Network Flow Applications

Admin

- Assignment 6 is due next Wed
- Midterm grading progress
- Colloquium today: **3.15 pm**
    - Talk by Williams alum **Daniel Seita**
    - Currently a PhD at UC Berkeley
    - Works on robotic manipulation and machine learning
- Cool opportunity to hear about what students (like you) end up doing post Williams!
- Cool opportunity to find out what CS research looks like
    - Zoom link sent by Lauren

Ford-Fulkerson Running Time

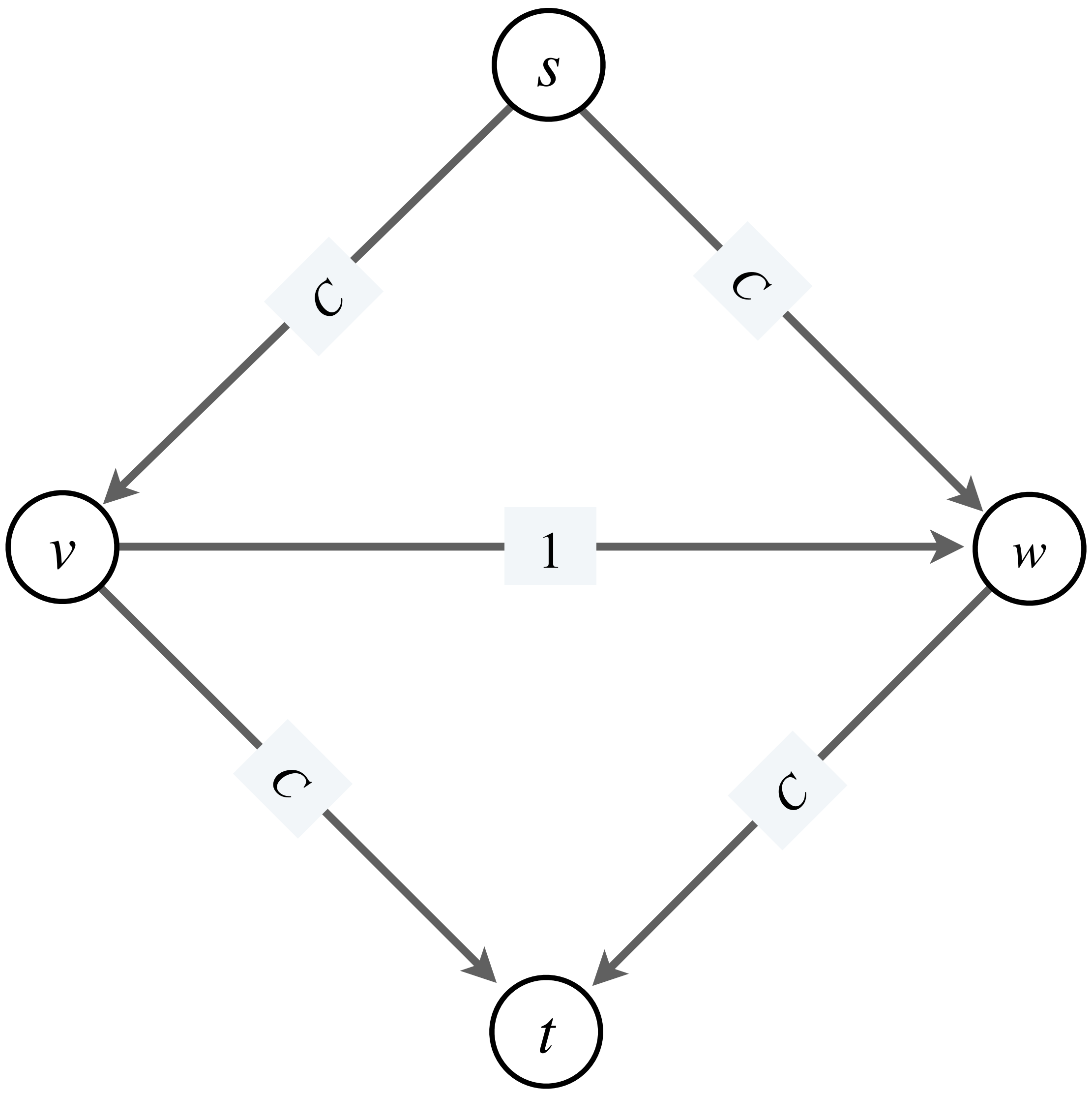
- **Claim.** Ford-Fulkerson can be implemented to run in time $O(nmC)$, where $m = |E| \geq n - 1$ and $C = \max_u c(s \to u)$.
- **Proof**. Time taken by each iteration:
- Finding an augmenting path in G_f
  - G_f has at most $2m$ edges, using BFS/DFS takes $O(m + n) = O(m)$ time
- Augmenting flow in P takes $O(n)$ time
- Given new flow, we can build new residual graph in $O(m)$ time
- Overall, $O(m)$ time per iteration and $O(nC)$ iterations ∎

[Digging Deeper] Polynomial time?

- Does the Ford-Fulkerson algorithm run in time polynomial in the input size?
- Running time is $O(nmC)$, where $C = \max_{u} c(s \to u)$
- What is the input size?
    - n vertices, m edges, m capacities
    - C represents the magnitude of the maximum capacity leaving the source node
    - How many bits to represent C?
- Let us take an example

[Digging Deeper] Polynomial time?

- **Question**. Does the Ford-Fulkerson algorithm run in polynomial-time in the size of the input? ← $\sim m$, n, and $\log C$
- **Answer**. No. if max capacity is C, the algorithm can take $\geq C$ iterations. Consider the following example.



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

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

[Digger Deeper] Pseudo-Polynomial

- Input graph has n nodes and $m = O(n^2)$ edges, each with capacity c_e
- $C = \max_{e \in E} c(e)$, then $c(e)$ takes $O(\log C)$ bits to represent
- Input size: $\Omega(n \log n + m \log n + m \log C)$ bits
- Running time: $O(nmC) = O(nm2^{\log C})$
    - Exponential in the *size* of C
- Such algorithms are called **pseudo-polynomial**
    - If the running time is polynomial in the **magnitude** but **not size** of an input parameter.
    - We saw this for knapsack as well!

Non-Integral Capacities?

- If the capacities are rational, can just multiply to obtain a large integer (massively increases running time)
- If capacities are irrational, Ford-Fulkerson can run infinitely!
    - Improvement at each step can be arbitrarily small
    - Can create bad instances where it doesn't terminate in finite steps

Network Flow:
Beyond Ford Fulkerson

Edmond and Karp's Algorithms

- Ford and Fulkerson's algorithm does not specify which path in the residual graph to augment
- Poor worst-case behavior of the algorithm can be blamed on bad choices on augmenting path
- **Better choice of augmenting paths.** In 1970s, Jack Edmonds and Richard Karp published two natural rules for choosing augmenting paths
  - Widest path first: paths with largest bottleneck capacity
  - Shortest (in terms of edges) augmenting paths first (Dinitz independently discovered & analyzed this rule)

Widest Augmenting Paths First

- Ford Fulkerson can be improved with a greedy algorithm way of choosing augmenting paths:
  - Choose the augmenting path with largest bottleneck capacity
- Largest bottleneck path can be computed in $O(m \log n)$ time in a directed graph
  - Similar to Dijkstra's analysis
- How many iterations if we use this rule?
  - Won't prove this: but takes $O(m \log C)$ iterations
- Overall running time is $O(m^2 \log n \log C)$ (polynomial time!)
  - Still depends on C though

Shortest Augmenting Paths First

- Choose the augmenting path with the smallest # of edges
- Can be found using BFS on G_f in $O(m+n) = O(m)$ time
- Surprisingly, this resulting a polynomial-time algorithm independent of the actual edge capacities !
    - Analysis looks at “level” of vertices in the BFS tree of G_f rooted at s —levels only grow over time
    - Analyzes # of times an edge $u \to v$ disappears from G_f
- Takes $O(mn)$ iterations overall
- Thus overall running time is $O(m^2n)$

Progress on Network Flows

1951	$O(m\, n^2\, C)$	Dantzig
1955	$O(m\, n\, C)$	Ford–Fulkerson
1970	$O(m\, n^2)$	Edmonds–Karp, Dinitz
1974	$O(n^3)$	Karzanov
1983	$O(m\, n \log n)$	Sleator–Tarjan
1985	$O(m\, n \log C)$	Gabow
1988	$O(m\, n \log (n^2 / m))$	Goldberg–Tarjan
1998	$O(m^{3/2} \log (n^2 / m) \log C)$	Goldberg–Rao
2013	$\boxed{O(m\, n)}$	Orlin
2014	$\tilde{O}(m\, n^{1/2} \log C)$	Lee–Sidford
2016	$\tilde{O}(m^{10/7}\, C^{1/7})$	Mądry
2017	$\tilde{O}(m^{10/7} \log W)$	Cohen-Mądr

For unit capacity networks

Progress on Network Flows

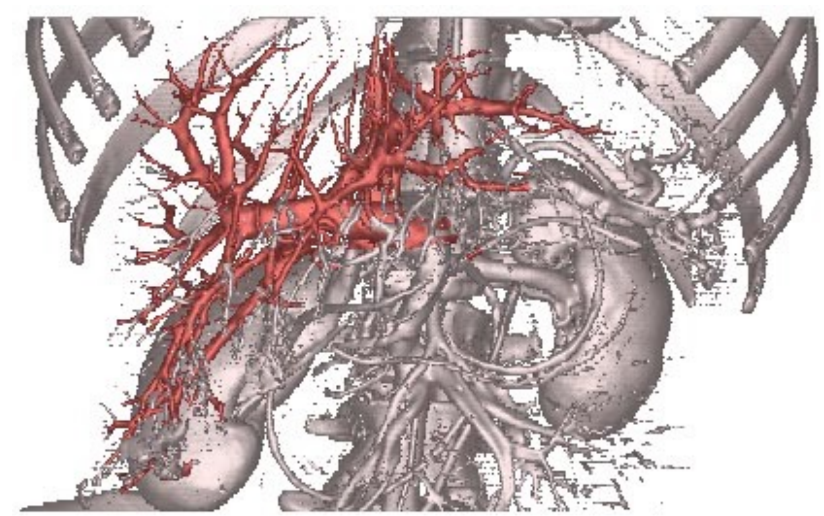
- Best known: $O(nm)$
- Best lower bound?
  - None known. (Needs $\Omega(n+m)$ just to look at the network, but that's it)
- Some of these algorithms do REALLY well in "practice" basically $O(n+m)$
- Well-known open problem

Applications of Network Flow:

Solving Problems by Reduction to Network Flows

Max-Flow Min-Cut Applications

- Data mining
- Bipartite matching
- Network reliability
- Image segmentation
- Baseball elimination
- Network connectivity
- Markov random fields
- Distributed computing
- Network intrusion detection
- **Many, many, more.**



liver and hepatic vascularization segmentation

Liver and hepatic vascularization segmentation using a Min-cut/Max-flow algorithm (S. Esneault, T. Pham, K. Torres)

Max-Flow Min-Cut Applications

- **Network flows** model a variety of optimization problems
- These optimization problems look complicated with lots of constraints; on the face of it seem to have nothing to do with networks or flows

Clients to base stations. Consider a set of mobile computing clients who each need to be connected to one of several possible base stations. We'll suppose there are n clients and k base stations; the position of each of these is specified by their (x, y) coordinates in the plane.

For each client, we wish to connect it to exactly one of the base stations, constrained in the following ways: a client can only be connected to a base station that is within distance r, and no more than L clients can be connected to any single base station. Design a polynomial time algorithm for the problem.

Max-Flow Min-Cut Applications

- **Network flows** model a variety of optimization problems
- These optimization problems look complicated with lots of constraints; on the face of it seem to have nothing to do with networks or flows

Survey design: Design survey asking n_1 consumers about n_2 products.

- Can survey consumer i about product j only if they own it.
- Ask consumer i between c_i and c_i' questions.
- Ask between p_j and p_j' consumers about product j.

Goal. Design a survey that meets these specs, if possible.

Max-Flow Min-Cut Applications

- **Network flows** model a variety of optimization problems
- These optimization problems look complicated with lots of constraints; on the face of it seem to have nothing to do with networks or flows

Airline scheduling: A very complicated scheduling problem but we can turn it into a simplified one:

Every day we have k flights and flight i leaves origin o_i at time s_i and arrives at destination d_i at time f_i.

