Portland State University

W'21 CS 584/684

Algorithm Design & Analysis

Fang Song

Lecture 15

- Bipartite matching
- Linear programming

Exercise

- For each of the following statements, decide TRUE or FALSE.
  - Let f be a flow and $G_f = (V, E_f)$ be the residual graph. Then $|E_f| \leq 2|E|$.
  - Any flow f and cut (A, B) satisfy that $\mathrm{cap}(\mathrm{A}, \mathrm{B}) \leq val(f)$.
  - f is a max flow iff. there is no augmenting path with respect to f.

Ford-Fulkerson algorithm: summary so far

Ford–Fulkerson
While you can
- Greedily push flow
- Update residual graph

- Correctness. Augmenting path theorem.
- Running time. Does it terminate at all?

Ford-Fulkerson algorithm: analysis

- **Assumption.** All capacities are integers between 1 and C.
- **Invariant.** Every flow value $f(e)$ and every residual capacity $c_f(e)$ remains an integer throughout the algorithm.

- **Theorem.** Ford-Fulkerson terminates in at most nC iterations.

  Pf.
  - Each augmentation increases flow value by at least 1.
  - There are at most nC units of capacity leaving source s.

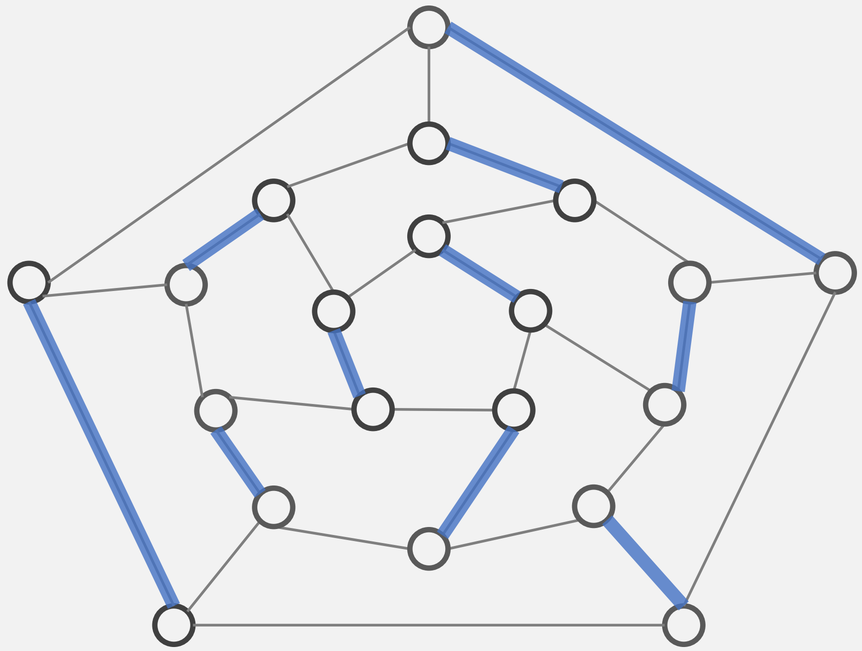
  Running time $O(mnC)$. Space $O(m+n)$

  Find an augmenting path in $O(m)$ time (by BFS/DFS).

- **Integrality theorem.** If all capacities are integers, then there exists a max flow f for which every flow value $f(e)$ is an integer.
- **Today.** Applications when $C = 1$ [N.B. running time $O(mn)$]

Matching

Def. Given an undirected graph $G = (V, E)$. A subset of edges $M \subseteq E$ is a matching if each node appears in at most one edge in M.



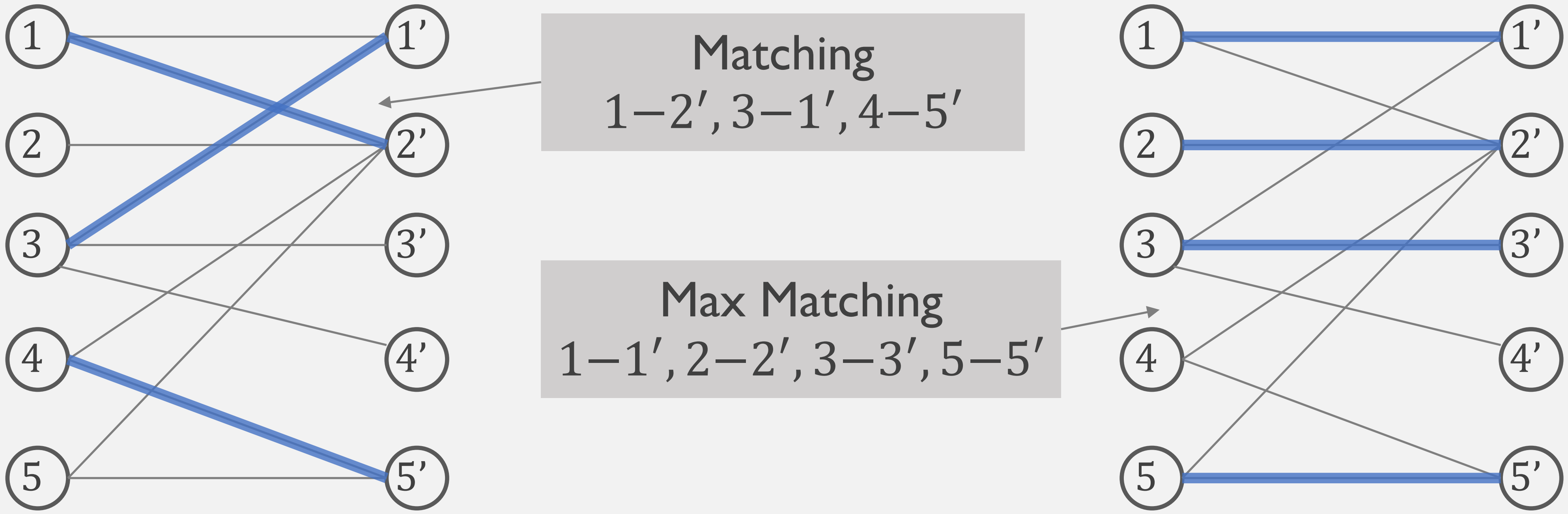
Max matching. Find a matching of max cardinality.

