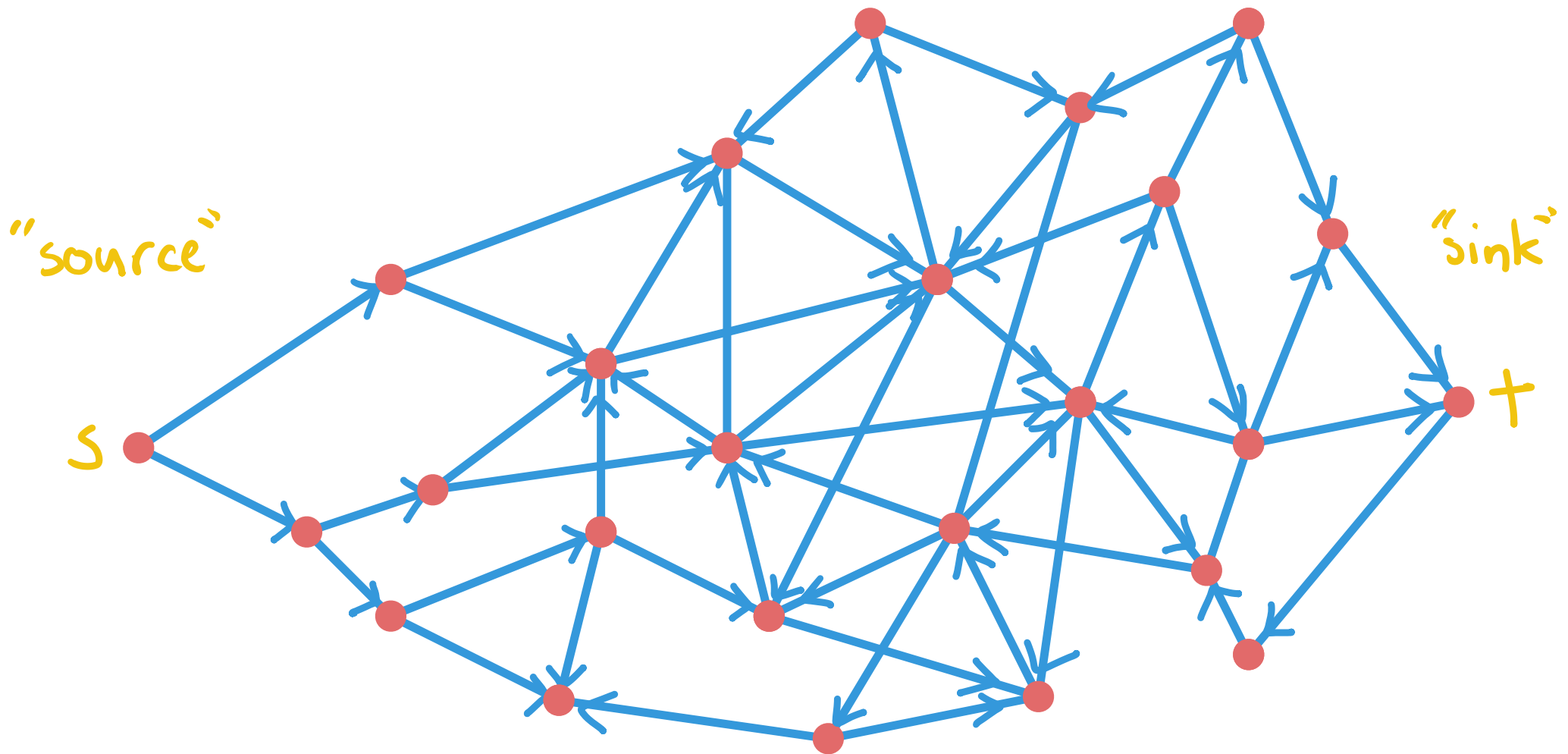


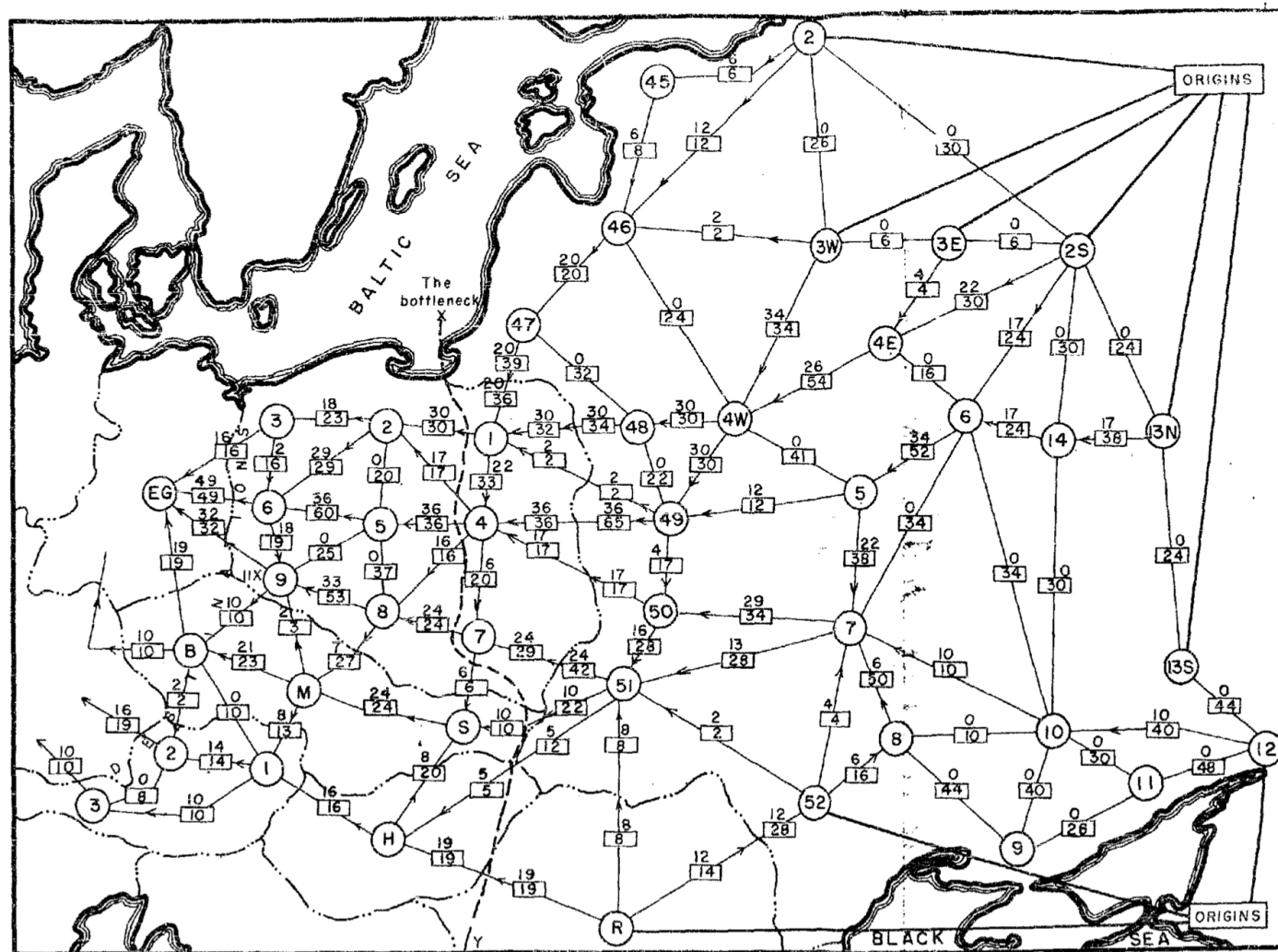
Applications of Flows + Cuts

Measuring connectivity



how well connected are s and t?

Physical infrastructure



SECRET
RM-1373
10-24-55
-33-

Fig. 7 — Traffic pattern: entire network available

Legend:

— International boundary

(B) Railway operating division

← 9/12 → Capacity: 12 each way per day. Required flow of 9 per day toward destinations (in direction of arrow) with equivalent number of returning trains in opposite direction

All capacities in $\sqrt{1000}$'s of tons each way per day

Origins: Divisions 2, 3W, 3E, 2S, 13N, 13S, 12, 52 (USSR), and Roumania

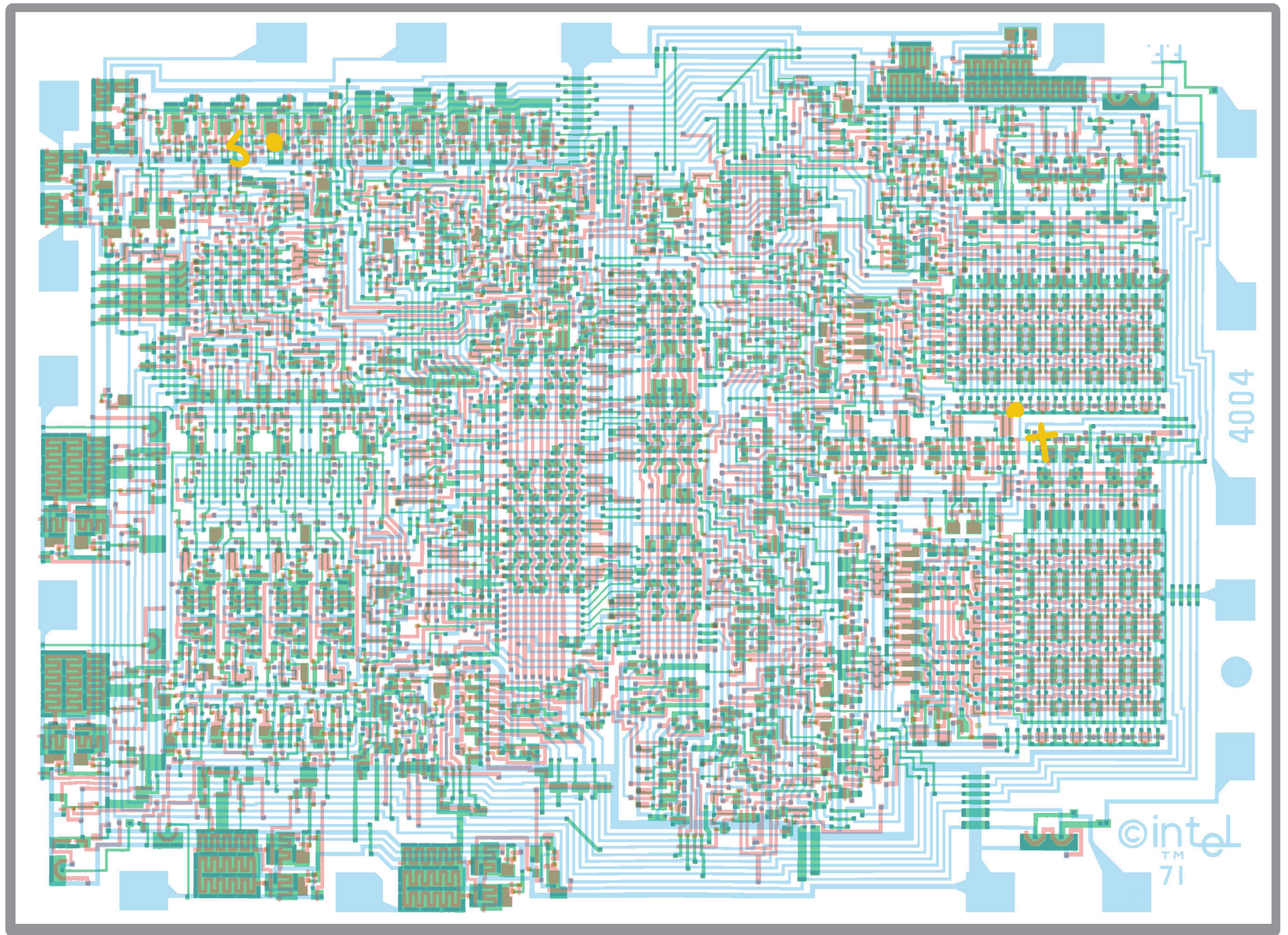
Destinations: Divisions 3, 6, 9 (Poland); B (Czechoslovakia); and 2, 3 (Austria)

Alternative destinations: Germany or East Germany

Note IIX of Division 9, Poland

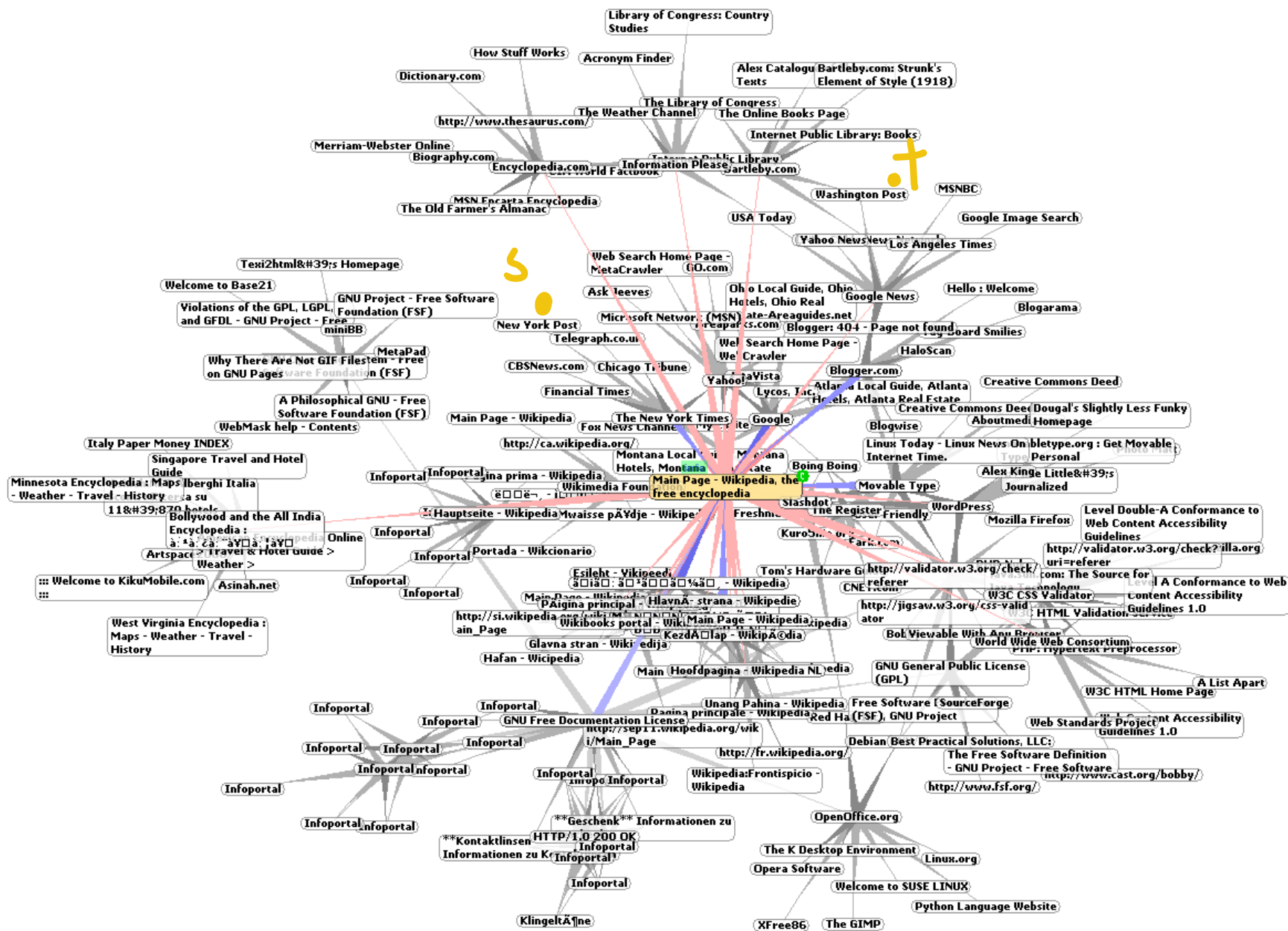
(from Jeff's notes)

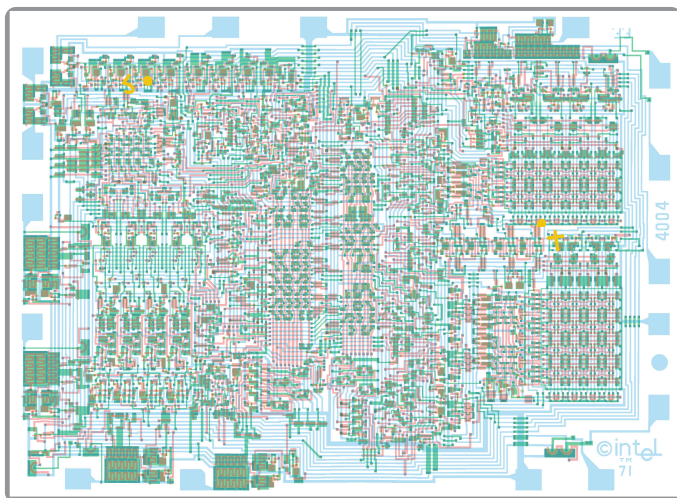
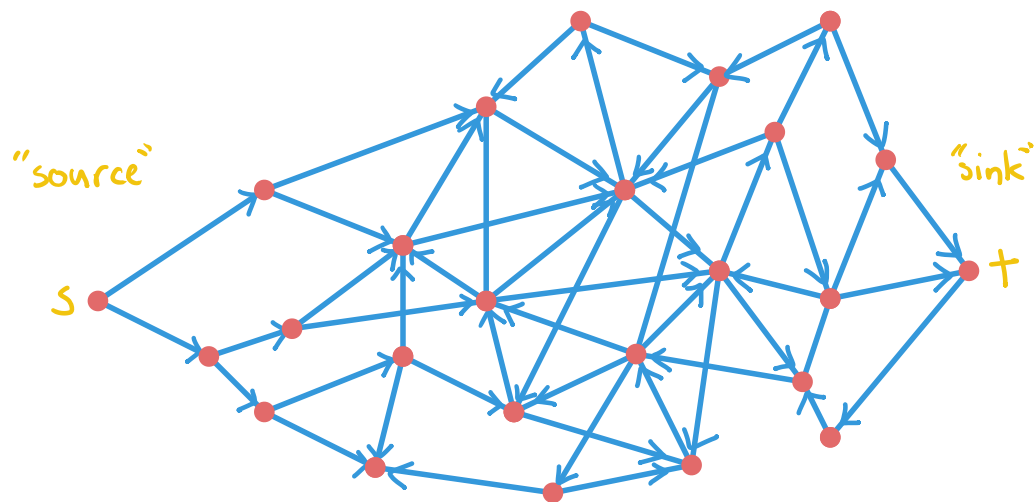
CPU's



social network

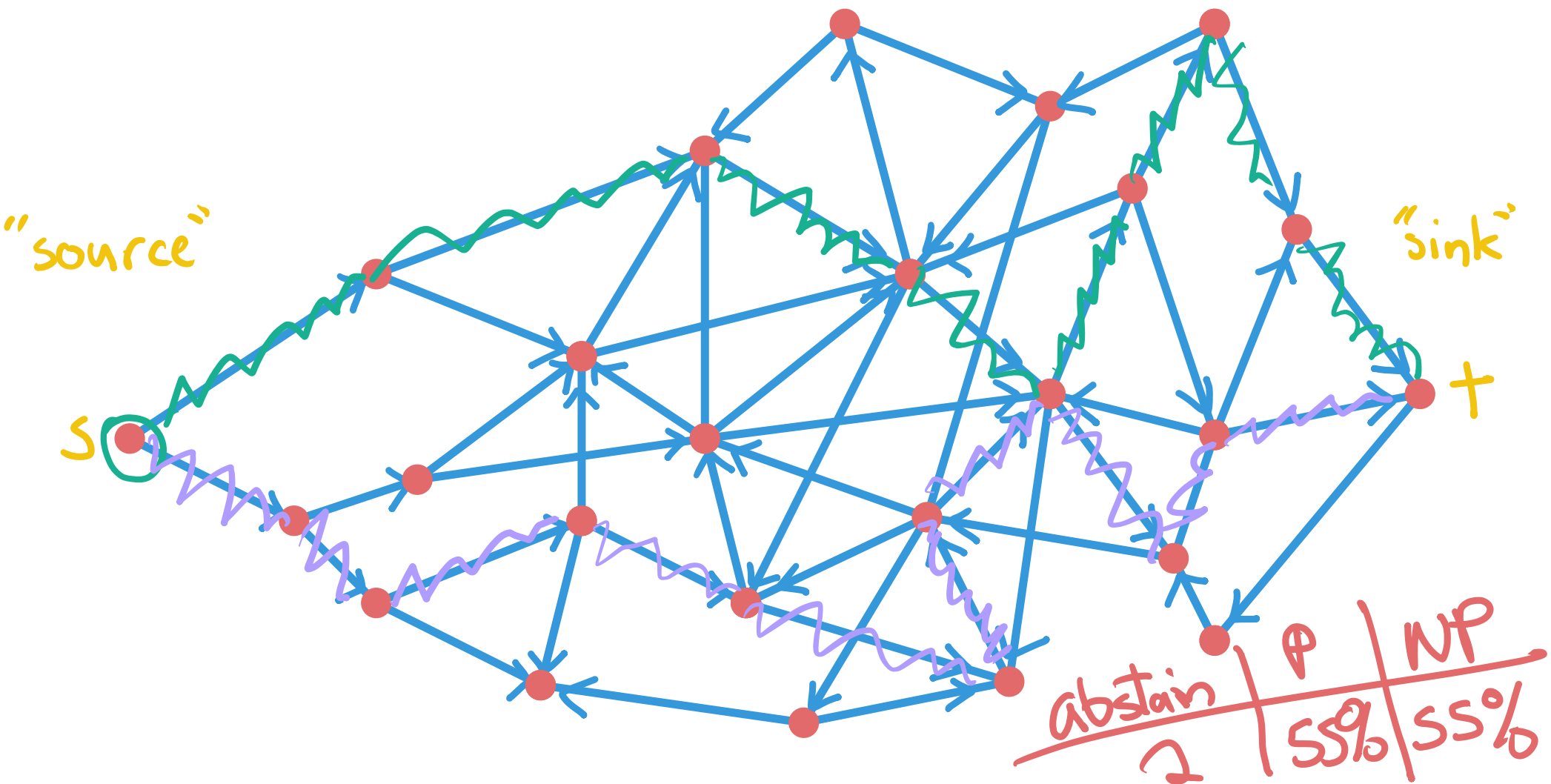






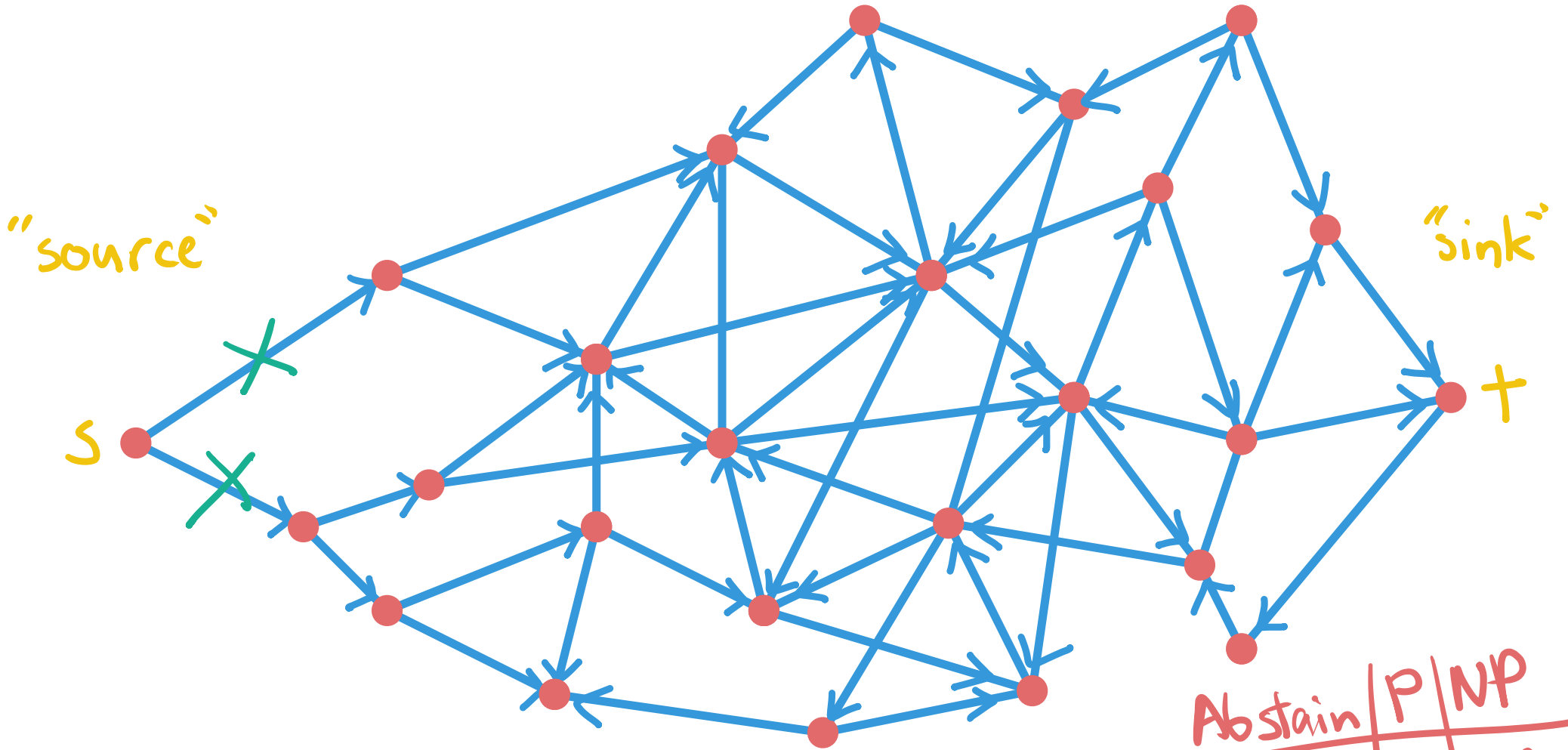
- ① Is there an $s \rightsquigarrow t$ path?
- ② How many distinct $s \rightsquigarrow t$ paths?
- ③ how many k -edge $s \rightsquigarrow t$ walks?
(k fixed)
- ④ length of shortest $s \rightsquigarrow t$ path?
- ⑤ how many shortest $s \rightsquigarrow t$ paths?
- ⑥ In a random walk starting from s , what is the expected # of steps until we reach t ?
- ⑦ max # of edge-disjoint $s \rightsquigarrow t$ paths?
- ⑧ min # of edge deletions to disconnect $s \rightsquigarrow t$?