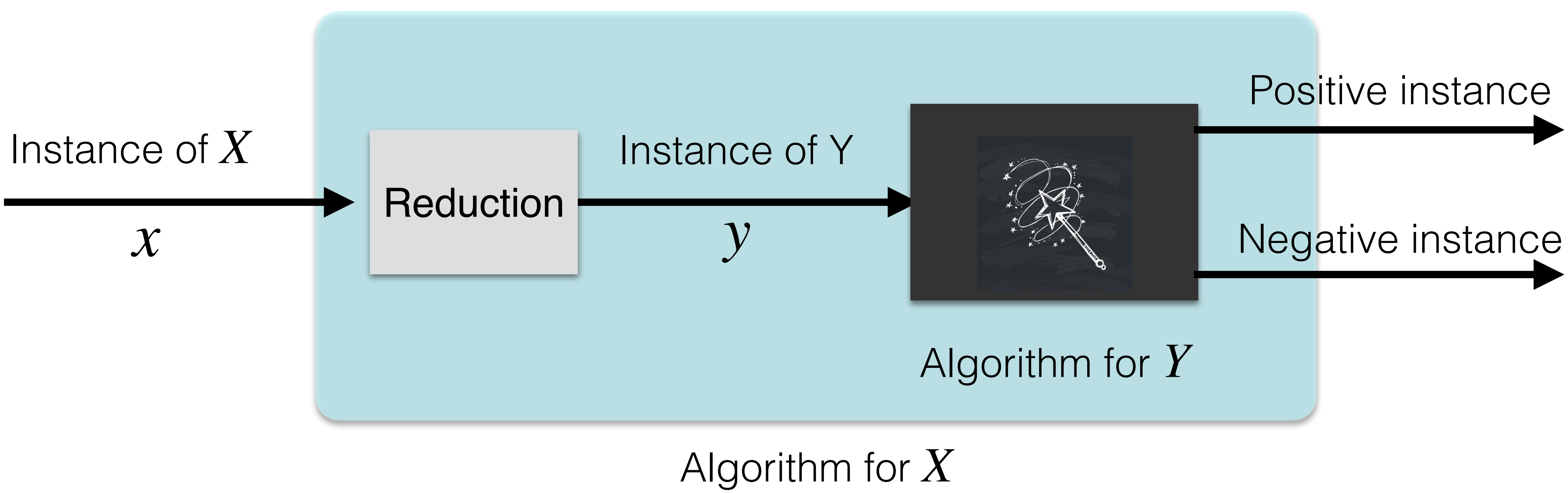
Goal. Minimize number of flight crews.

Reductions

- We will solve all these problems by reducing them to a network flow problem
- We'll focus on the concept of **problem reductions**

Anatomy of Problem Reductions

- At a high level, a problem X reduces to a problem Y if an algorithm for Y can be used to solve X
- **Reduction.** Convert an arbitrary instance x of X to a special instance y of Y such that there is a 1-1 correspondence between them



Anatomy of Problem Reductions

- **Claim.** x satisfies a property iff y satisfies a corresponding property
- Proving a reduction is correct: prove both directions
- x has a property (e.g. has matching of size k) $\implies$ y has a corresponding property (e.g. has a flow of value k)
- x does not have a property (e.g. does not have matching of size k) $\implies$ y does not have a corresponding property (e.g. does not have a flow of value k)
- Or equivalently (and this is often easier to prove):
    - y has a property (e.g. has flow of value k) $\implies$ x has a corresponding property (e.g. has a matching of value k)

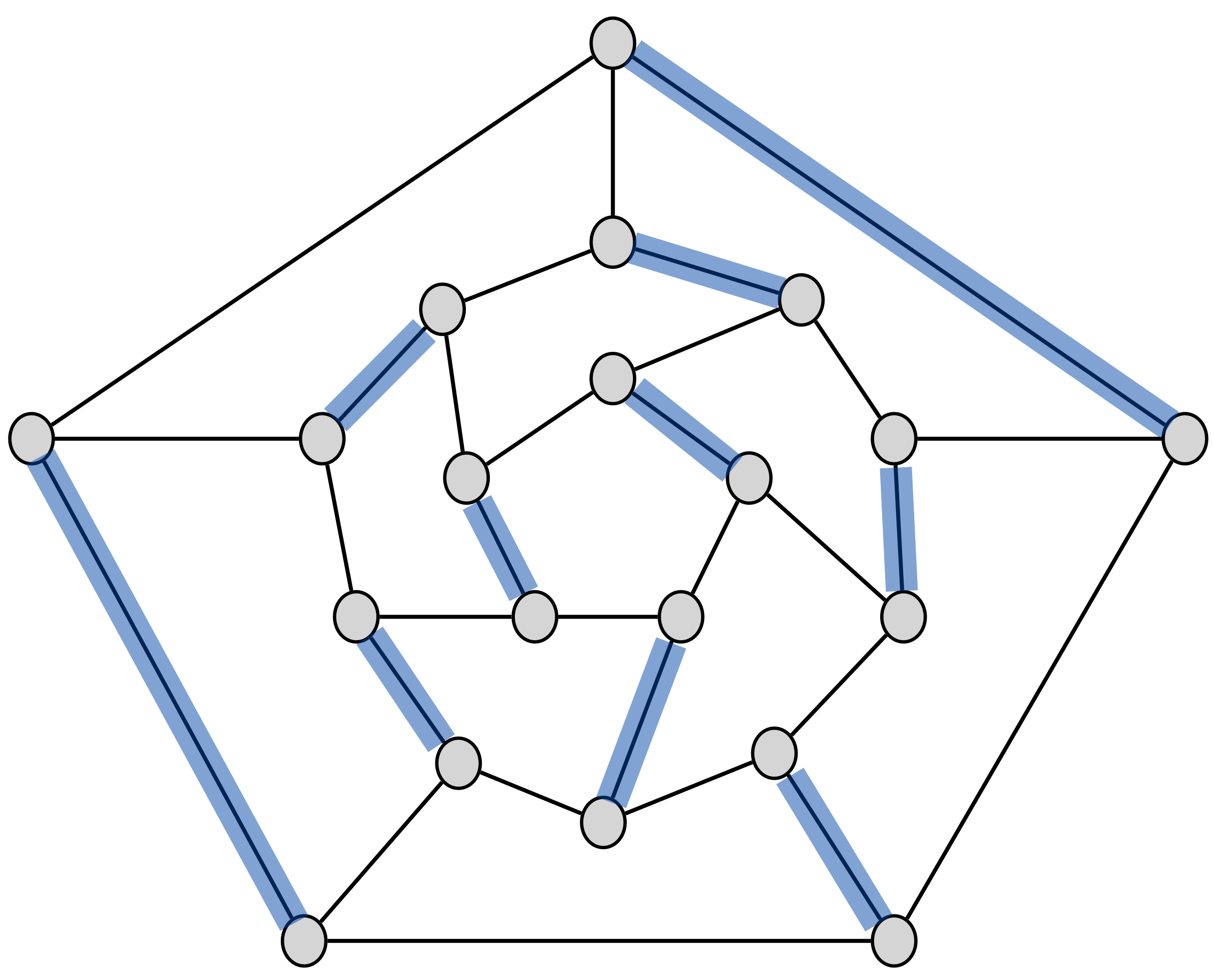
Today's Plan

- We will explore one application of network flow in detail today
    - Matching in bipartite graphs
    - Matchings are super practical have many applications
    - We have already seen one, can you remember?
- Next lecture: more applications of network flow
    - More practice with reductions
    - We will do some in-class exercises of reductions next time

Bipartite Matching

Review: Matching in Graphs

- **Definition.** Given an undirected graph $G = (V, E)$, a matching $M \subseteq E$ of G is a subset of edges such that no two edges in M are incident on the same vertex.
  - In other words, each node appears in at most one edge in M

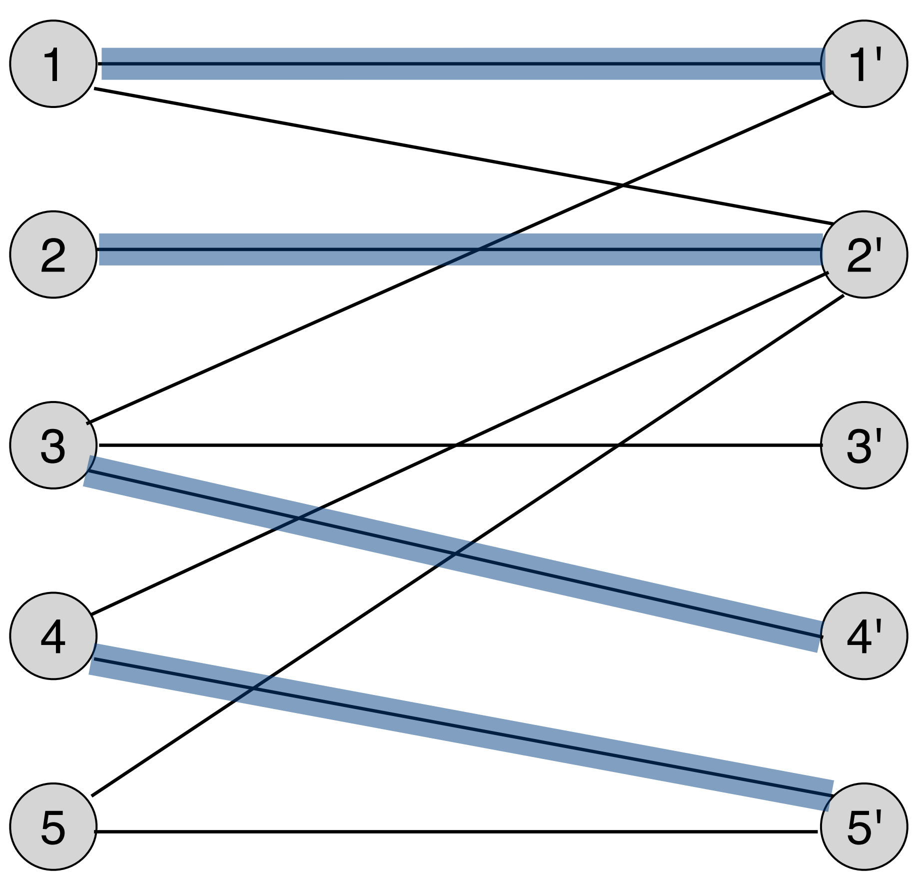


Review: Matching in Graphs

- **Definition.** Given an undirected graph $G = (V, E)$, a matching $M \subseteq E$ of G is a subset of edges such that no two edges in M are incident on the same vertex.
  - In other words, each node appears in at most one edge in M
- A perfect matching matches all nodes in G
- **Max matching problem.** Find a matching of maximum cardinality for a given graph, that is, a matching with maximum number of edges
  - A perfect matching if it exists is maximum!

Review: Bipartite Graphs

- A graph is **bipartite** if its vertices can be partitioned into two subsets X, Y such that every edge $e = (u, v)$ connects $u \in X$ and $v \in Y$
- **Bipartite matching problem.** Given a bipartite graph $G = (X \cup Y, E)$ find a maximum matching.



Bipartite Matching Examples

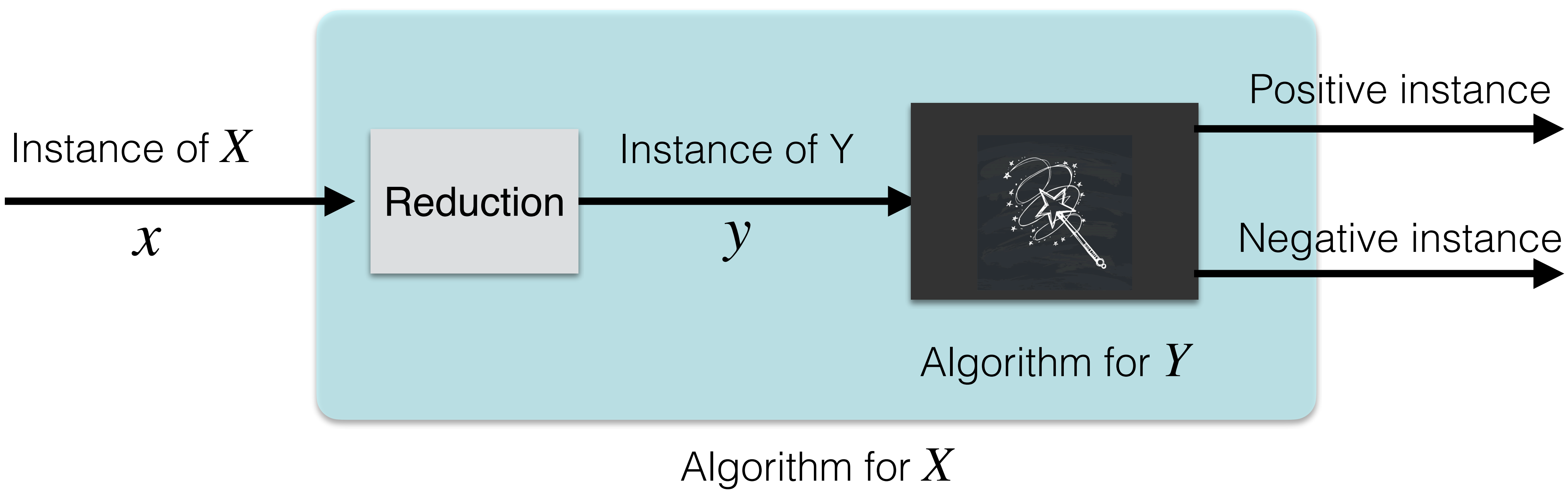
- Models many assignment problems
  - A is a set of jobs, B as a set of machines
  - Edge (a_i, b_j) indicates where machine b_j is able to process job a_i
  - Perfect matching: way to assign each job to a machine that can process it, such that, each machine is assigned exactly one job
- Assigning customers to stores, students to dorms, etc
- **Note.** This is a different problem than the one we studied for Gale-Shapely matching!