- i.e., adding any edge will make it no longer a matching

Bipartite Matching

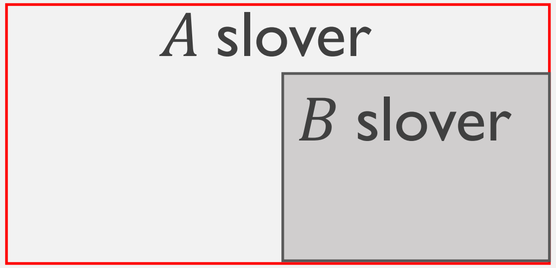
Bipartite graph. A graph G is bipartite if the nodes V can be partitioned into two subsets L and R such that every edge connects a node in L with a node in R.

(Max) Bipartite matching. Given a bipartite graph $G = (L \cup R, E)$, find a max cardinality matching.

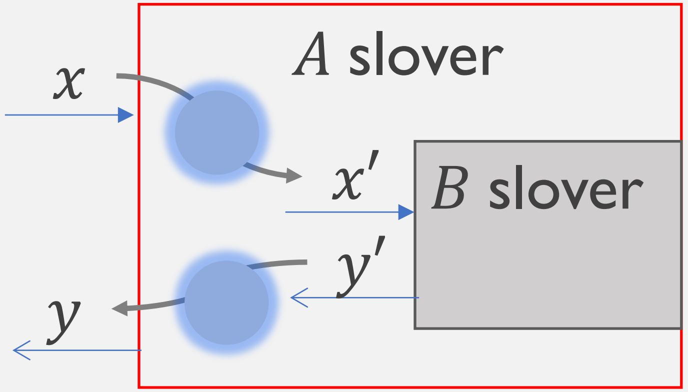


Reductions

- Informal. Problem A reduces to problem B if there is a simple algorithm for A that uses an algorithm for B as a subroutine.



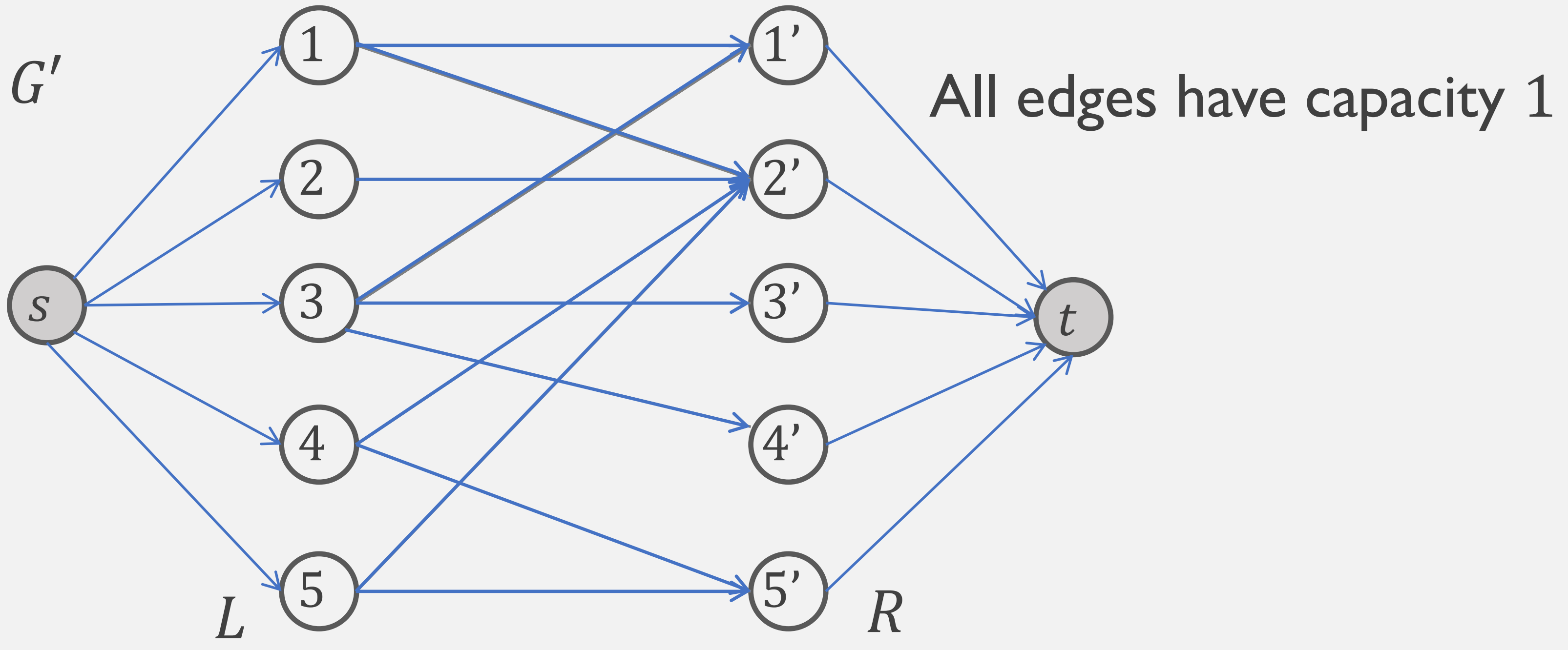
- Common scenario [a.k.a. Karp reduction]
  - Given instance x of problem A.
  - Convert x to an instance x' and solve it.
  - Use the solution to x' to build a solution for x.



- Useful skill
  - Quickly identifying problems where existing solutions may be applied
  - Good programmers do this all the time [don't reinvent wheels]

Reducing bipartite matching to max flow

- Reduction to max flow
  - Create directed graph $G' = (L \cup R \cup \{s,t\}, E')$
  - Direct all edges from L to R, and assign capacity 1
  - Add source s, and capacity 1 edges to every node in L
  - Add sink t, and capacity 1 edges from each node in R to t.