Measuring connectivity



⑦ max # of edge-disjoint $s \leadsto t$ paths?

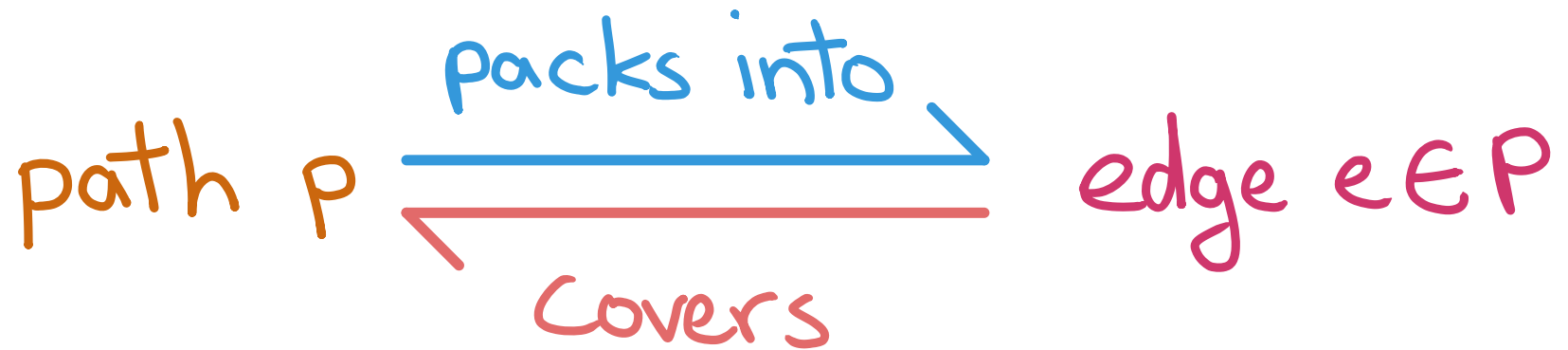
Measuring connectivity

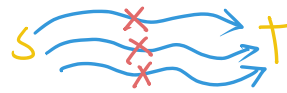


⑧ min # of edge deletions to disconnect $s \rightarrow t$?

Abstain	P	NP
2	45%	55%

Duality





Duality
 path p $\xrightarrow{\text{packs into}}$ edge $e \in P$
 $\xrightarrow{\text{covers}}$

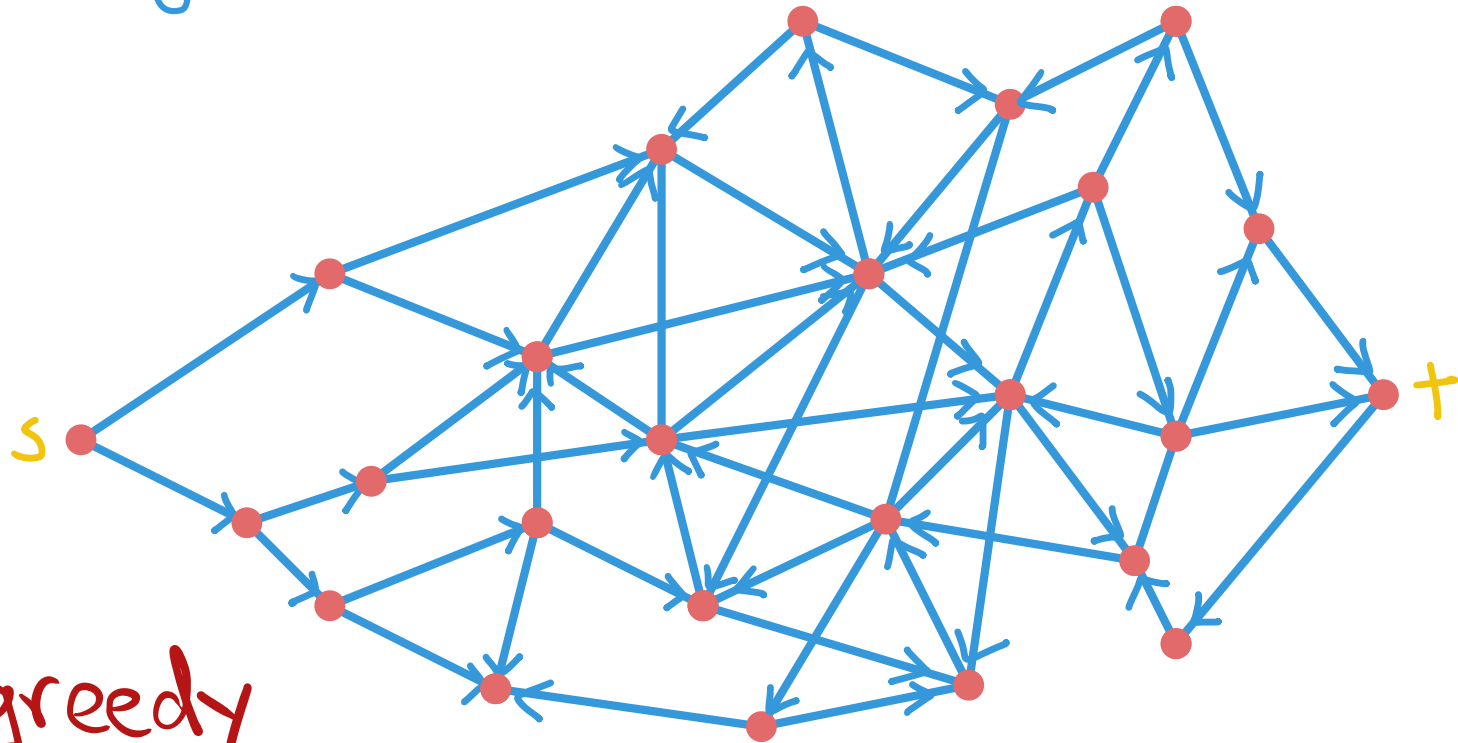
Lemma

$$\left(\begin{array}{c} \text{max \#} \\ \text{disjoint} \\ s \rightsquigarrow t \text{ paths} \end{array} \right) \leq \left(\begin{array}{c} \text{min \# edges} \\ \text{in any } s \rightsquigarrow t \text{ cut} \end{array} \right)$$

* 2nd example of
 packing $\leq$ covering

How to compute

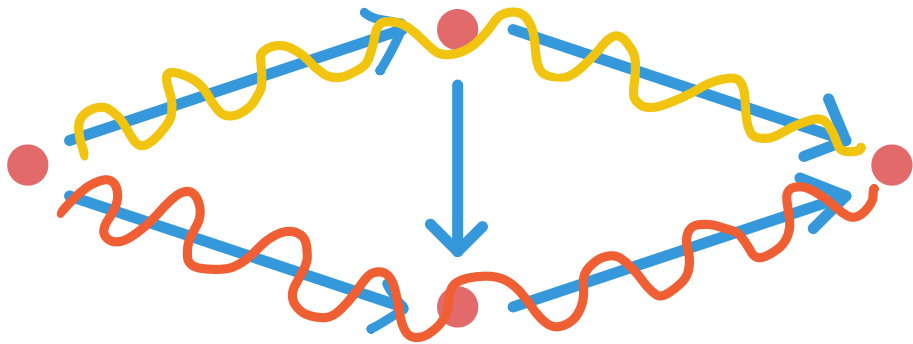
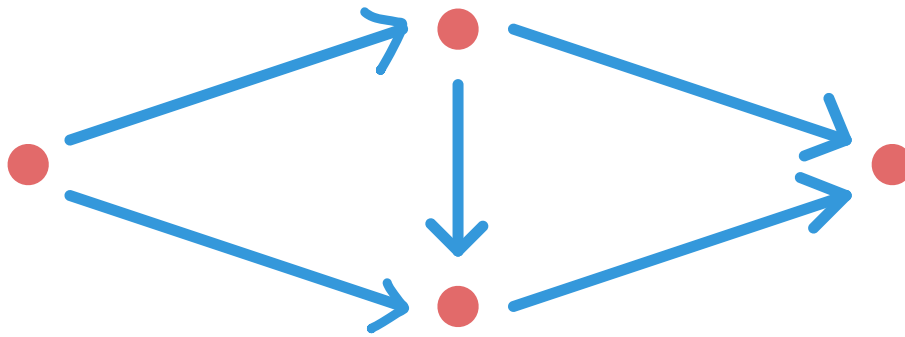
⑦ max # of edge-disjoint $s \rightsquigarrow t$ paths?



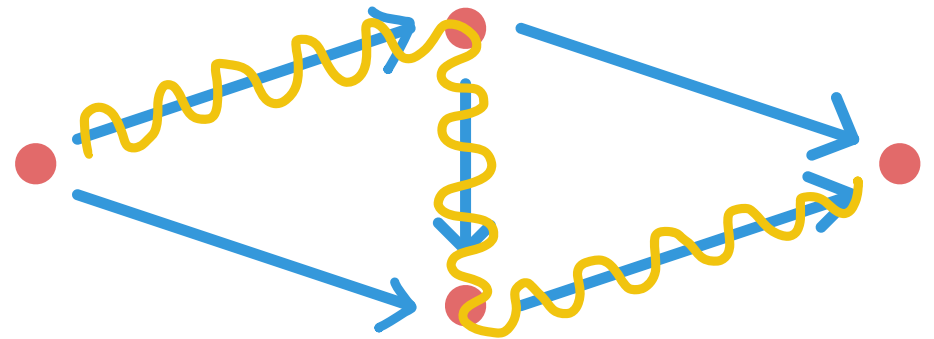
Iterative/greedy

① while there is a path $p: s \rightsquigarrow t$

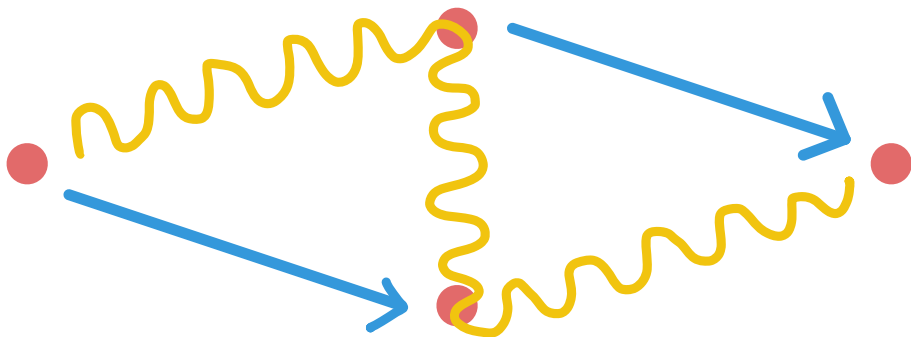
② take p and remove all its edges from the graph



(good)

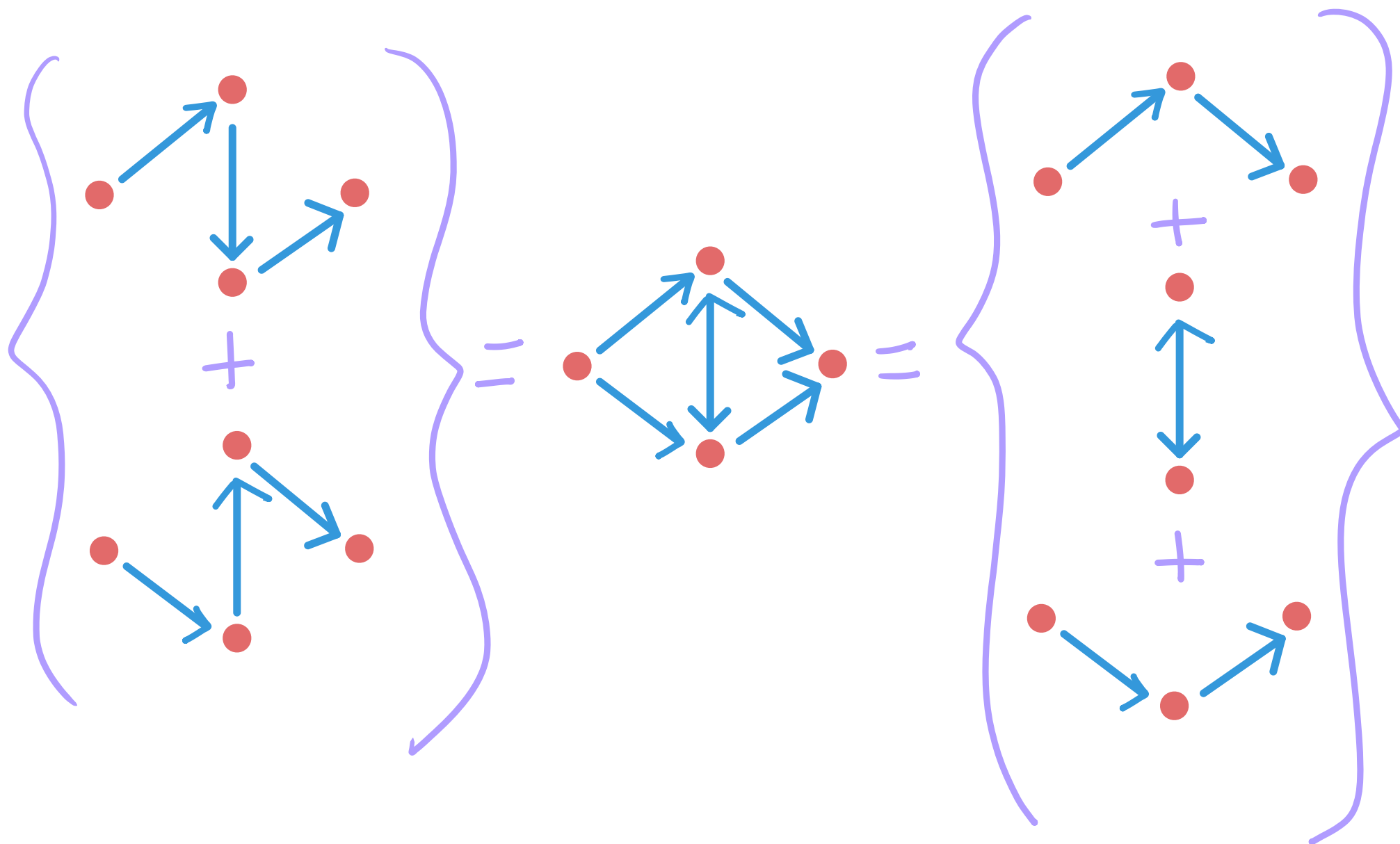


(bad)

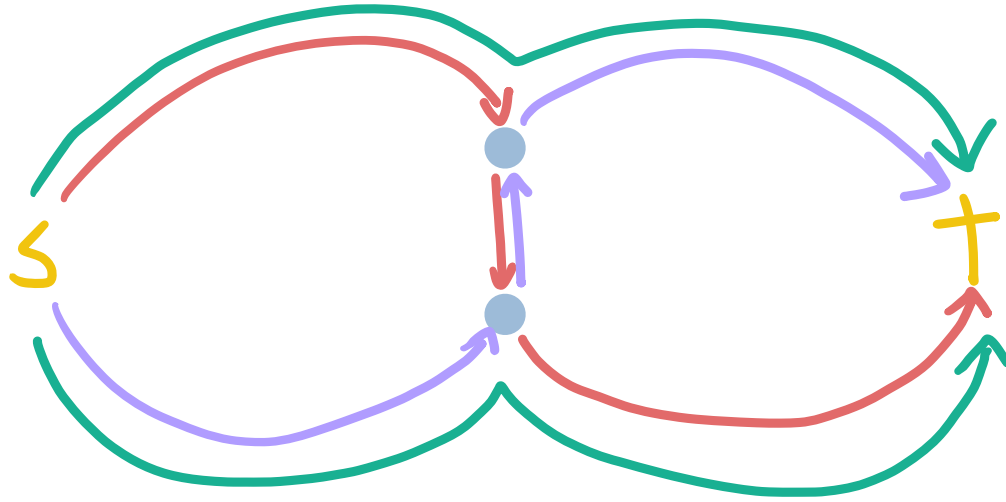


- ① while there is a path $p: s \rightsquigarrow t$
 - @ take p and remove all its edges from the graph

Consider:



Lemma let p, q be edge-disjoint $s \rightsquigarrow t$ paths



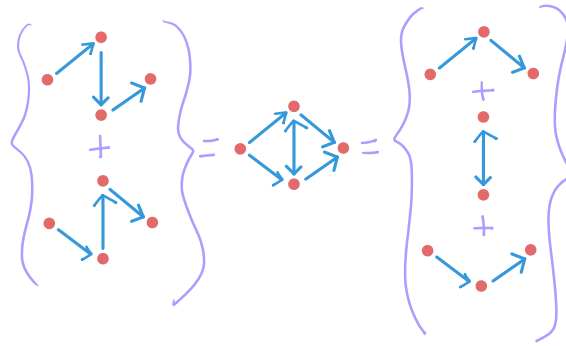
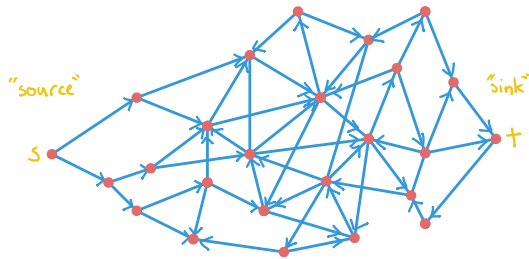
then there is p', q' : $s \rightsquigarrow t$ disjoint
without opposite pairs of edges

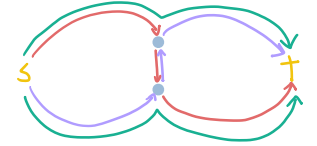
Proof: try it!

hint: by induction on # of opposite pairs

How to compute

⑦ max # of edge-disjoint $s \rightsquigarrow t$ paths?



Lemma let p, q be edge-disjoint $s \rightsquigarrow t$ paths

then there is $p', q' : s \rightsquigarrow t$ disjoint
without opposite pairs of edges

augmenting-paths($G = (V, E), s, t$)

/ Ford and Fulkerson [FF56]*

1. $P \leftarrow \emptyset$

2. While there is an (s, t) -path $p : s \rightsquigarrow t$:

A. Remove all of p 's edges from the graph.

B. Add the reverse of all of p 's edges to the graph.

C. Uncross p with previously selected paths, and add p to P .

"residual graph"



④ let

S = vertices reachable from s in residual graph

T = all other vertices (incl. t)

C = edges from S to T

⑤ each path decrease $|C|$ by 1.

⑥ at end, $|C| = 0$.

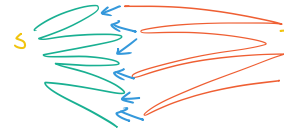
$$\Rightarrow \left(\begin{array}{c} \# \text{ paths} \\ \text{packed} \end{array} \right) = \left(\begin{array}{c} \# \text{ edges} \\ \text{initially in } C \end{array} \right)$$

Recall Lemma

$$\left(\begin{array}{c} \max \# \\ \text{disjoint} \\ s \rightsquigarrow t \text{ paths} \end{array} \right) \leq \left(\begin{array}{c} \min \# \\ \text{edges in any} \\ s \rightsquigarrow t \text{ cut} \end{array} \right)$$

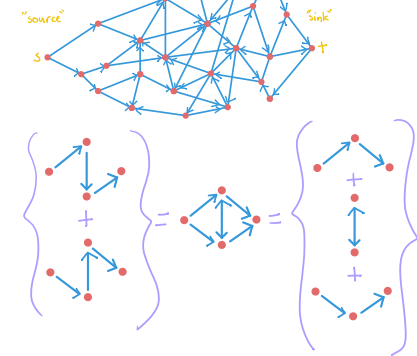
$$\left(\begin{array}{c} \# \text{ paths} \\ \text{packed} \end{array} \right) \leq \left(\begin{array}{c} \max \# \\ \text{disjoint} \\ s \rightsquigarrow t \text{ paths} \end{array} \right) \leq \left(\begin{array}{c} \min \# \\ \text{edges in any} \\ s \rightsquigarrow t \text{ cut} \end{array} \right) \leq \left(\begin{array}{c} \# \text{ edges} \\ \text{initially in } C \end{array} \right)$$

equal

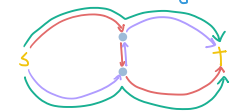


How to compute

⑦ max # of edge-disjoint $s \rightsquigarrow t$ paths?