Maximum & Perfect Matchings

- One of the oldest problems in combinatorial algorithms:
    - Determine the largest matching in a bipartite graph
- This doesn't seem like a network flow problem
    - But we will turn it into one
- Special case: Find a perfect matching in G if it exists
    - What conditions do we need for perfect matching?
    - Certainly need $|A| = |B|$
    - What are the necessary and sufficient conditions?
    - Will use network flow to determine

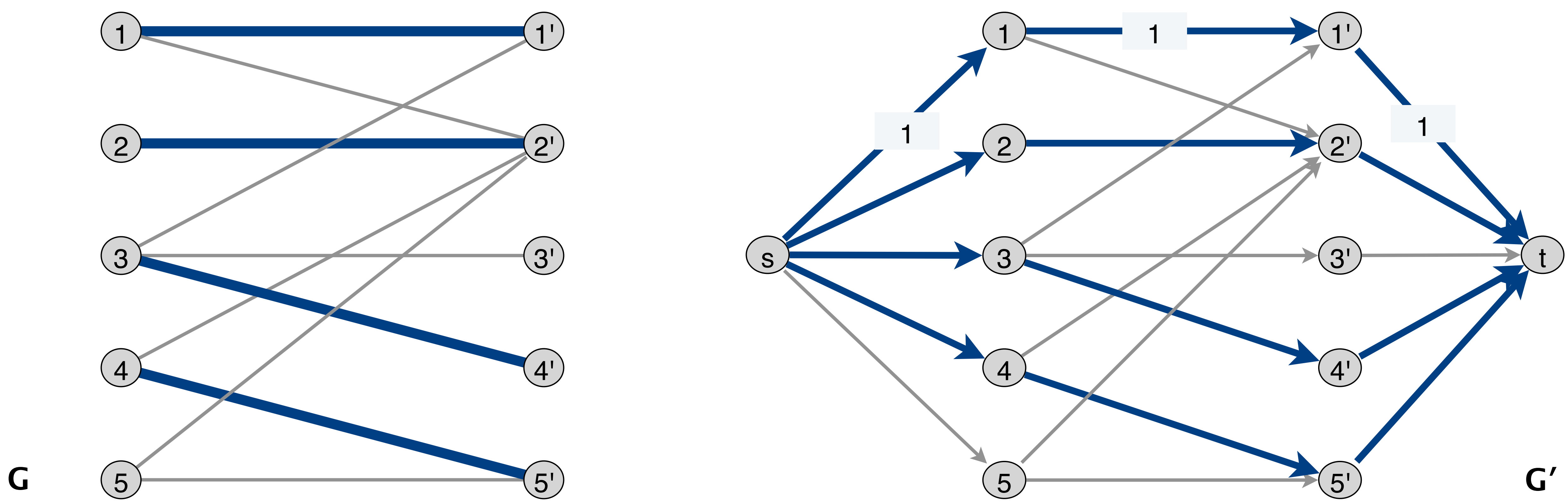
Reduction to Max Flow

- Given arbitrary instance x of bipartite matching problem (X): A, B and edges E between A and B
- **Goal.** Create a special instance y of a max-flow problem (Y): flow network: $G(V, E, c)$, source s, sink $t \in V$ s.t.
- **1-1 correspondence.** There exists a matching of size k iff there is a flow of value k



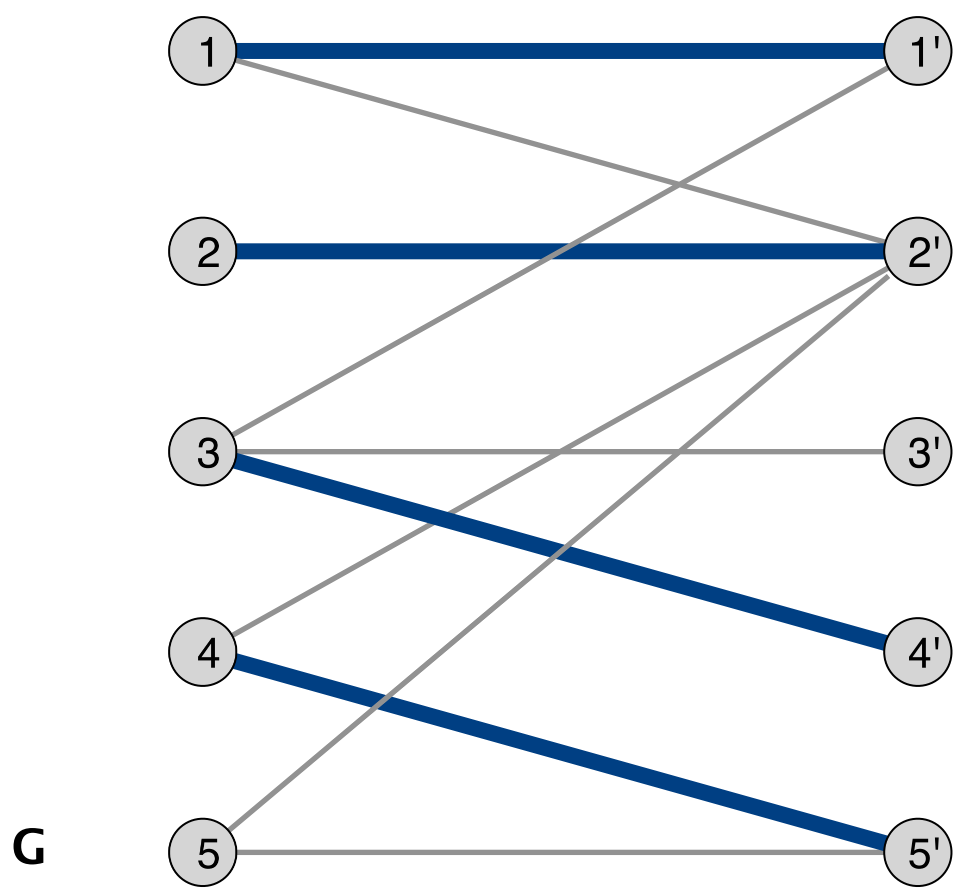
Reduction to Max Flow

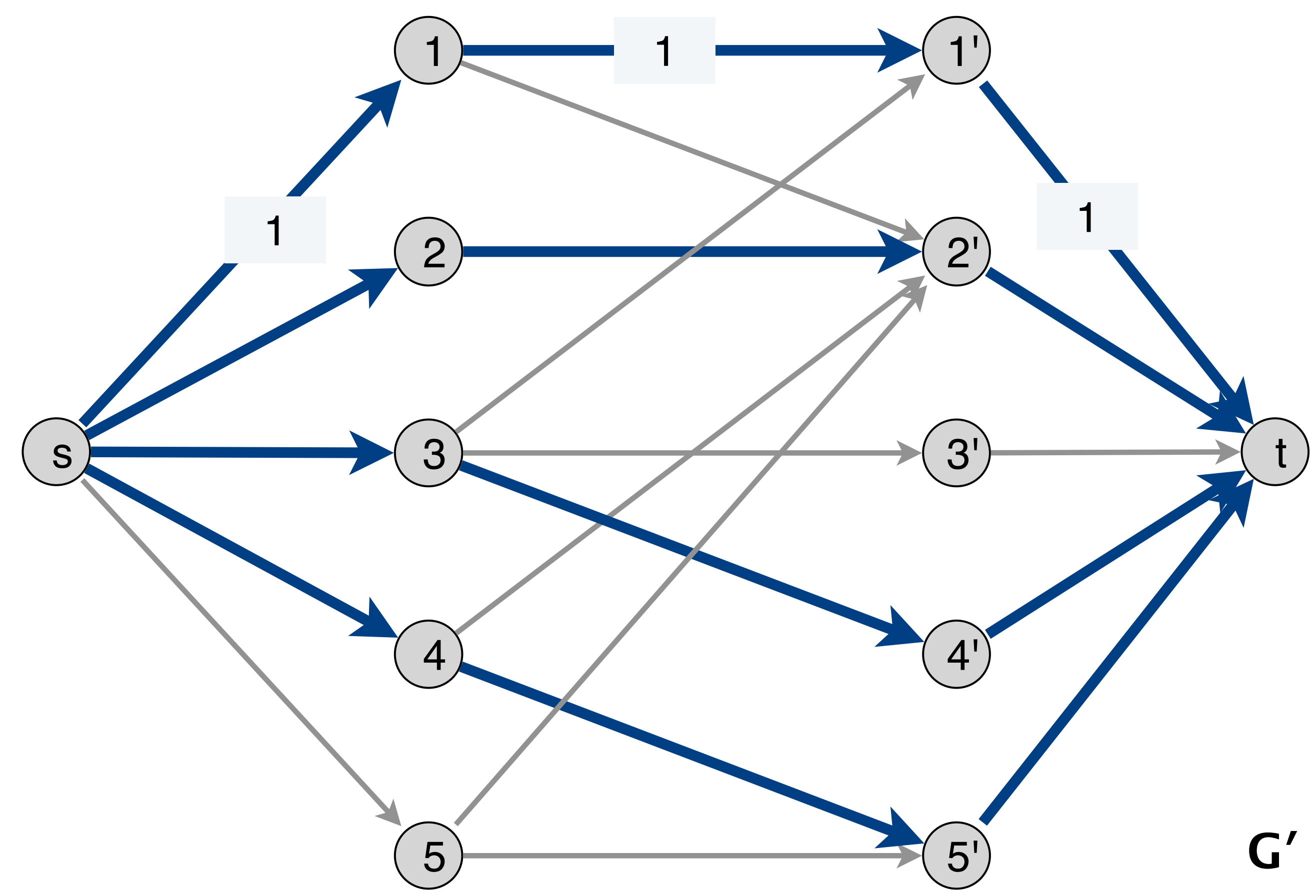
- Create a new directed graph $G' = (A \cup B \cup \{s, t\}, E', c)$
- Add edge $s \to a$ to E' for all nodes $a \in A$
- Add edge $b \to t$ to E' for all nodes $b \in B$
- Direct edge $a \to b$ in E' if $(a, b) \in E$
- Set capacity of all edges in E' to 1



Correctness of Reduction

- **Claim ($\Rightarrow$).**
  If the bipartite graph (A, B, E) has matching M of size k then flow-network G' has an integral flow of value k.





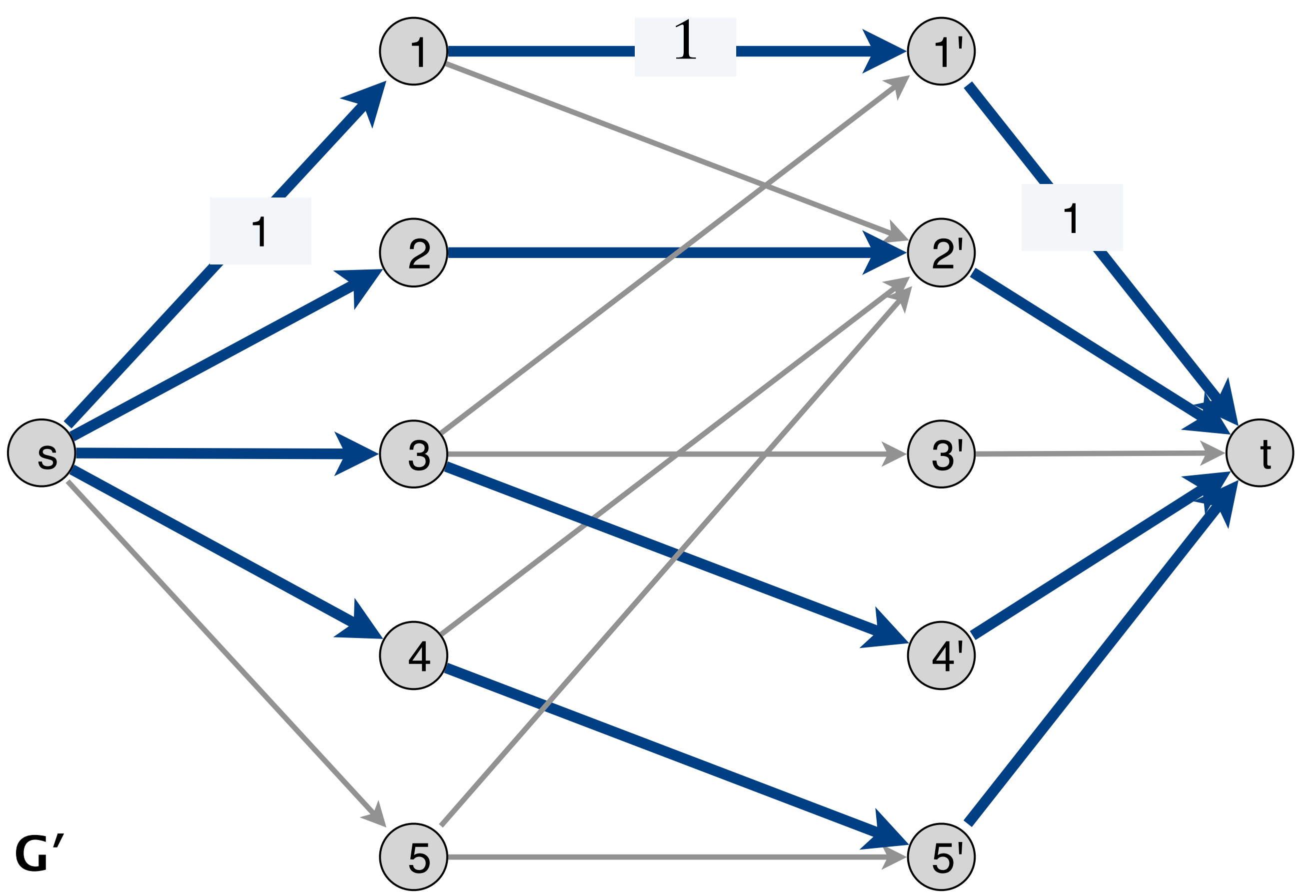
Correctness of Reduction

- **Claim** ($\Rightarrow$)**.**
  If the bipartite graph (A, B, E) has matching M of size k then flow-network G' has an integral flow of value k.