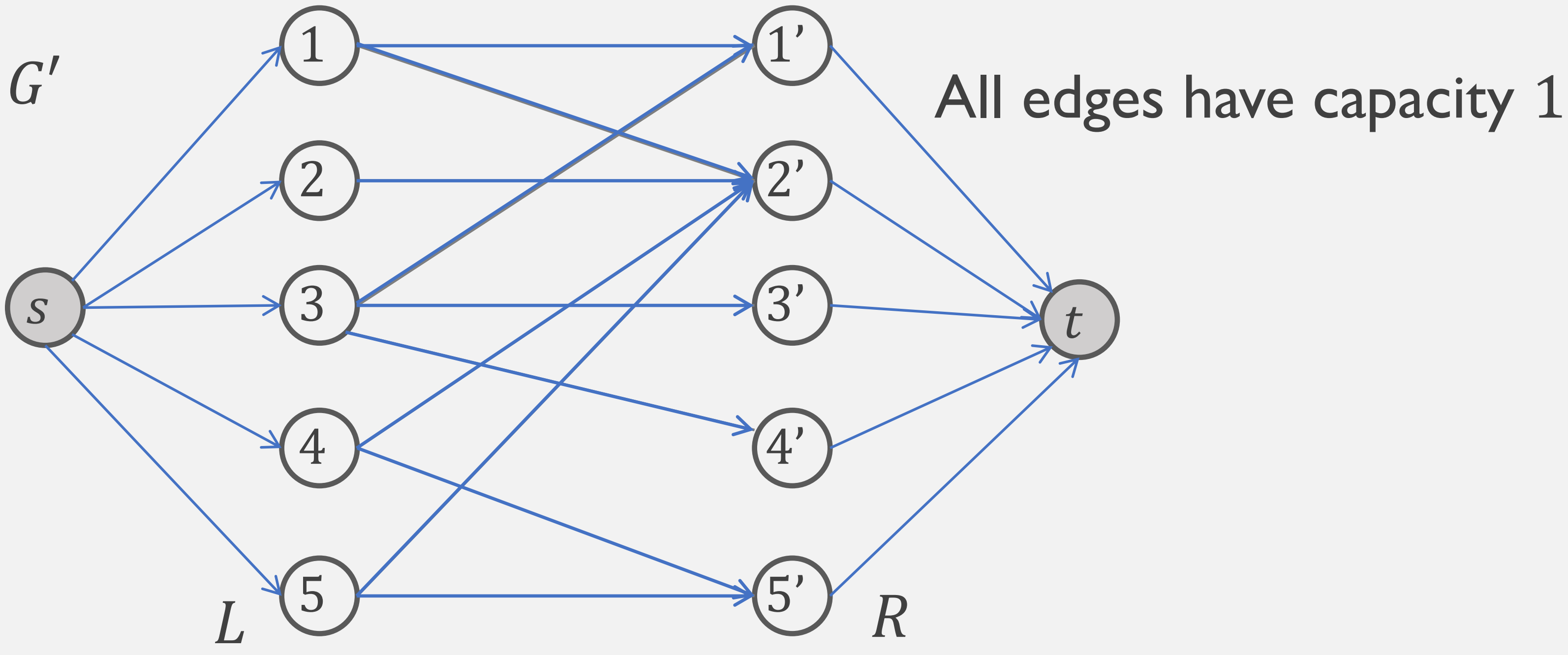


Bipartite matching: proof of correctness

Theorem. Max cardinality matching in G = value of max flow in G'

Proof. We show two claims

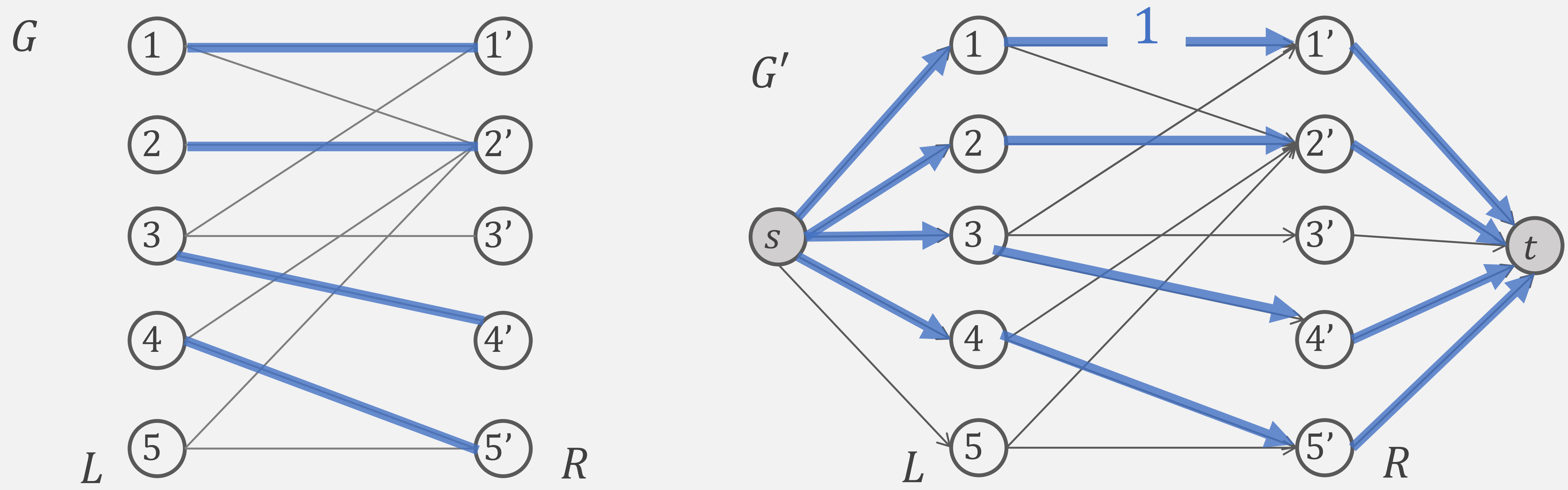
- Max matching in $G \leq$ max flow in G'
- Max matching in $G \geq$ max flow in G'