Lemma let p, q be edge-disjoint $s \rightsquigarrow t$ paths



then there is p', q' $s \rightsquigarrow t$ disjoint without opposite pairs of edges

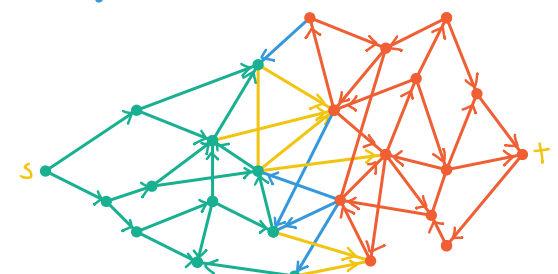
augmenting-paths($G = (V, E), s, t$)

/* Ford and Fulkerson [FF56]

1. $P \leftarrow \emptyset$
2. While there is an (s, t) -path $p : s \rightsquigarrow t$:
 - A. Remove all of p 's edges from the graph.
 - B. Add the reverse of all of p 's edges to the graph.
 - C. Uncross p with previously selected paths, and add p to P .

let (S, I) partition V st. $s \in S, t \in I$

C = edges from S to I (is on (s, t) -cut)



$$(\# \text{ paths packed}) \stackrel{!}{=} \left(\max \# \text{ disjoint } s \rightarrow t \text{ paths} \right) \stackrel{!}{=} \left(\min \# \text{ edges in any } s \rightarrow t \text{ cut} \right) \stackrel{!}{=} \left(\# \text{ edges initially in } C \right)$$

Theorems

$$\textcircled{1} \left(\max \# \text{ disjoint } s \rightarrow t \text{ paths} \right) = \left(\min \# \text{ edges in any } s \rightarrow t \text{ cut} \right)$$

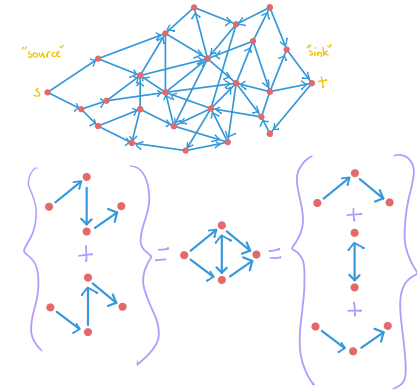
$$\textcircled{2} \left(\# \text{ paths packed} \right) = \left(\max \# \text{ disjoint } s \rightarrow t \text{ paths} \right)$$

$$\textcircled{3} \left(\min \# \text{ edges in any } s \rightarrow t \text{ cut} \right) = \left(\# \text{ edges initially in } C \right)$$

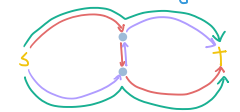
(where $\left(\# \text{ paths packed} \right)$ and $\left(\# \text{ edges initially in } C \right)$ refers to the outcome of iterative finding paths in the residual graph)

How to compute

⑦ max # of edge-disjoint $s \rightarrow t$ paths?



Lemma let p, q be edge-disjoint $s \rightarrow t$ paths



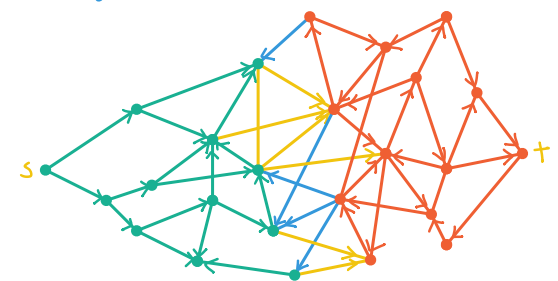
then there is p', q' : $s \rightarrow t$ disjoint without opposite pairs of edges

augmenting-paths($G = (V, E), s, t$)

/* Ford and Fulkerson [FF56]

1. $P \leftarrow \emptyset$
2. While there is an (s, t) -path $p: s \rightsquigarrow t$:
 - A. Remove all of p 's edges from the graph.
 - B. Add the reverse of all of p 's edges to the graph.
 - C. Uncross p with previously selected paths, and add p to P .

let (S, I) partition V st. $s \in S, t \in I$
 $C =$ edges from S to I (is an (s, t) -cut)



augmenting-paths($G = (V, E), s, t$)

/* Ford and Fulkerson [FF56]

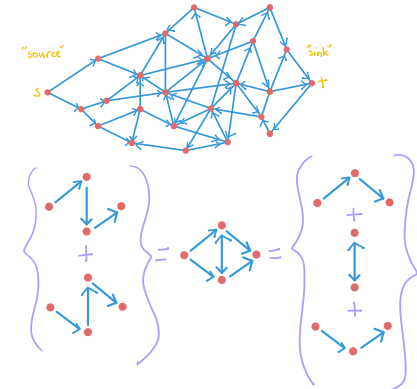
*/

1. $P \leftarrow \emptyset$
2. While there is an (s, t) -path $p : s \rightsquigarrow t$:
 - A. Remove all of p 's edges from the graph.
 - B. Add the reverse of all of p 's edges to the graph.
 - C. Uncross p with previously selected paths, and add p to P .³
3. $S \leftarrow \{\text{vertices still reachable from } s\}$.
4. Return the (s, t) -path packing P and the (s, t) -cut $\partial^+(S)$.

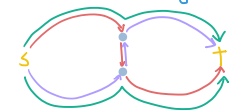
Theorem 14.5 ([FF56]). Let $G = (V, E)$ be a directed graph with m edges, and let $s, t \in V$ be distinct vertices. Let λ be the size of the max-flow/min-cut from s to t . Then augmenting-paths(G, s, t) produces both a maximum packing and a minimum cut in $O(m\lambda)$ time.

How to compute

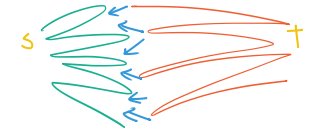
⑦ max # of edge-disjoint $s \rightsquigarrow t$ paths?



Lemma let p, q be edge-disjoint $s \rightsquigarrow t$ paths



then there is $p', q' : s \rightsquigarrow t$ disjoint without opposite pairs of edges



$$\textcircled{1} \left(\begin{array}{c} \text{max \#} \\ \text{disjoint} \\ s \rightsquigarrow t \text{ paths} \end{array} \right) = \left(\begin{array}{c} \text{min \#} \\ \text{edges in any} \\ s \rightsquigarrow t \text{ cut} \end{array} \right)$$

$$\textcircled{2} \left(\begin{array}{c} \# \text{ paths} \\ \text{packed} \end{array} \right) = \left(\begin{array}{c} \text{max \#} \\ \text{disjoint} \\ s \rightsquigarrow t \text{ paths} \end{array} \right)$$

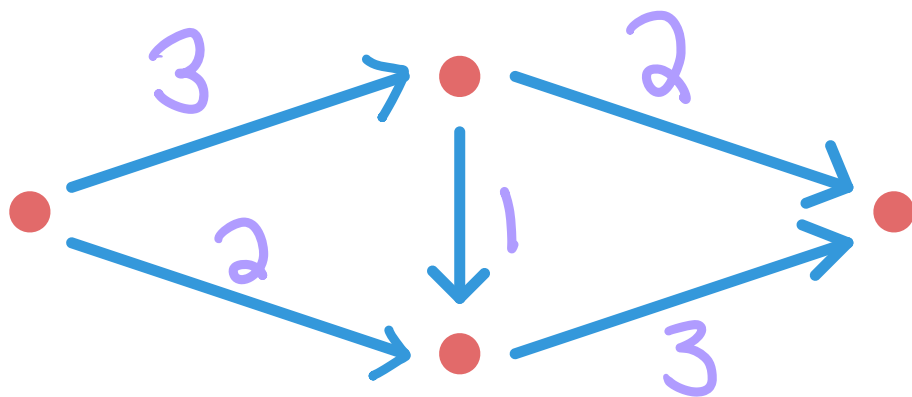
$$\textcircled{3} \left(\begin{array}{c} \text{min \#} \\ \text{edges in any} \\ s \rightsquigarrow t \text{ cut} \end{array} \right) = \left(\begin{array}{c} \# \text{ edges} \\ \text{initially in } C \end{array} \right)$$

1. $P \leftarrow \emptyset$
2. While there is an (s, t) -path $p: s \rightsquigarrow t$:
 - A. Remove all of p 's edges from the graph.
 - B. Add the reverse of all of p 's edges to the graph.
 - C. Uncross p with previously selected paths, and add p to P .³
3. $S \leftarrow \{\text{vertices still reachable from } s\}$.
4. Return the (s, t) -path packing P and the (s, t) -cut $\partial^+(S)$.

*/

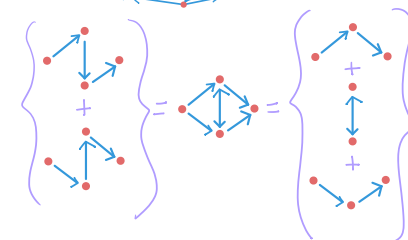
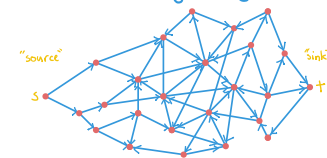
Theorem 14.5 ([FF56]). Let $G = (V, E)$ be a directed graph with m edges, and let $s, t \in V$ be distinct vertices. Let λ be the size of the max-flow/min-cut from s to t . Then $\text{augmenting-paths}(G, s, t)$ produces both a maximum packing and a minimum cut in $O(m\lambda)$ time.

edge capacities,
fractional packings, & network flow

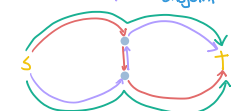


How to compute

⑦ max # of edge-disjoint $s \rightsquigarrow t$ paths?



Lemma let p, q be edge-disjoint $s \rightsquigarrow t$ paths



then there is $p', q': s \rightsquigarrow t$ disjoint without opposite pairs of edges



$$\textcircled{1} \left(\begin{array}{c} \text{max \#} \\ \text{disjoint} \\ s \rightsquigarrow t \text{ paths} \end{array} \right) = \left(\begin{array}{c} \text{min \#} \\ \text{edges in any} \\ s \rightsquigarrow t \text{ cut} \end{array} \right)$$

$$\textcircled{2} \left(\begin{array}{c} \text{\# paths} \\ \text{packed} \end{array} \right) = \left(\begin{array}{c} \text{max \#} \\ \text{disjoint} \\ s \rightsquigarrow t \text{ paths} \end{array} \right)$$

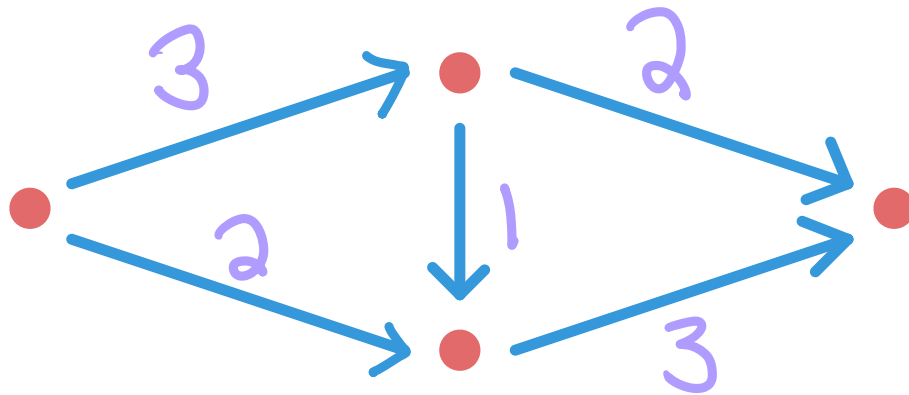
$$\textcircled{3} \left(\begin{array}{c} \text{min \#} \\ \text{edges in any} \\ s \rightsquigarrow t \text{ cut} \end{array} \right) = \left(\begin{array}{c} \text{\# edges} \\ \text{initially in } C \end{array} \right)$$

capacity $c(e) > 0$: we can reuse
edge e up to $c(e)$ times

1. $P \leftarrow \emptyset$
2. While there is an (s, t) -path $p: s \rightsquigarrow t$:
 - A. Remove all of p 's edges from the graph.
 - B. Add the reverse of all of p 's edges to the graph.
 - C. Uncross p with previously selected paths, and add p to P .³
3. $S \leftarrow \{\text{vertices still reachable from } s\}$.
4. Return the (s, t) -path packing P and the (s, t) -cut $\partial^+(S)$.

*/

Theorem 14.5 ([FF56]). Let $G = (V, E)$ be a directed graph with m edges, and let $s, t \in V$ be distinct vertices. Let λ be the size of the max-flow/min-cut from s to t . Then $\text{augmenting-paths}(G, s, t)$ produces both a maximum packing and a minimum cut in $O(m\lambda)$ time.



capacity $c(e) > 0$: reuse edge e up to $c(e)$ times

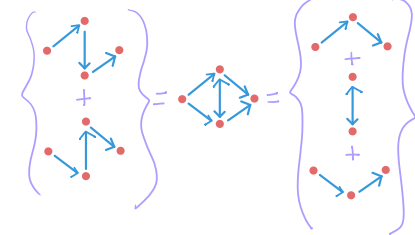
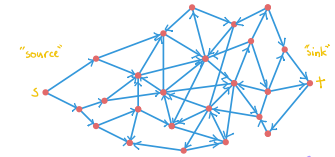
• if $c(e)$ large, Augmenting-Paths not polynomial time

• if $c(e)$ not integral (and just real-valued)

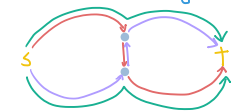
what about max fractional path packing

How to compute

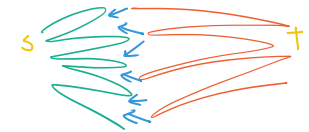
⑦ max # of edge-disjoint $s \rightsquigarrow t$ paths?



Lemma let p, q be edge-disjoint $s \rightsquigarrow t$ paths



then there is p', q' : $s \rightsquigarrow t$ disjoint without opposite pairs of edges



$$\textcircled{1} \left(\begin{array}{l} \text{max \#} \\ \text{disjoint } s \rightsquigarrow t \text{ paths} \end{array} \right) = \left(\begin{array}{l} \text{min \#} \\ \text{edges in any } s \rightsquigarrow t \text{ cut} \end{array} \right)$$

$$\textcircled{2} \left(\begin{array}{l} \text{\# paths} \\ \text{packed} \end{array} \right) = \left(\begin{array}{l} \text{max \#} \\ \text{disjoint } s \rightsquigarrow t \text{ paths} \end{array} \right)$$

$$\textcircled{3} \left(\begin{array}{l} \text{min \#} \\ \text{edges in any } s \rightsquigarrow t \text{ cut} \end{array} \right) = \left(\begin{array}{l} \text{\# edges} \\ \text{initially in } C \end{array} \right)$$

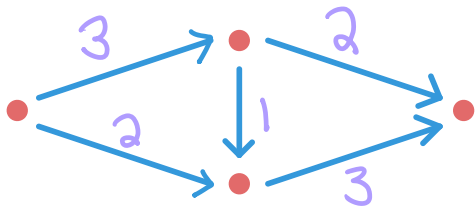
augmenting-paths($G = (V, E), s, t$)

/* Ford and Fulkerson [FF56]

1. $P \leftarrow \emptyset$
2. While there is an (s, t) -path $p : s \rightsquigarrow t$:
 - A. Remove all of p 's edges from the graph.
 - B. Add the reverse of all of p 's edges to the graph.
 - C. Uncross p with previously selected paths, and add p to P .³
3. $S \leftarrow \{\text{vertices still reachable from } s\}$.
4. Return the (s, t) -path packing P and the (s, t) -cut $\partial^+(S)$.

*/

Theorem 14.5 ([FF56]). Let $G = (V, E)$ be a directed graph with m edges, and let $s, t \in V$ be distinct vertices. Let λ be the size of the max-flow/min-cut from s to t . Then $\text{augmenting-paths}(G, s, t)$ produces both a maximum packing and a minimum cut in $O(m\lambda)$ time.



capacity $c(e) > 0$: reuse edge e
up to $c(e)$ times

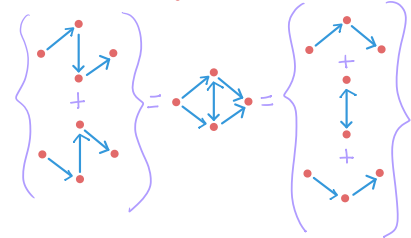
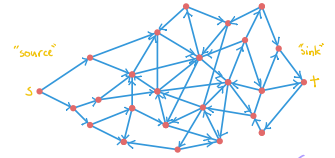
- if $c(e)$ large, Augmenting-Paths not polynomial time
- if $c(e)$ not integral (and just real-valued) what about max fractional path packing

Good news:

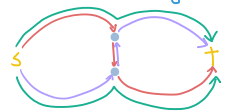
our duality theorems still hold

How to compute

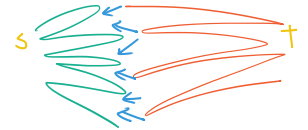
⑦ max # of edge-disjoint $s \rightsquigarrow t$ paths?



Lemma let p, q be edge-disjoint $s \rightsquigarrow t$ paths



then there is $p', q' : s \rightsquigarrow t$ disjoint
without opposite pairs of edges



$$\textcircled{1} \left(\begin{array}{c} \text{max \#} \\ \text{disjoint} \\ s \rightsquigarrow t \text{ paths} \end{array} \right) = \left(\begin{array}{c} \text{min \#} \\ \text{edges in any} \\ s \rightsquigarrow t \text{ cut} \end{array} \right)$$

$$\textcircled{2} \left(\begin{array}{c} \# \text{ paths} \\ \text{packed} \end{array} \right) = \left(\begin{array}{c} \text{max \#} \\ \text{disjoint} \\ s \rightsquigarrow t \text{ paths} \end{array} \right)$$

$$\textcircled{3} \left(\begin{array}{c} \text{min \#} \\ \text{edges in any} \\ s \rightsquigarrow t \text{ cut} \end{array} \right) = \left(\begin{array}{c} \# \text{ edges} \\ \text{initially in } C \end{array} \right)$$

remaining issues algorithmic

Max Flow "liquid" version of path packing

Input $G=(V,E)$ directed capacities $c:E \rightarrow \mathbb{R}_{>0}$

source $s \in V$, sink $t \in V$

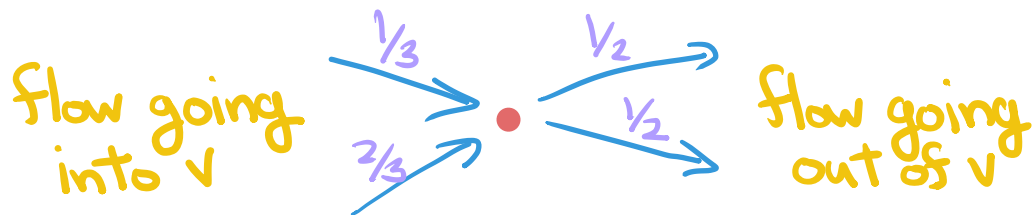
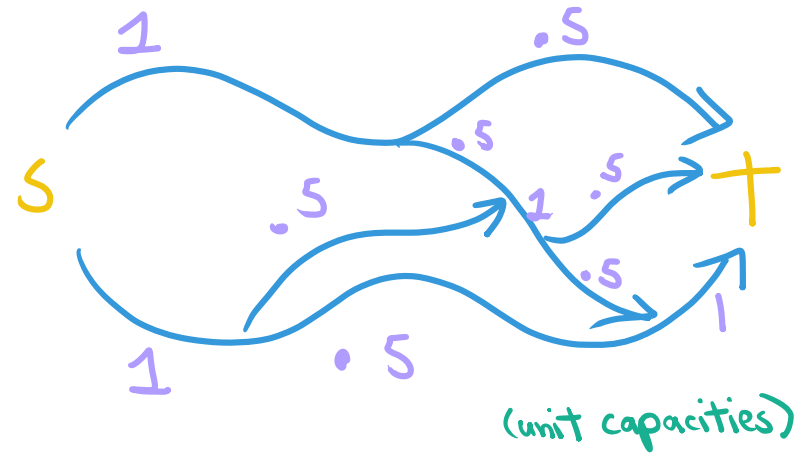
"s-t Flow" $f:E \rightarrow \mathbb{R}_{\geq 0}$ s.t.

(a) "capacity constraints"

$$0 \leq f(e) \leq c(e) \quad \forall e \in E$$

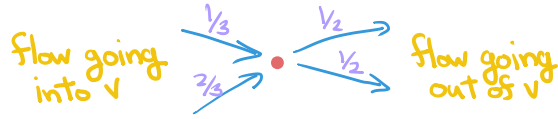
(b) "conservation of flow"

$$\sum_{(u,v) \in E} f(u,v) = \sum_{(v,w) \in E} f(v,w) \quad \forall v \notin \{s,t\}$$



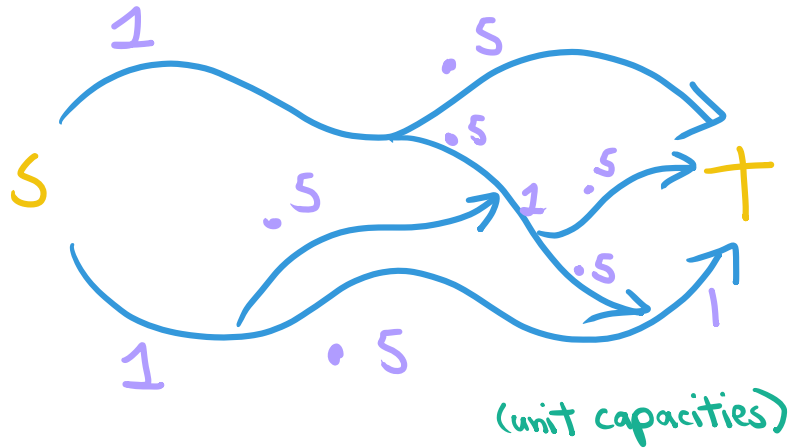
(b) "Conservation of flow"

$$\sum_{(u,v) \in E} f(u,v) = \sum_{(v,w) \in E} f(v,w) \quad \forall v \notin \{s,t\}$$



$$\text{net flow out of } s = \text{net flow into } t$$

||



"net flow from s to t",
"size of flow"

Max flow problem

find $s \rightarrow t$ flow of maximum size

"linear program"

size of flow

$$\max \sum_{(s \rightarrow w) \in E} f(s, w) - \sum_{(u \rightarrow s) \in E} f(u, s) \quad \text{over } f \in \mathbb{R}^E$$

"capacity constraints" s.t. $0 \leq f(e) \leq c(e) \quad \forall e \in E$

"conservation of flow" $\sum_{(u, v) \in E} f(u, v) = \sum_{(v, w) \in E} f(v, w) \quad \forall v \notin \{s, t\}$

"linear program" because objective, constraints, all linear functions of f

max path packings vs Max Flow

let $x: P_{st} \rightarrow \mathbb{R}_{\geq 0}$ encode a path packing

define $f: E \rightarrow \mathbb{R}_{\geq 0}$ as:

$$f(e) = \sum_{p \ni e} x_p$$

then f is feasible flow of same size

min cut vs Max Flow

let $C = \mathcal{J}(A)$ min-cut, $f: E \rightarrow \mathbb{R}_{\geq 0}$ max flow

(MFC of unit flow)

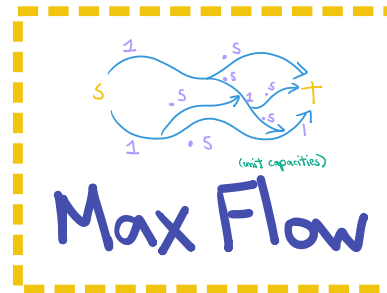


(size of f) = (net flow out of A) $\leq$ (capacity of $\mathcal{J}(A)$)

Theorem



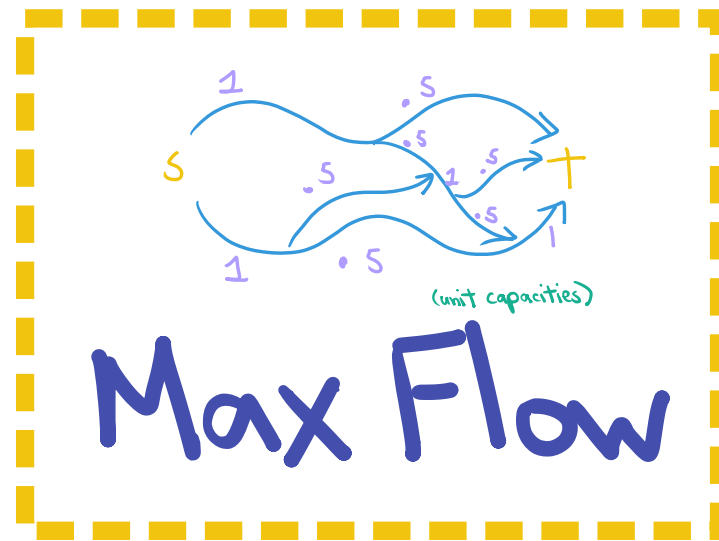
=



=

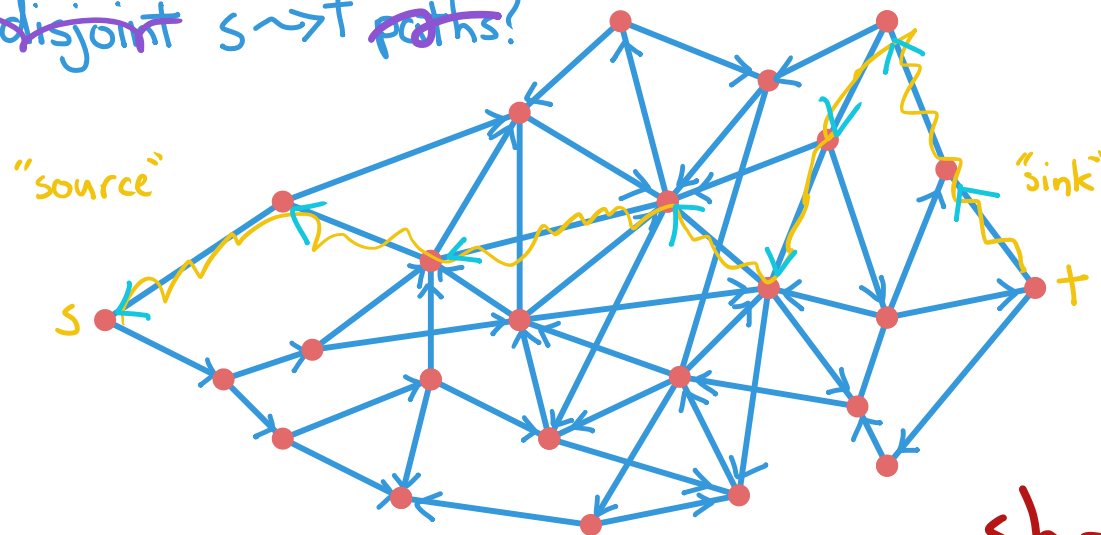


Next: strongly polynomial times



How to compute

⑦ max ~~# of edge-disjoint $s \rightsquigarrow t$ paths?~~ ^{flow}



shortest path
(Edmonds, Karp)