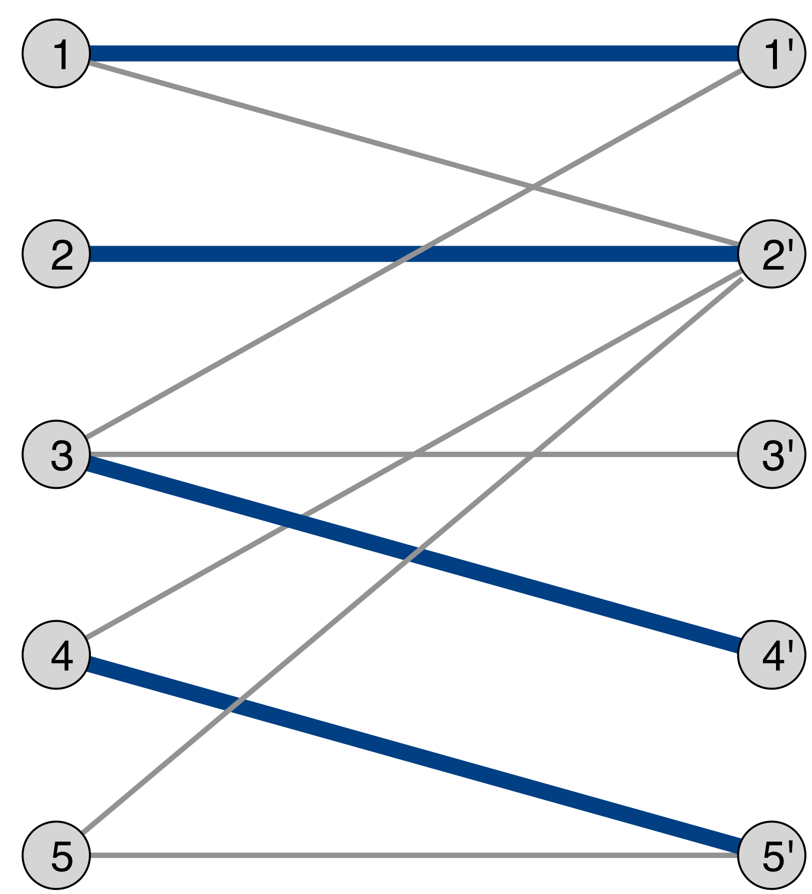
- **Proof**.
  - For every edge $e = (a, b) \in M$, let f be the flow resulting from sending 1 unit of flow along the path
    $s \to a \to b \to t$
  - f is a feasible flow (satisfies capacity and conservation) and integral
  - $v(f) = k$

Correctness of Reduction

- **Claim ($\Leftarrow$).**
  If flow-network G' has an integral flow of value k, then the bipartite graph (A, B, E) has matching M of size k.



G′



G

Correctness of Reduction

- **Claim** ($\Leftarrow$).
  If flow-network G' has an integral flow of value k, then the bipartite graph (A, B, E) has matching M of size k.

- **Proof**.
  - Let $M =$ set of edges from A to B with $f(e) = 1$.
  - No two edges in M share a vertex, why?
  - $|M| = k$
    - $v(f) = f_{out}(S) - f_{in}(S)$ for any $(S, V - S)$ cut
    - Let $S = A \cup \{s\}$

Summary & Running Time

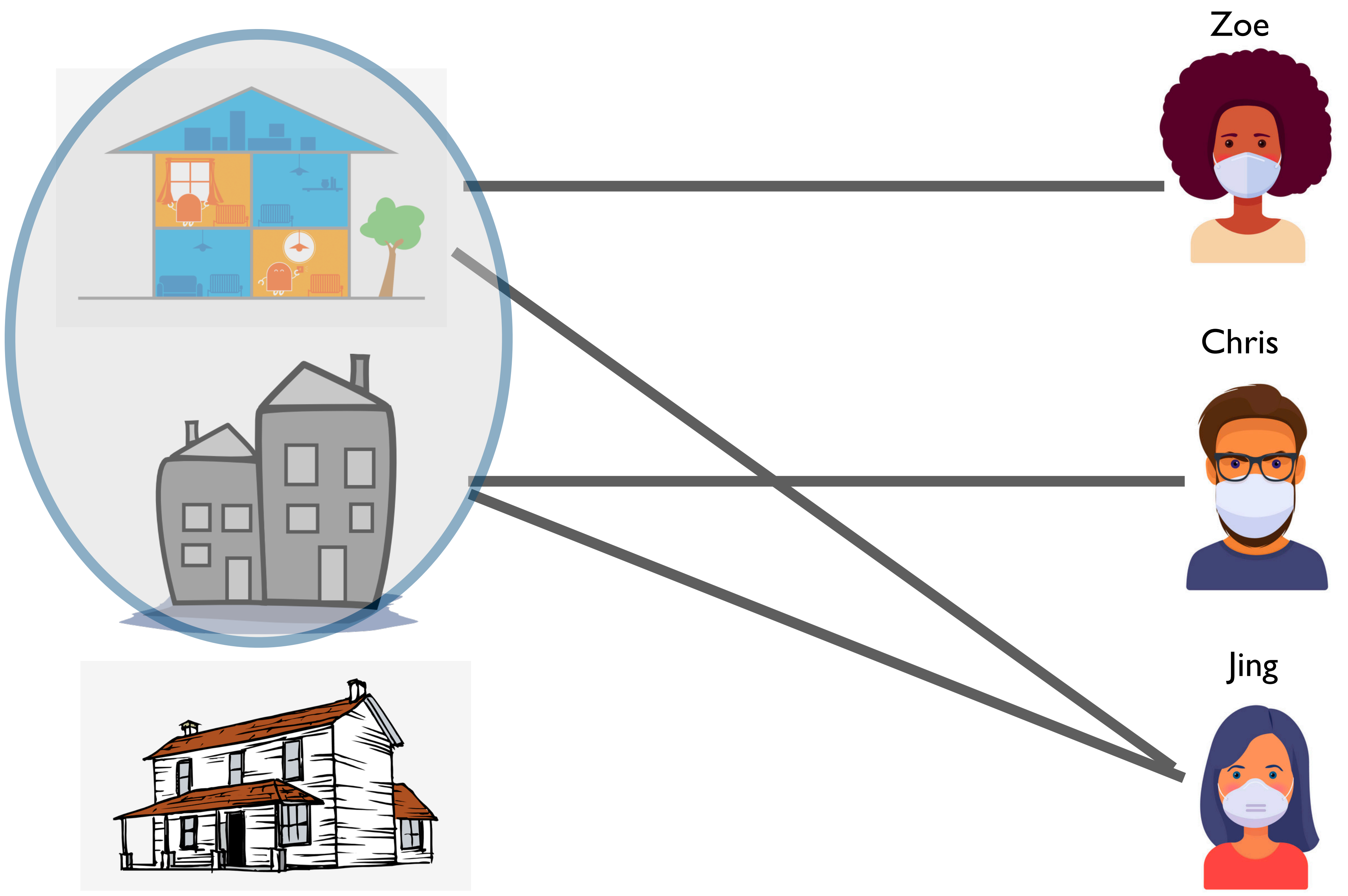
- Proved matching of size k iff flow of value k
- Thus, max-flow iff max matching
- Running time of algorithm overall:
  - Running time of reduction + running time of solving the flow problem (dominates)
- What is running time of Ford–Fulkerson algorithm for a flow network with all unit capacities?
  - $O(nm)$
- Overall running time of finding max-cardinality bipartite matching: $O(nm)$

Understanding Bipartite Matchings Better

Perfect Matchings

- Suppose we want a perfect matching: a matching that matches all vertices
- If the maximum matching produced by our algorithm has size $n = |A| = |B|$: we know we have a perfect matching
- Suppose the maximum matching produced is smaller
    - How can we give a certificate that shows it is impossible to find a perfect matching in such a graph?
- Let's think about **necessary** and **sufficient** conditions for a graph to have a perfect matching

Perfect Matching?



Aside: Necessary & Sufficient

Suppose $P \implies Q$ (If P, then Q)

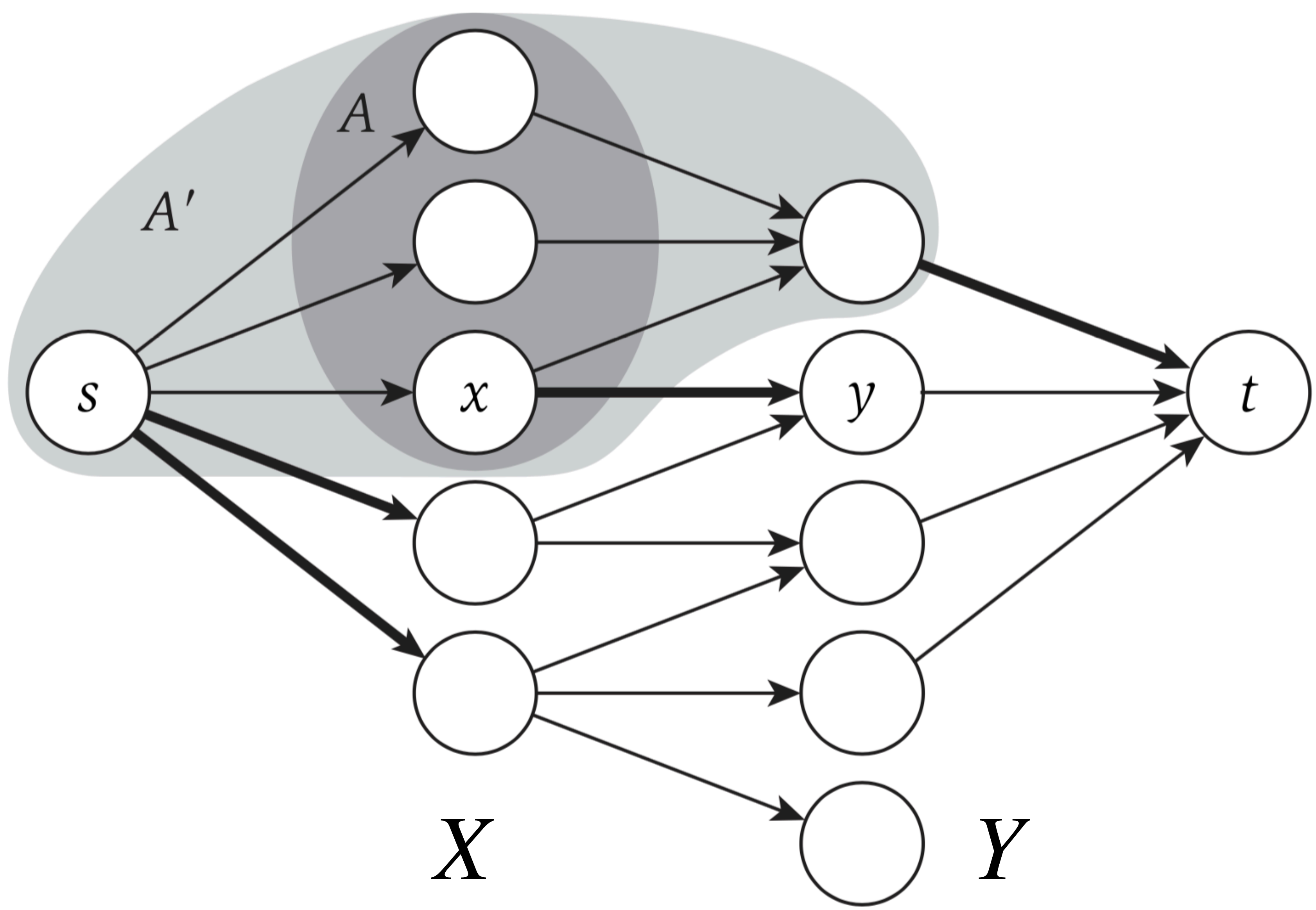
- We say Q is necessary for P
  - In other words, it is impossible to have P without Q
- Similarly, we say P is sufficient for Q
  - P being true, implies Q is true
  - But if P is false, we can't say anything about Q
- Necessary and sufficient conditions:
  - $P \iff Q$, if and only if characterization

