Graph partitioning I: Dense Sub-Graphs

Class	Algorithmic Methods of Data Mining
Program	M. Sc. Data Science
University	Sapienza University of Rome
Semester	Fall 2015
Lecturer	Carlos Castillo http://chato.cl/

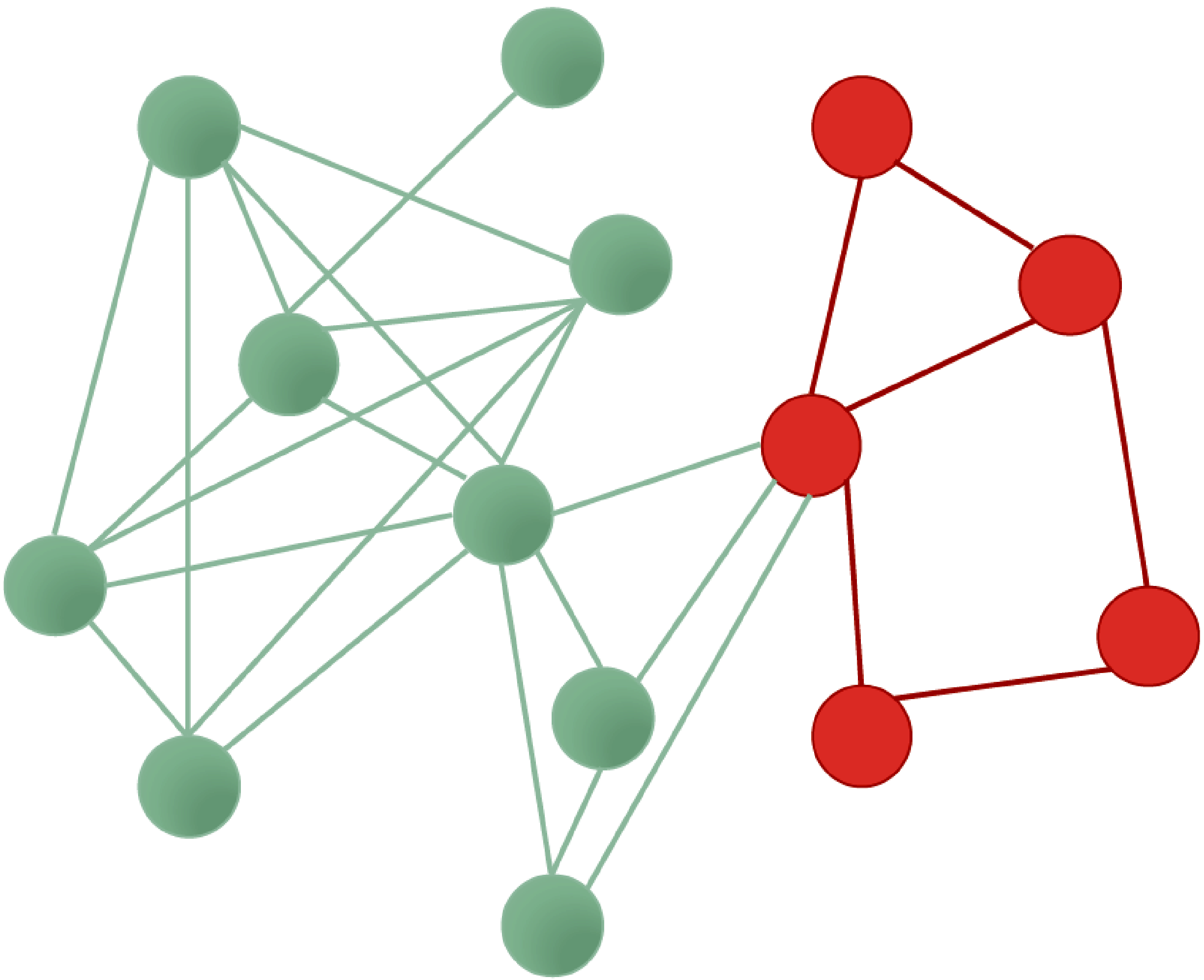
Sources:

- Tutorial by A. Beutel, L. Akoglu, C. Faloutsos [Link]
- Frieze, Gionis, Tsourakakis: “Algorithmic techniques for modeling and mining large graphs (AMAzING)” [Tutorial]
- A survey of algorithms for dense sub-graph discovery [link]