① while there is a path $p: s \rightsquigarrow t$

Ⓐ take p and ~~remove all its edges from the graph~~ route as much flow as possible

Ⓑ add the reverse of all of p 's edges in the graph

Ⓒ ~~uncross p w/ previously selected paths~~

update residual graph

Analysis

① while there is a path $p: s \rightsquigarrow t$

Ⓐ take shortest path p and
route as much flow as possible

Ⓑ update residual graph

Key
Lemma $\forall v \in V$, distance from s to v
in residual graph never decreases



④ while there is a path $p: s \rightsquigarrow t$

⑤ take shortest path p and
route as much flow as possible

⑥ update residual graph

Key Lemma $\forall v \in V$, distance from s to v
in residual graph never decreases

① e removed from ^(residual graph) RG ,
 $\Rightarrow e$ in shortest $s \rightsquigarrow t$ path



$\Rightarrow level(v) = level(u) + 1$

② $e = (u \rightarrow v)$ added back in to RG
 $\Rightarrow (v \rightarrow u)$ was in shortest $s \rightsquigarrow t$ path



$\Rightarrow level(u) = level(v) + 1$

each event ①/② $\Rightarrow level(u), level(v)$ have switched order.

level's nondecreasing by key lemma

$\Rightarrow e$ gets deleted/added at most n times

Analysis

Edmonds, Karp

① while there is a path $p: s \rightsquigarrow t$

② take shortest path p and
route as much flow as possible

③ update residual graph

Key
Lemma $\forall v \in V$, distance from s to v
in resid. graph never decreases

Running time

$O(m)$ per iteration

$O(mn)$ iterations

$O(m^2n)$ time

Applications of Flows + Cuts

Bipartite
Matching

Vertex
Disjoint
Paths

Path
Covers

Project
Selection

Assignment
Problems

Edge
Orientation

Densest
Subgraph

Playoff
Elimination

scheduling

Bipartite
Vertex
Cover

(high-level comment)

*(mostly; we also had a little divide + conquer extend. merge-sort)

"only two algorithms"*

① identify subproblems

② change the input

Applications of Flows + Cuts

Gameplan

1. Survey a bunch of problems
2. start solving them one by one

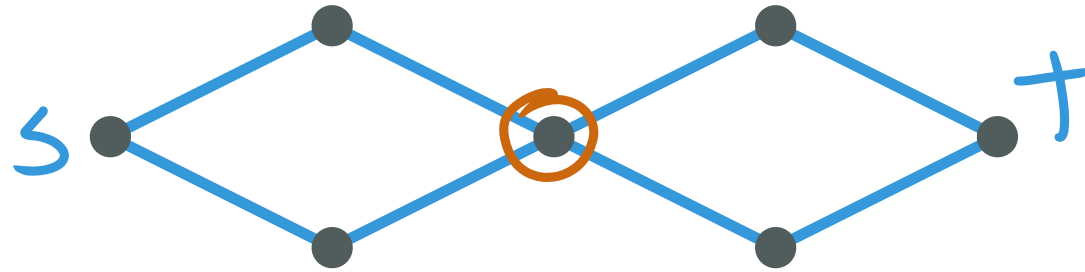


"only two algorithms"

① identify subproblems

② change the input

Vertex Disjoint Paths



2 edge disjoint paths

1 vertex disjoint path

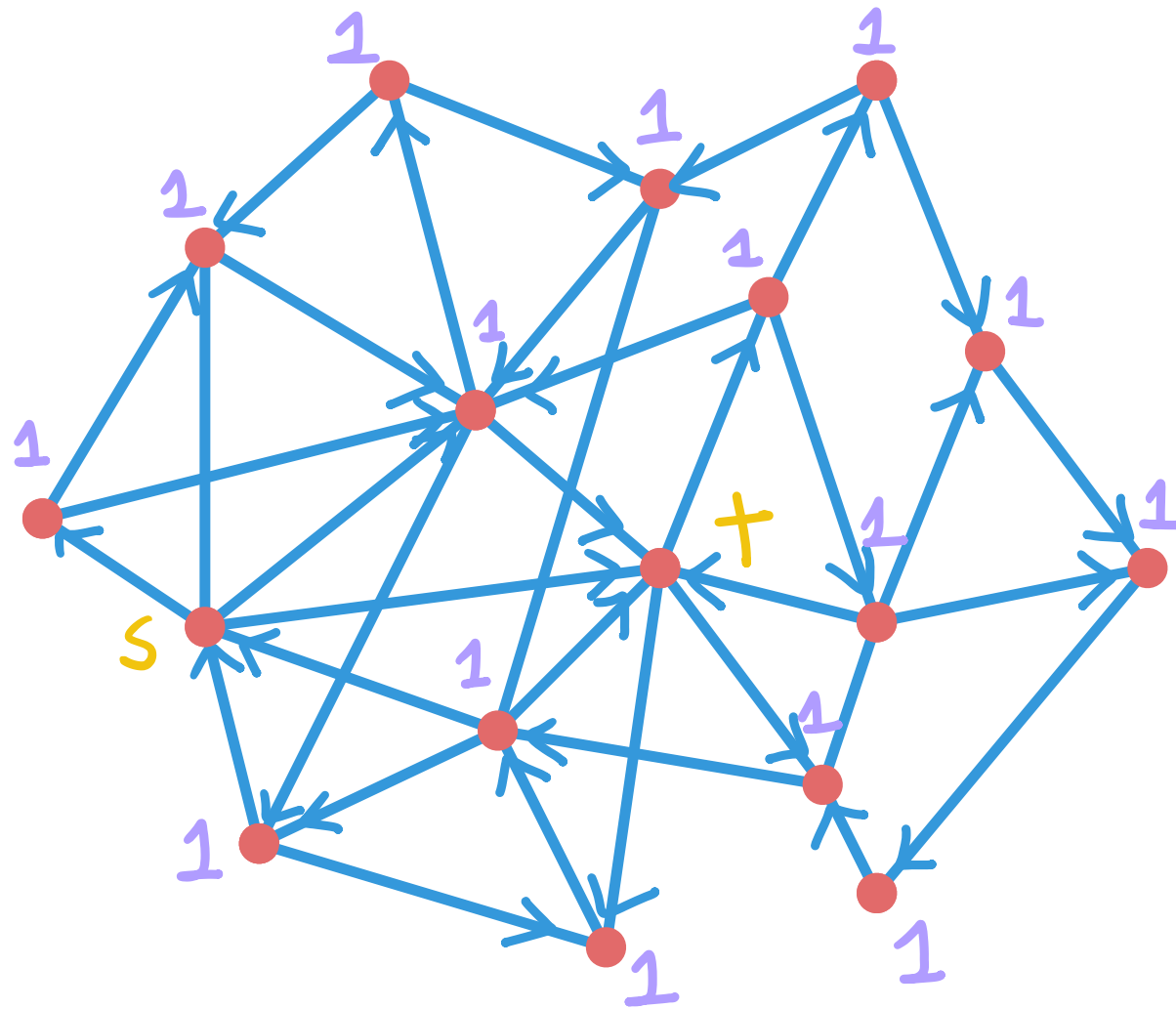
vertex cut = 1

"vertex capacities" $c: V \rightarrow \mathbb{R}_{>0}$

$c(v)$ = limit of flow through v

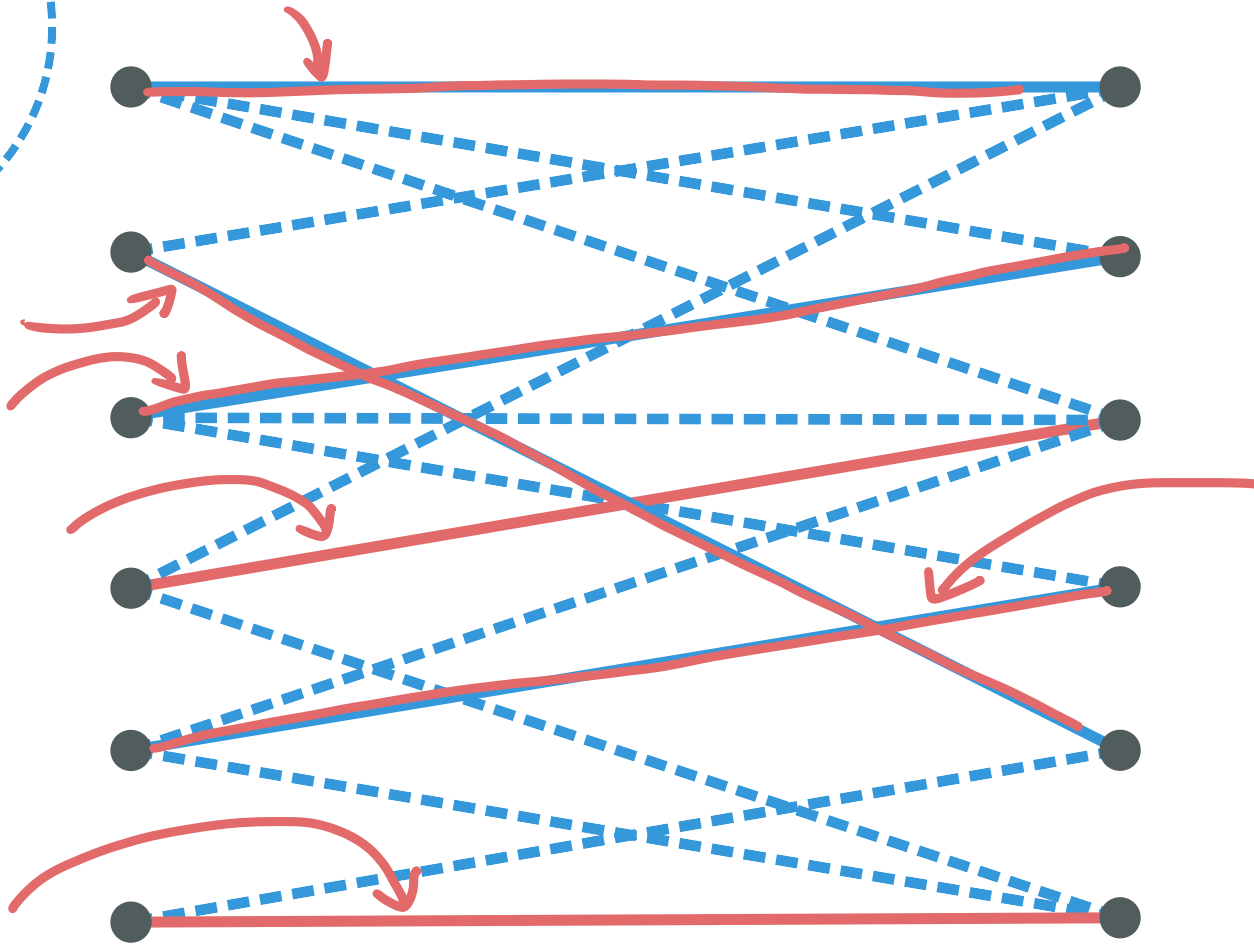
e.g. $c(v) = 1$
 $\forall v \notin \{s, t\}$

$\Rightarrow$ vertex
disjoint paths



Bipartite Matching

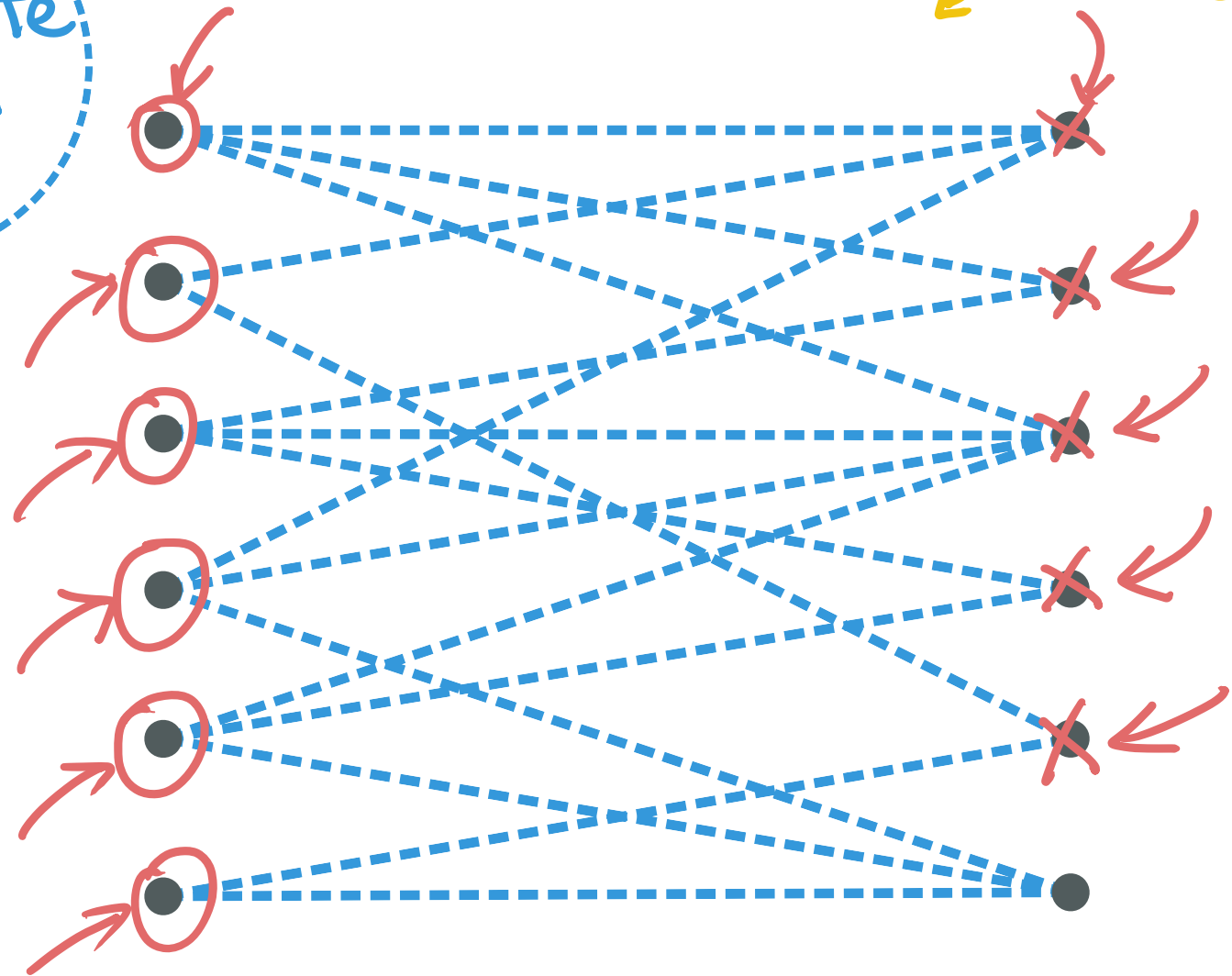
(bipartite graph)



Goal: max set of edges w/ distinct endpoints
"matching"

Bipartite
Vertex
Cover

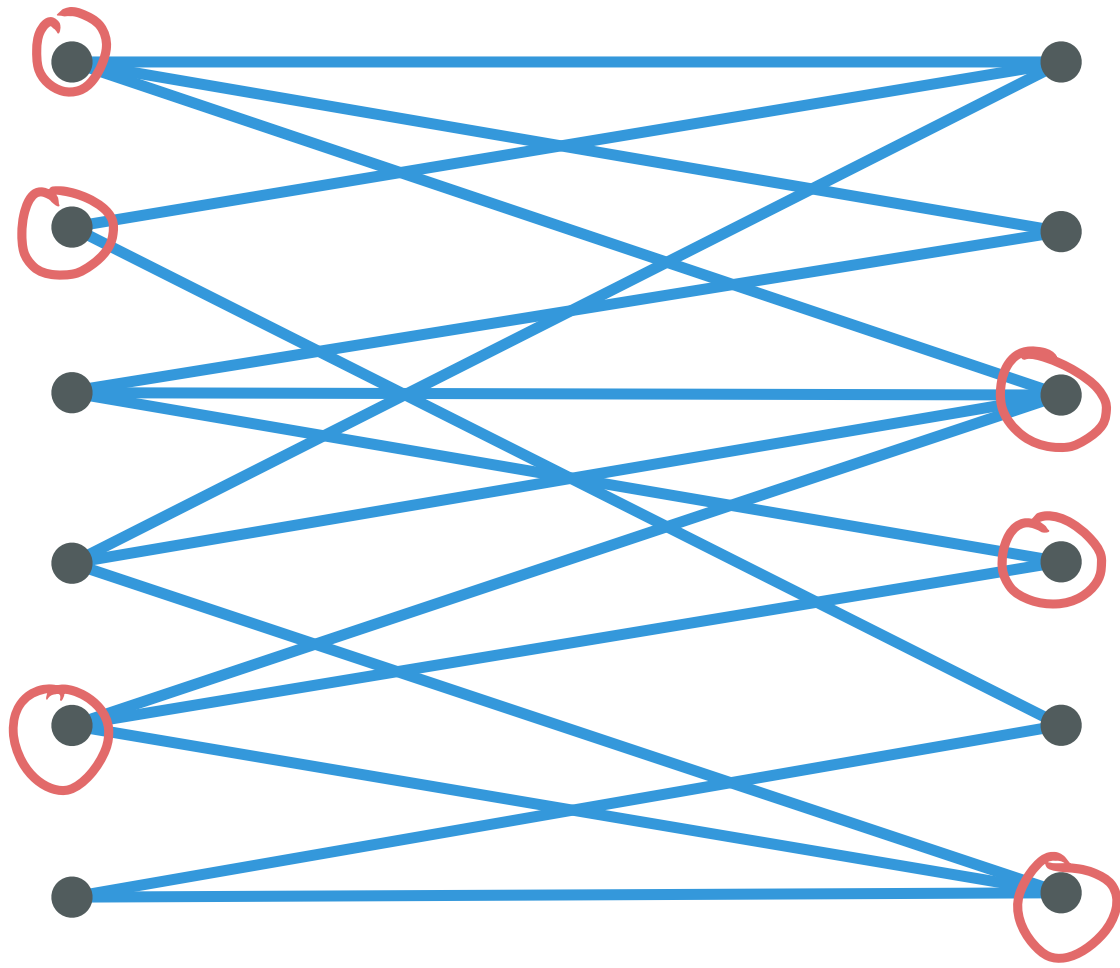
(bipartite graph)



Goal: min set of vertices incident to all edges
"vertex cover"

Bipartite
Vertex
Cover

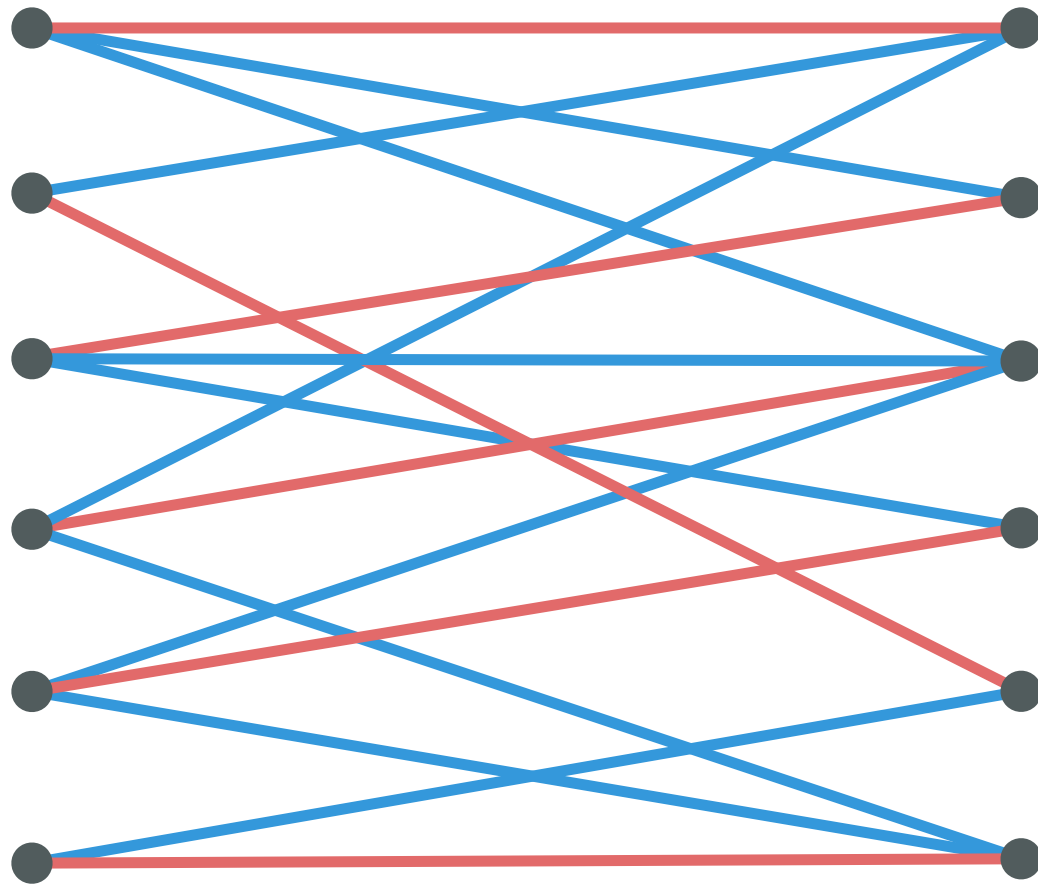
(bipartite graph)
↓



Goal: min set of vertices incident to all edges
"vertex cover"

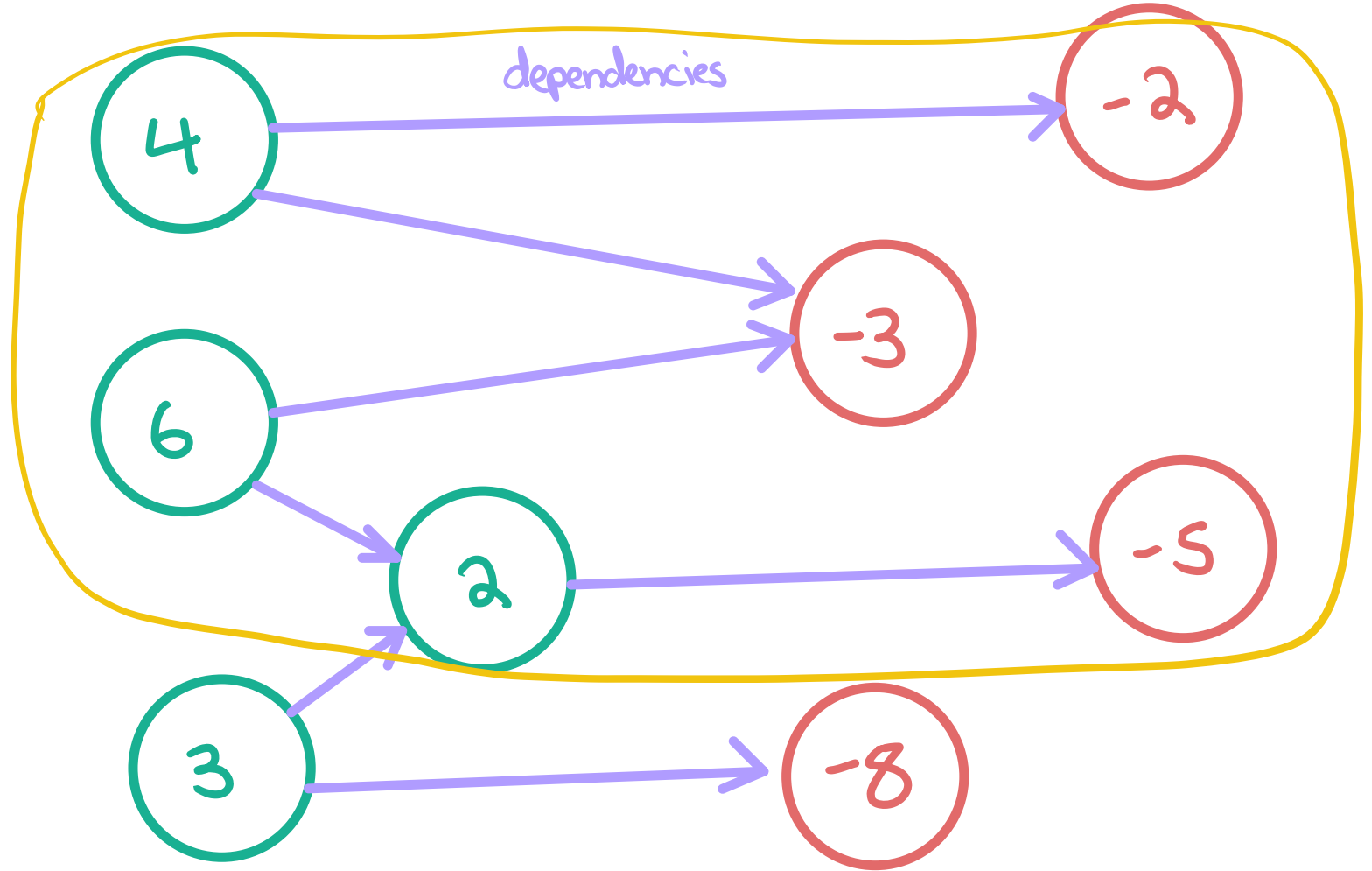
Bipartite Matching

(bipartite graph)



Goal: max set of edges w/ distinct endpoints
"matching"

Project
Selection



Projects have net profits
and dependencies

(can be < 0)
(for costs)

Goal: select projects (including dependencies)
w/ max total profits

Playoff
Elimination



Playoff
Elimination



can Boston catch the Yankees?