Perfect Matchings

- **Notation**. Let S be a subset of nodes in G, and let $N(S)$ be the set of nodes adjacent to nodes in S in G.
- **[Halls marriage theorem.]** Let $G = (X \cup Y, E)$ be a bipartite graph with $|X| = |Y|$. Then, graph G has a perfect matching iff $|N(S)| \geq |S|$ for all subsets $S \subseteq X \cup Y$.
- Proof.
- ($\Rightarrow$). In a perfect matching, each node in S needs to be matched with a different node in $N(S)$
- ($\Leftarrow$). Suppose G does not have a perfect matching
- In our max-flow problem, this means max flow is less than n

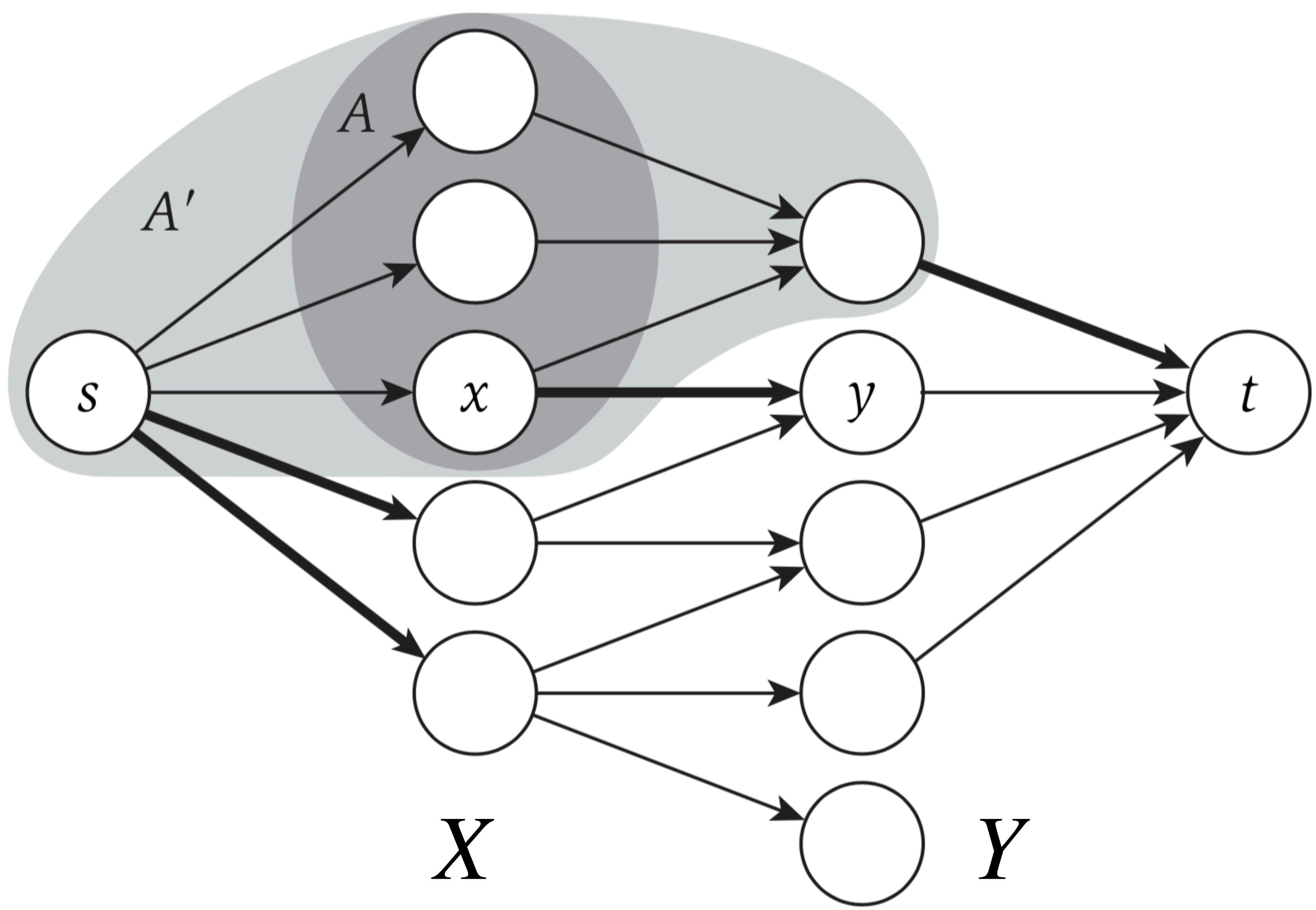
Perfect Matchings & Cuts

- ($\Leftarrow$). Suppose G does not have a perfect matching
- The capacity of the min-cut (A', B') is less than n
- A' may contain nodes from both X and Y



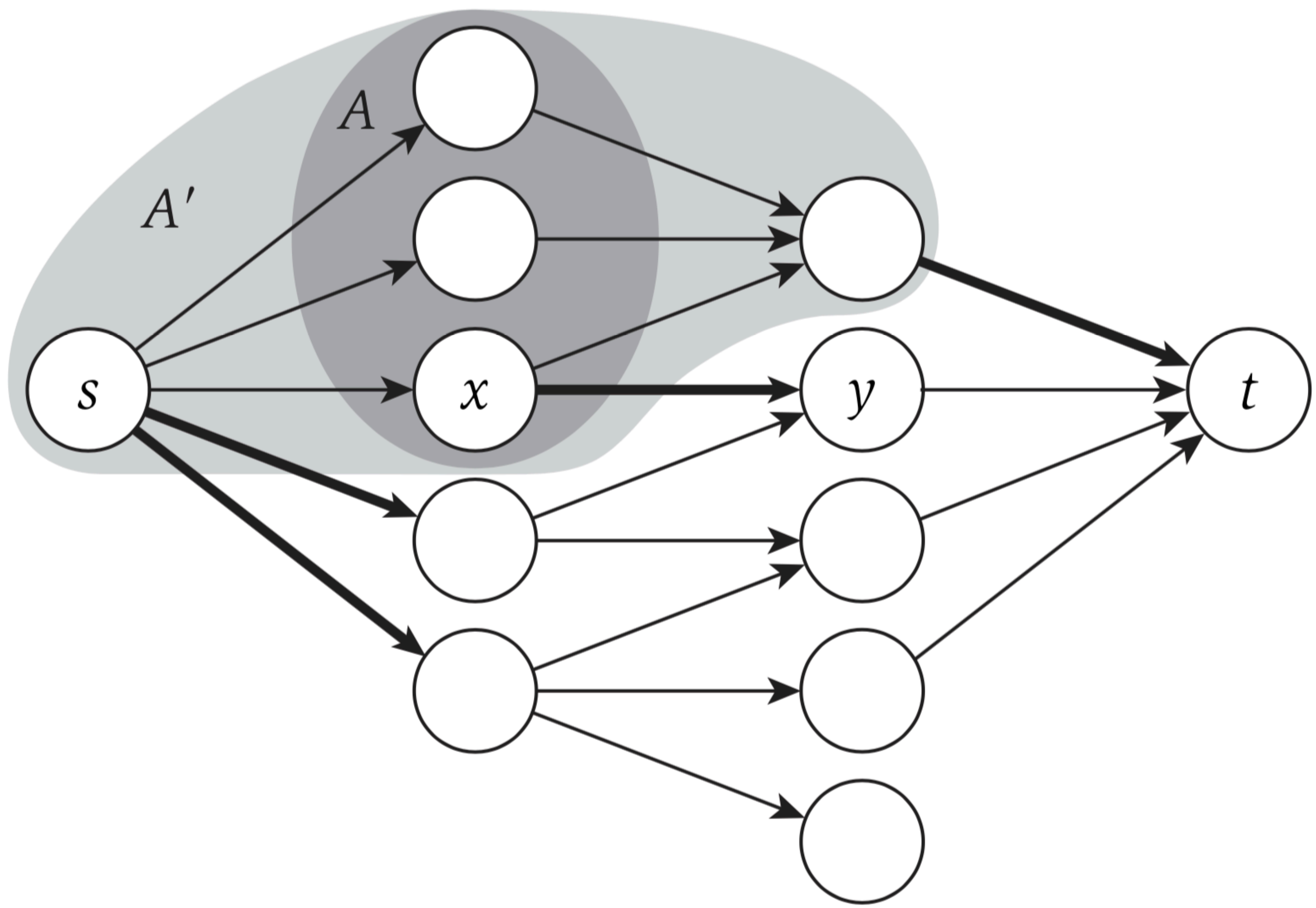
Perfect Matchings & Cuts

- A' may contain nodes from both X and Y
    - We want a subset A, s.t. $|N(A)| < |A|$
- Claim. $A = X \cap A'$ has this property.