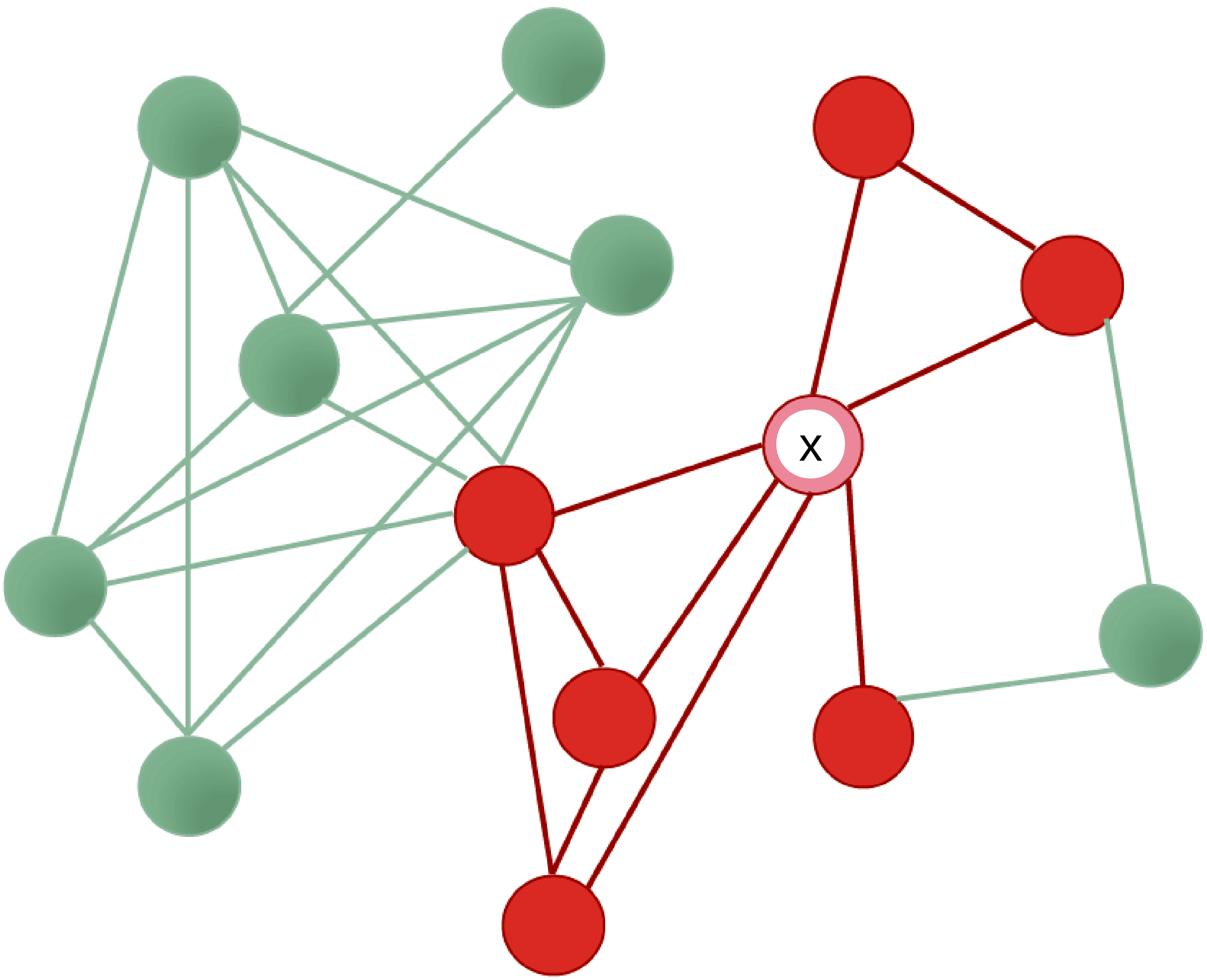
Sub-graphs

Subgraph

Subset of nodes, and edges among those nodes

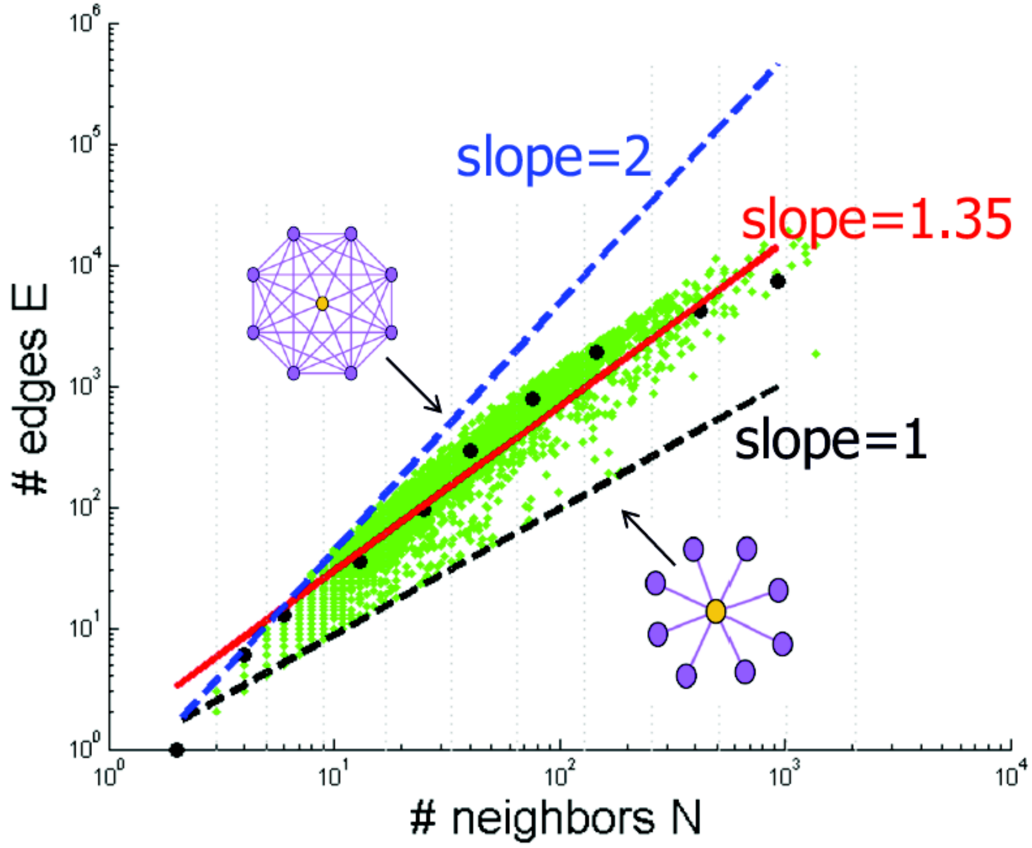


Ego network



Ego graph of node x = neighbors and the links between them

Typical pattern



$$E_i \propto N_i^{\alpha}$$

$$1 \leq \alpha \leq 2$$

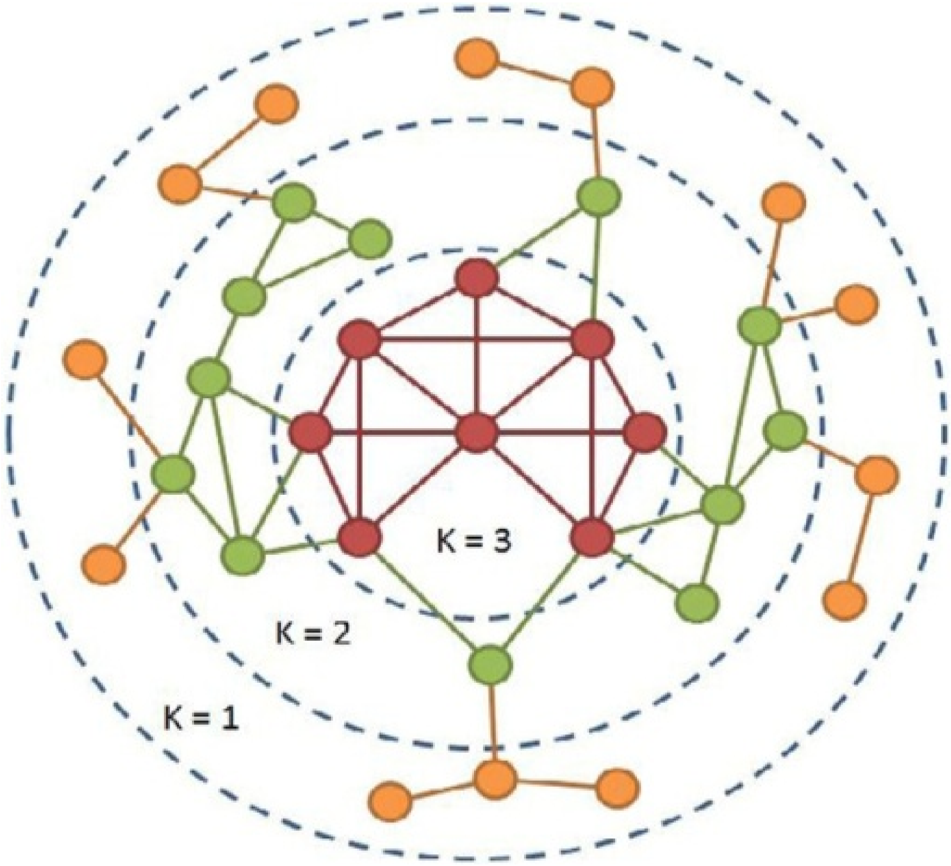
Oddball: Spotting anomalies in weighted graphs
Leman Akoglu, Mary McGlohon, Christos Faloutsos
PAKDD 2010

k-core decomposition

k-core decomposition

- Remove all nodes having degree 1
  - Those are in the 1-core
- Remove all nodes having degree 2 *in the remaining graph*
  - Those nodes are in the 2-core
- Remove all nodes having degree 3 *in the remaining graph*
  - Those nodes are in the 3-core
- Etc.

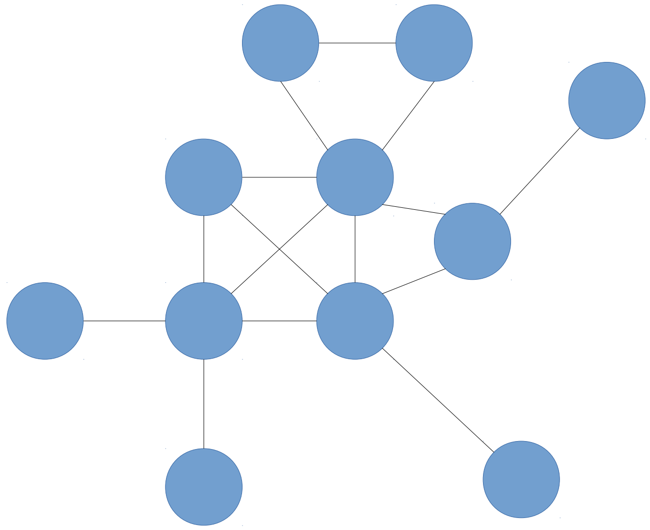
Example



https://openi.nlm.nih.gov/detailedresult.php?img=3368241_fnagi-04-00013-g0001&req=4

Try it!

How many nodes are there in the each core of this graph?



http://www.cpt.univ-mrs.fr/~barrat/NHM.pdf

Graph s-t cuts

Min s-t cut

Given a weighted graph $G(V,E)$, $W{:}E\rightarrow R$

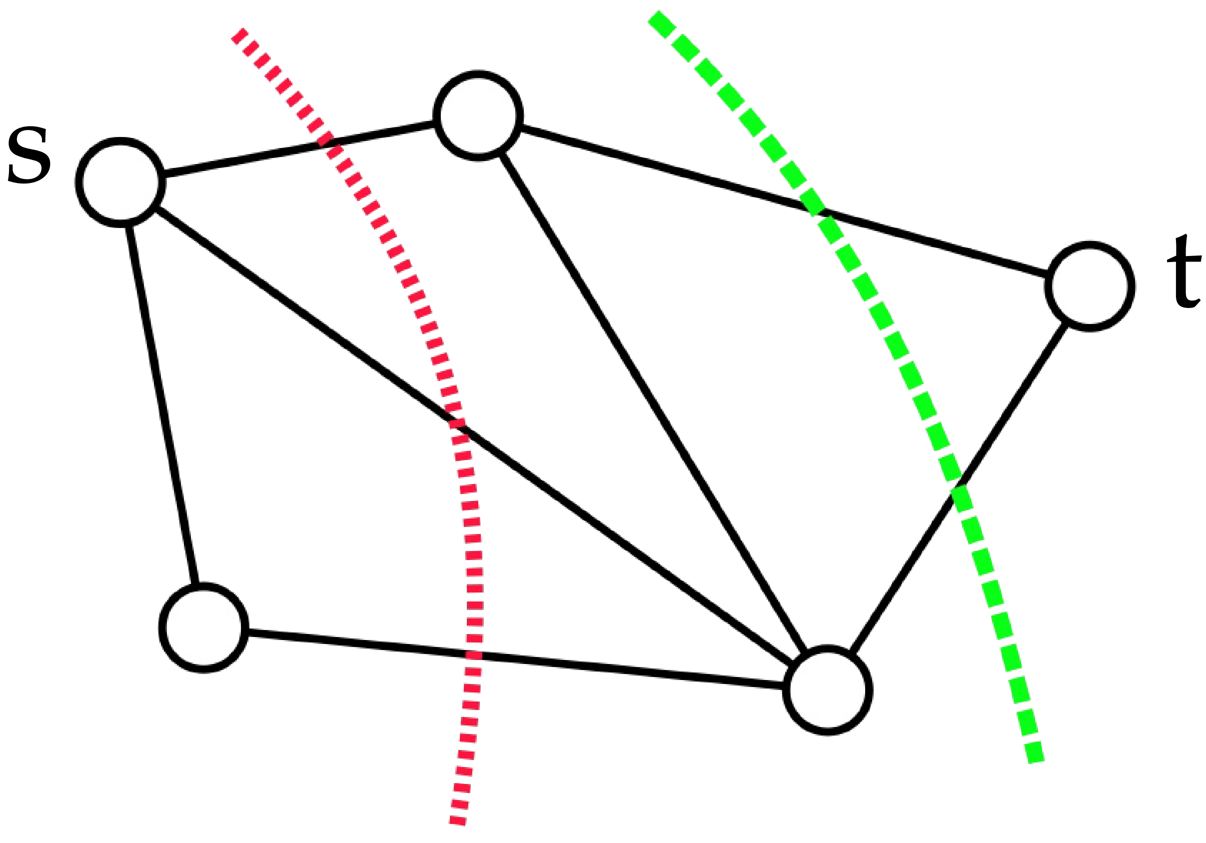
An $(s\text{-}t)$-cut $C=(S,T)$ is such that

- $S \cup T=V$
- $s \in S,\ t \in T$

The cost of a cut is $\sum_{(u,v)\in E, u\in S, v\in T} w(u,v)$

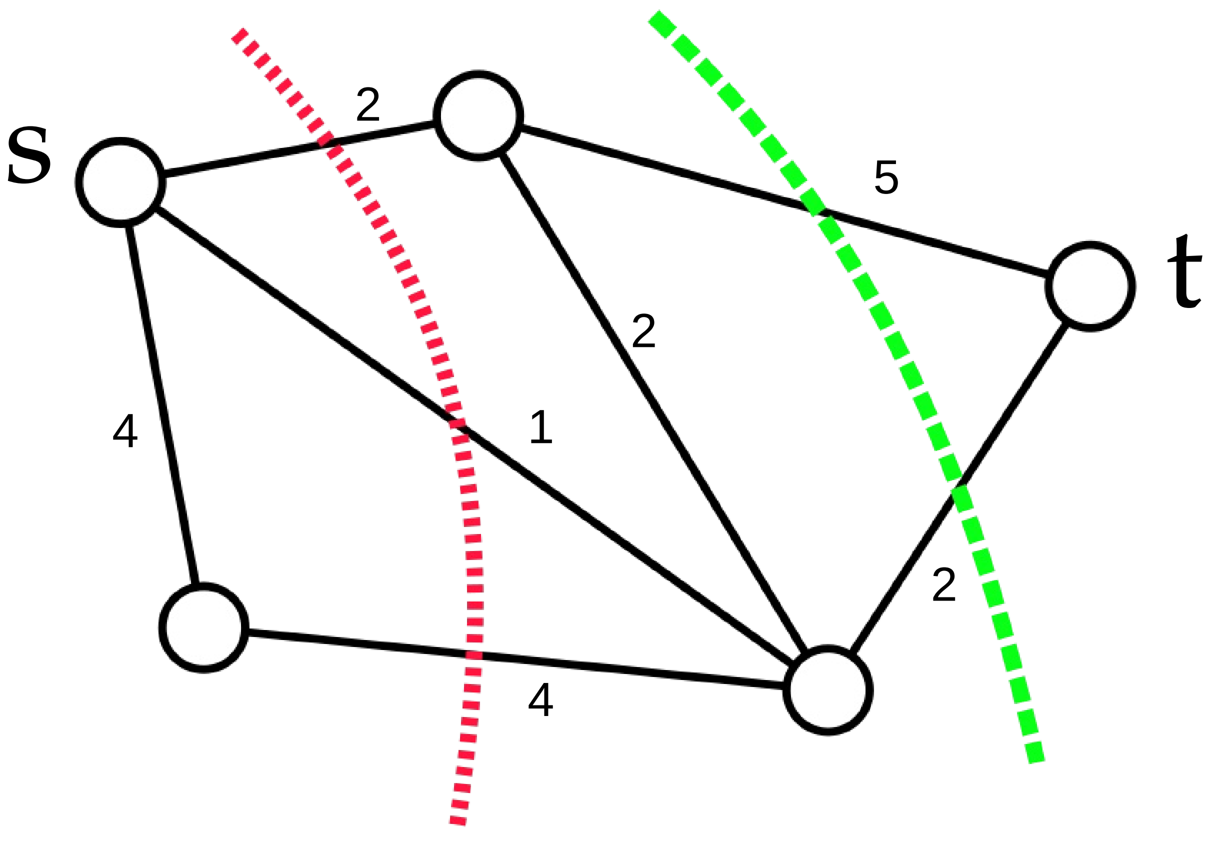
Key problem: given G, s, t, find min weight s-t cut

Example of two s-t cuts



If all edge weights are equal, which one is a smaller cut, the red or the green? Is this the smaller cut in this case?