Team	Wins	Games left
New York	92	2
Baltimore	91	3
Toronto	91	3
Boston	90	2

Remaining schedule

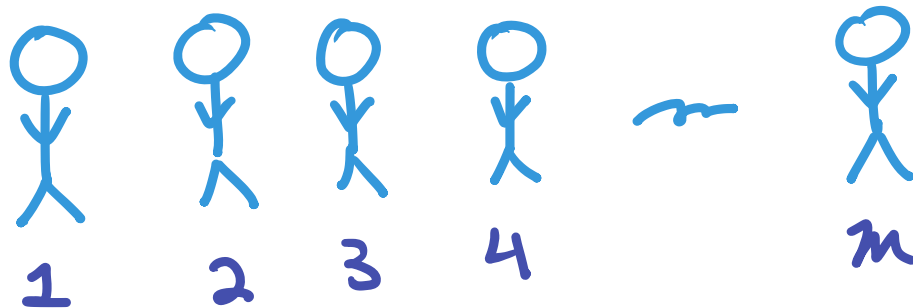
(wins)	(92) New York	(91) Baltimore	(91) Toronto	(90) Boston
New York	X	1	1	0
Baltimore	1	X	1	1
Toronto	1	1	X	1
Boston	0	1	1	X

Hand-drawn red arrows indicating a sequence of wins:

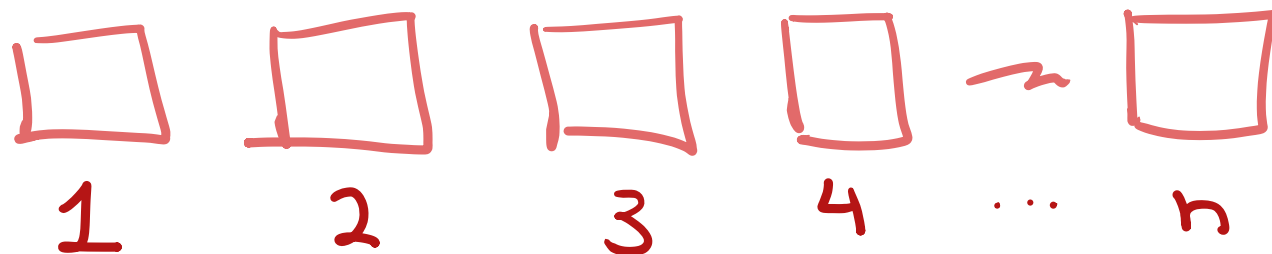
- From Boston to Toronto
- From Toronto to Baltimore
- From Baltimore to New York

Assignment Problems

m individuals



n jobs



goal: fill as many jobs as possible
w/ individuals

the catch: not everyone is qualified for all jobs

Assignment Problems

"Qualification Matrix"

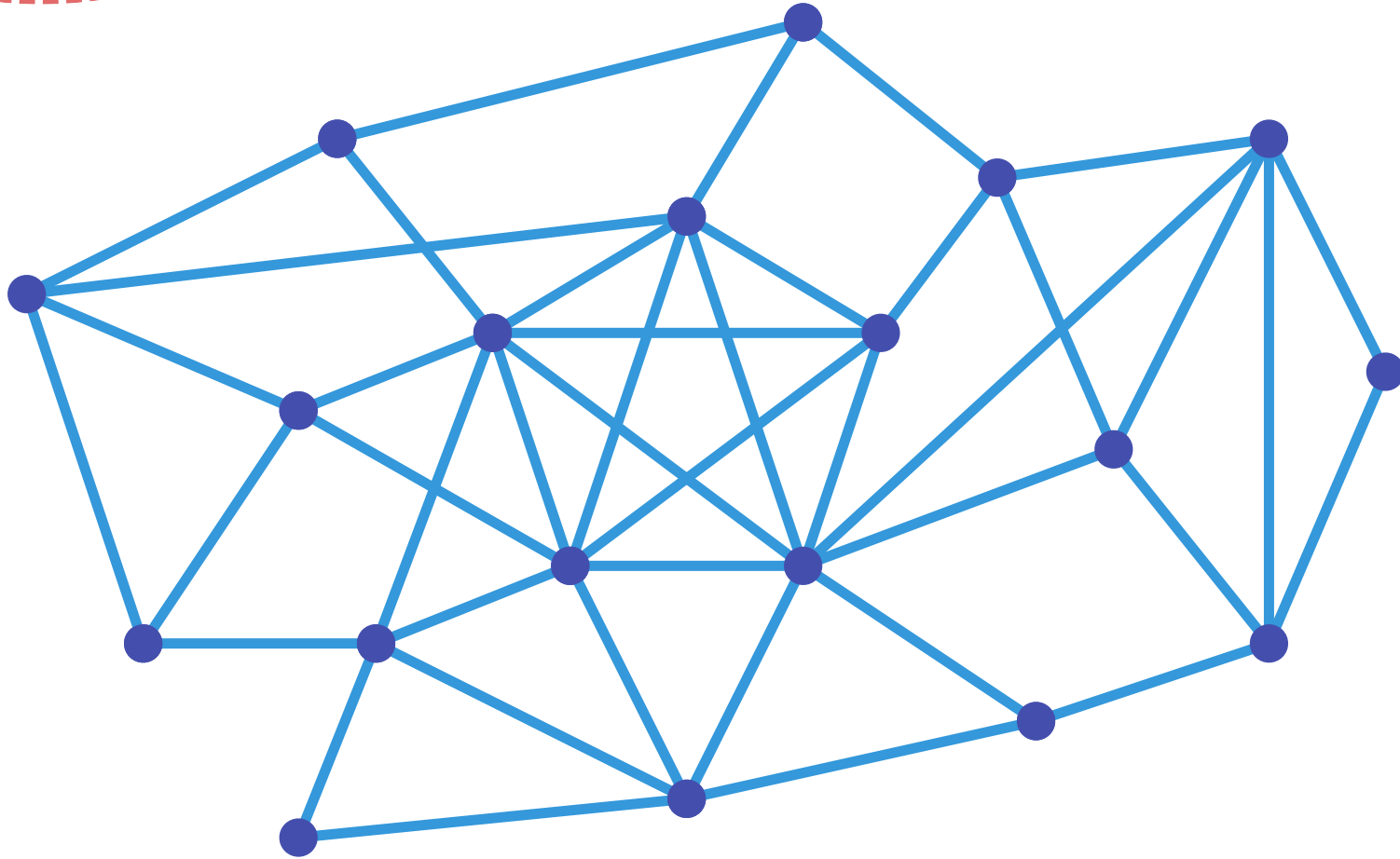
	☐	☐	☐	☐
	1	1	1	0
	0	0	1	0
	0	0	0	1
	0	0	0	1

"1" = qualified

"0" = not qualified

Edge
Orientation

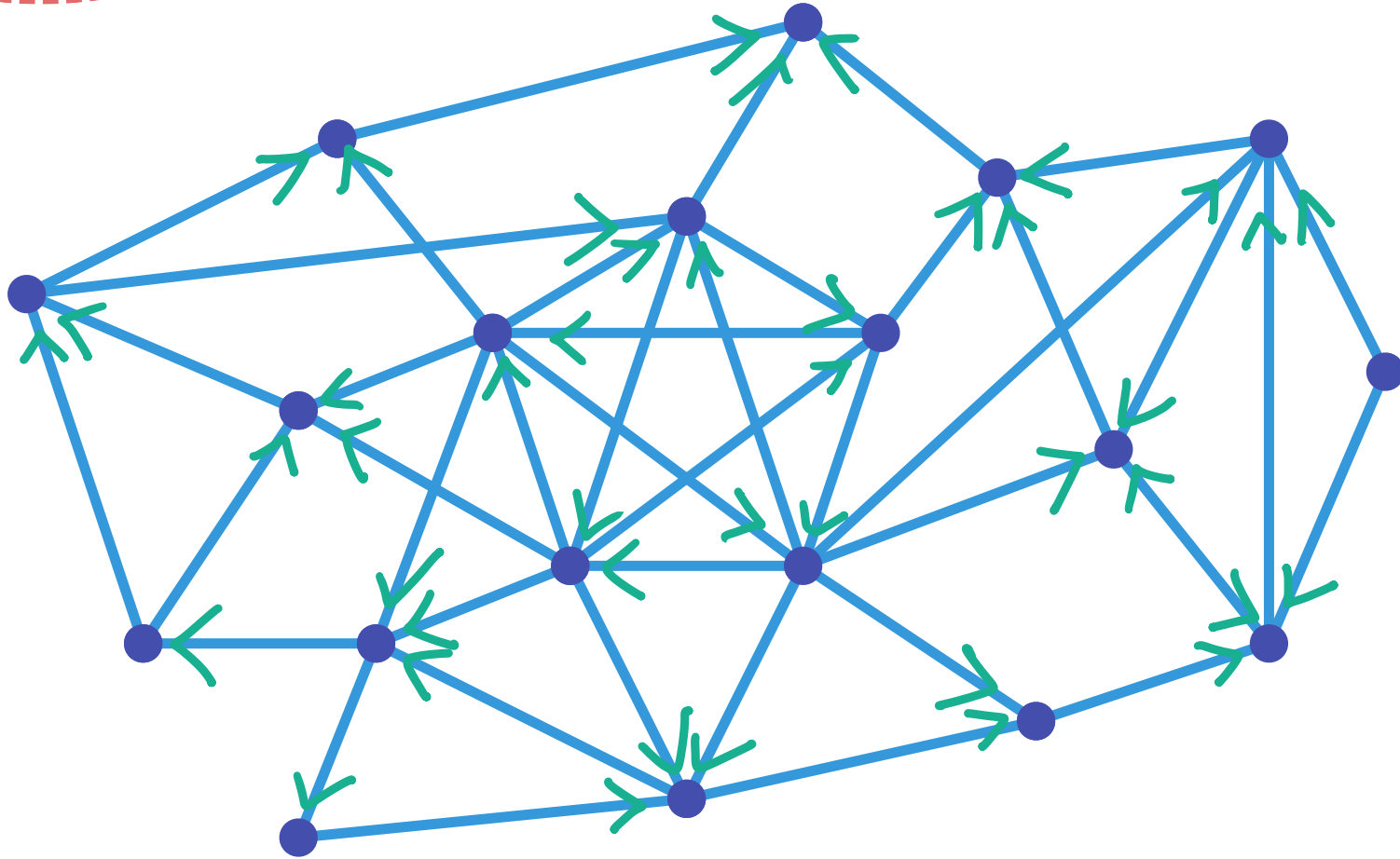
input: undirected graph



Goal: direct the edges to
to minimize the max in-degree

Edge
Orientation

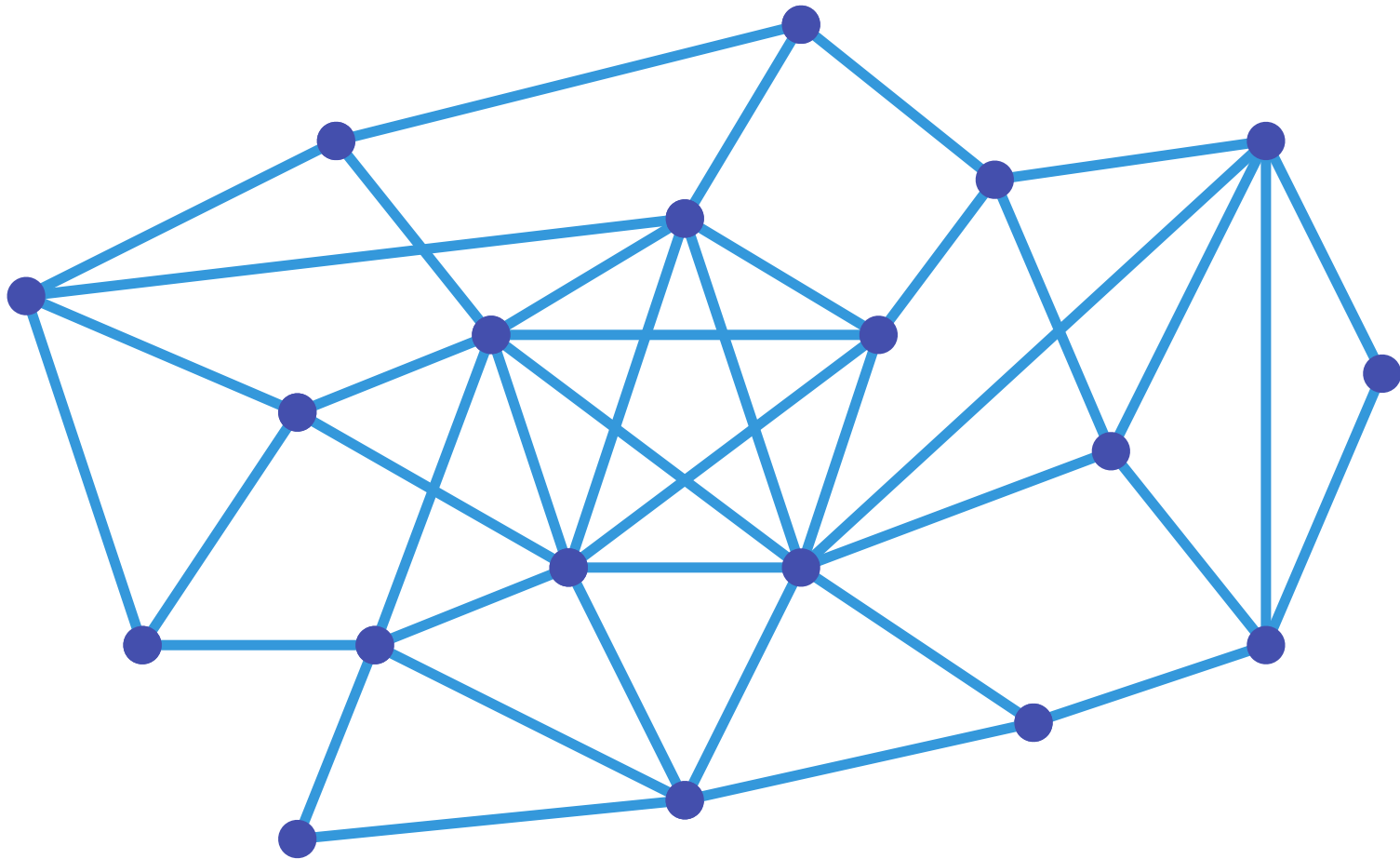
input: undirected graph



Goal: direct the edges to
minimize the max in-degree

Densest
Subgraph

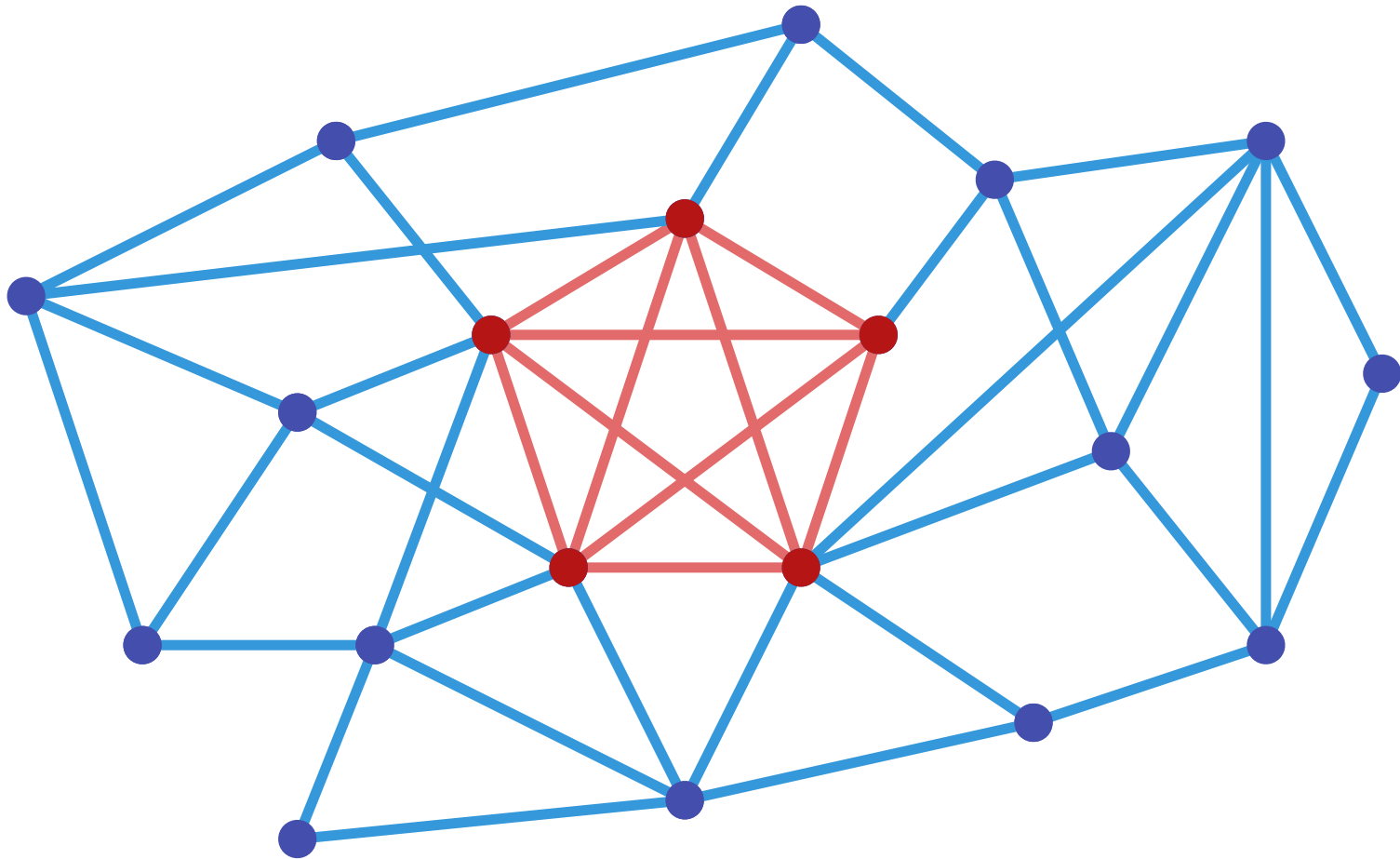
$$\text{"density"} = \frac{\# \text{ edges}}{\# \text{ vertices}}$$



Goal: compute densest subgraph

Densest
Subgraph

$$\text{"density"} = \frac{\# \text{ edges}}{\# \text{ vertices}}$$



Goal: compute densest subgraph

Applications of Flows + Cuts

Gameplan

1. Survey a bunch of problems

2. start solving them one by one

Bipartite Matching

Vertex Disjoint Paths

Path Covers

Project Selection

Assignment Problems

Edge Orientation

Densest Subgraph

Playoff Elimination

scheduling

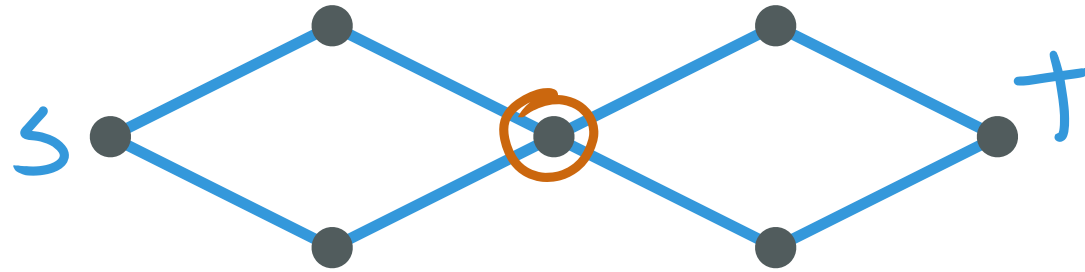
Bipartite Vertex Cover

"only two algorithms"