Bipartite matching: proof of correctness

Proof. (Part 1) Max matching in G $\leq$ max flow in G'

- Given max matching M of cardinality k
- Consider flow f that send 1 unit along each of k paths
- f is a flow of value k



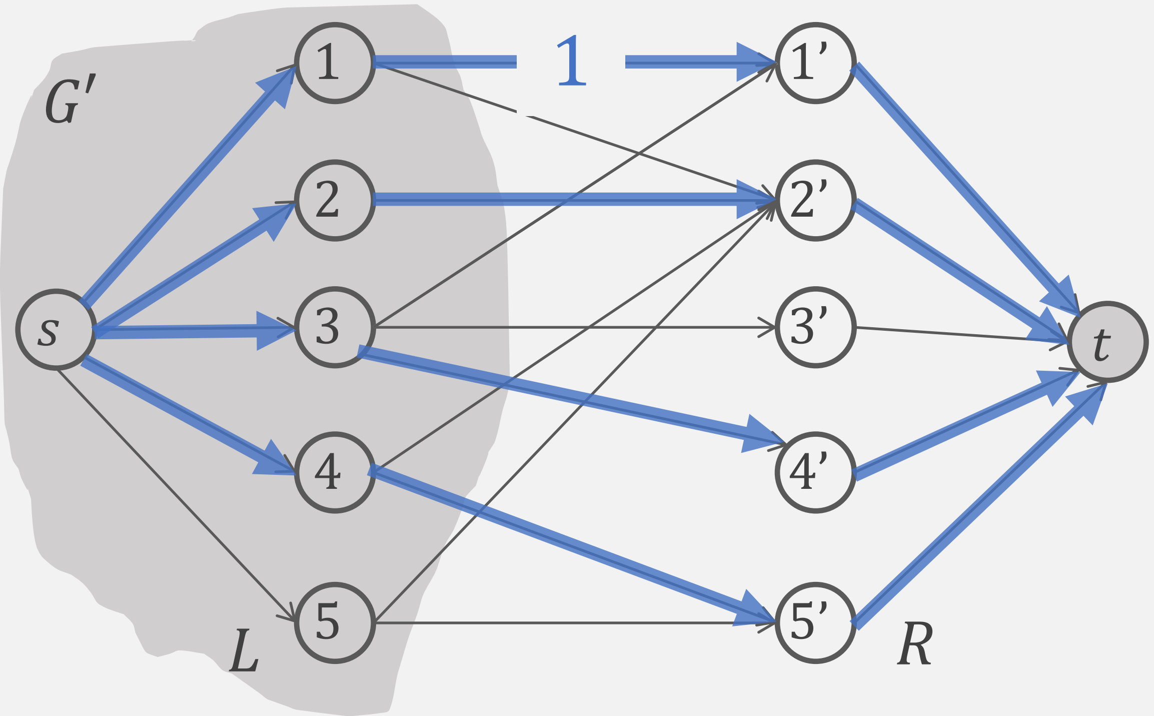
Bipartite matching: proof of correctness

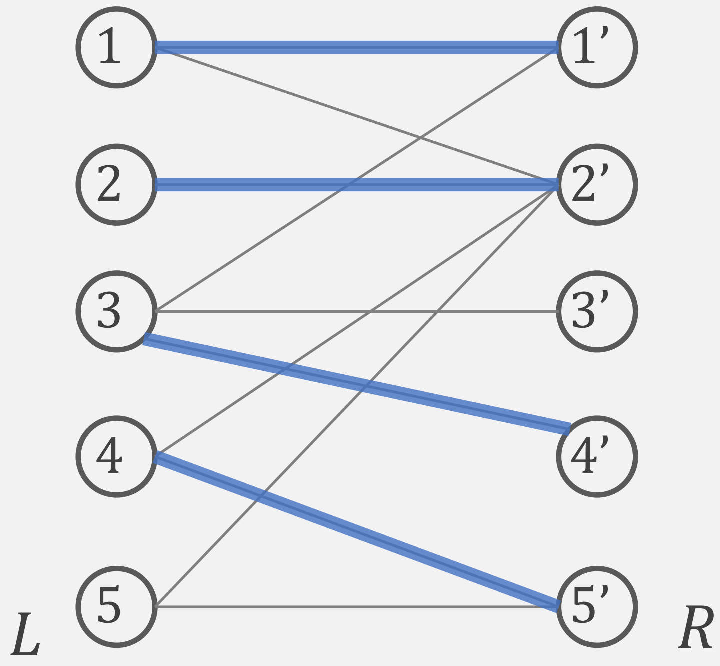
Proof. (Part 2) Max matching in $G \geq$ max flow in G'

- Given max flow f in G' of integer value k [Exists by Integrality theorem]
- All capacities are $1 \Rightarrow f(\mathrm{e})$ is 0 or 1. Let $M =$ edges from L to R with $f(e) = 1$

$\Rightarrow M$ is a matching (each node in L and R participate in at most one edge)

& M has size k (consider cut $(s \cup L, R \cup t)$).





Perfect matching

Def. A matching $M \subseteq E$ is perfect if each node appears in exactly one edge in M.

When does a bipartite graph have a perfect matching?

Structure of bipartite graphs with perfect matchings.