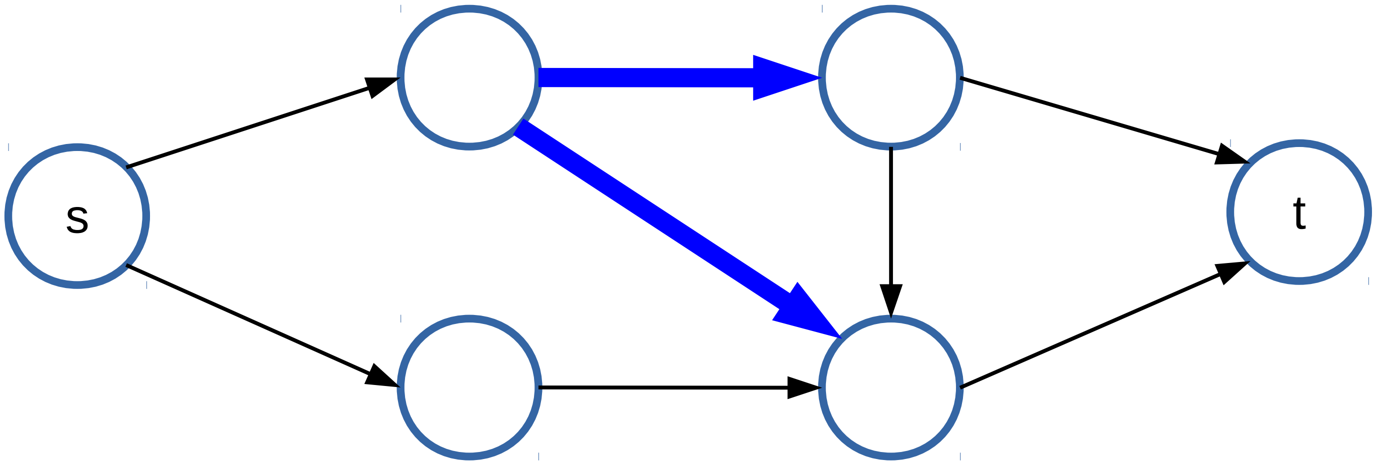
Example of two s-t cuts



What about now, what is the smaller s-t cut in the graph?

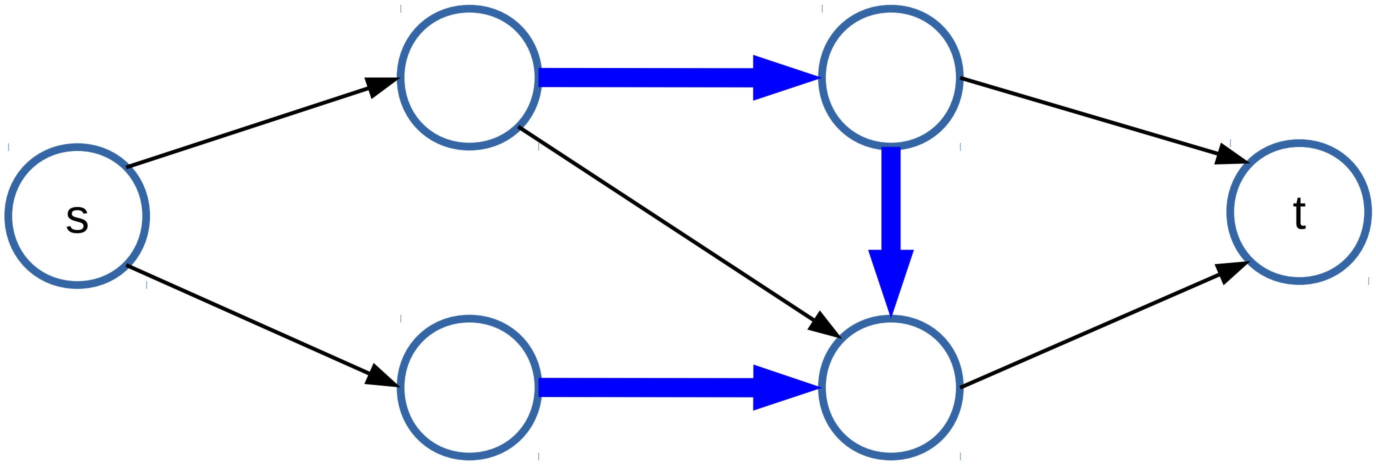
What defines an s-t cut?

- Can I take an arbitrary set of edges and claim it is an s-t cut?
- Is this an s-t cut? Why? Why not?



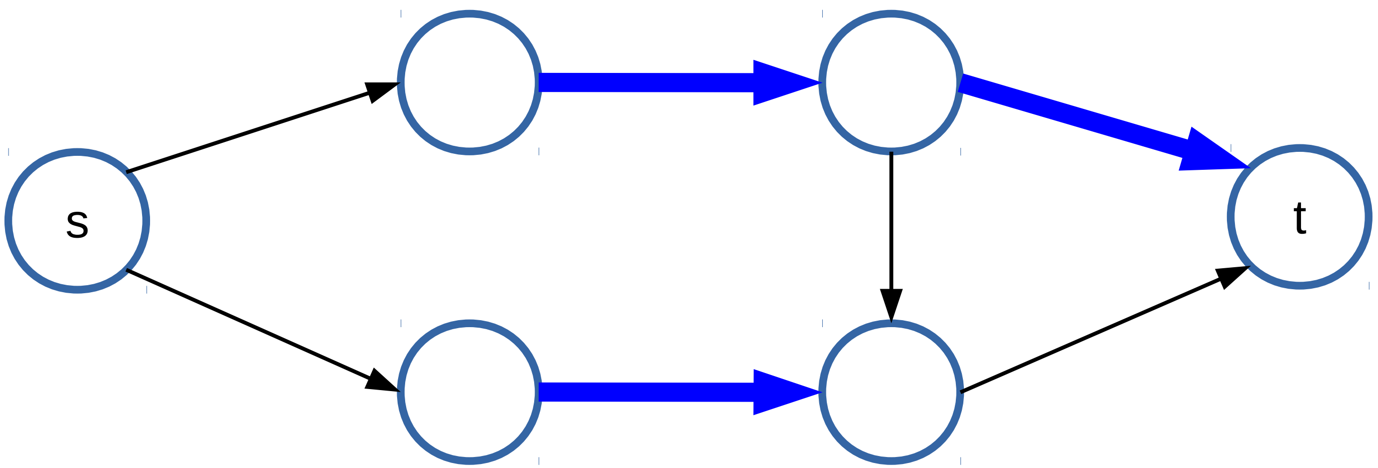
What defines an s-t cut?

- Can I take an arbitrary set of edges and claim it is an s-t cut?
- Is this an s-t cut? Why? Why not?



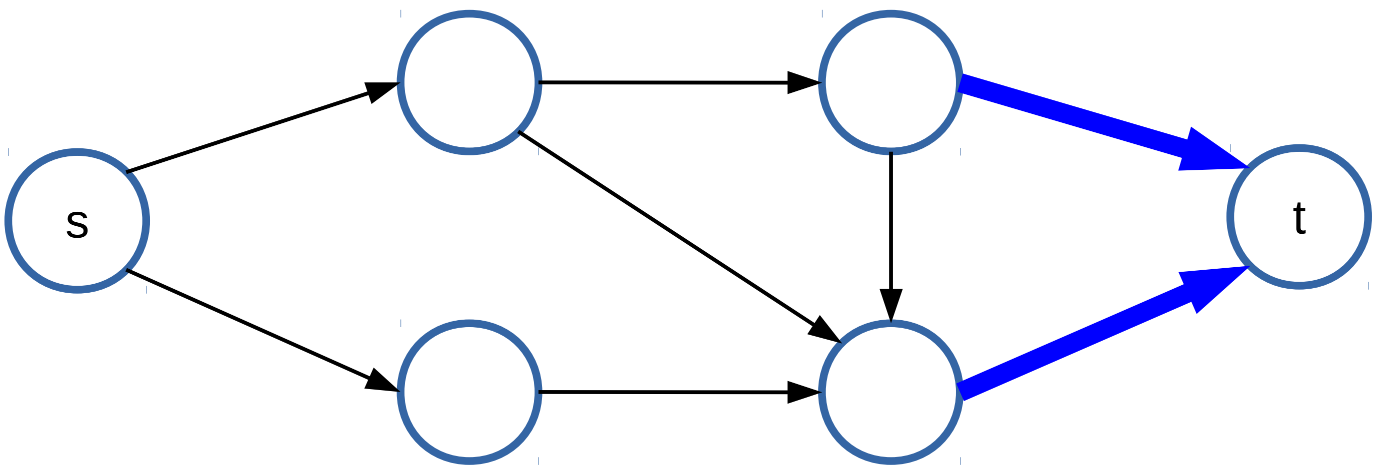
What defines an s-t cut?

- Can I take an arbitrary set of edges and claim it is an s-t cut?
- Is this an s-t cut? Why? Why not?



What defines an s-t cut?

- Can I take an arbitrary set of edges and claim it is an s-t cut?
- Is this an s-t cut? Why? Why not?



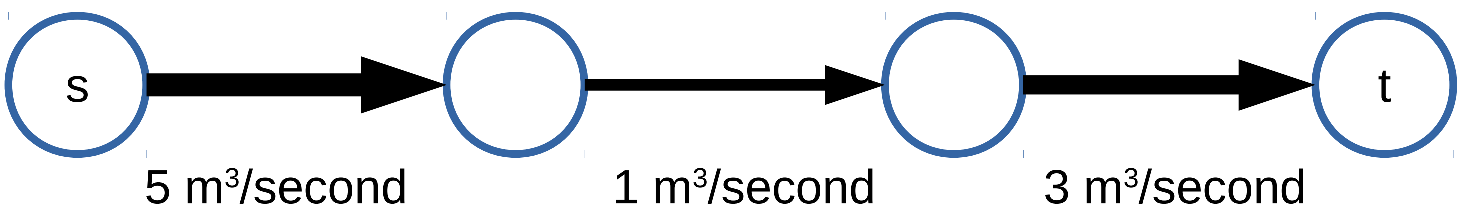
Simple s-t paths and s-t cuts

- For a subset of edges S of a graph to be a cut, every simple path between s and t should contain exactly one edge in E

Maximum flows

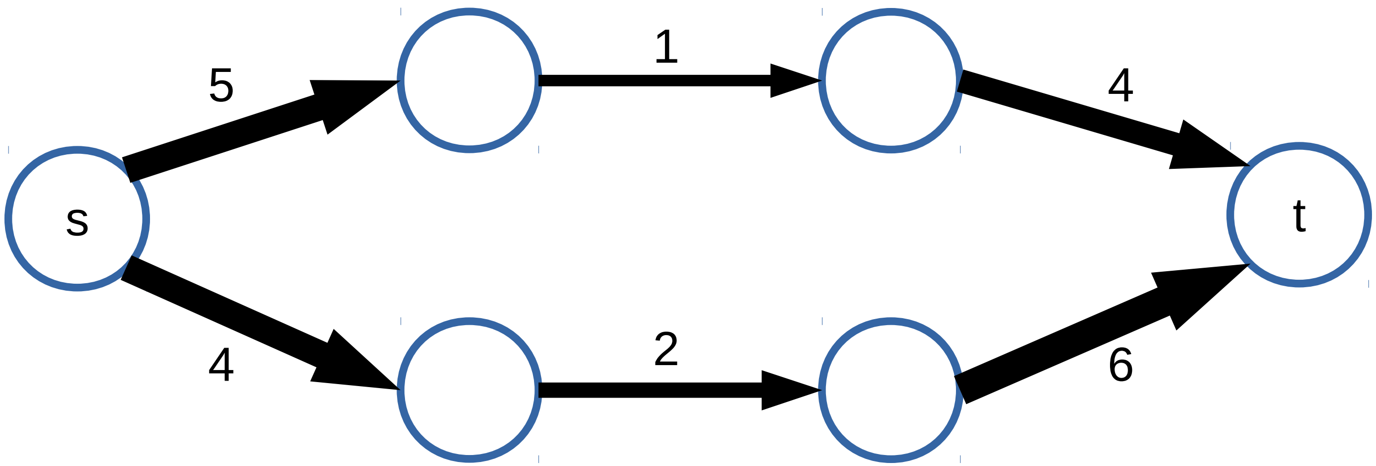
Maximum flow: example 1

- If edge weights were capacities, what is the maximum flow that can be sent from s to t?



Maximum flow: example 2

- If edge weights were capacities, what is the maximum flow that can be sent from s to t?