① identify subproblems

② change the input

Vertex Disjoint Paths



2 edge disjoint paths

1 vertex disjoint path

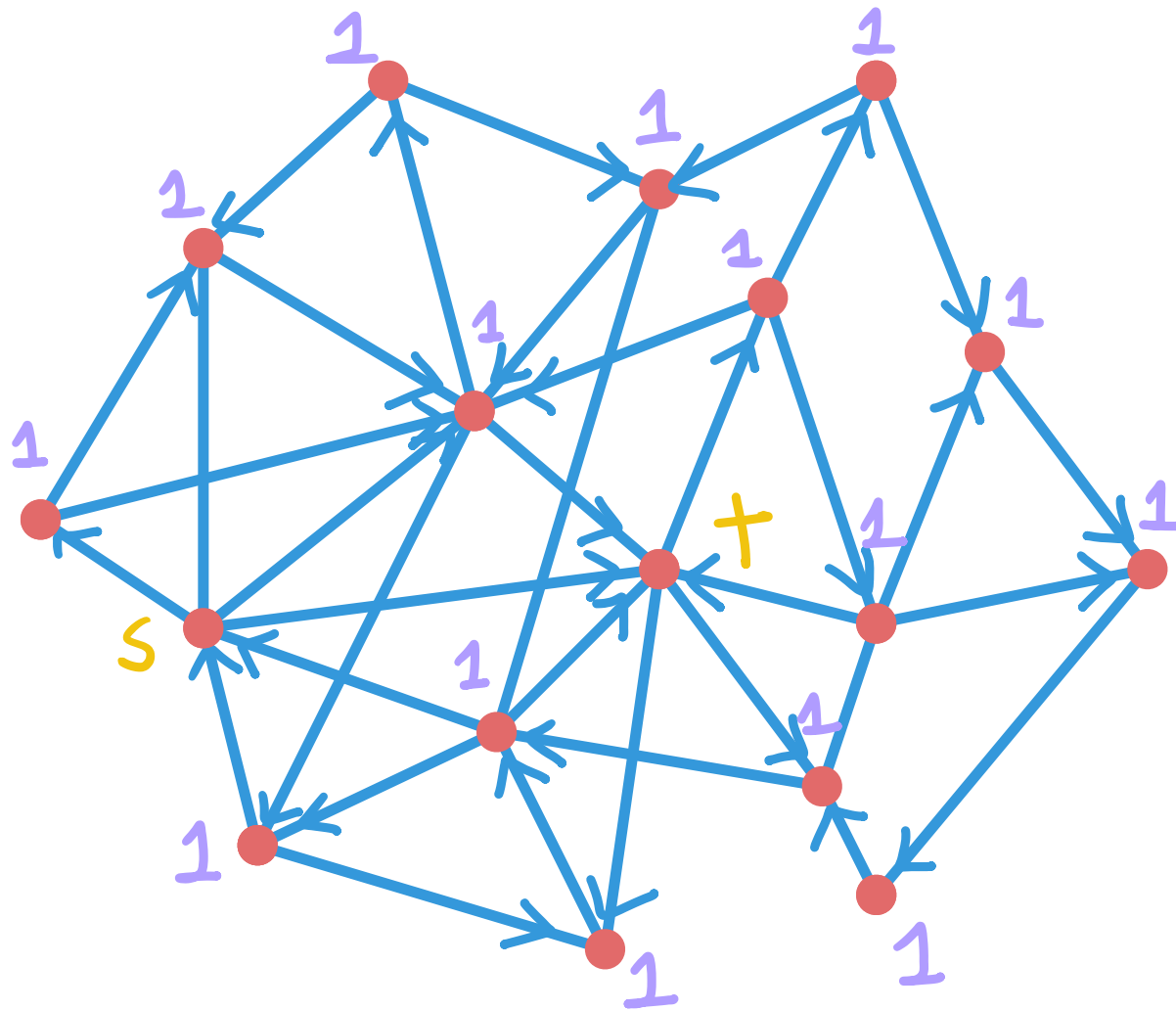
vertex cut = 1

"vertex capacities" $c: V \rightarrow \mathbb{R}_{>0}$

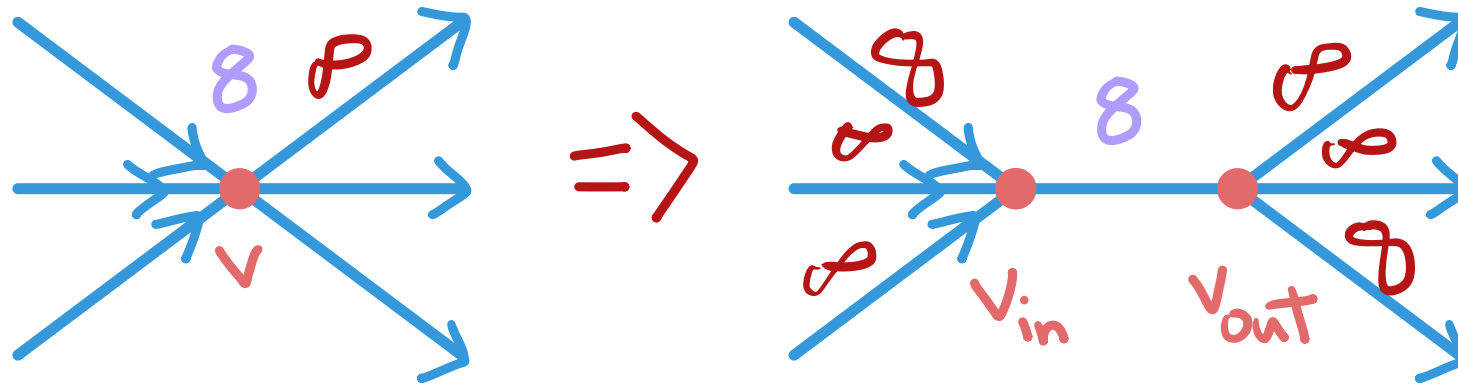
$c(v)$ = limit of flow through v

e.g. $c(v) = 1$
 $\forall v \notin \{s, t\}$

$\Rightarrow$ vertex
disjoint paths

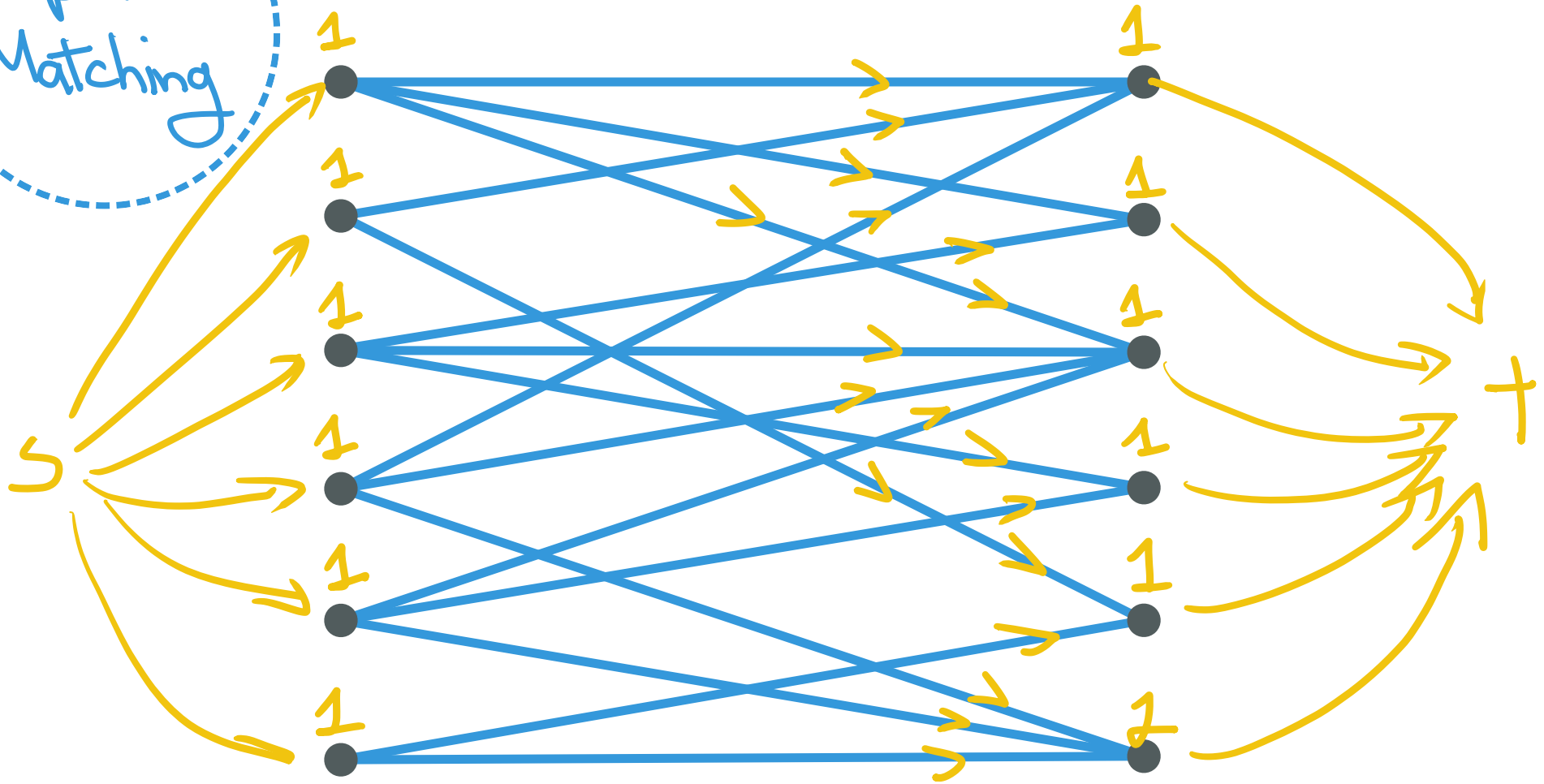


reduction to edge capacities



Bipartite
Matching

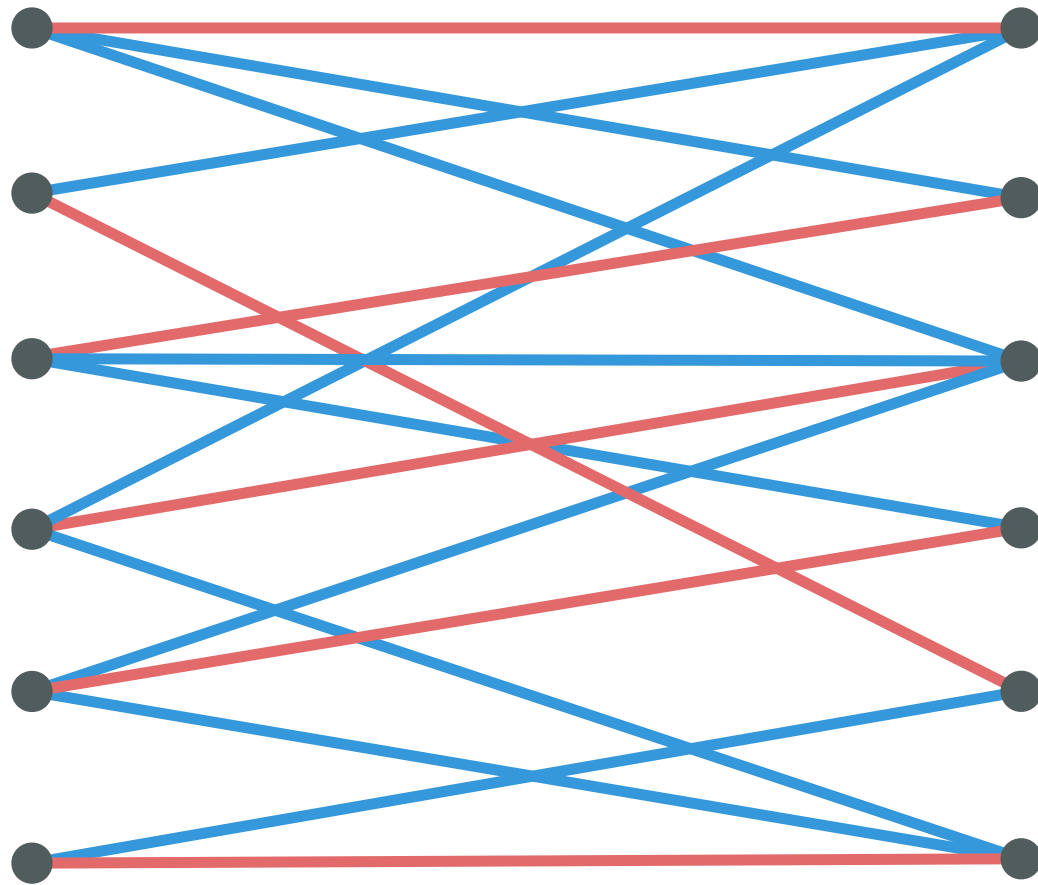
(bipartite graph)



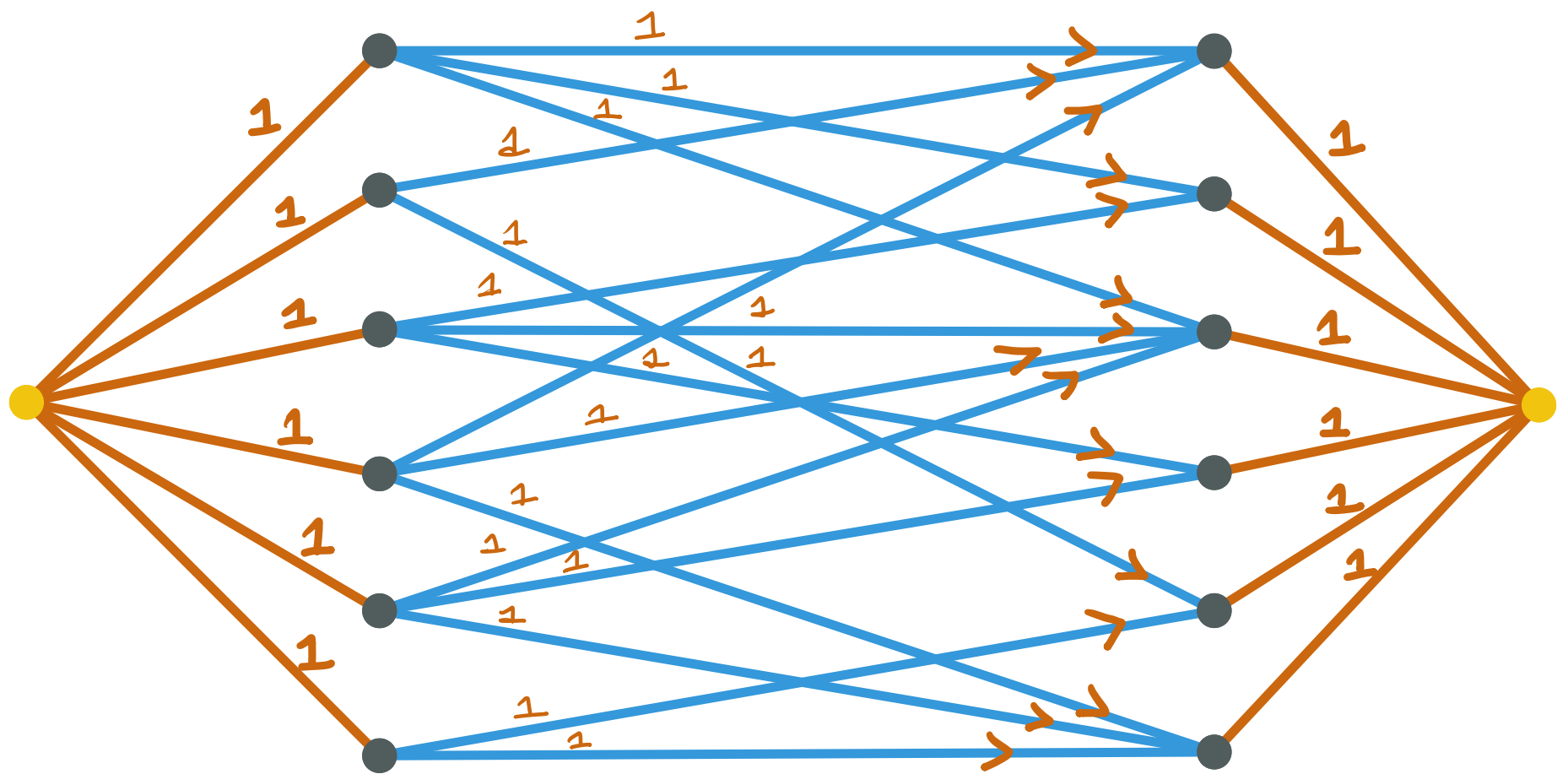
Goal: max set of edges w/ distinct endpoints
"matching"

Bipartite Matching

(bipartite graph)



Goal: max set of edges w/ distinct endpoints
"matching"



Matching w/
k edges

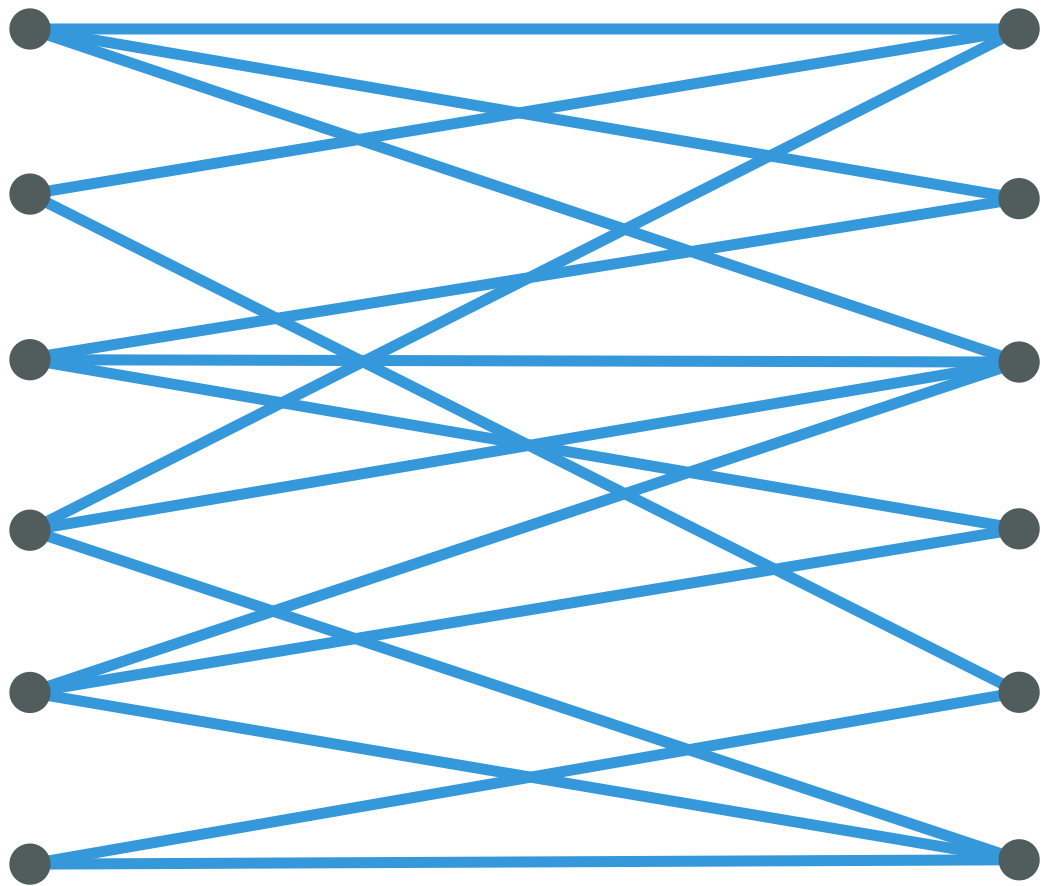


Integral flow
of size k

(Augmenting paths)

Bipartite
Vertex
Cover

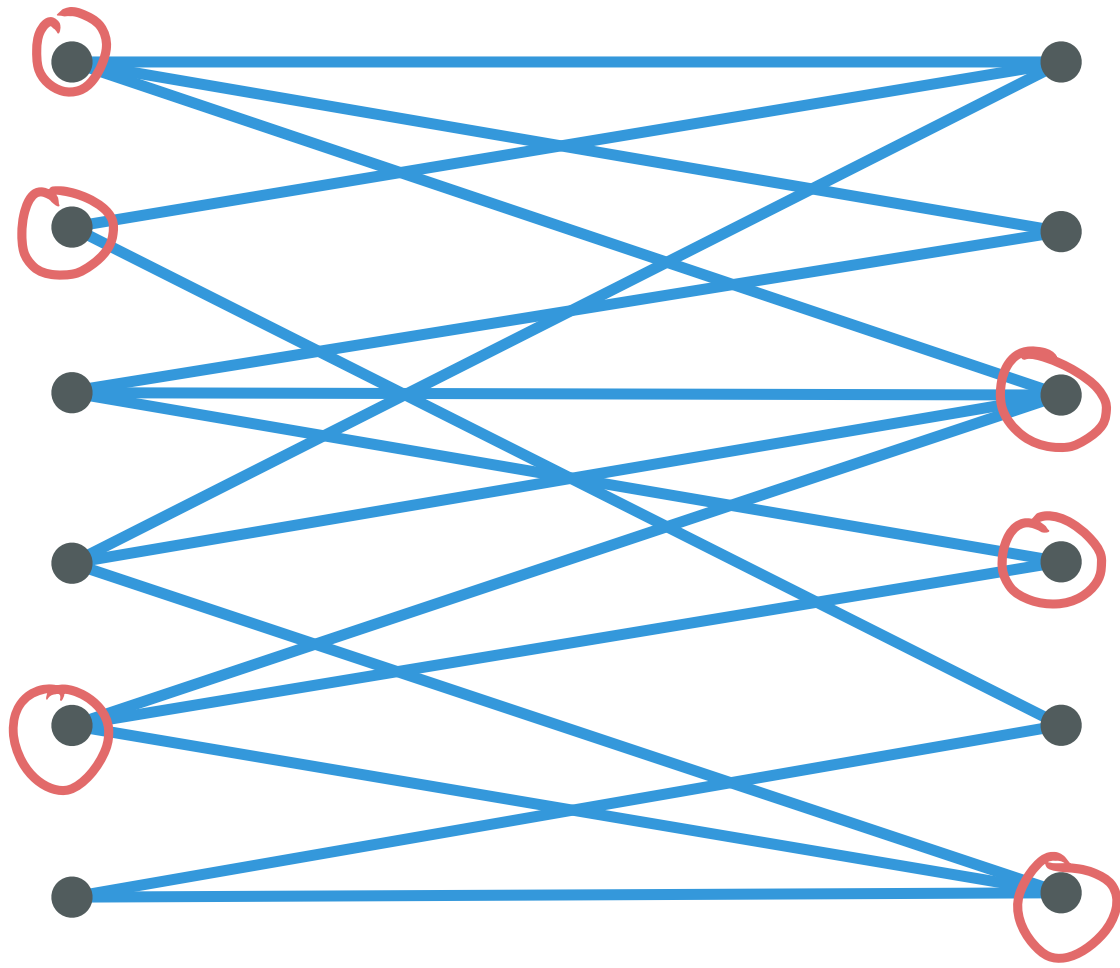
(bipartite graph)
↓



Goal: min set of vertices incident to all edges
"vertex cover"

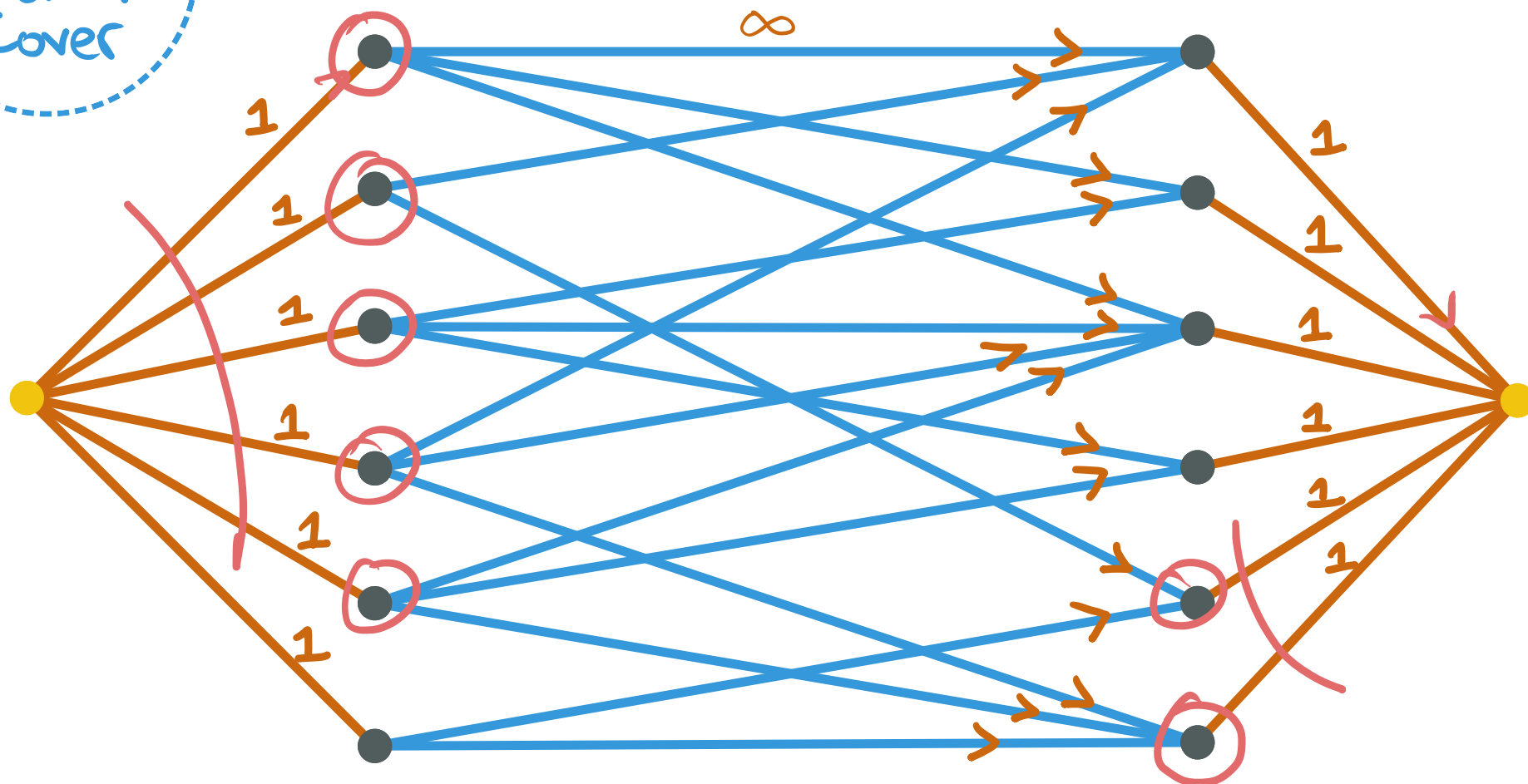
Bipartite
Vertex
Cover

(bipartite graph)
↓



Goal: min set of vertices incident to all edges
"vertex cover"