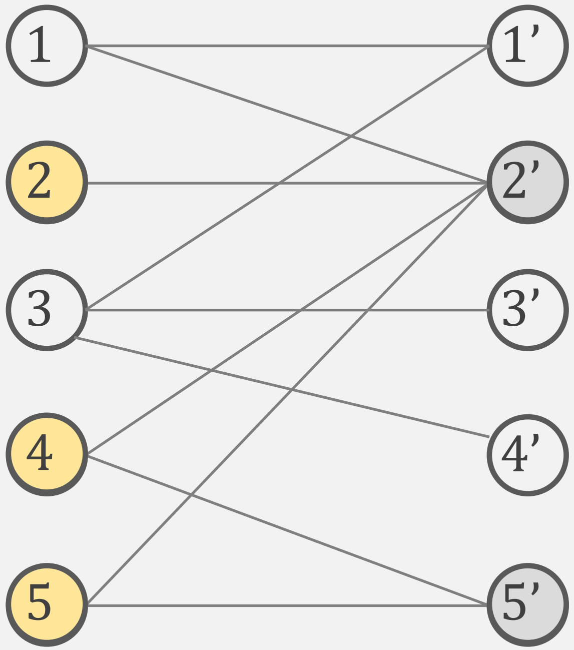
- Clearly we must have $|L| = |R|$
- What other conditions are necessary?
- What conditions are sufficient?

Perfect matching

Notation. Let S be a subset of nodes. Let $N(S)$ be the set of nodes adjacent to nodes in S.

Observation. If a bipartite graph $G = (L \cup R, E)$, has a perfect matching, then $|N(S)| \geq |S|$ for all subsets $S \subseteq L$.

- Pf. Each node in S has to be matched to a different node in $N(S)$.



No perfect matching
$S = \{2,4,5\}, N(S) = \{2', 5'\}$

Marriage theorem

Actually, this is also a sufficient condition …

Marriage Theorem [Frobenius1917, Hall1935]

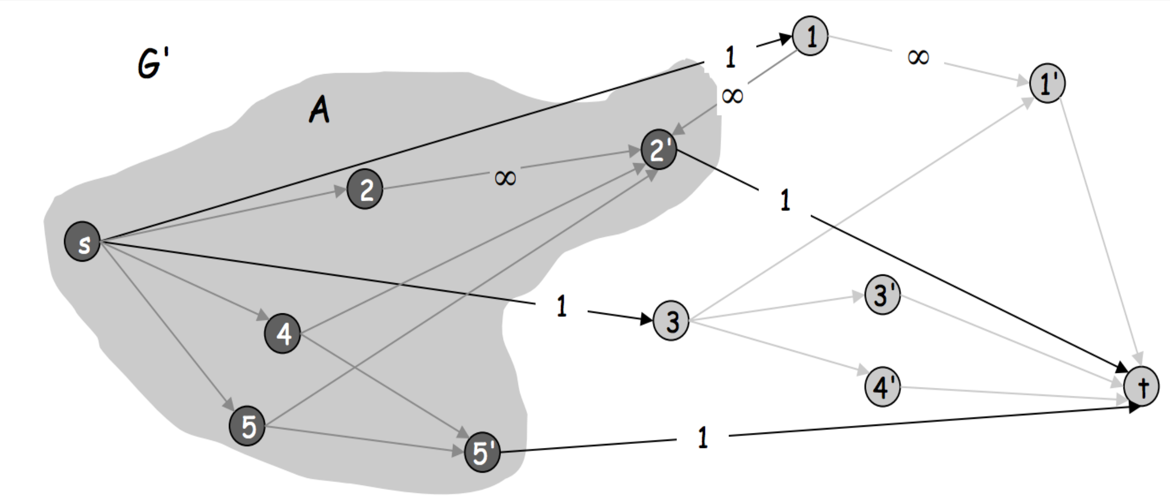
Let $G = (L \cup R, E)$ be a bipartite graph with $|L| = |R|$. Then G has a perfect matching if and only if $|N(S)| \geq |S|$ for all subsets $S \subseteq L$.

Pf. $\Rightarrow$ ("only if") This is the previous observation.

Marriage theorem: proof

Pf. $\Leftarrow$ ("if") Suppose for contradiction $|N(S)| \geq |S|$ for all subsets $S \subseteq L$, but G does not contain a perfect matching.

- Formulate as a max flow problem in G' with ∞ capacities on edges from L to R.
- Let (A, B) be min cut in G'. By max-flow min-cut, $cap(A,B) < |L|$
- Let $L_A = L \cap A, L_B = L \cap B, R_A = R \cap A$. Then $cap(A,B) = |L_B| + |R_A|$.
  $\Rightarrow |R_A| = cap(A,B) - |L_B| < |L| - |L_B| = |L_A|$
- Since min cut cannot use ∞ edges, $N(L_A) \subseteq R_A$. $|N(L_A)| \leq |R_A| < |L_A|$!!!



$$L = \{1 \ldots 5\}, R = \{1' \ldots 5'\}$$
$$L_A = \{2,4,5\}, L_B = \{1,3\}, R_A = \{2', 5'\}$$
$$N(L_A) = \{2', 5'\}$$

Additional remarks on Max flow algorithms

Ford-Fulkerson augmenting-path algorithm

$\boldsymbol{Ford\!-\!Fulkerson}(G, s, t, c)$
For each $e \in E$ $f(e) \leftarrow 0$, $G_f \leftarrow residual\ graph$
While there is an augmenting path P in G_f
  $f \leftarrow Augment(f, c, P)$
  Update G_f
return f

Theorem. Ford-Fulkerson terminates in at most nC iterations.