Maximum flow problem

- What is the maximum “flow” that can be carried from s to t?
  - Think of edge weights as capacities (e.g. m^3/s of water)
- What is the flow of an edge?
  - The amount sent through that edge (an assignment)
- What is the net flow of a node?
  - The amount exiting the node minus the amount entering the node

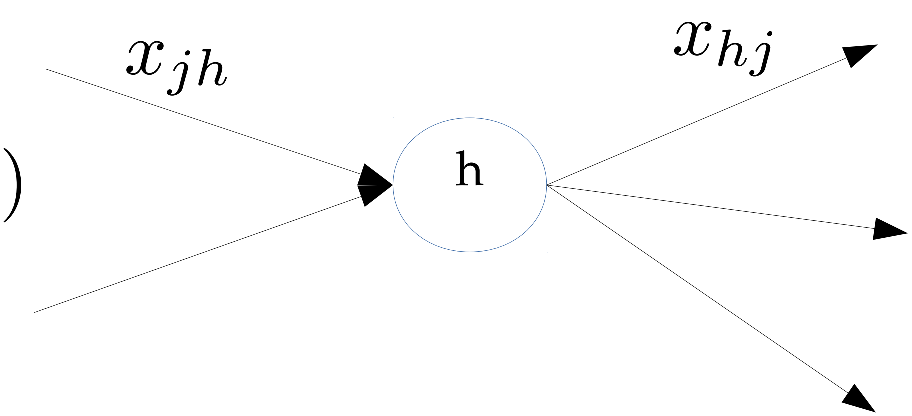
Formulating the max flow problem

- The flow through each edge should be $\leq k_{ij}$
- Net flow node h = OUT(h) – IN(h)
- Node *s* should have positive flow *v*
- Node *t* should have negative flow *-v*
- *What should be the flow of the other nodes?*

Formulating the max flow problem

- Net flow node h = OUT(h) - IN(h)
- Node *s* should have positive flow *v*
- Node *t* should have negative flow *-v*

$2 \text{ nodes in } \delta^-(h)$



$3 \text{ nodes in } \delta^+(h)$

- *What should be the flow of a node?*

$$\sum_{(h,j)\in\delta^+(h)} x_{hj} - \sum_{(i,h)\in\delta^-(h)} x_{ij} = \ ?$$

Max flow as a linear program

$$
\begin{aligned}
\max \quad & v & (1)\\
\sum_{(s,j)\in\delta^+(s)} x_{sj} &= v & (2)\\
-\sum_{(i,t)\in\delta^-(t)} x_{it} &= -v & (3)\\
\sum_{(h,j)\in\delta^+(h)} x_{hj} - \sum_{(i,h)\in\delta^-(h)} x_{ih} &= 0, \quad h \in N - \{s,t\} & (4)\\
x_{ij} &\leq k_{ij} \quad (i,j) \in A & (5)\\
x_{ij} &\geq 0 \quad (i,j) \in A & (6)
\end{aligned}
$$

http://www.dii.unisi.it/~agnetis/mincutENG.pdf

Writing the dual

$$\max \quad v \tag{1}$$

$$\sum_{(s,j)\in\delta^+(s)} x_{sj} = v \qquad \text{variable } u_s \tag{2}$$

$$-\sum_{(i,t)\in\delta^-(t)} x_{it} = -v \qquad \text{variable } u_t \tag{3}$$

$$\sum_{(h,j)\in\delta^+(h)} x_{hj} - \sum_{(i,h)\in\delta^-(h)} x_{ih} = 0, \quad h \in N - \{s,t\} \qquad \text{variables } u_j \tag{4}$$

$$x_{ij} \leq k_{ij} \quad (i,j) \in A \qquad \text{variables } y_{ij} \tag{5}$$

$$x_{ij} \geq 0 \quad (i,j) \in A \tag{6}$$

Writing the dual

- *Remember: the infimum of the solutions of the dual is the supremum of the solutions of primal*

$$\min \sum_{(i,j)\in A} k_{ij} y_{ij}$$

$$\begin{aligned} u_i - u_j + y_{ij} &\geq 0, (i,j) \in A \\ -u_s + u_t &= 1 \end{aligned}$$

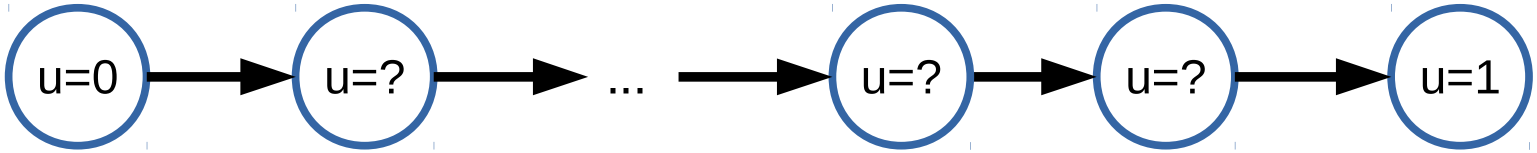
$$y_{ij} \geq 0$$

- *Variables u_i don't enter the objective, only their difference is in the constraints*
- *We can set them arbitrarily, in particular $u_s = 0$, $u_t = 1$*

Dual (after simplification)

$$\begin{aligned} min \sum_{(i,j)\in A} k_{ij} y_{ij} & & \\ u_i - u_j + y_{ij} & \geq & 0, (i,j) \in A \\ y_{ij} & \geq & 0 \\ u_s = 0, u_t = 1 & & \end{aligned}$$

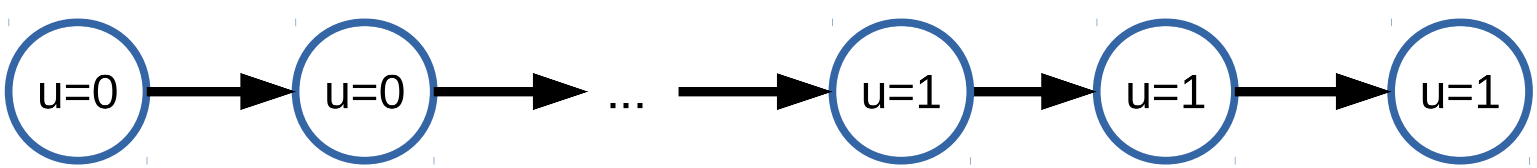
- Observe what happens with the values of u in every path going from s to t



Dual (after simplification)

$$\begin{aligned} min \sum_{(i,j)\in A} k_{ij}y_{ij} & & \\ u_i - u_j + y_{ij} &\geq& 0, (i,j) \in A \\ y_{ij} &\geq& 0 \\ u_s = 0, u_t = 1 & & \end{aligned}$$

- Given these constraints, the sequence **must** increase, and can only increase **once**



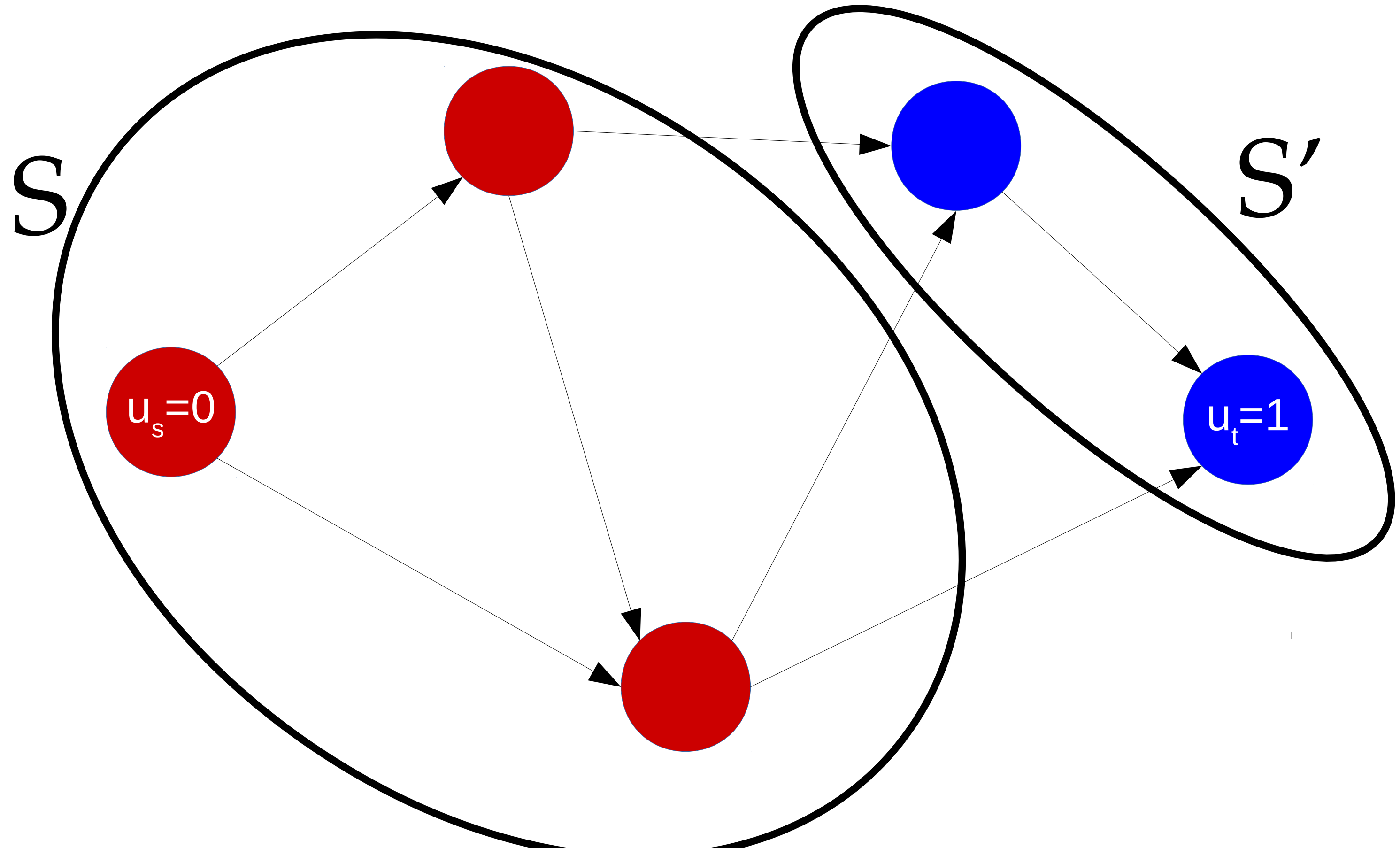
Dual (after simplification)

$$
\begin{aligned}
min \sum_{(i,j)\in A} k_{ij} y_{ij} & & \\
u_i - u_j + y_{ij} &\geq& 0, (i,j) \in A \\
y_{ij} &\geq& 0 \\
u_s = 0, u_t = 1 & &
\end{aligned}
$$