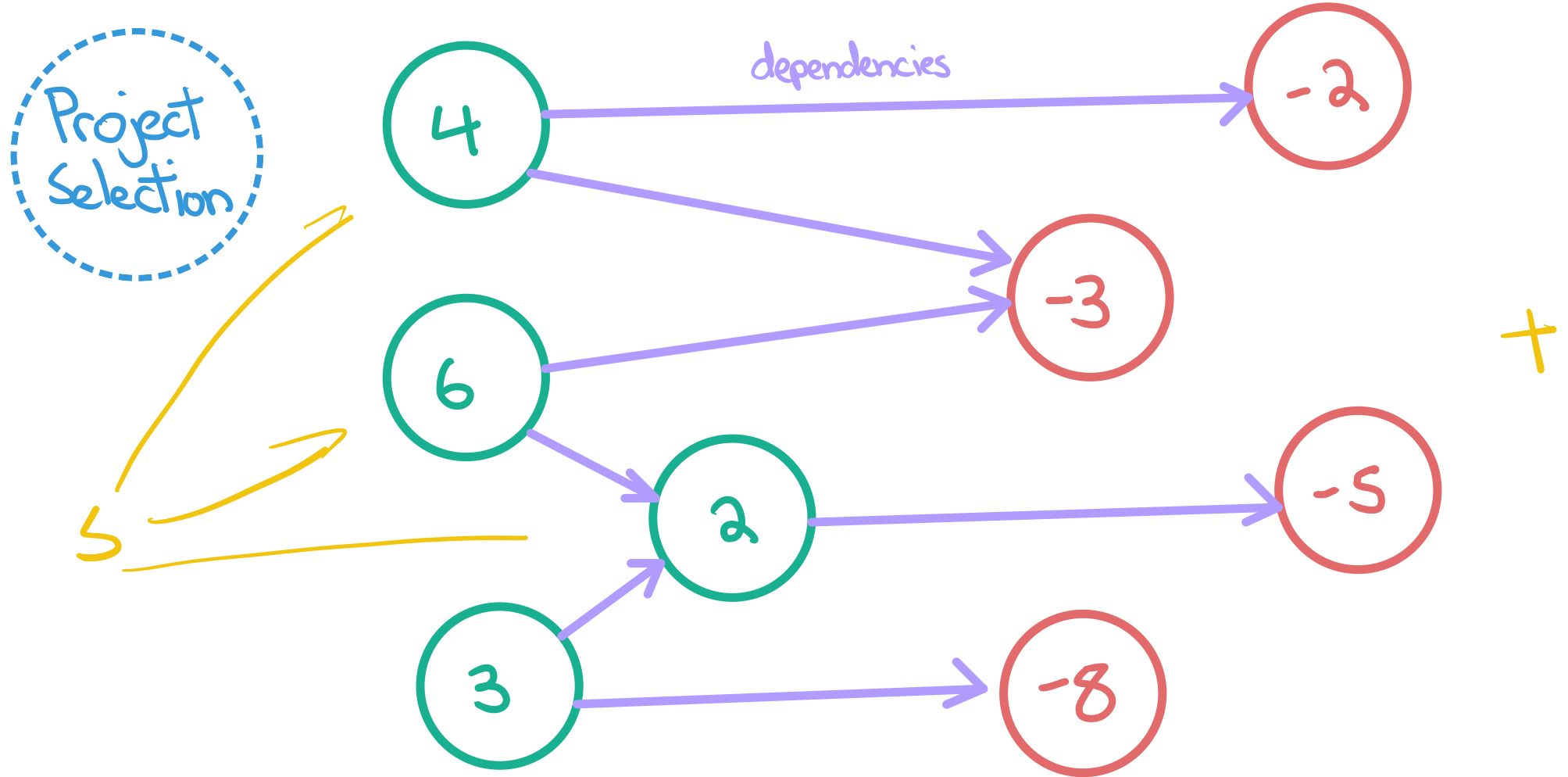
Bipartite
Vertex
Cover



cut w/
weight k

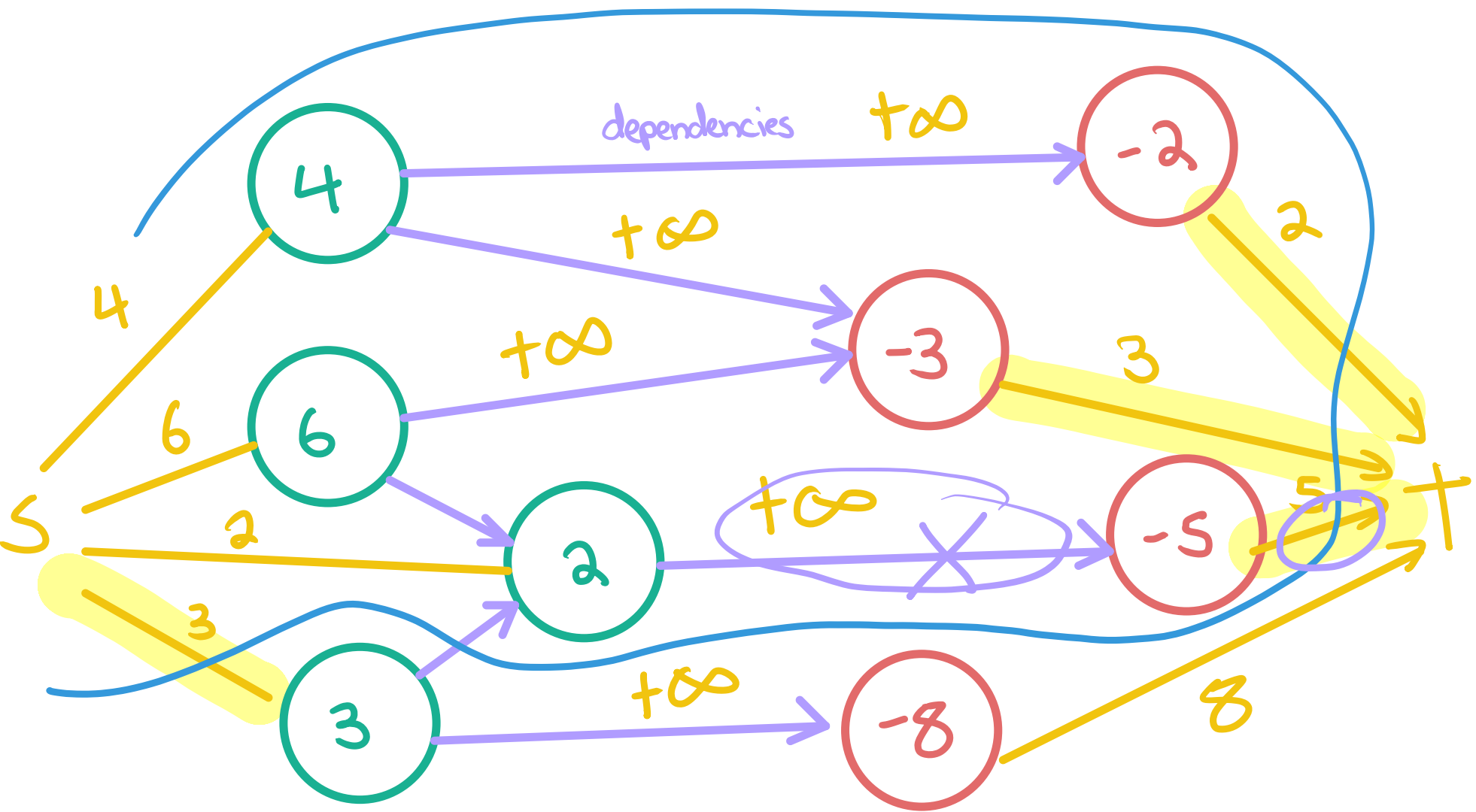


vertex cover
of size k



Projects have net profits
and dependencies

Goal: select projects (including dependencies)
w/ max total profits



claim: min-cut $\Rightarrow$ best project selection

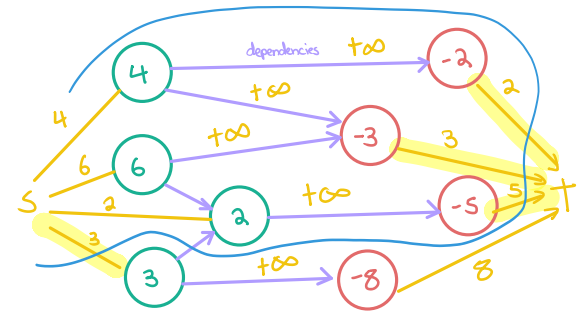
cut capacity

= (profits dont take)

+ (cost we take)

= (all possible profits) - (profits we take)
+ (cost we take)

= (all possible profits) - (net profit we take)



Playoff
Elimination



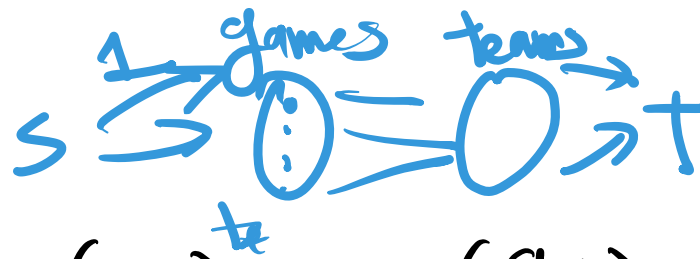
Playoff Elimination



can Boston catch the Yankees?

Team	Wins	Games left
New York	92	2
Baltimore	91	3
Toronto	91	3
Boston	90	2

Remaining schedule

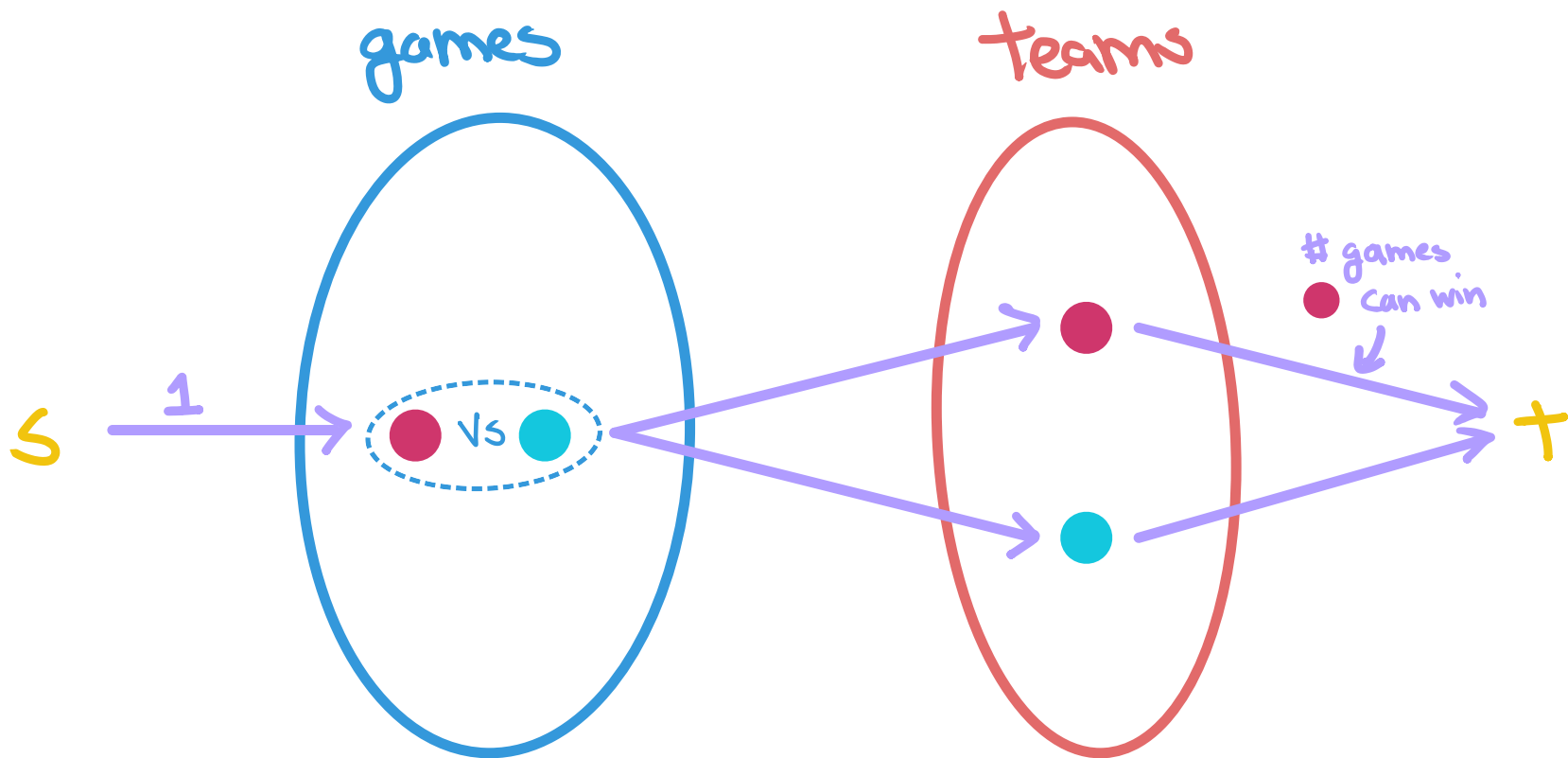


(wins)	(92) New York	(91) ^{te} Baltimore	(91) Toronto	(90) Boston
--------	------------------	---------------------------------	-----------------	----------------

New York	X	1	1	0
Baltimore	1	X	1	1
Toronto	1	1	X	1
Boston	0	1	1	X

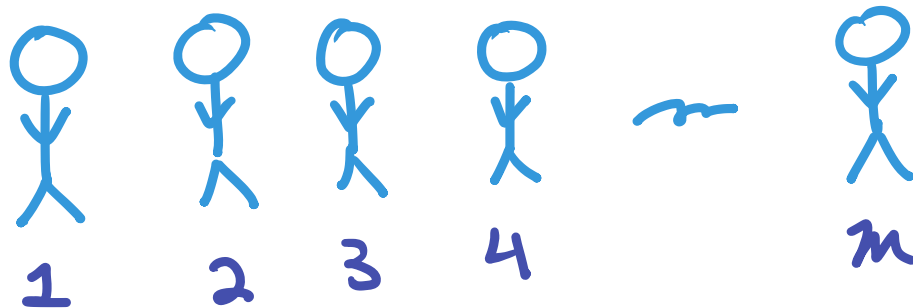
(wins)	(92) New York	(91) Baltimore	(91) Toronto	(90) Boston
New York	×	1	1	0
Baltimore	1	×	1	1
Toronto	1	1	×	1
Boston	0	1	1	×



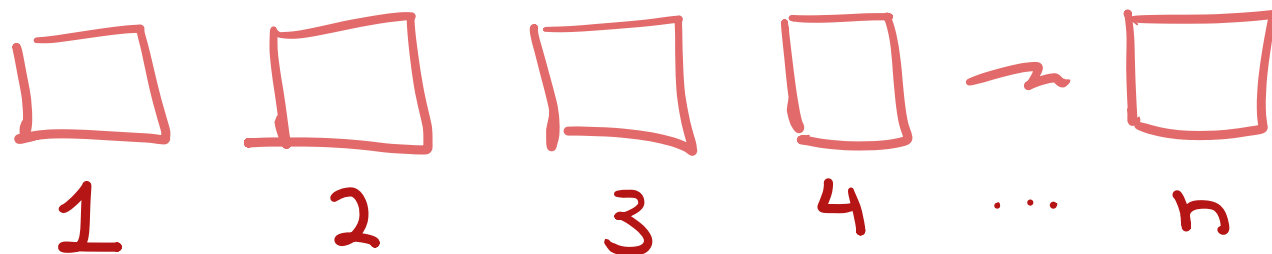


Assignment Problems

m individuals



n jobs

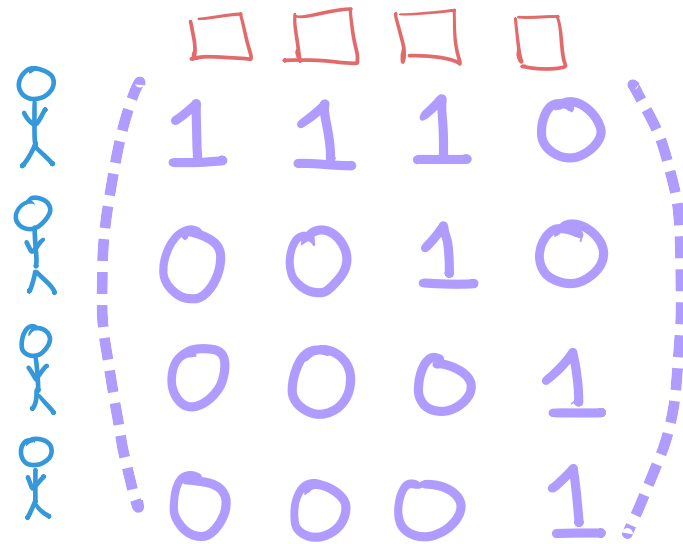


goal: fill as many jobs as possible
w/ individuals

the catch: not everyone is qualified for all jobs

Assignment Problems

"Qualification Matrix"



A qualification matrix diagram. On the left, four stick figures represent people. Above the matrix, four red squares represent tasks. The matrix itself is a 4x4 grid of numbers (1 or 0) enclosed in large purple parentheses. The numbers indicate whether a person is qualified for a specific task.