Running time. $O(mnC)$

Exponential in input size:
$\log C$ bits (to represent C)

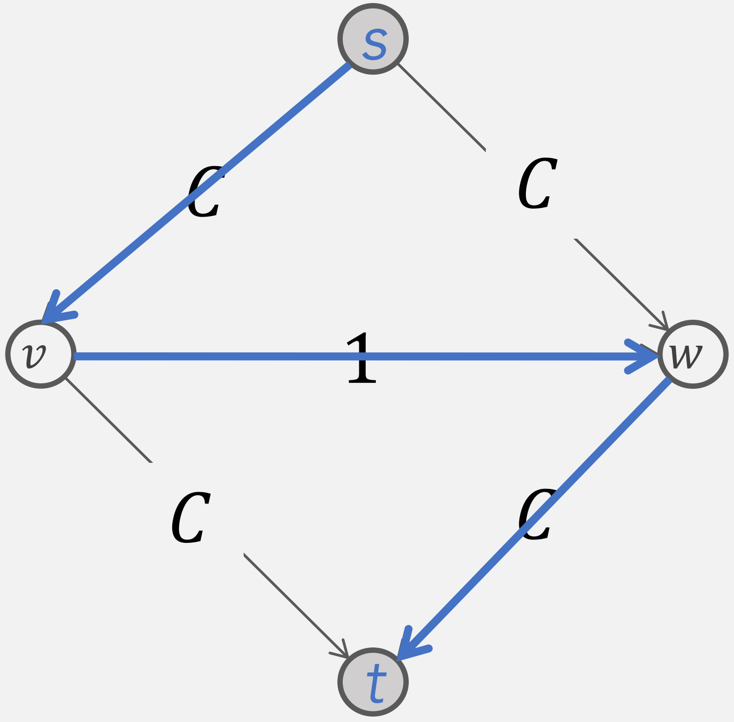
Can it be this bad?

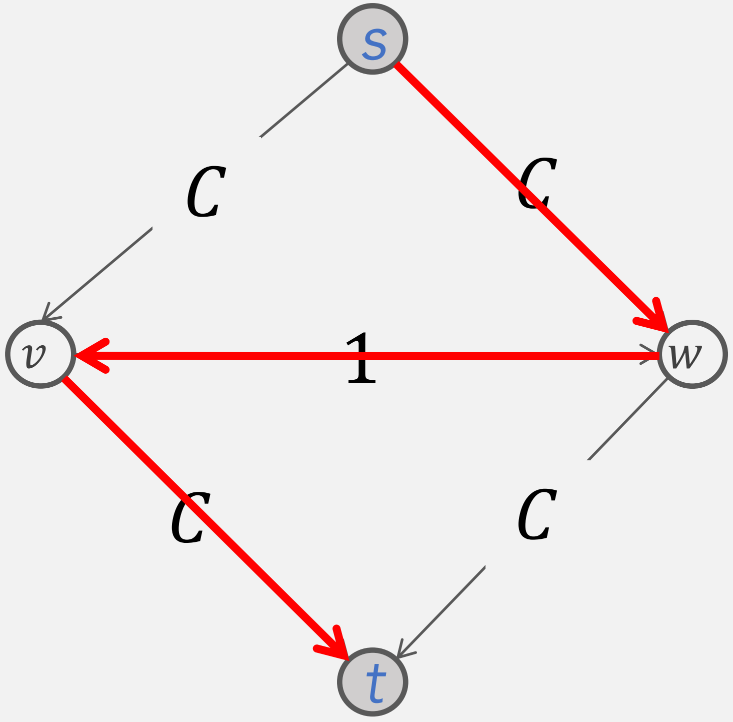
Ford-Fulkerson: exponential example

Obs. If max capacity is C, then FF can take $\geq C$ iterations.

- $s \to v \to w \to t$
- $s \to w \to v \to t$
- $s \to v \to w \to t$
- $s \to w \to v \to t$
- $\ldots$
- $s \to v \to w \to t$
- $s \to w \to v \to t$

Each augmenting path sends only 1 unit of flow (# augmenting paths = $2C$)





Choosing good augmenting paths

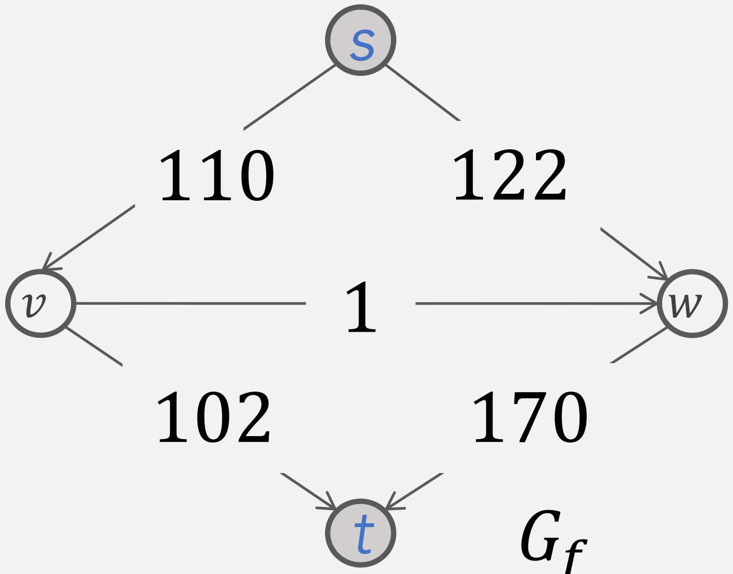
- Use care when selecting augmenting paths
  - Some choices lead to exponential algorithms
  - Clever choices lead to polynomial algorithms
  - If capacities are irrational, algorithm not guaranteed to terminate!

- Good choices of augmenting paths [EdmondsKarp’72,Dinitz’70]
  - Max bottleneck capacity [Next]
  - Fewest edges (shortest) [CLRS 26.2]

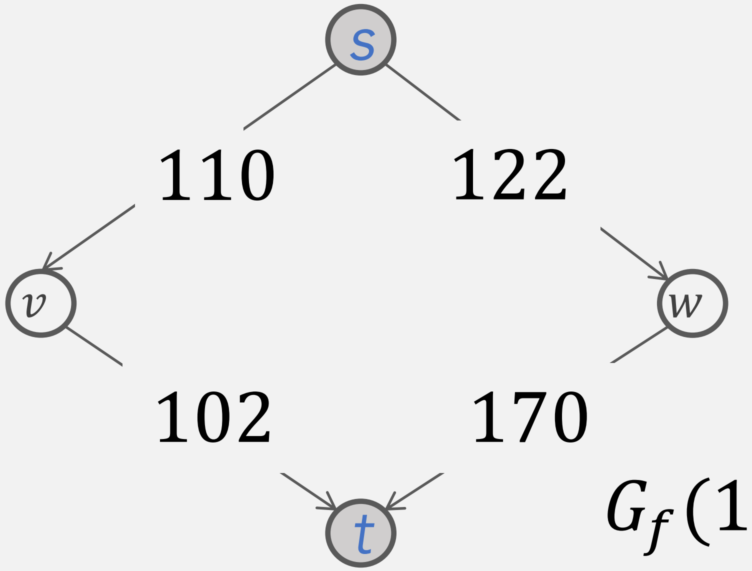
Capacity scaling

Intuition. Choosing path with highest bottleneck capacity increases the flow by max possible amount