- Important theorem: **every feasible solution can be written as a cut (S, S')**

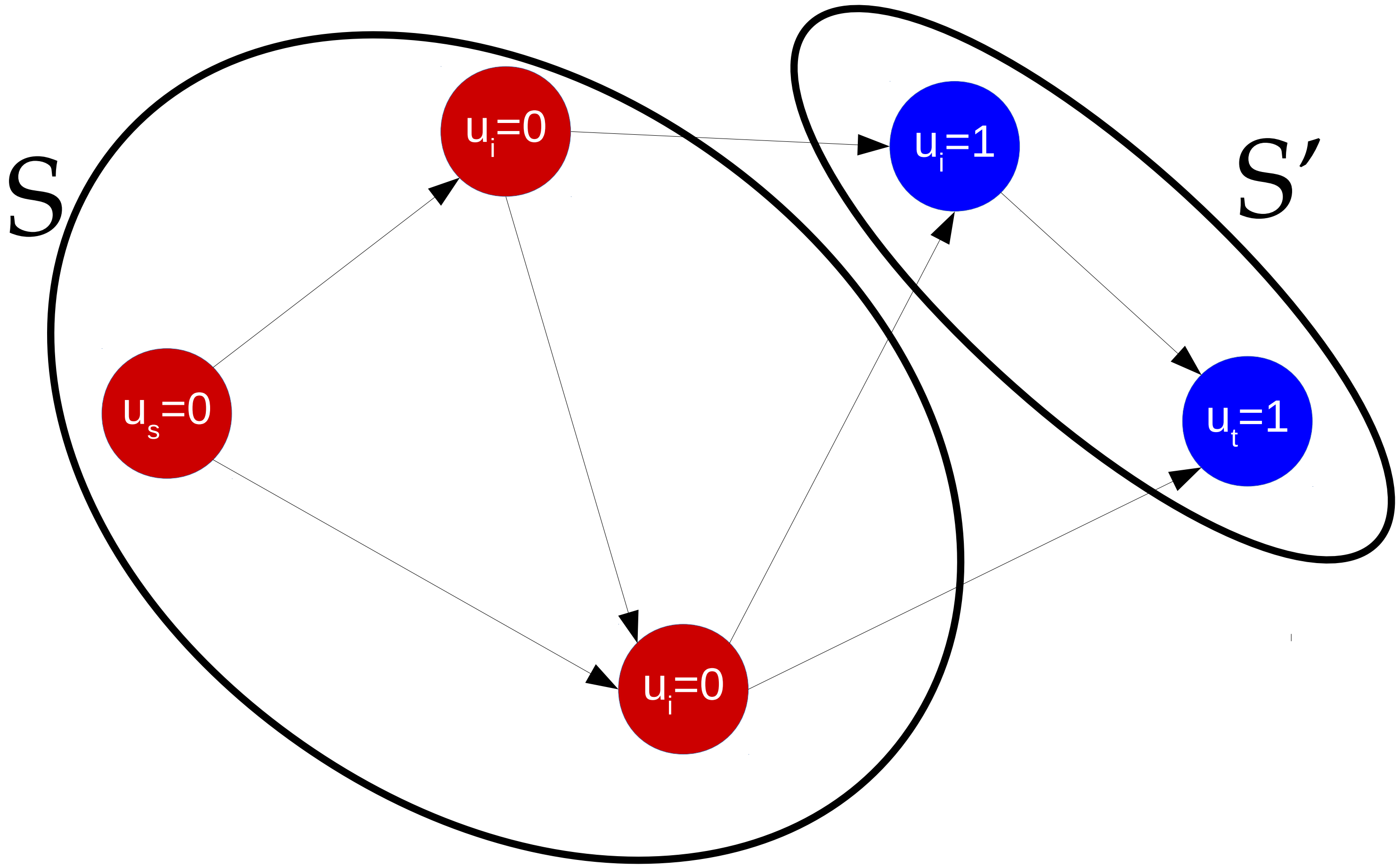
Dual solutions are cuts

- Every feasible solution of the dual has the form of a cut (S, S')



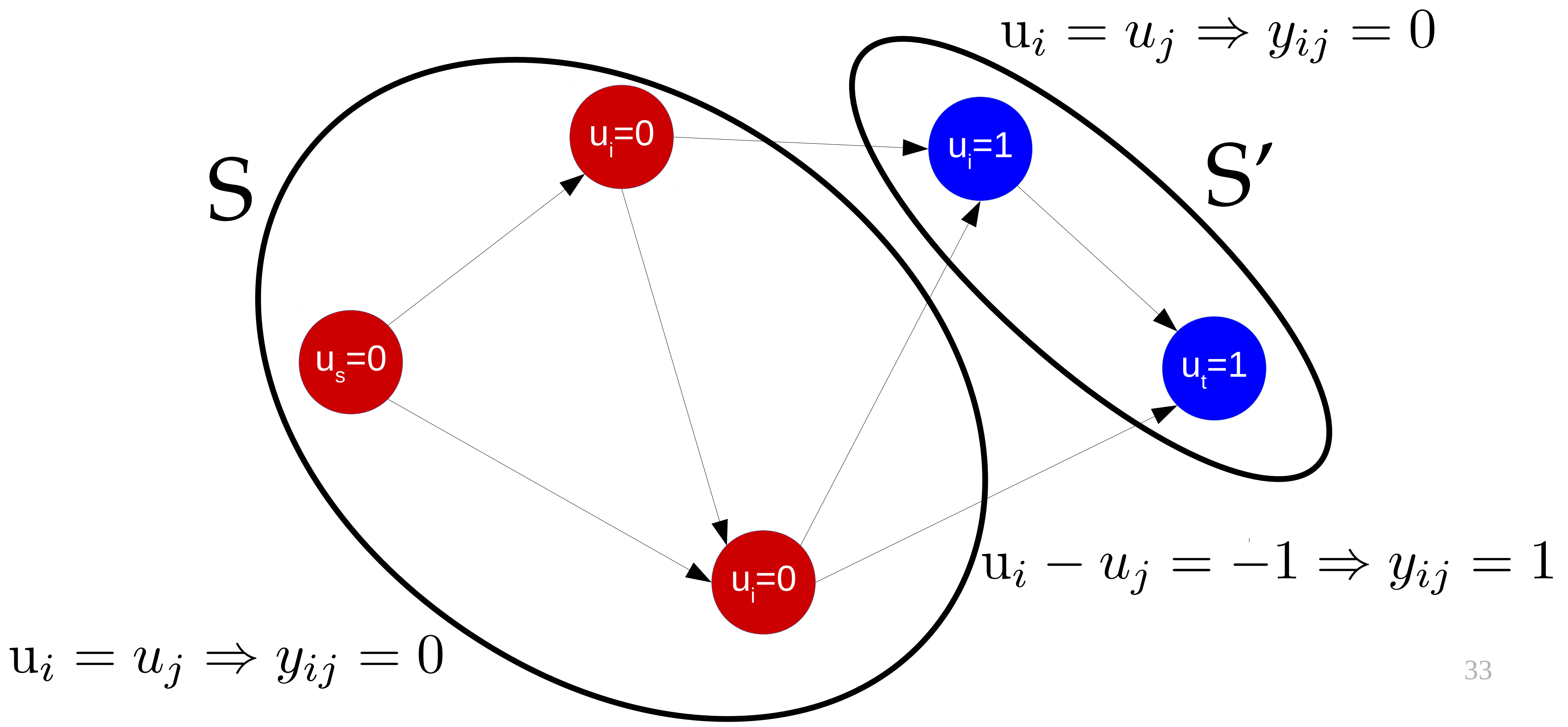
Dual solutions are cuts

- Every feasible solution of the dual has the form of a cut (S, S')



Dual solutions are (s-t)-cuts

$u_i - u_j + y_{ij} \geq 0$ and remember we're trying to minimize $\sum k_{ij} y_{ij}$



One more thing about the solution

$$\begin{aligned} min \sum_{(i,j)\in A} k_{ij} y_{ij} & & \\ u_i - u_j + y_{ij} & \geq & 0, (i,j) \in A \\ y_{ij} & \geq & 0 \\ u_s = 1, u_t = 0 & & \end{aligned}$$

y_{ij} is a dual variable corresponding to primal constraint $x_{ij} \leq k_{ij}$
If y_{ij} is non-zero, then the corresponding constraint is tight

Summarizing

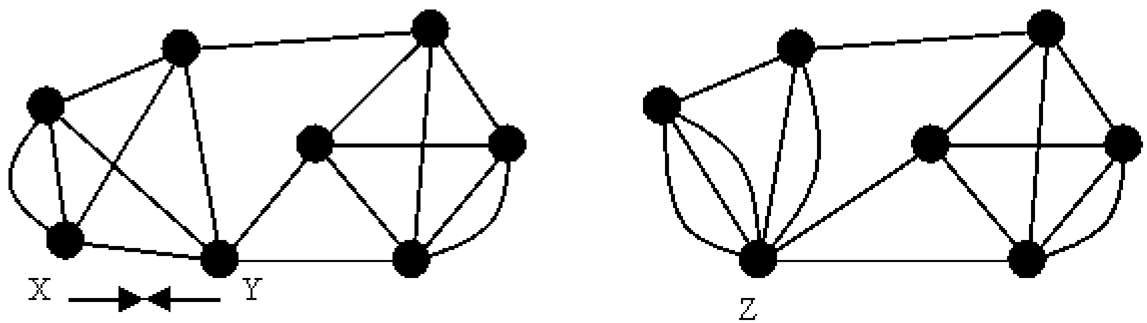
- Min-cut and Max-flow are equivalent problems
  - Their solutions are also equal: the value of the maximum flow is equivalent to the minimum cut
- Think of a chain that breaks at the weakest link
- Both can be solved exactly in polynomial time

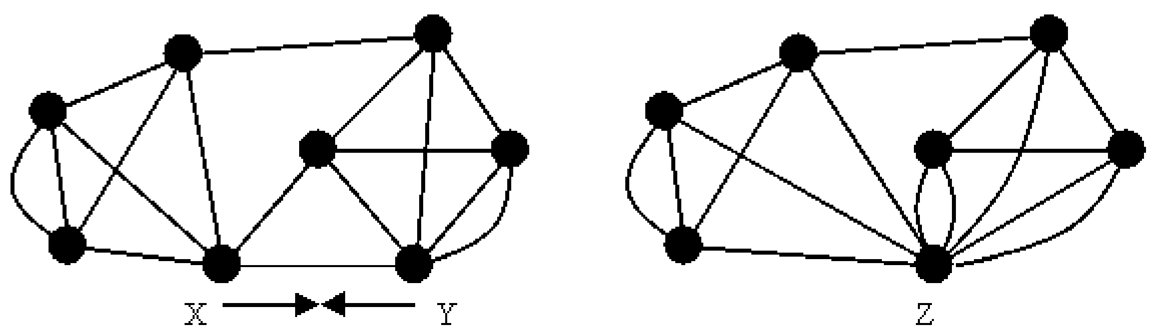
A simple randomized algorithm

- Pick an edge at random (u,v)
- Merge u and v in new vertex uv
- Edges between u and v are removed
- Edges pointing to u or v are added as multi-edges to vertex uv
- When only s and t remain, the multi-edges are a cut, probably the minimum one