				
	1	1	1	0
	0	0	1	0
	0	0	0	1
	0	0	0	1

"1" = qualified

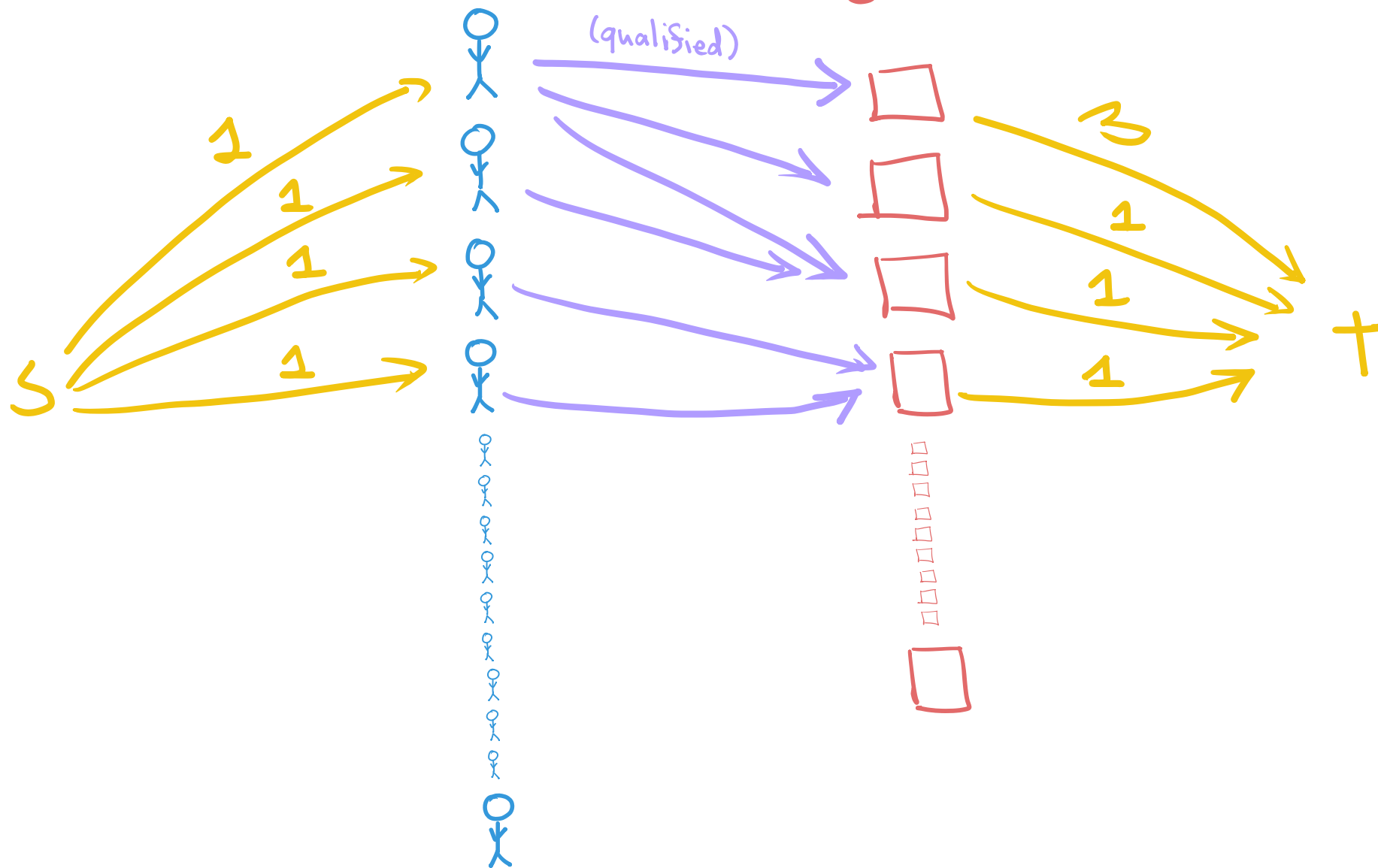
"0" = not qualified

Assignment Problems

	☐	☐	☐	☐
☐	1	1	1	0
☐	0	0	1	0
☐	0	0	0	1
☐	0	0	0	1

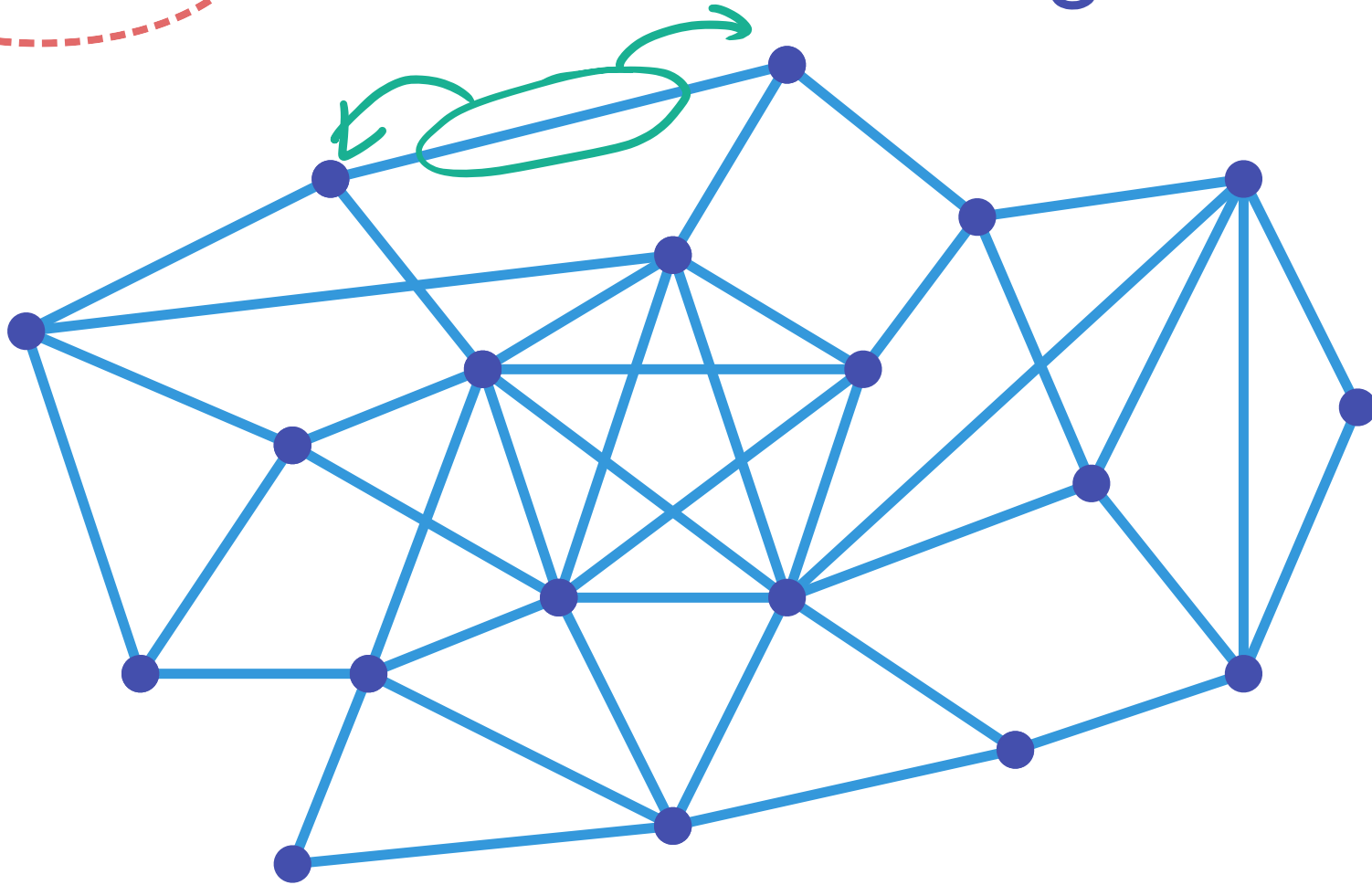
individuals

jobs



Edge
Orientation

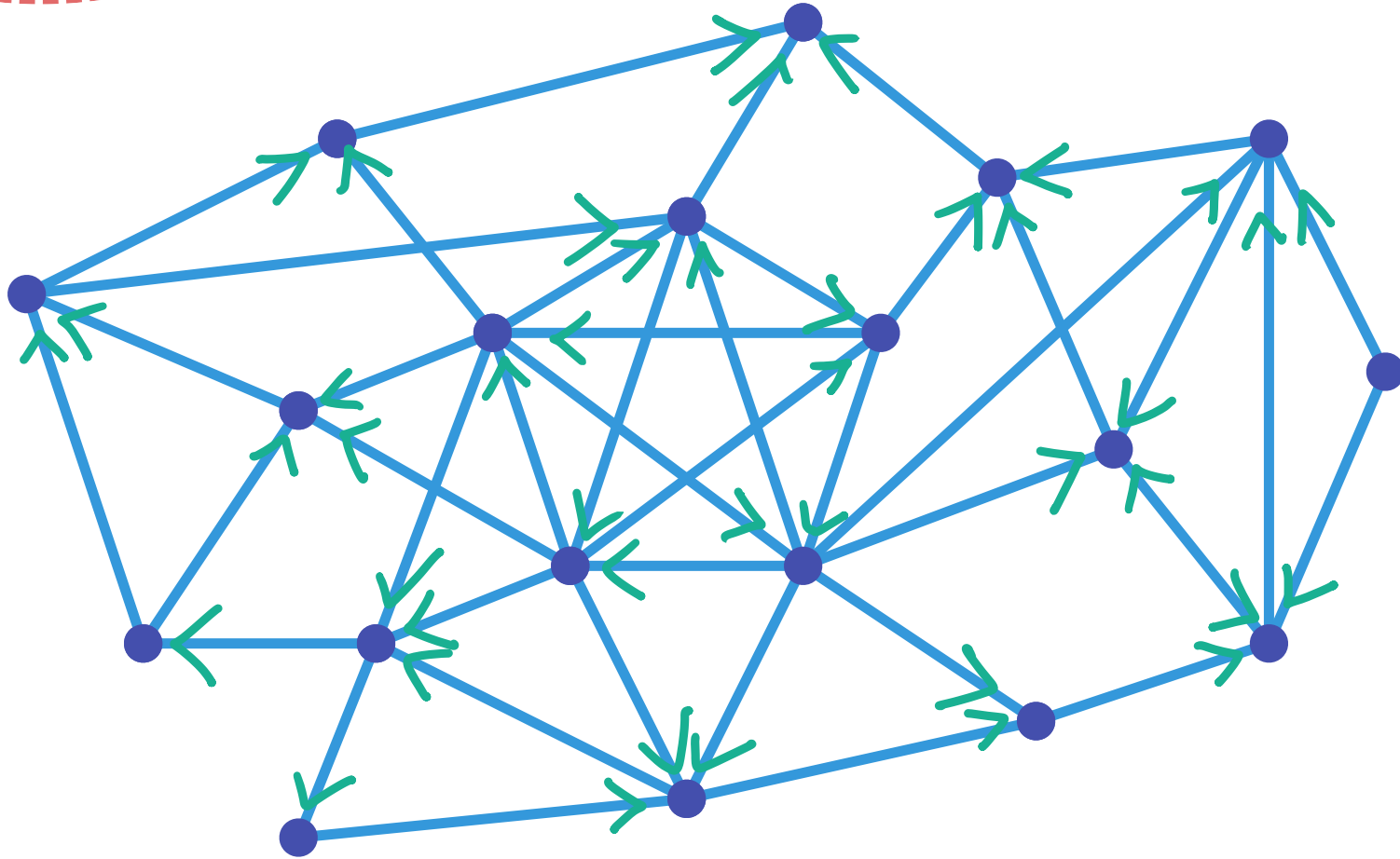
input: undirected graph



Goal: direct the edges to
to minimize the max in-degree

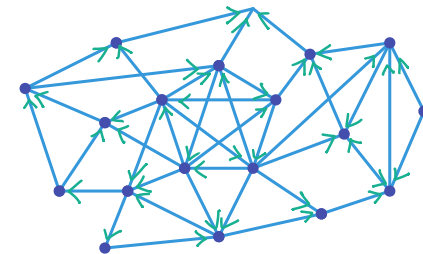
Edge
Orientation

input: undirected graph



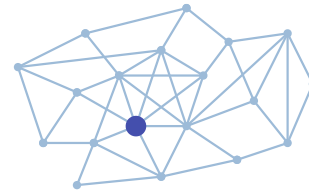
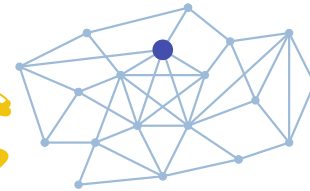
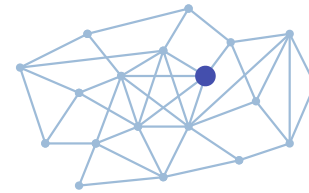
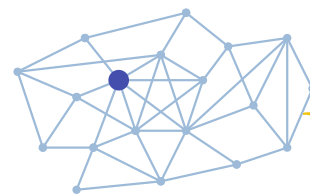
Goal: direct the edges to
minimize the max in-degree

Edge Orientation



edges

vertices



3 $h = \text{target in-degree}$

0

+

S

4

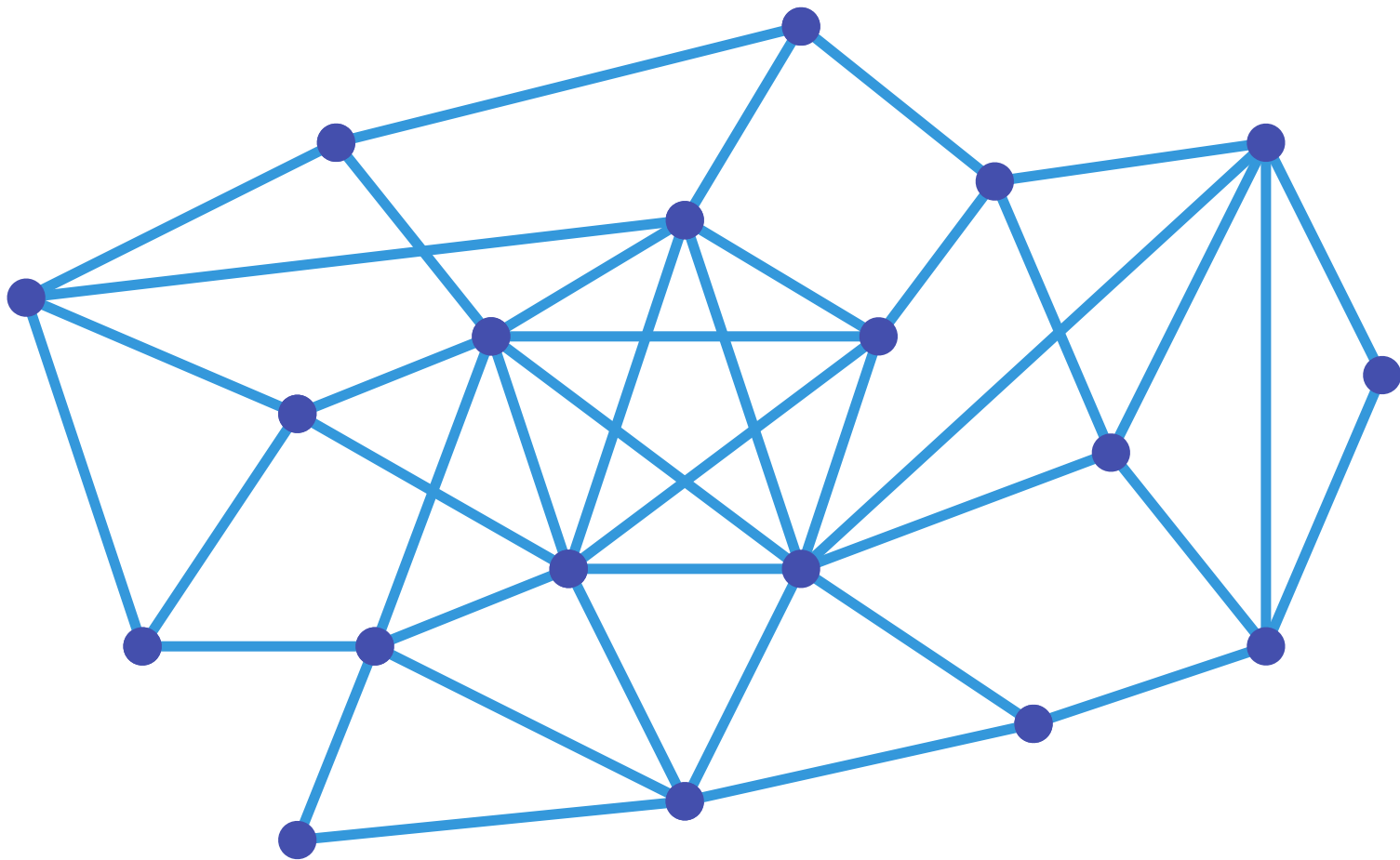
2

4

7

Densest
Subgraph

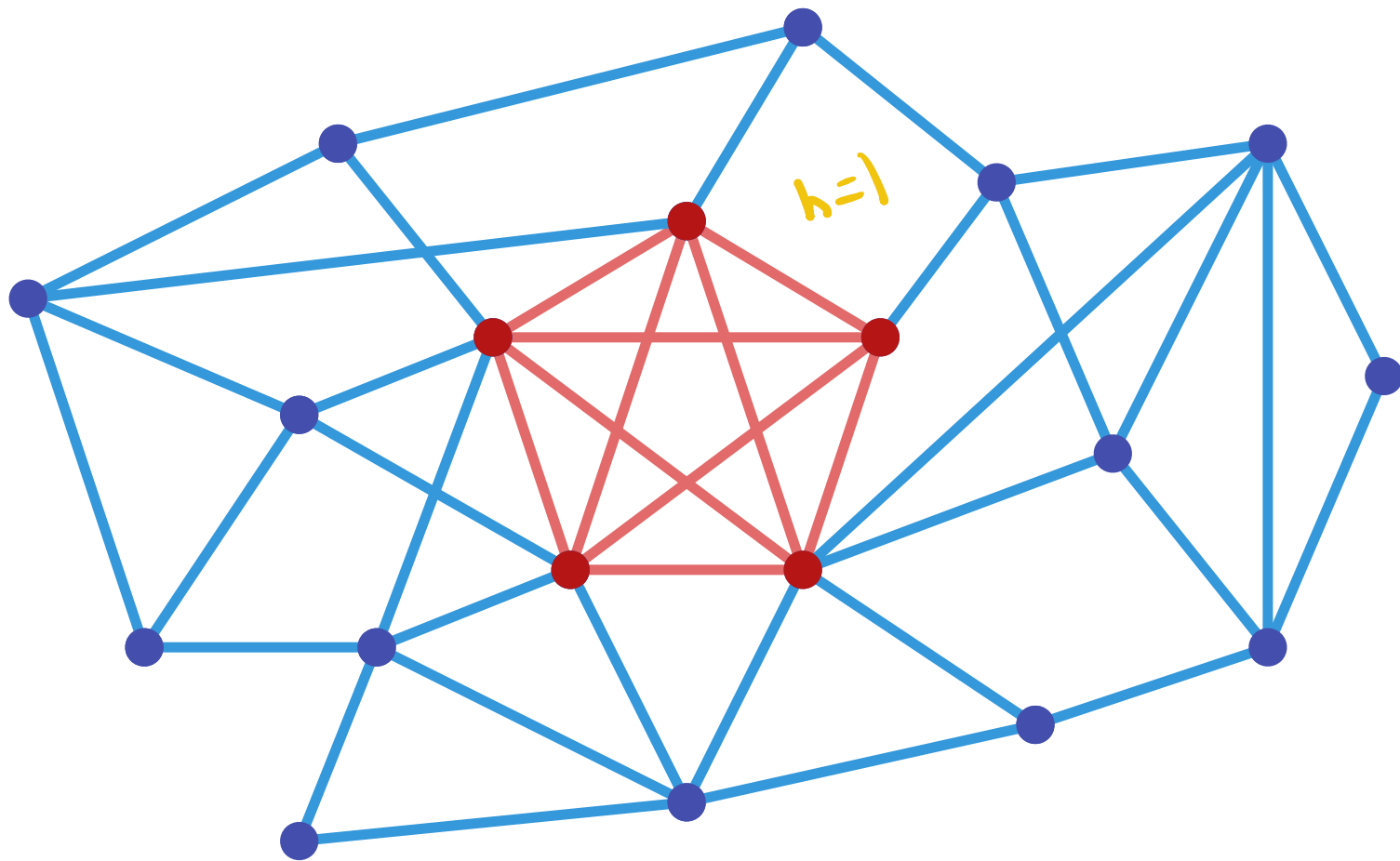
$$\text{"density"} = \frac{\# \text{ edges}}{\# \text{ vertices}}$$



Goal: compute densest subgraph

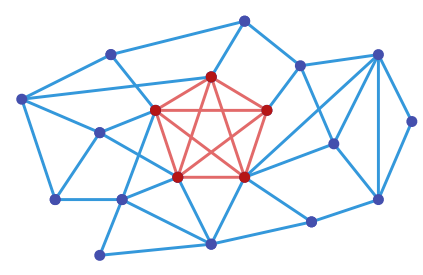
Densest
Subgraph

$$\text{"density"} = \frac{\# \text{ edges}}{\# \text{ vertices}}$$



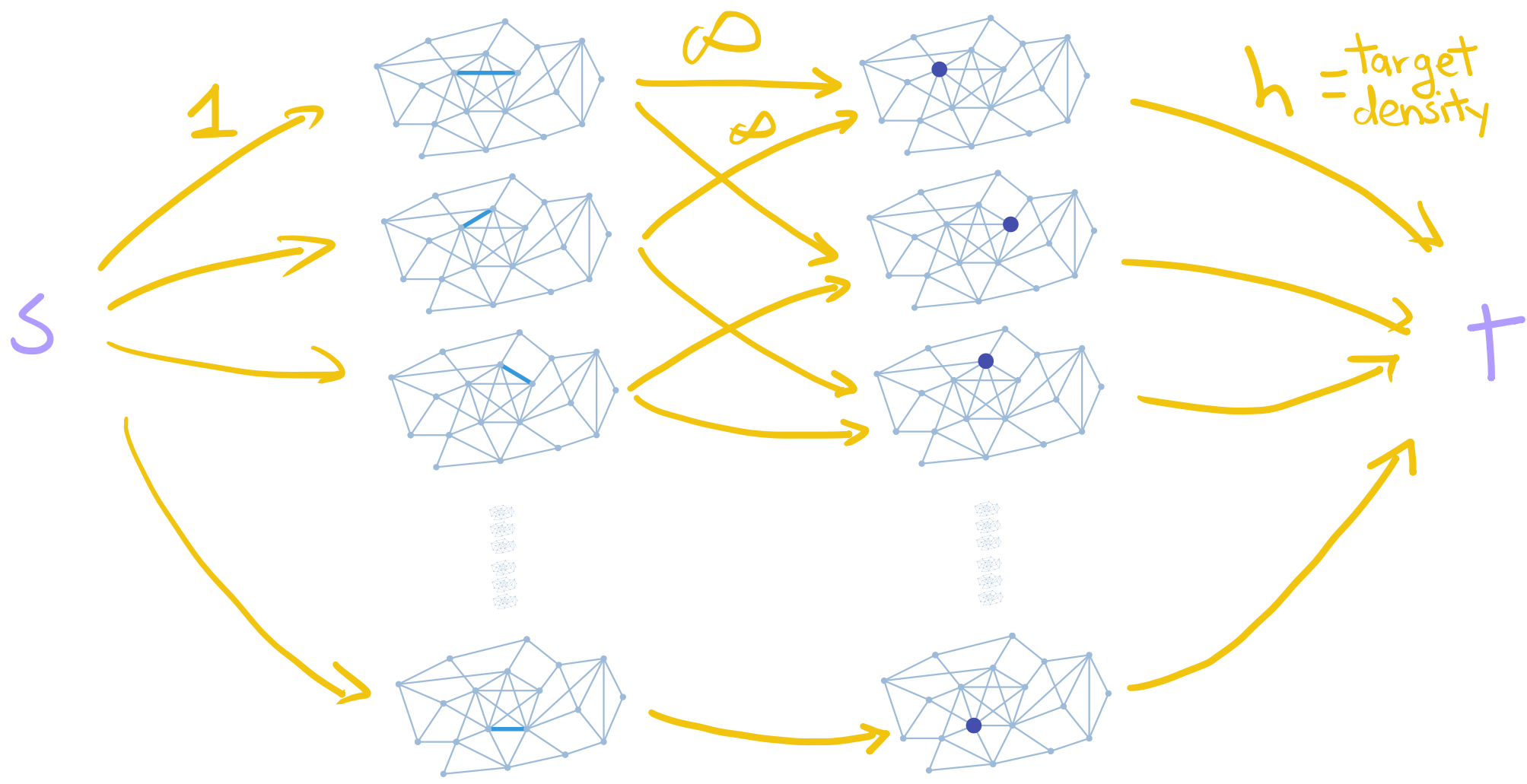
Goal: compute densest subgraph

Densest Subgraph

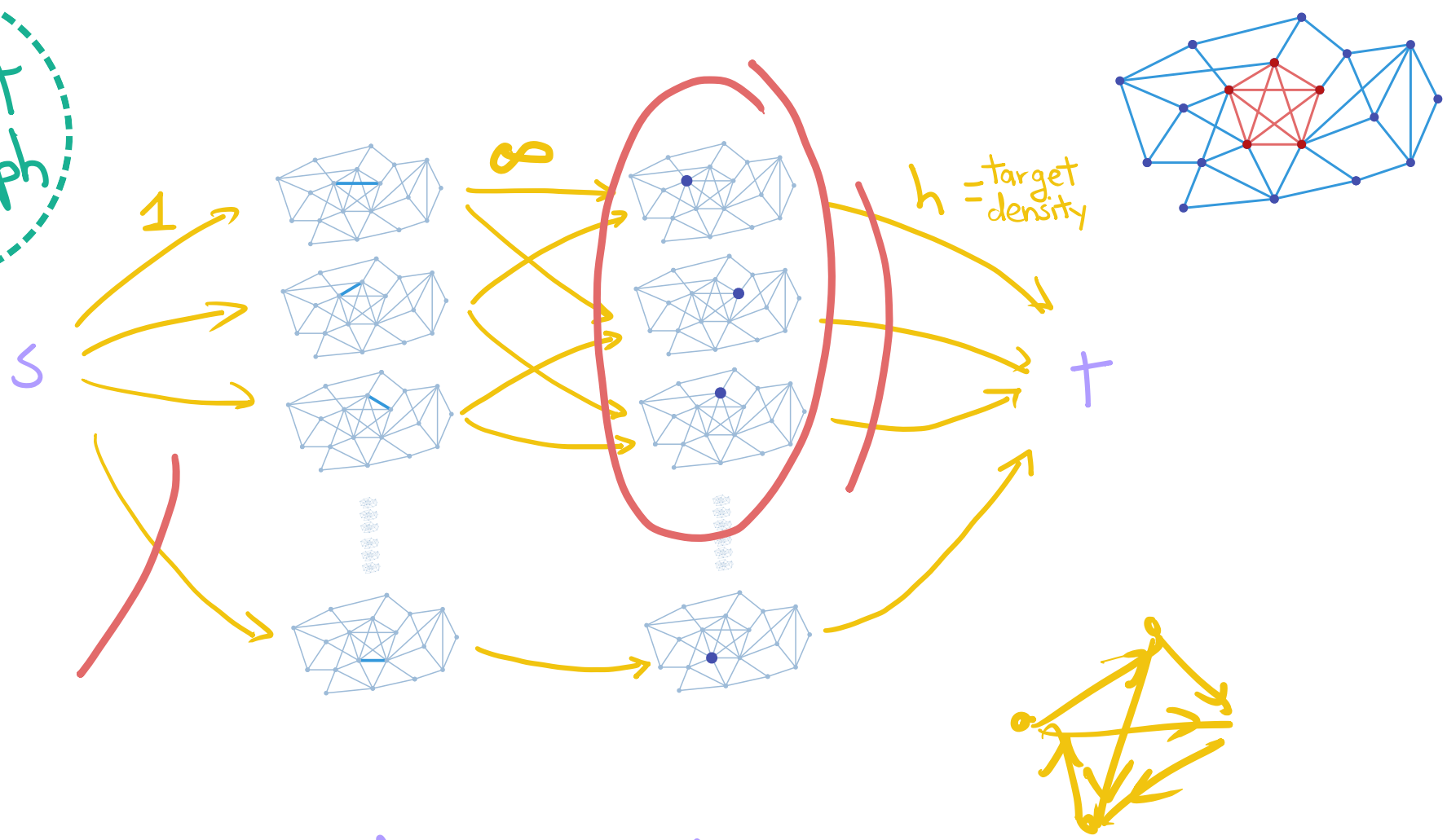


edges

vertices



Densest Subgraph



Claim: $\max \text{ density} \leq h$

$\Leftrightarrow$ there is a flow of size m

Densest Subgraph

Claim: $\max \text{ density} \leq h$
 $\Leftrightarrow$ there is a flow of size m

