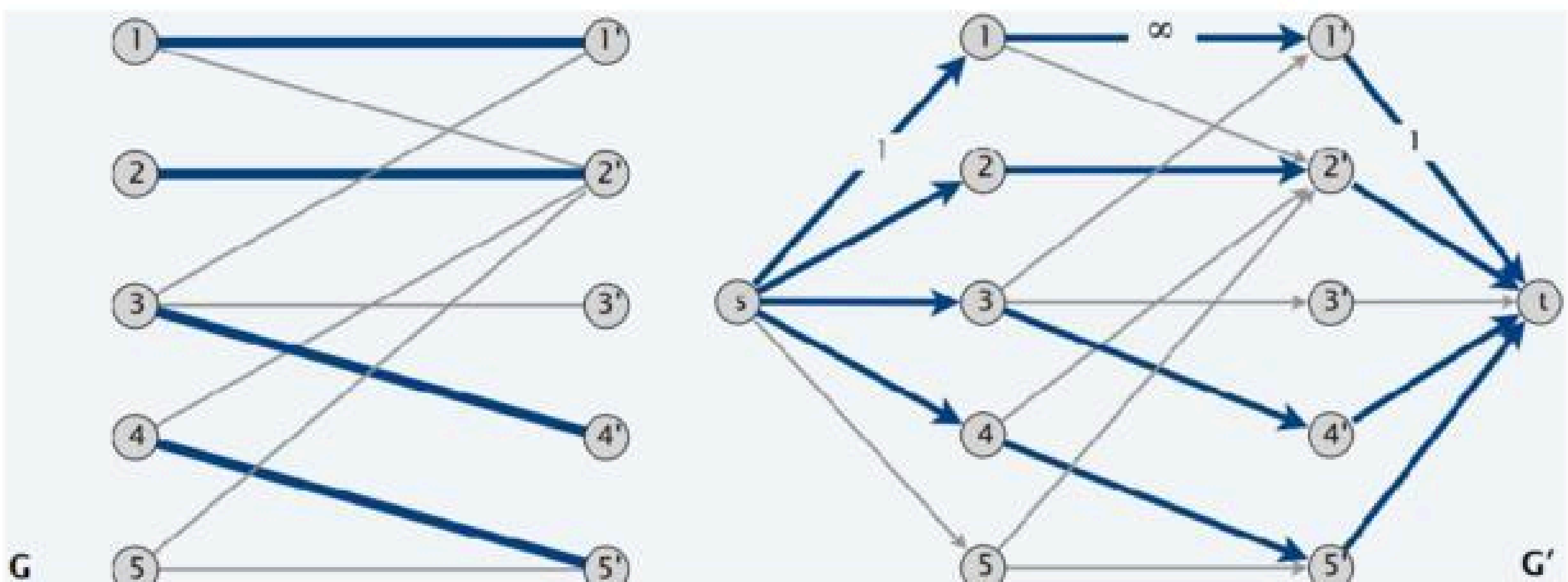
Theorem. [König-Egerváry] In a *bipartite* graph, the max cardinality of a matching is equal to the min cardinality of a vertex cover.



König-Egerváry Theorem: proof

Theorem. [König-Egerváry] In a *bipartite* graph, the max cardinality of a matching is equal to the min cardinality of a vertex cover.

- Suffices to find matching M and cover S such that $|M| = |S|$.
- Formulate max flow problem as for bipartite matching.
- Let M be max cardinality matching and let (A, B) be min cut.



König-Egerváry Theorem: proof (cont.)

Theorem. [König-Egerváry] In a *bipartite* graph, the max cardinality of a matching is equal to the min cardinality of a vertex cover.

- Suffices to find matching M and cover S such that $|M| = |S|$.
- Formulate max flow problem as for bipartite matching.
- Let M be max cardinality matching and let (A, B) be min cut.
- Define $L_A = L \cap A, L_B = L \cap B, R_A = R \cap A, R_B = R \cap B$.
- **Claim 1**. $S = L_B \cup R_A$ is a vertex cover.
  - consider $(u, v) \in E$
  - $u \in L_A, v \in R_B$ impossible since infinite capacity
  - thus, either $u \in L_B$ or $v \in R_A$ or both
- **Claim 2**. $|M| = |S|$.
  - max-flow min-cut theorem $\Rightarrow |M| = cap(A, B)$
  - only edges of form (s, u) or (v, t) contribute to $cap(A, B)$
  - $|M| = cap(A, B) = |L_B| + |R_A| = |S|$.