http://www.cs.berkeley.edu/~jfc/cs174lecs/lec18/lec18.html

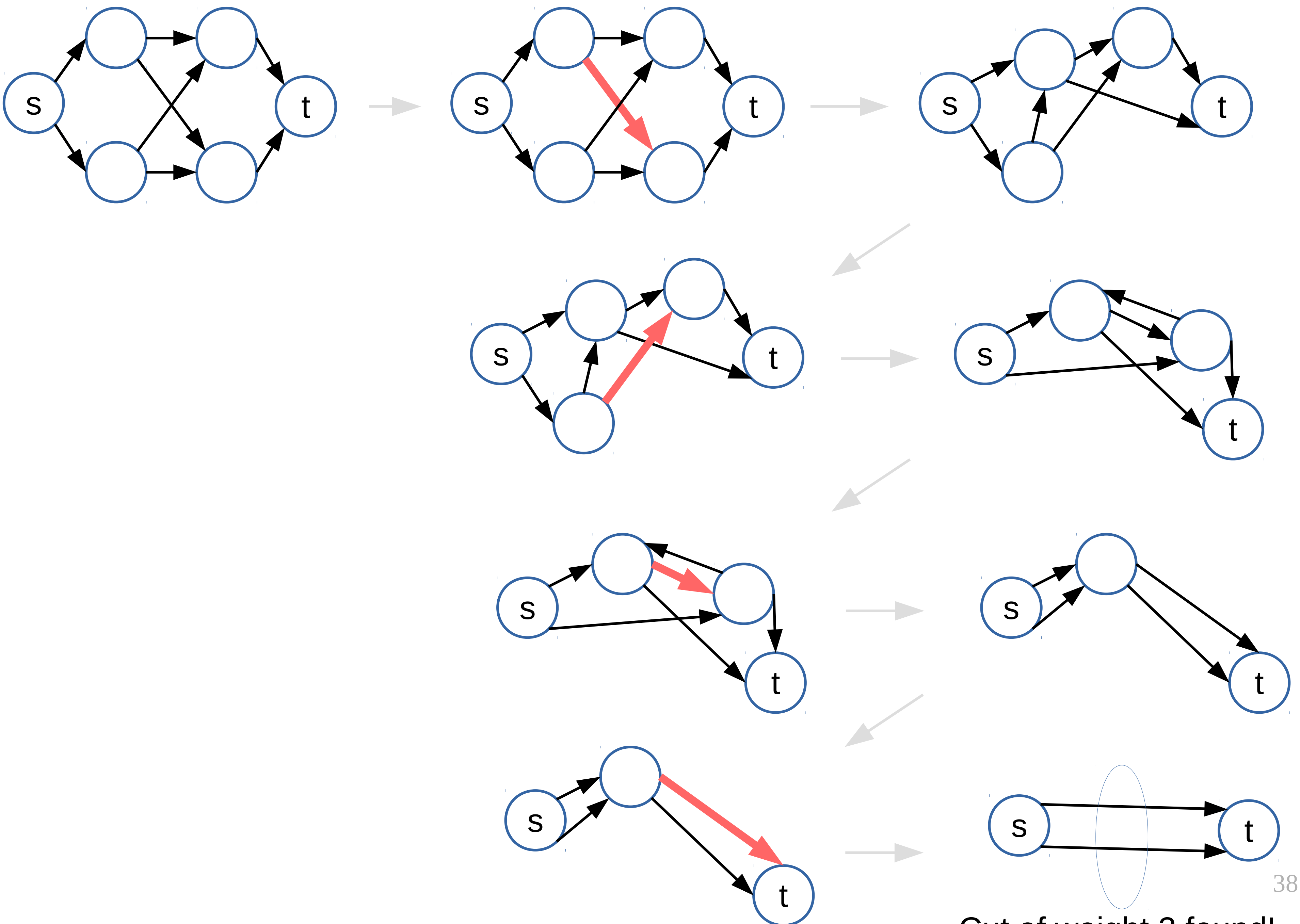
Example contractions





http://www.cs.berkeley.edu/~jfc/cs174lecs/lec18/lec18.html

Example run



Cut of weight 2 found!

Randomized algorithm might miss the min cut

- Multiple runs are required
- The probability that this finds the min cut in one run is about 1/log(n), so O(log n) iterations are required to find min cut
- Each iteration costs $O(n^2 \log n)$
- $O(n^2 \log^2 n)$ operations needed to find min cut
- Exact algorithm: $O(n^3 + n^2 \log n)$; the n^3 is because of |V||E| operations required

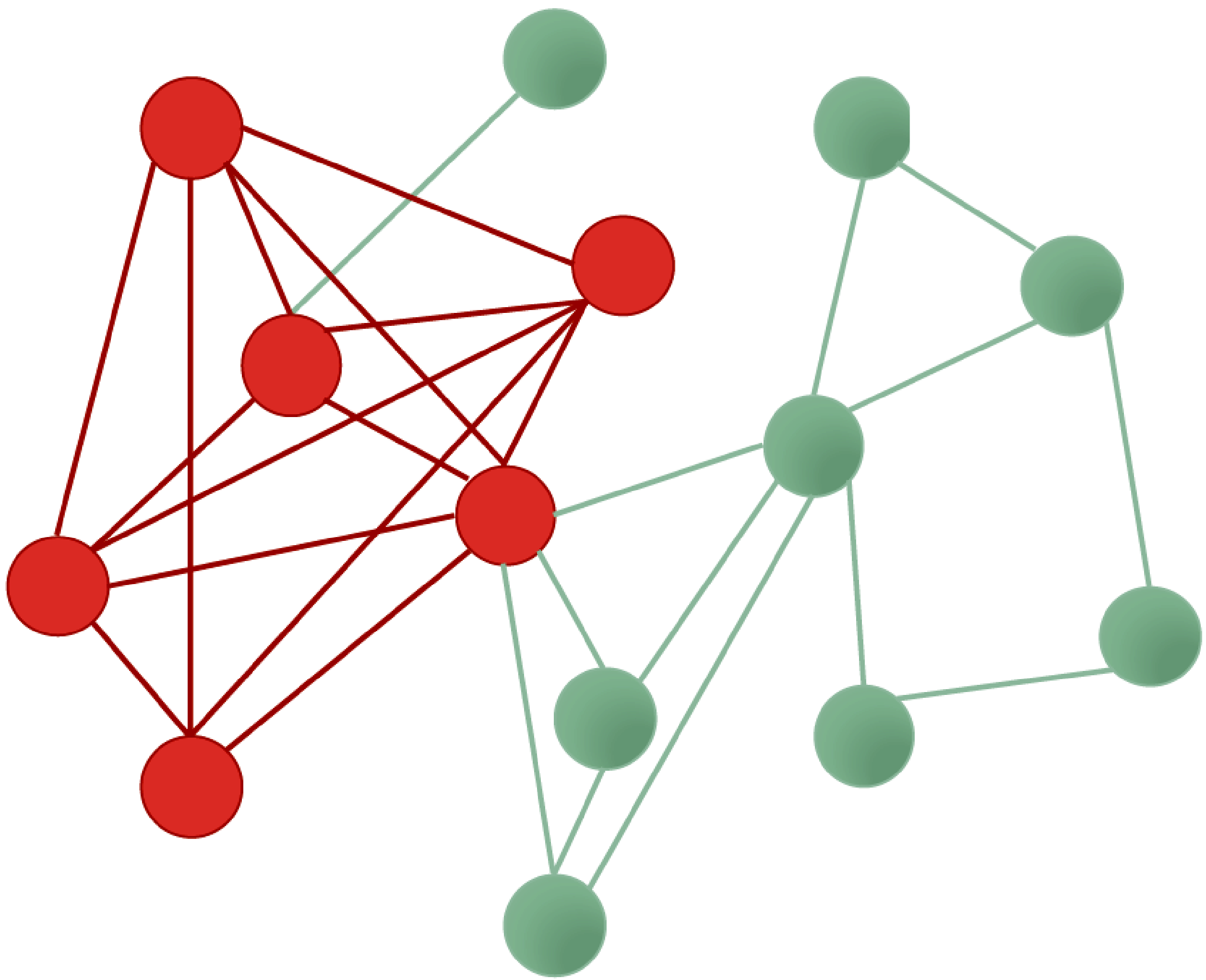
http://www.cs.berkeley.edu/~jfc/cs174lecs/lec18/lec18.html

Densest sub-graph

Density measures

- Density = Average degree = 2|E|/|V|
  - Sometimes just |E|/|V|
- Edge ratio = (2|E|)/(|V|(|V|-1))
  - What is |V|(|V|-1|)/2?

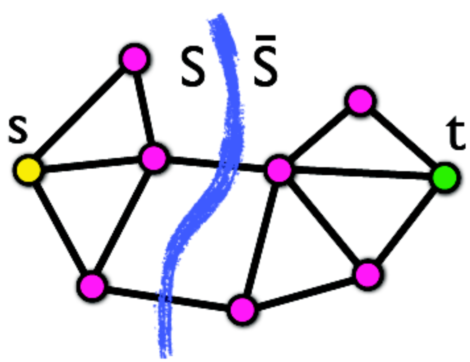
Densest sub-graph



Goldberg's algorithm for densest subgraph

- Requires: min-cut problem

min-cut problem

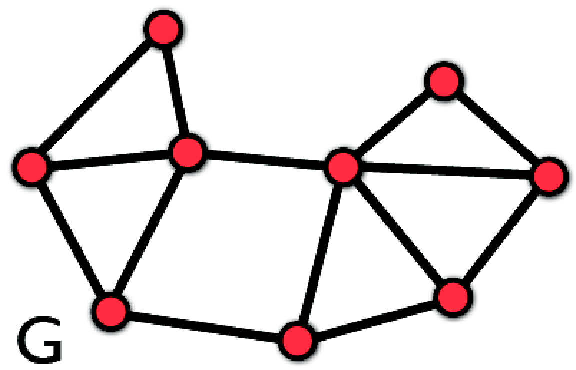


- source $s \in V$, destination $t \in V$
- find $S \subseteq V$, s.t.,
- $s \in S$ and $t \in \bar{S}$, and
- minimize $e(S, \bar{S})$
- polynomially-time solvable
- equivalent to max-flow problem

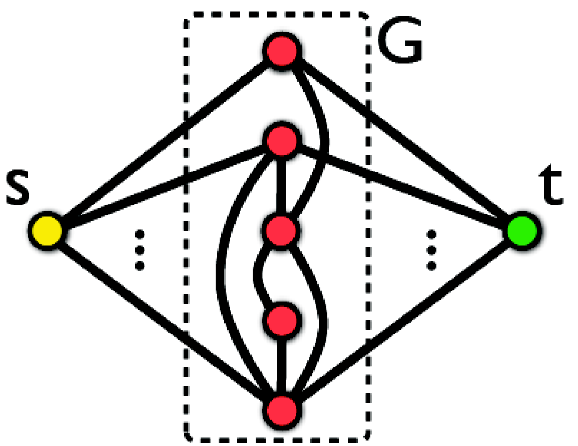
Slides on this section from: http://www.math.cmu.edu/~ctsourak/amazing.html

Goldberg's algorithm (1)

- consider first degree density d



- is there a subgraph S with $d(S) \geq c$?
- transform to a min-cut instance



- on the transformed instance:
- is there a cut smaller than a certain value?

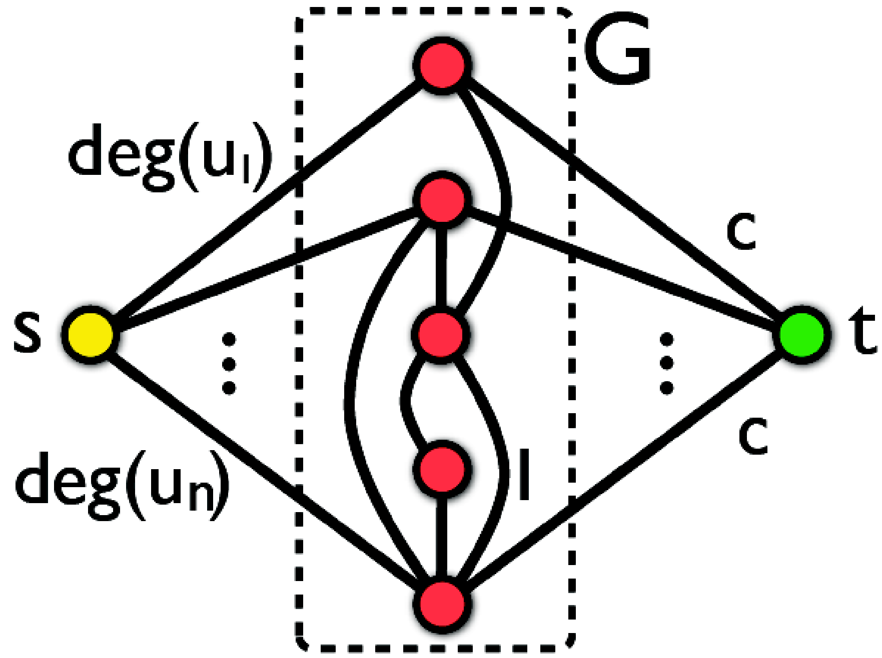
Goldberg's algorithm (2)

is there S with $d(S) \geq c$?

$$\begin{aligned}
\frac{2\,|E(S,S)|}{|S|} &\geq c \\
2\,|E(S,S)| &\geq c|S| \\
\sum_{u \in S} \deg(u) - |E(S,\bar{S})| &\geq c|S| \\
\sum_{u \in S} \deg(u) + \sum_{u \in \bar{S}} \deg(u) - \sum_{u \in \bar{S}} \deg(u) - |E(S,\bar{S})| &\geq c|S| \\
\sum_{u \in \bar{S}} \deg(u) + |E(S,\bar{S})| + c|S| &\leq 2\,|E|
\end{aligned}$$