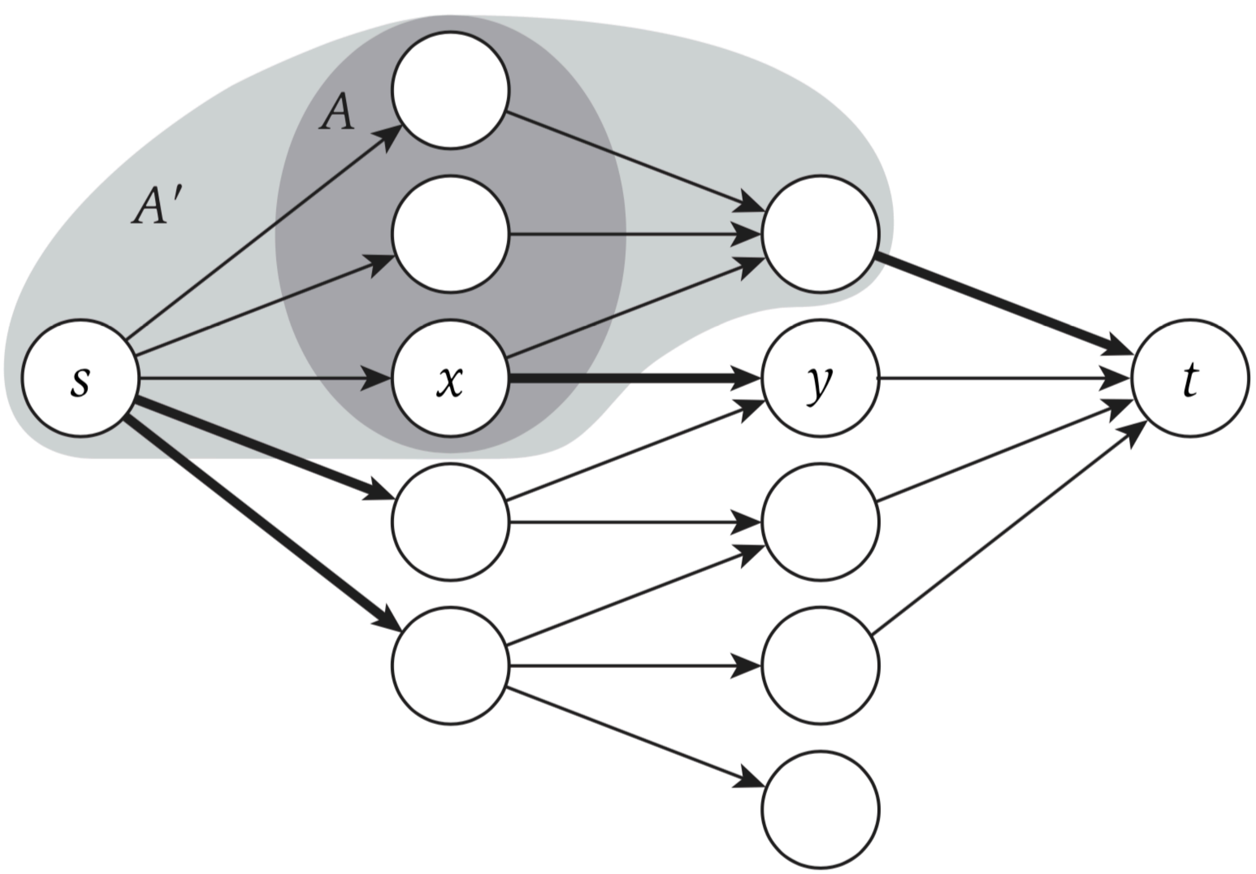
Perfect Matchings & Cuts

- Claim. $A = X \cap A'$ has the property that $|N(A)| < |A|$

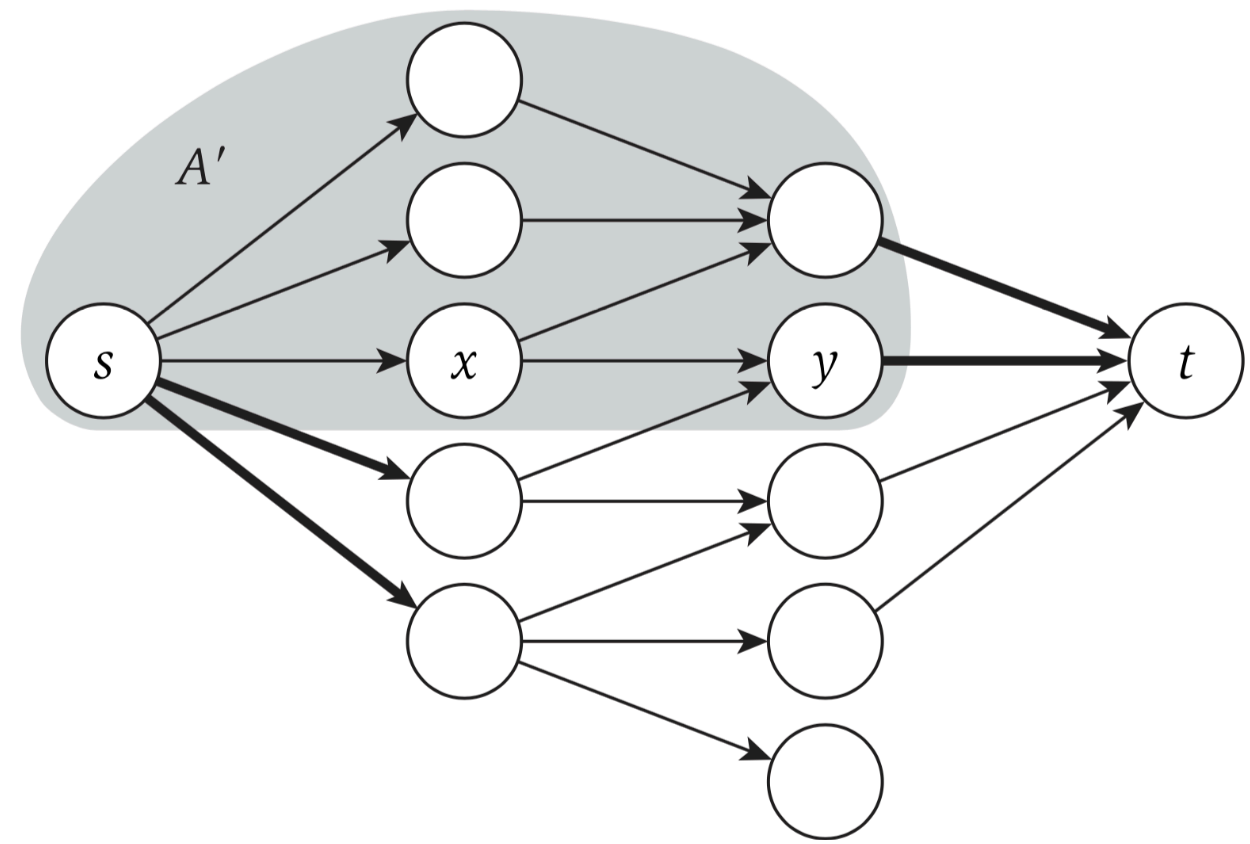


Perfect Matchings & Cuts

- Cases: $N(A) \subseteq A'$ or $N(A) \subsetneq A'$



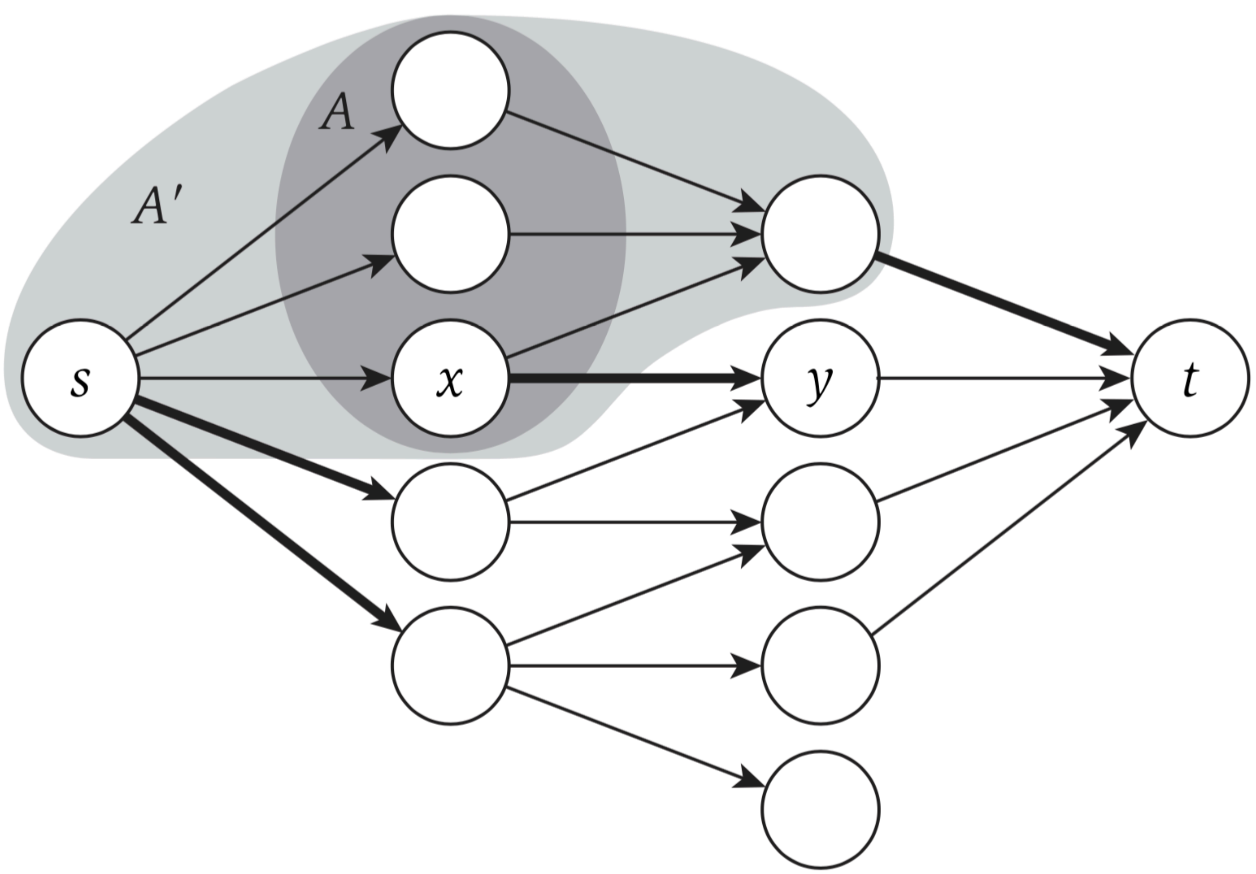
$N(A) \subsetneq A'$



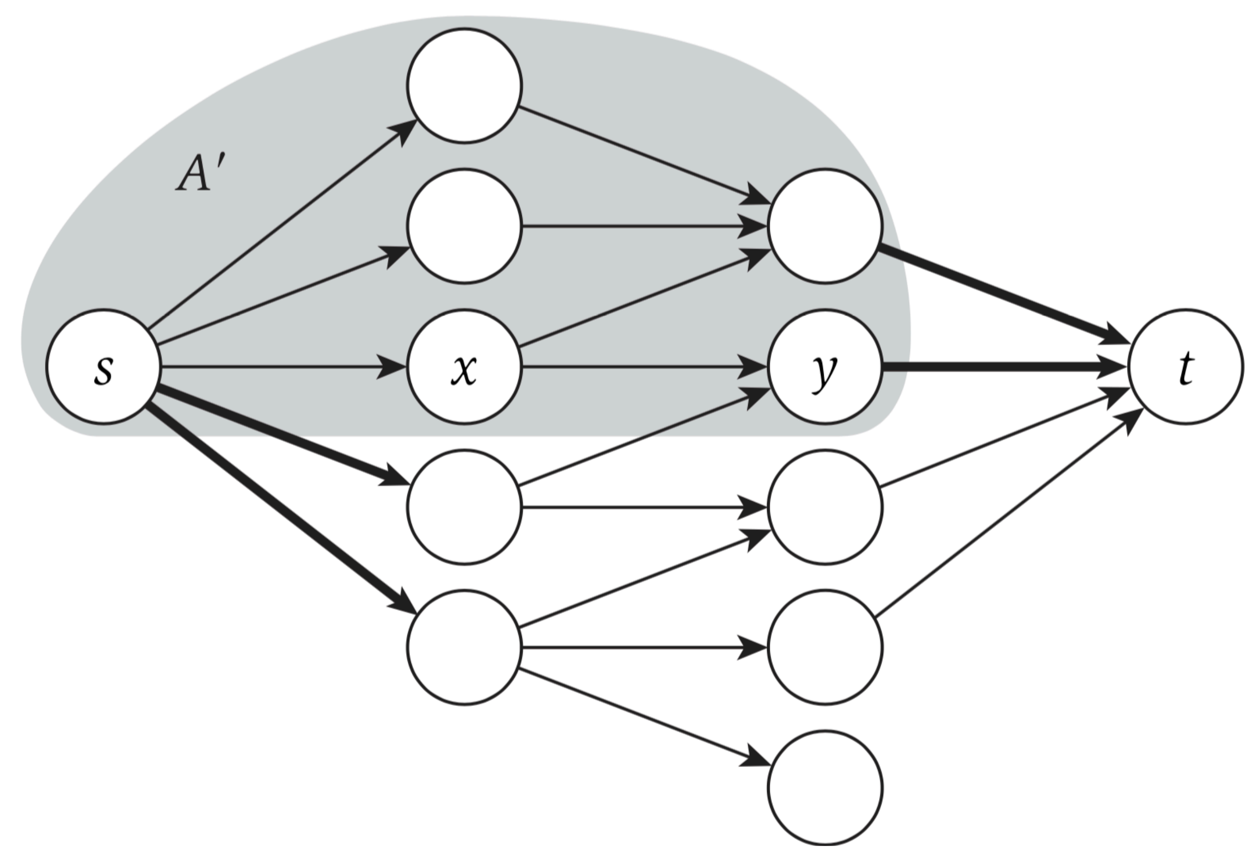
$N(A) \subseteq A'$

Perfect Matchings & Cuts

- We will show, if a mincut (A', B') doesn't have the property that $N(A) \subseteq A'$. we can find a new cut that does, that is, wlog we can assume $N(A) \subseteq A'$,   where $A = X \cap A'$