- OK to choose sufficiently large bottleneck: scaling parameter Δ
- Let $G_f(\Delta)$ be the subgraph of the residual graph consisting of only arcs with capacity at least Δ



G_f



$G_f(100)$

Capacity scaling algorithm

Scaling–Max–Flow (G, s, t, c)
For each $e \in E$ $f(e) \leftarrow 0$,
$G_f \leftarrow residual\ graph$
$\Delta \leftarrow$ smallest power of 2 & $\geq C$

While $\Delta \geq 1$
    $G_f(\Delta) \leftarrow \Delta$–residual graph
    While there is an augmenting path P in $G_f(\Delta)$
        $f \leftarrow Augment(f, c, P)$ // augment flow by $\geq \Delta$
        Update $G_f(\Delta)$
    $\Delta \leftarrow \Delta/2$
return f

Exercise. Prove correctness

Capacity scaling algorithm: running time

While $\Delta \geq 1$
    $G_f(\Delta) \leftarrow \Delta$−residual graph
    While there is P in $G_f(\Delta)$
        $f \leftarrow \mathit{Augment}(f, c, P)$
        Update $G_f(\Delta)$
    $\Delta \leftarrow \Delta/2$
…

Lemma1 Outer loop runs $1 + \log C$ times.

Pf. Initially $C \leq \Delta \leq 2C$, decreases by a factor of 2 each iteration

Lemma2. Let f be the flow at the end of a Δ-scaling phase. Then the value of the maximum flow f^* is at most $v(f) + m\Delta$.

Lemma3. There are at most $2m$ augmentations per scaling phase.

Pf. Let f be the flow at end of previous scaling

- [Lemma2] $\Rightarrow v(f^*) \leq v(f) + m(2\Delta)$
- Each augmentation in Δ-scaling increases f by Δ

Theorem. Scaling-max-flow finds a max flow in $O(m^2 \log C)$ time.

Completing the proof

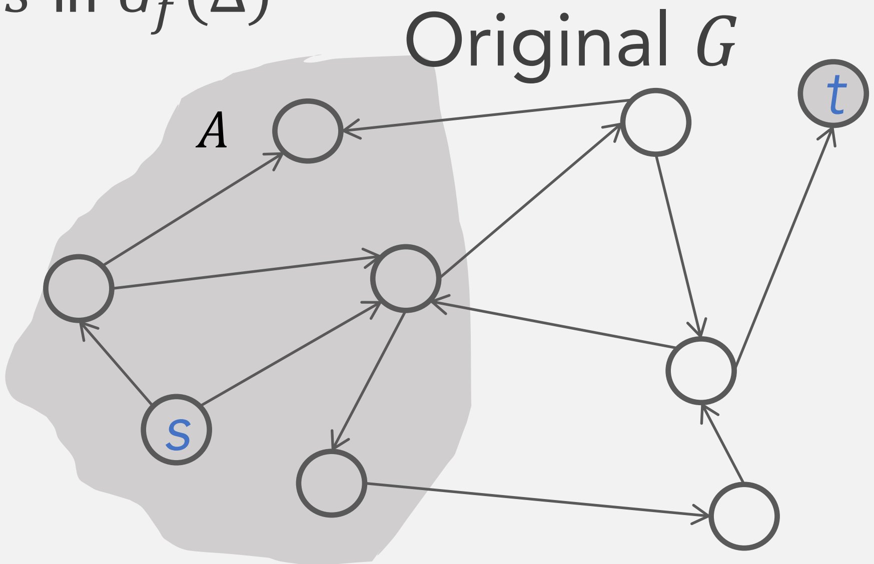
Lemma2. Let f be the flow at the end of a Δ-scaling phase. Then the value of the maximum flow f^* is at most $v(f) + m\Delta$.

Pf. [Almost identical to proof of max-flow min-cut theorem]

Show cut (A, B) w. $cap(A, B) \leq v(f) + m\Delta$ at the end of a Δ−phase.

- Choose A to be the set of nodes reachable from s in $G_f(\Delta)$
- By definition $s \in A$ & $t \notin A$

$$
\begin{aligned}
v(f) &= \textstyle\sum_{e \text{ outof } A} f(e) - \sum_{e \text{ into } A} f(e) \\
&\geq \textstyle\sum_{e \text{ outof } A} (c(e) - \Delta) - \sum_{e \text{ into } A} \Delta \\
&= \textstyle\sum_{e \text{ outof } A} c(e) - \sum_{e \text{ outof } A} \Delta - \sum_{e \text{ into } A} \Delta \\
&\geq cap(A, B) - m\Delta
\end{aligned}
$$



Augmenting-path algorithms: summary

Year	Method	# augmentations	Running time
1955	Augmenting path	nC	$O(mnC)$
1972	Fattest path	$m \log mC$	$O(m^2 \log n \log mC)$
1972	Capacity scaling	$m \log C$	$O(m^2 \log C)$
1985	Improved CapS	$m \log C$	$O(mn \log C)$
1970	Shortest path	mn	$O(m^2 n)$
1970	level graph	mn	$O(mn^2)$
1983	dynamic trees	mn	$O(mn \log n)$

and the show goes on …

Year	Method	Worst case	Discovered by
1951	Simplex	$O(mn^2C)$	Dantzig
1955	Augmenting path	$O(mnC)$	Ford-Fulkerson
…			
1988	Push-relabel	$O(mn\log(n^2/m))$	Goldberg-Tarjan
…			
2013	Compact networks	$O(mn)$	Orlin
2016	Electrical flows	$\tilde{O}(m^{10/7}C^{1/7})$	Madry
20XX			