Goldberg's algorithm (3)

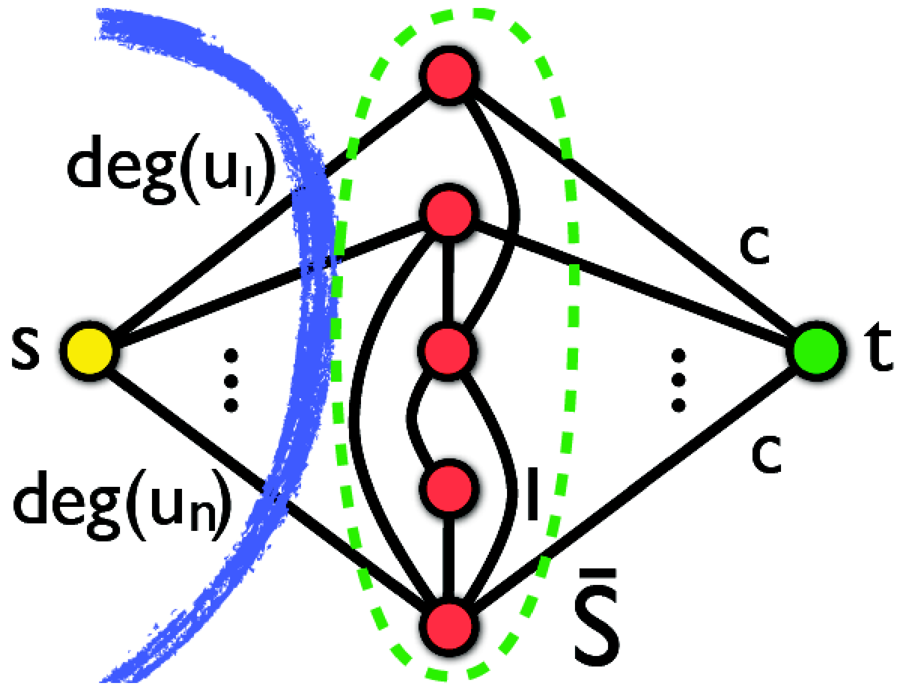
- transformation to min-cut instance



- is there S s.t. $\sum_{u \in \bar{S}} \deg(u) + |e(S, \bar{S})| + c|S| \leq 2\,|E|$?

Goldberg's algorithm (4)

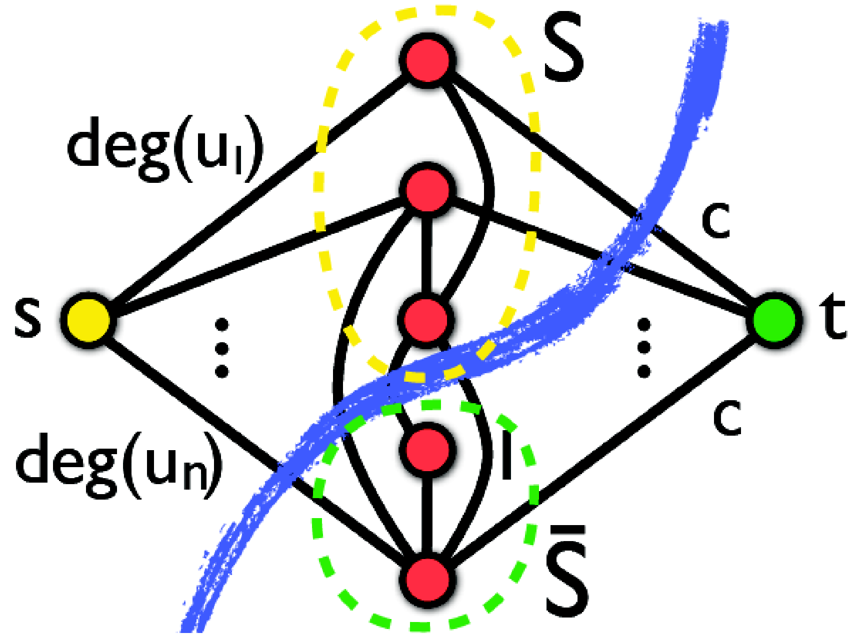
- transform to a min-cut instance



- is there S s.t. $\sum_{u \in \bar{S}} \deg(u) + |e(S, \bar{S})| + c|S| \leq 2|E|$?
- a cut of value $2|E|$ always exists, for $S = \emptyset$

Goldberg's algorithm (5)

- transform to a min-cut instance



- is there S s.t. $\sum_{u \in \bar{S}} \deg(u) + |e(S, \bar{S})| + c|S| \leq 2|E|$?
- $S \neq \emptyset$ gives cut of value $\sum_{u \in \bar{S}} \deg(u) + |e(S, \bar{S})| + c|S|$

If this exists for non-empty S, then S is a sub-graph of density c

Goldberg's algorithm (6)

- to find the densest subgraph perform binary search on c
  - logarithmic number of min-cut calls
  - each min-cut call requires O(|V||E|) time
- problem can also be solved with one min-cut call using the parametric max-flow algorithm

A faster algorithm

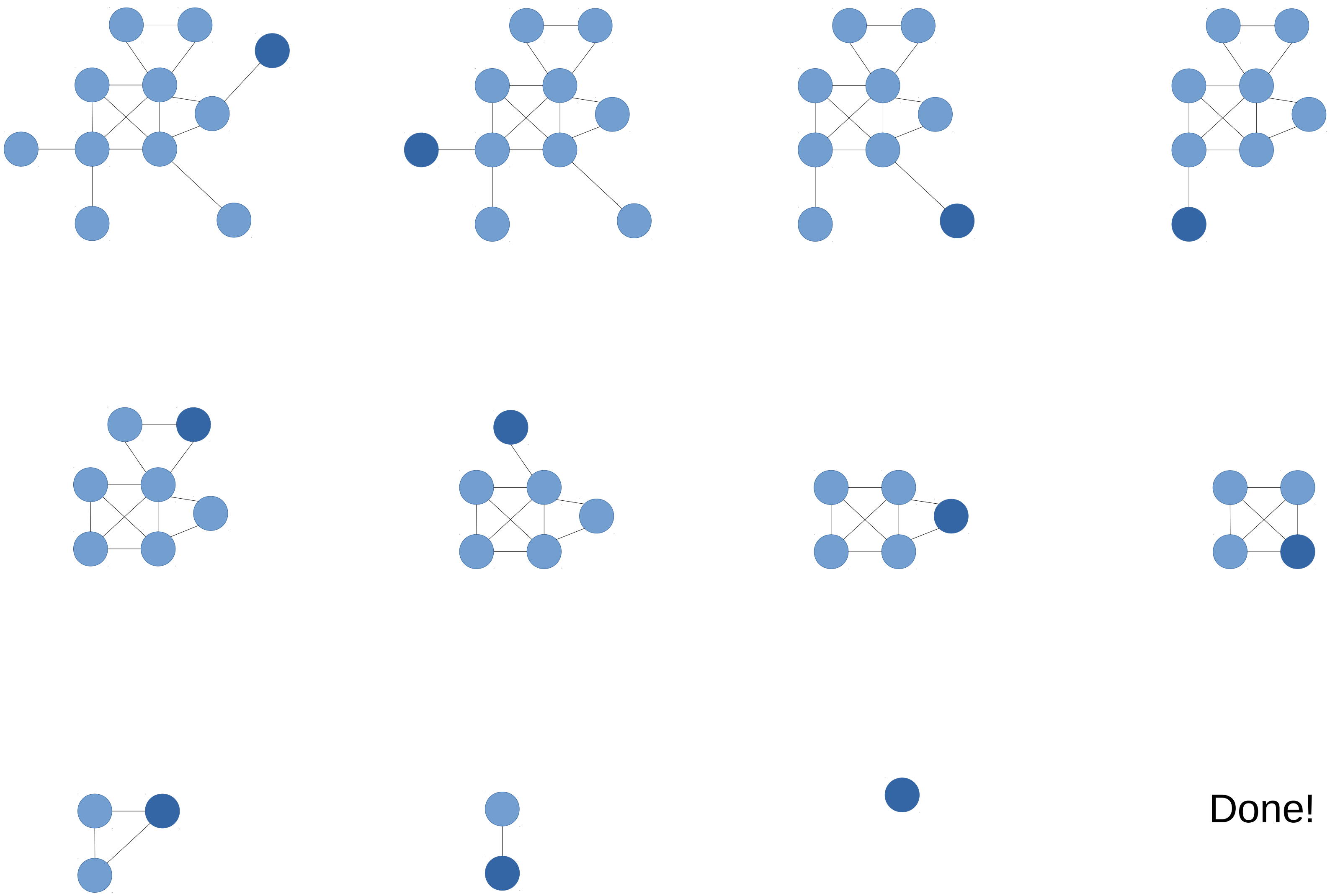
- Charikar, M. (2000). Greedy approximation algorithms for nding dense components in a graph. In APPROX.
- Approximate algorithm (by a factor of 2)

Greedy algorithm

input: undirected graph $G = (V, E)$
output: S, a dense subgraph of G

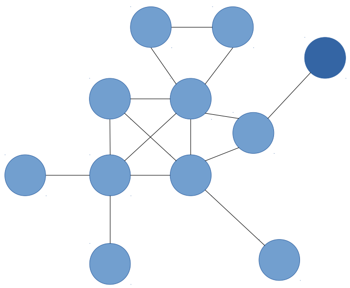
1 set $G_n \leftarrow G$
2 for $k \leftarrow n$ downto 1
2.1 let v be the smallest degree vertex in G_k
2.2 $G_{k-1} \leftarrow G_k \setminus \{v\}$
3 output the densest subgraph among $G_n, G_{n-1}, \ldots, G_1$

Example run of Greedy Algorithm

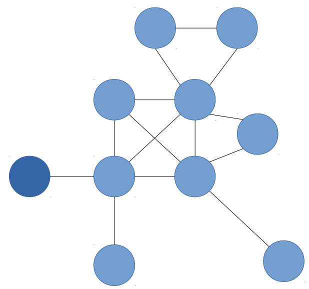


Done!

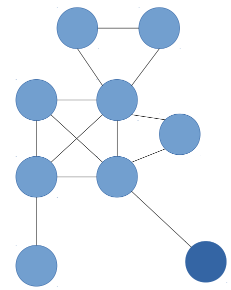
Example run of Greedy Algorithm



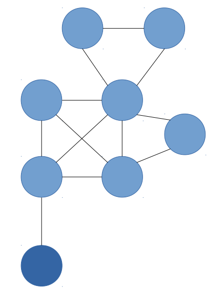
13/11=1.18



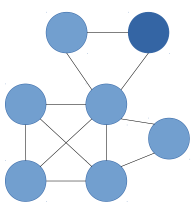
14/10=1.40



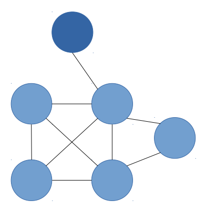
13/9=1.44



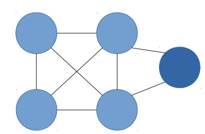
12/8=1.50



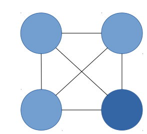
11/7=1.57



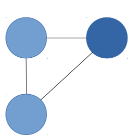
9/6=1.50



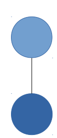
8/5=1.60



6/4=1.50



3/3=1.00

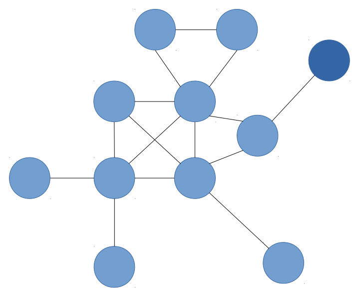


1/2=0.50

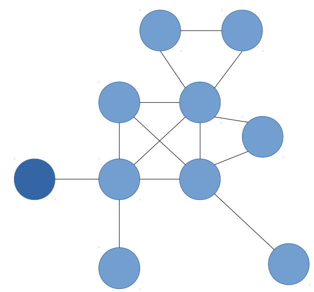
0/1=0.00

Density computed as |E|/|V|

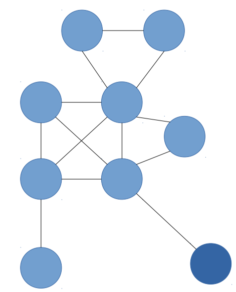
Example run of Greedy Algorithm



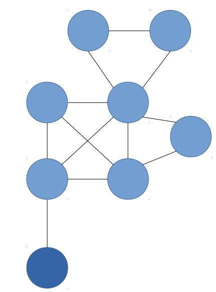
13/11=1.18



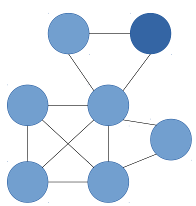
14/10=1.40



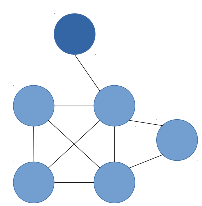
13/9=1.44



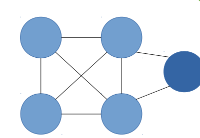
12/8=1.50



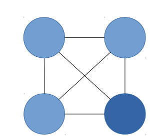
11/7=1.57



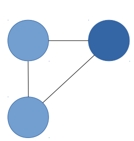
9/6=1.50



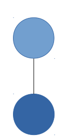
8/5=1.60



6/4=1.50



3/3=1.00



1/2=0.50

0/1=0.00

Done!