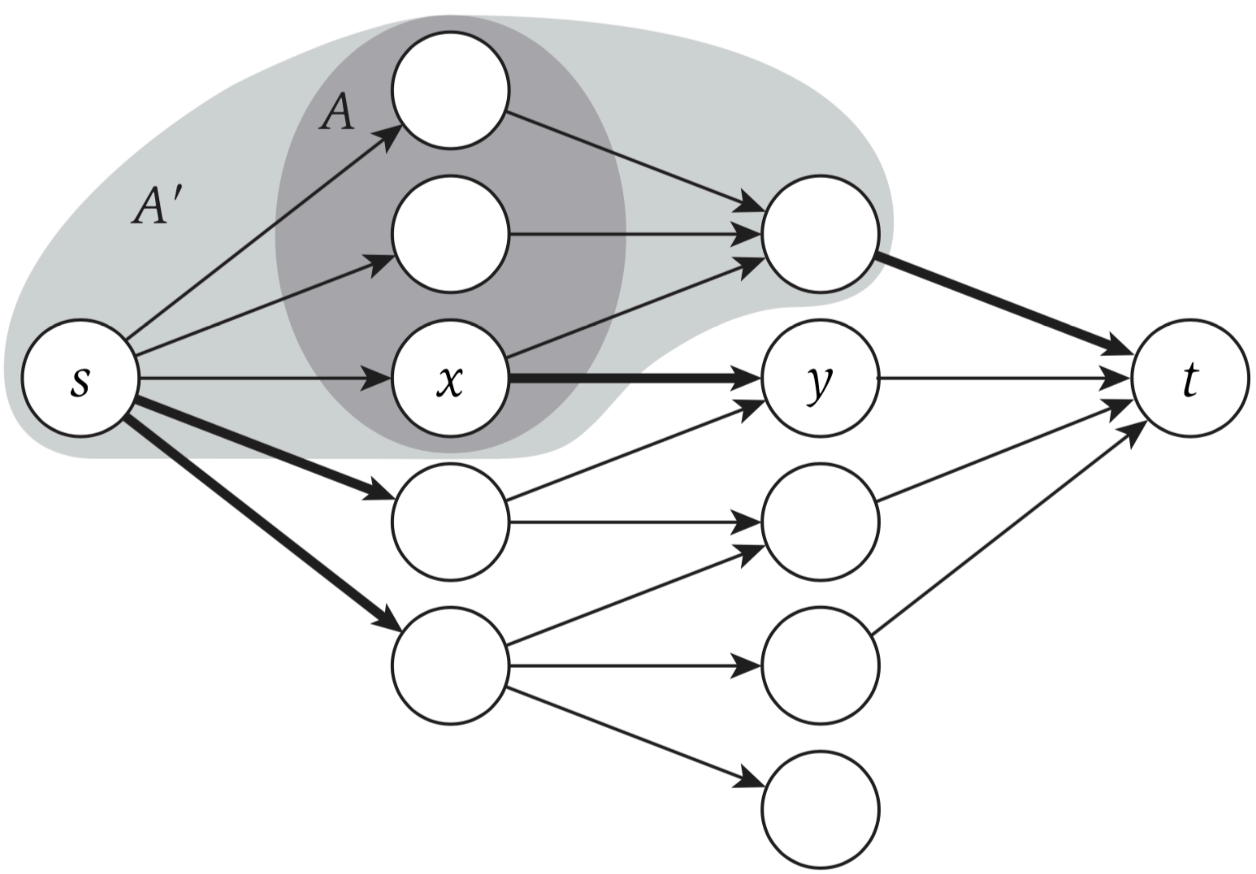
$N(A) \subsetneq A'$



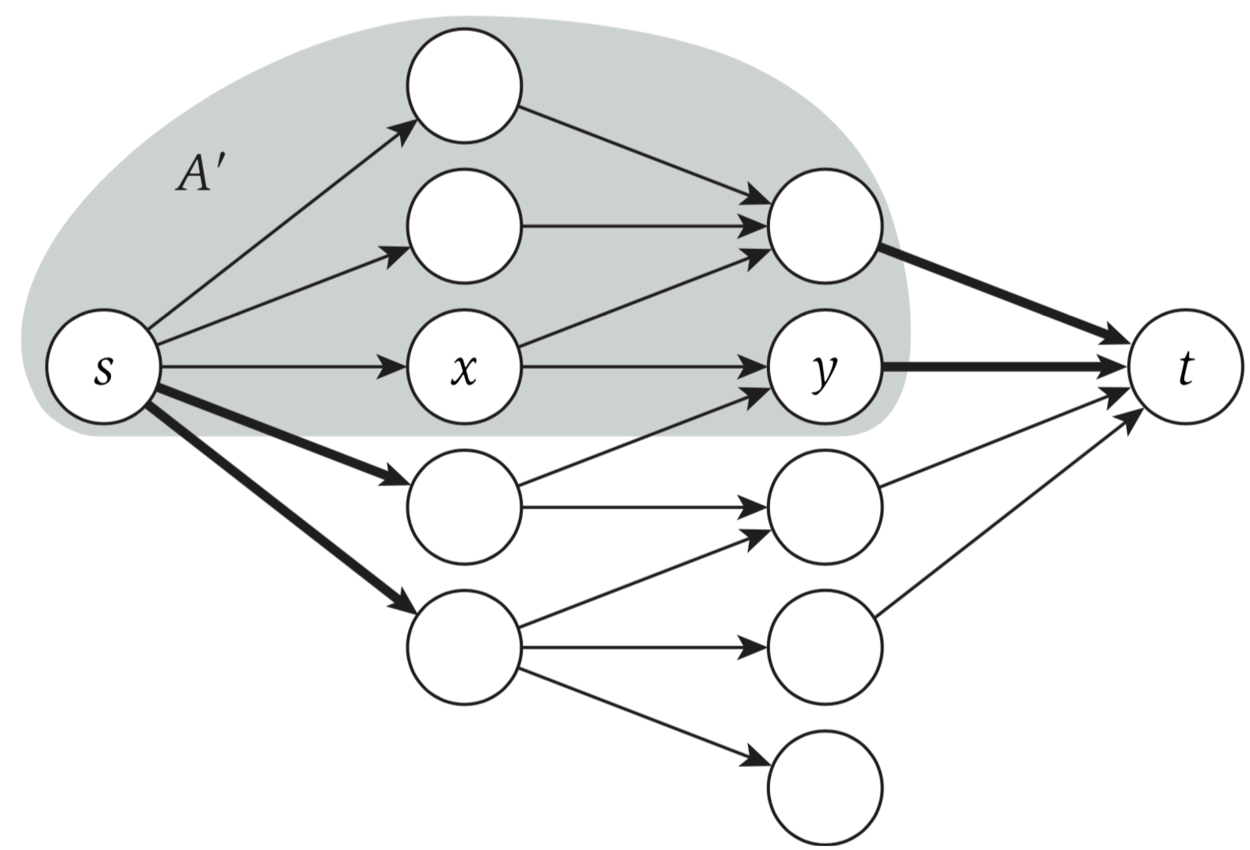
$N(A) \subseteq A'$

Perfect Matchings & Cuts

- Pick an edge (x, y) s.t. $x \in A$ and $y \notin A'$
- **Claim**: moving y to A' doesn't increase capacity of the cut



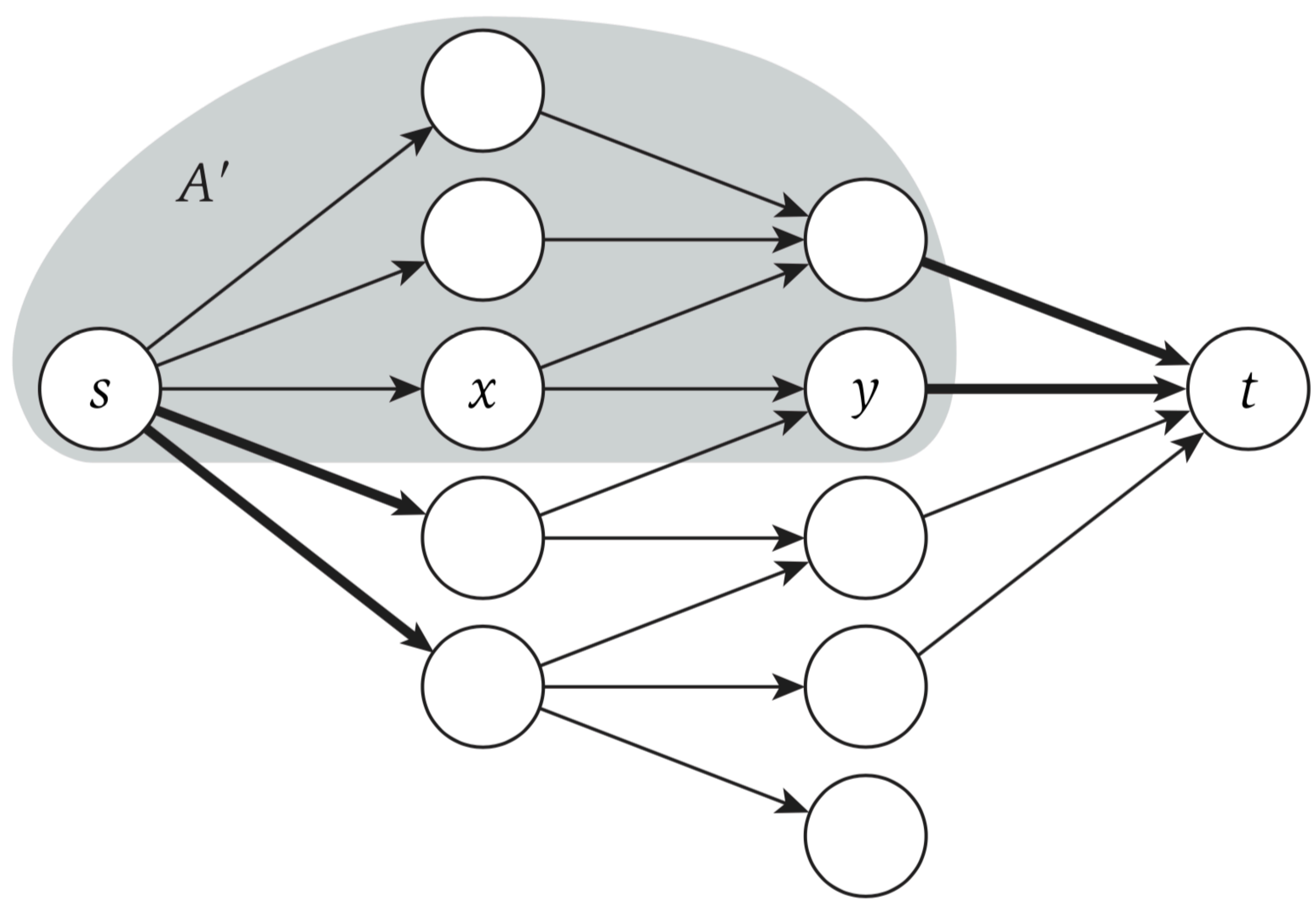
$N(A) \subsetneq A'$



$N(A) \subseteq A'$

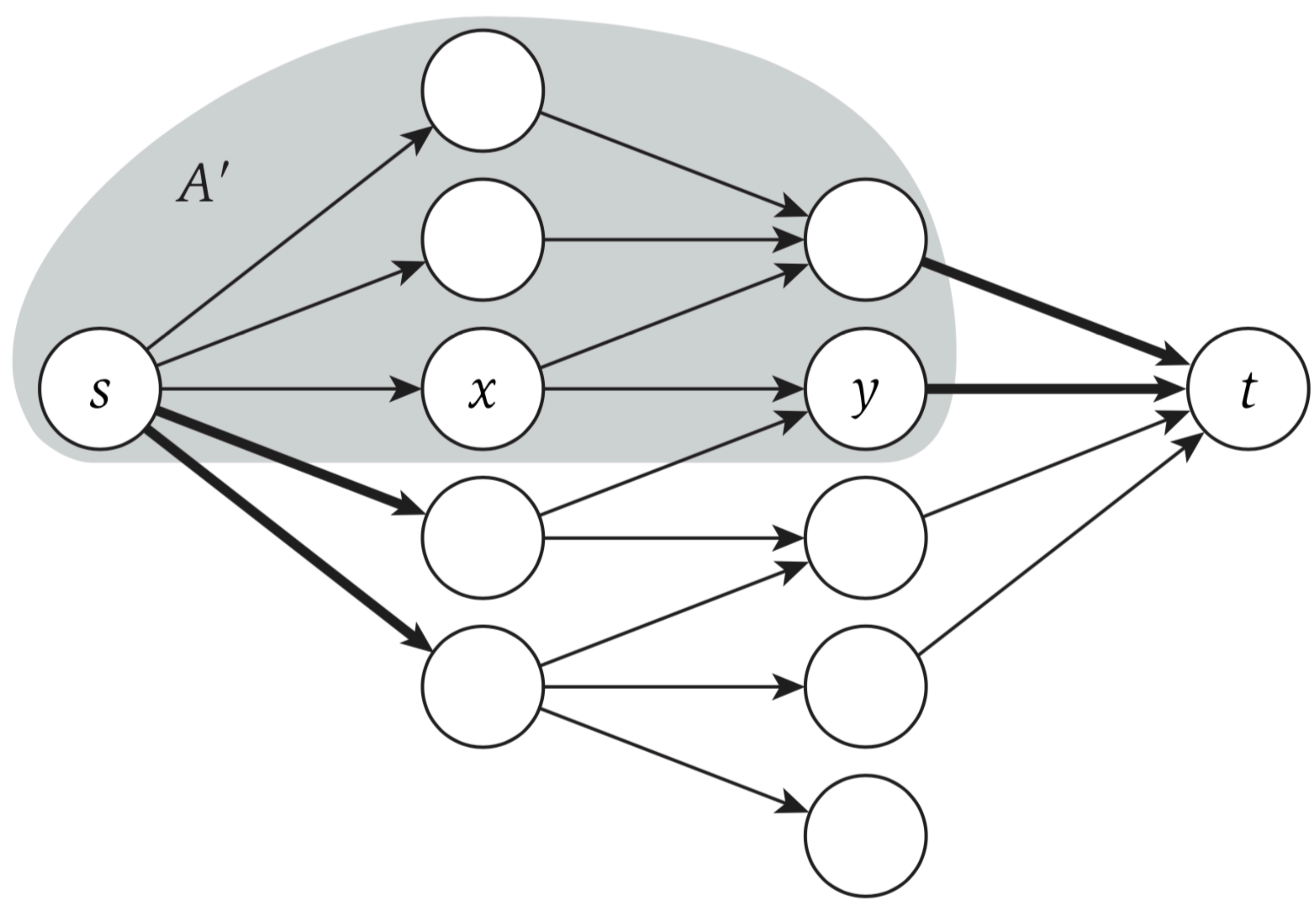
Perfect Matchings & Cuts

- Now wlog, assume $N(A) \subseteq A'$
- $c(A', B') =$ # edges that leave the source to nodes in X outside A' + # edges from nodes in Y and A' to the sink