To keep it simple, cite below when you invoke a max-flow subroutine in hw/exam

Maximum flows can be computed in $O(mn)$ time

Another formulation of max-flow problem

Recall. An $s{-}t$ flow is a function $f: E \to \mathbb{R}$ satisfying

- [Capacity] $\forall e \in E: 0 \le f(e) \le c(e)$
- [Conservation] $\forall v \in V \backslash \{s,t\}$: $\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$

The value of a flow f is $v(f) \coloneqq \sum_{e \text{ out of } s} f(e)$

Max−Flow Problem

Real-value variables $\vec{f} = \{f_e : e \in E\}$

Maximize: $v(\vec{f})$

Subject to:

$$0 \le f_e \le c(e), \ \forall e \in E$$

$$\sum_{e \text{ into } v} f_e - \sum_{e \text{ out of } v} f_e = 0, \ \forall v \in V \backslash \{s,t\}$$

Linear constraints: no $x^2, xy, \sin(x)$, ...

Grade maximization

Input. HW from two courses (xxx & 584/684) due in one day

- Every hour you spend, you earn 1pts on xxx or 5pts on 584/684
- Your brain will explode if you work more than 12hrs on xxx or 15hrs on 5/684
- Of course, there are only 24 hrs in a day

Goal. Maximize the total pts you can earn

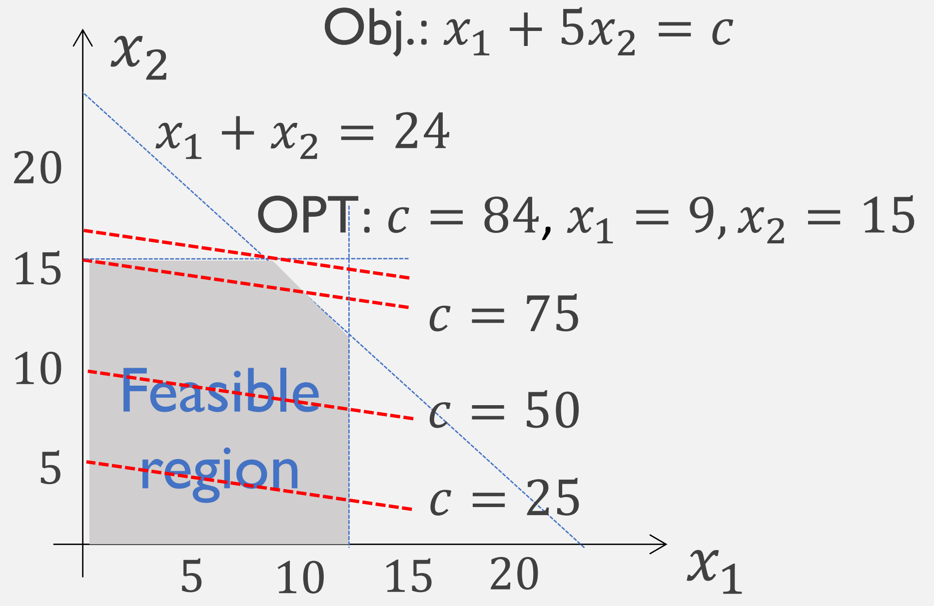
Grade—Maximization

Variables: x_1 (xxx hrs); x_2 (5/684 hrs)

Maximize: $x_1 + 5x_2$

Subject to: // linear constraints

$$0 \leq x_1 \leq 12$$
$$0 \leq x_2 \leq 15$$
$$x_1 + x_2 \leq 24$$



Linear programming

Linear programming. Optimize a linear objective function subject to linear inequalities.

- Formal definition and representations
- Duality
- Algorithms: simplex, ellipsoid, interior point

Why significant?

- Design poly-time algorithms & approximation algorithms
- Wide applications: math, economics, business, transportation, energy, telecommunications, and manufacturing

Ranked among most important scientific advances of 20th century

Linear programming

- "Standard form" of an LP
  - m = # constraints, n = # decision variables. $i = 1, \dots, m, j = 1, \dots, n$
  - Input: real numbers c_j, a_{ij}, b_i
  - Output: real numbers x_j
  - Maximize linear objective function subject to linear inequalities
  - Feasible vs. optimal soln's.

Max $\sum_{j=1}^{n} c_j x_j$

Subject to: // linear constraints

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i \quad 1 \le i \le m$$

$$x_j \ge 0 \quad 1 \le j \le n$$

$\equiv$

Max $\boldsymbol{c}^T \boldsymbol{x}$

Subject to: $A\boldsymbol{x} \le \boldsymbol{b}$

$\boldsymbol{x} \ge \boldsymbol{0}$

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ a_{21} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}$$

$$\boldsymbol{c} = \begin{pmatrix} c_1 \\ c_2 \\ \cdots \\ c_n \end{pmatrix} \quad \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \\ \cdots \\ b_m \end{pmatrix} \quad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{pmatrix} \quad \boldsymbol{0} = \begin{pmatrix} 0 \\ 0 \\ \cdots \\ 0 \end{pmatrix}$$

Linear programming: variants

- "Slack form" of an LP: linear equalities

Max $\sum_{j=1}^{n} c_j x_j$

Subject to: // linear constraints

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i \quad 1 \le i \le m$$

$$x_j \ge 0 \quad 1 \le j \le n$$

$\Rightarrow$

Max $\sum_{j=1}^{n} c_j x_j$

Subject to: // linear constraints

$$s_i = b_i - \sum_{j=1}^{n} a_{ij} x_j \quad 1 \le i \le m$$

$$\text{(slack vars) } s_i \ge 0 \quad 1 \le i \le m$$

$$x_j \ge 0 \quad 1 \le j \le m$$

- Other equivalent variations
  - Minimization vs. maximization
  - Variables without nonnegativity constraints
  - $\ge$ vs. $\le$