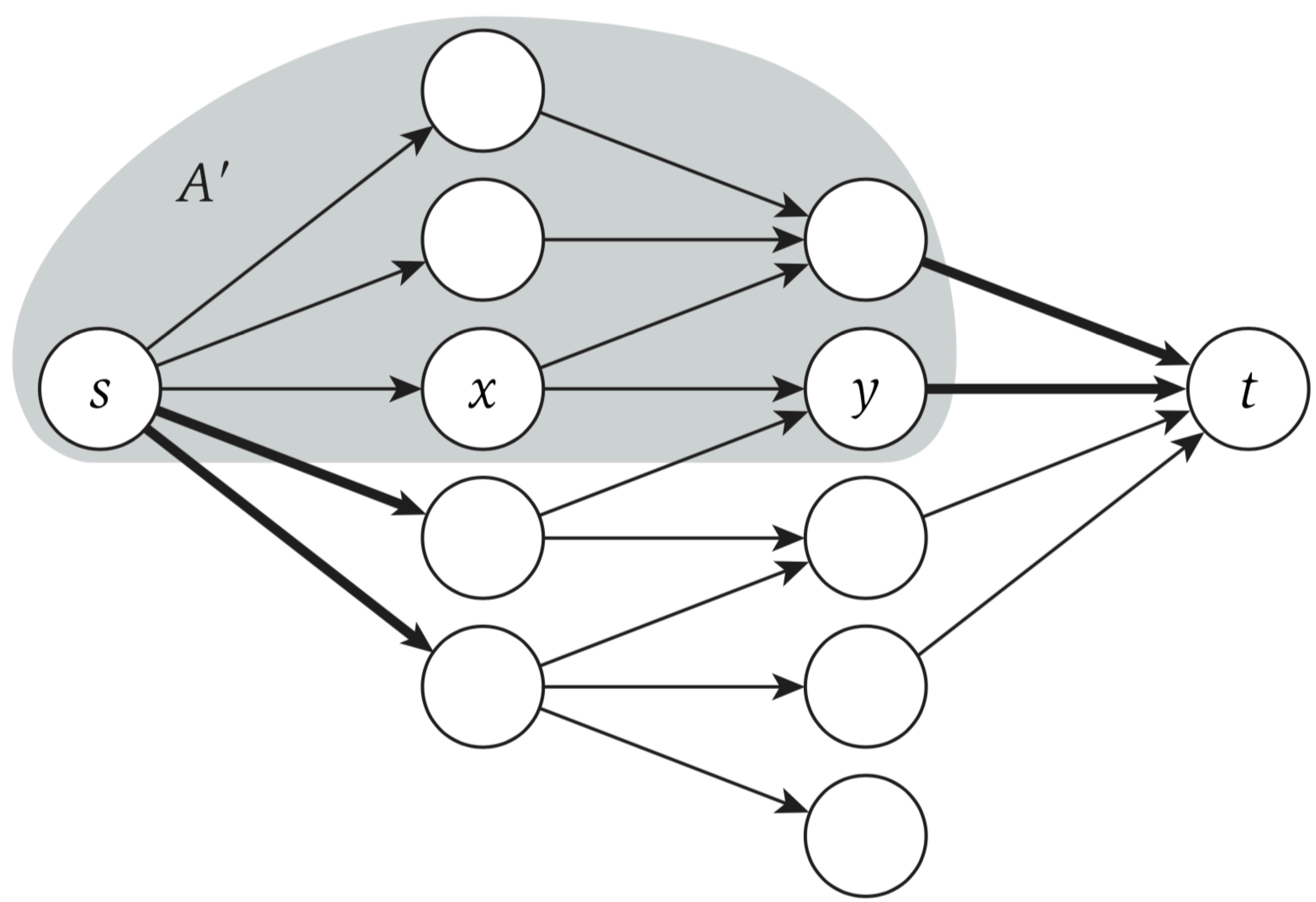
Perfect Matchings & Cuts

- Now wlog, assume $N(A) \subseteq A'$
- $c(A', B') = \ |X \cap B'| + |Y \cap A'|$
- Now, $|X \cap B'| = n - |A|$ and $|Y \cap A'| \geq |N(A)|$



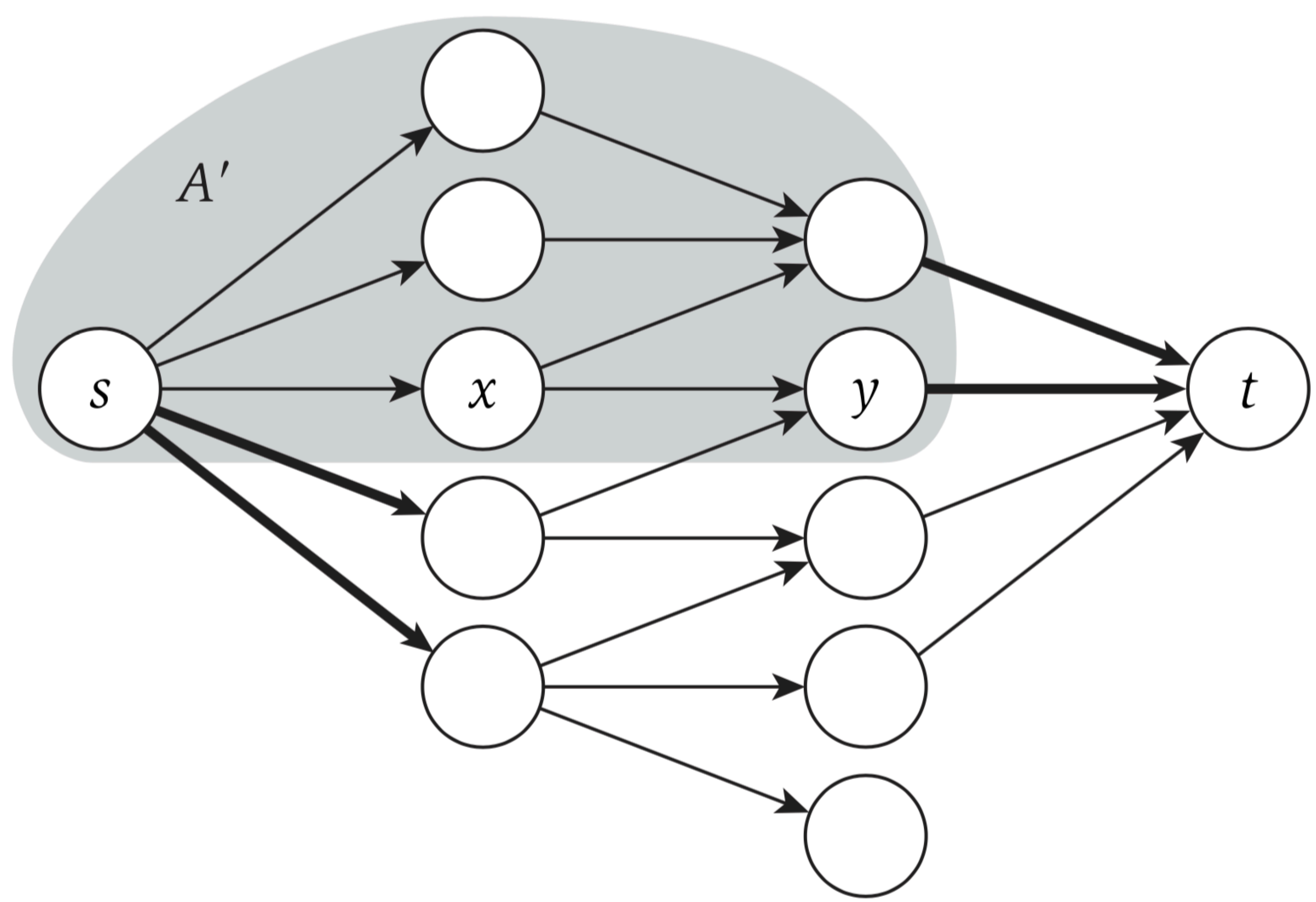
Perfect Matchings & Cuts

- $c(A', B') = \ |X \cap B'| + |Y \cap A'|$
- $n - |A| + |N(A)| \leq c(A', B')$



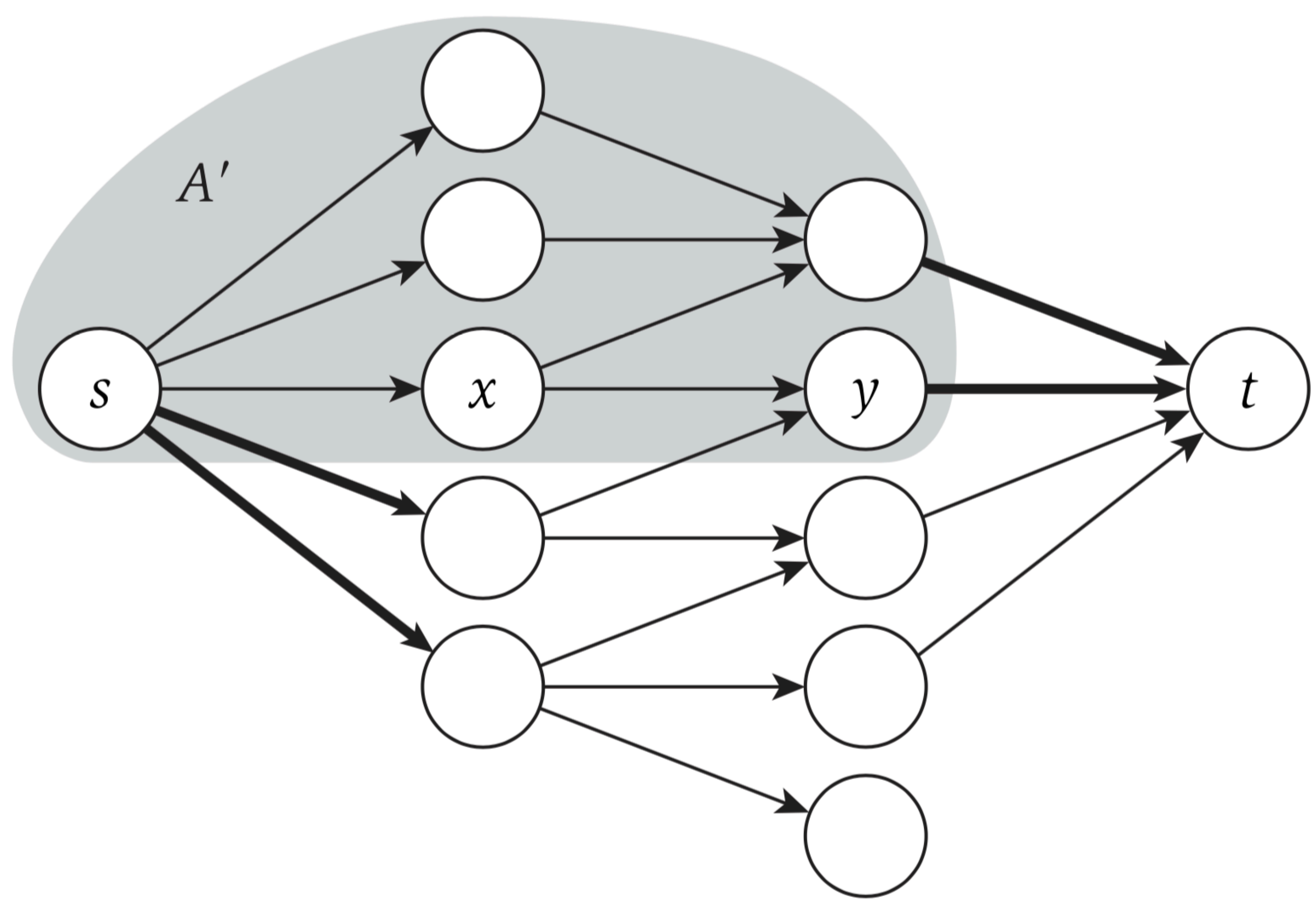
Perfect Matchings & Cuts

- $c(A', B') = \ |X \cap B'| + \ |Y \cap A'|$
- $n - |A| + |N(A)| \leq c(A', B') < n$



Perfect Matchings & Cuts

- $n - |A| + |N(A)| \leq c(A', B') < n$
- $|N(A)| < |A|$ ∎



Summary: Flows and Matching

- We have proved Hall's theorem using network flows!

- **[Halls marriage theorem.]** Let $G = (X \cup Y, E)$ be a bipartite graph with $|X| = |Y|$. Then, graph G has a perfect matching iff $|N(S)| \geq |S|$ for all subsets $S \subseteq X \cup Y$.

- If G has a perfect matching, we can find one using flow!

- If G doesn't have a perfect matching, we can find a certificate for this: a subset of nodes that violate Hall's condition!

- **Takeaway**. Algorithms can be useful in proving purely combinatorial math theorems!

Next Class:

More Reductions/
Applications of Flow

Acknowledgments

- Some of the material in these slides are taken from
  - Kleinberg Tardos Slides by Kevin Wayne (https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf)
  - Jeff Erickson's Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)