Approximation guarantee

- S* = optimal sub-graph (highest density)
- density(S_*) = λ = $|e(S_*)| / |S_*|$
- For all v in S*, deg(v) >= λ, because

$$\frac{|e(S^*)|}{|S^*|} \geq \frac{|e(S^*\backslash v)|}{|S^*\backslash v|} = \frac{|e(S^*)| - deg_{S^*}(v)}{|S^*| - 1}$$

Because of optimality of S*

https://people.cs.umass.edu/~barna/paper/dense-subgraph-full.pdf

Approximation guarantee (cont)

$$\frac{|e(S^*)|}{|S*|} \geq \frac{|e(S^* \backslash v)|}{|S* \backslash v|} = \frac{|e(S^*)| - deg_{S^*}(v)}{|S*| - 1}$$

Hence,

$$deg_{S^*}(v) \geq \frac{|e(S^*)|}{|S*|} = d(S^*) = \lambda$$

Approximation guarantee (cont.)

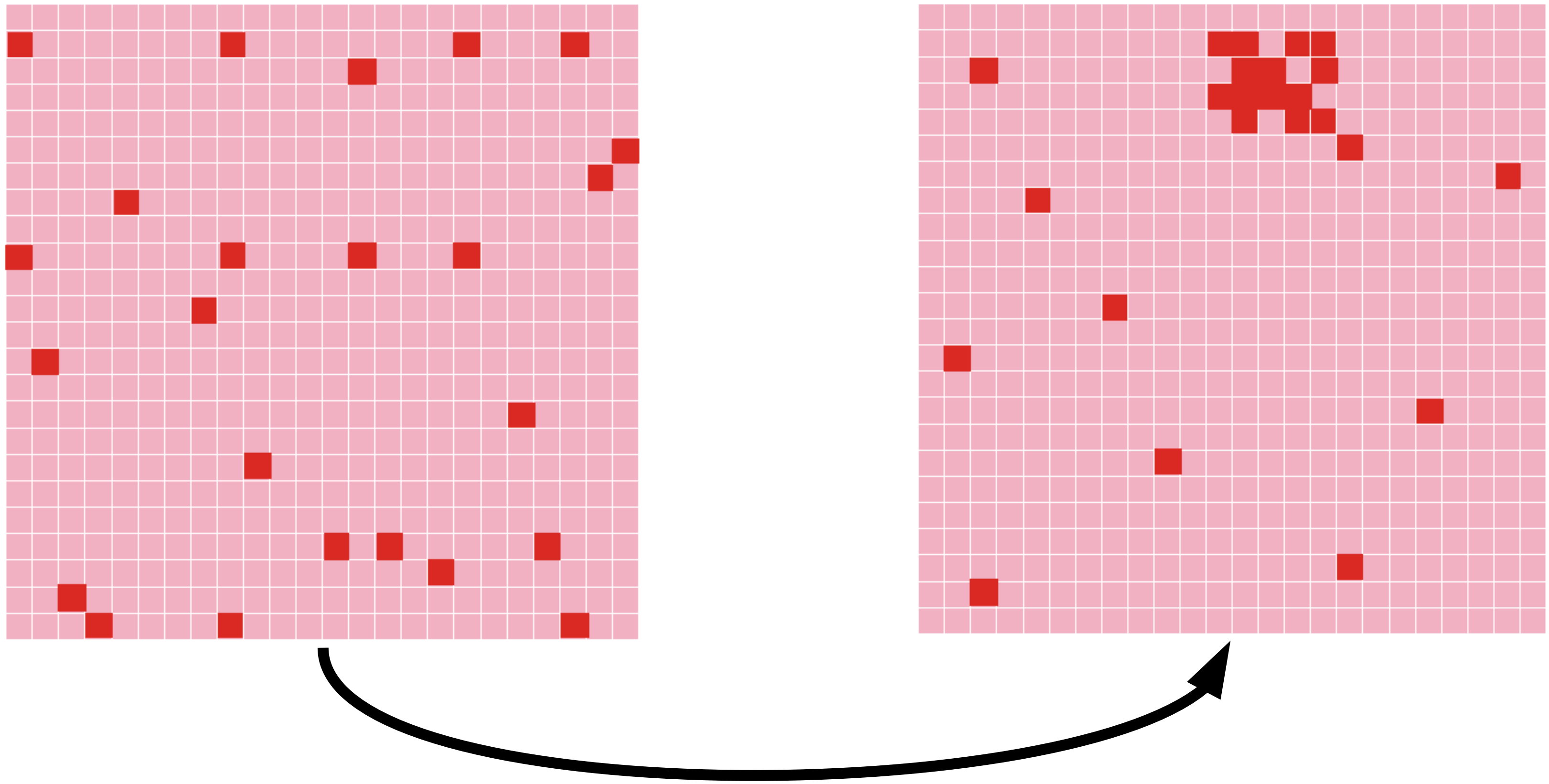
- Now, let's consider when greedy removes the **first** vertex of the optimal solution $v \in S^*$
- At that point, all the vertices of the remaining subgraph (S) have degree >= λ, because v has degree >= λ
- Hence, this subgraph has more than $\frac{\lambda|S|}{2}$ edges, and density more than

$$\frac{\frac{\lambda|S|}{2}}{|S|} = \frac{\lambda}{2}$$

- Hence this is a 2-approximate algorithm

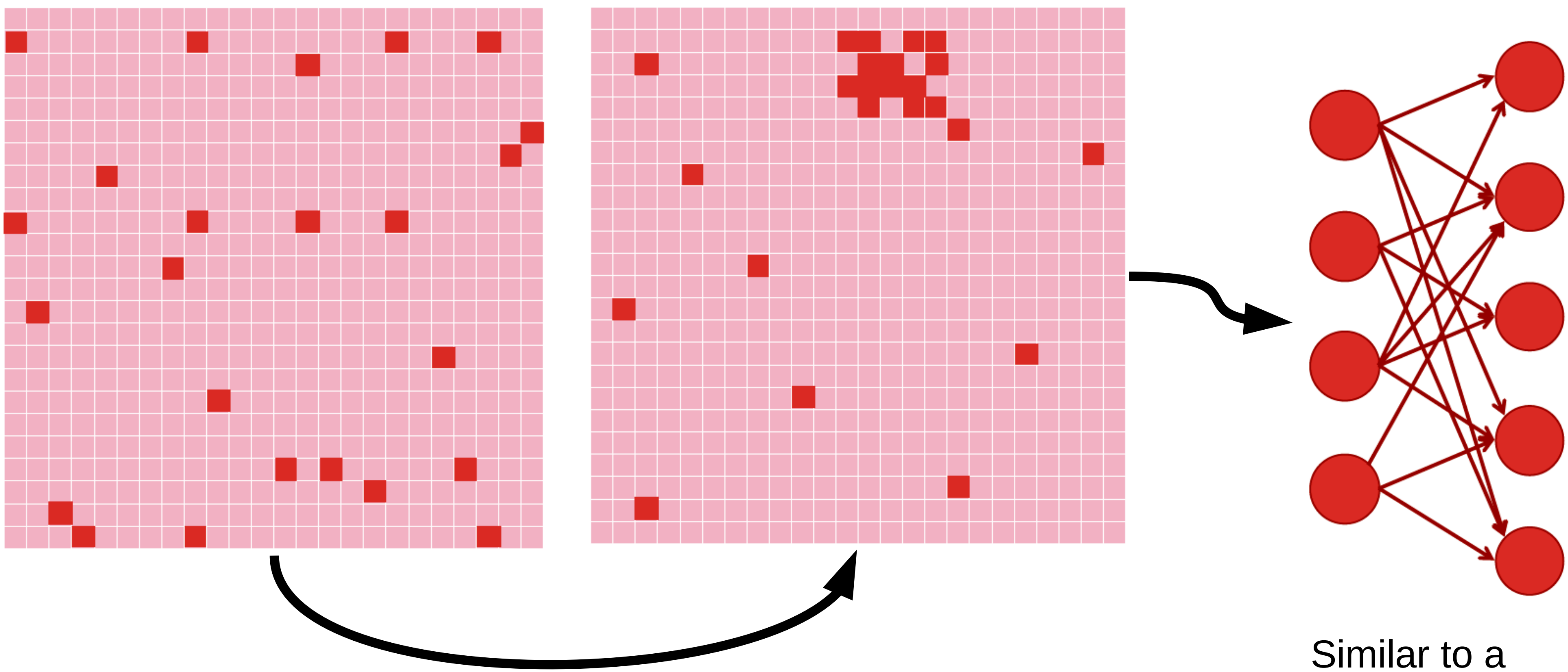
Bi-partite near cliques

Dense subgraphs in matrix representation of a graph



Re-arrange rows and columns

Dense subgraphs in matrix representation of a graph



Re-arrange rows and columns

Similar to a
bi-partite clique

Example of bi-partite near-cliques

e-commerce

- weighted bipartite graph $G(A \cup Q, E, w)$
- set A corresponds to advertisers
- set Q corresponds to queries
- each edge (a, q) has weight $w(a, q)$ equal to the amount of money advertiser a is willing to spend on query q

large almost bipartite cliques correspond to sub-markets

Fans and artists in cultural products also create bi-partite near-cliques.

Scalable method for dense sub-graphs

- *D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. In VLDB ’05: Proc. 31st Intl. Conf. on Very Large Data Bases, pages 721–732. ACM, 2005.*
- Can be applied to arbitrarily large graphs

Shingling algorithm

- Take a permutation π and apply it to both sets
- Take the minimum element in each set under this permutation
- The probability of the two minima matching is the Jaccard coefficient of A and B

$$Pr[\pi^{-1}(min_{a\in A}\{\pi(a)\}) = \pi^{-1}(min_{b\in B}\{\pi(b)\})] = \frac{|A\cap B|}{|A\cup B|}$$

A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. Comput. Netw. ISDN Syst., 29(8-13):1157–1166, 1997.

Example

- A = {dcab, abcd, cabb, aabd}
- B = {abcd, dabc, abbd, badd, dcab}
- Suppose permutation = “sort by second character, then by fourth”
  - Minimum(A) = cabb
  - Minimum(B) = dabc
  - Bad luck this time, however …
- If you use many permutations, you can get good estimates of Jaccard coefficient

How to build the permutations

- *What is a natural family of permutations to use?*

Yes

- That's why this method is often referred to as *min-hashing*

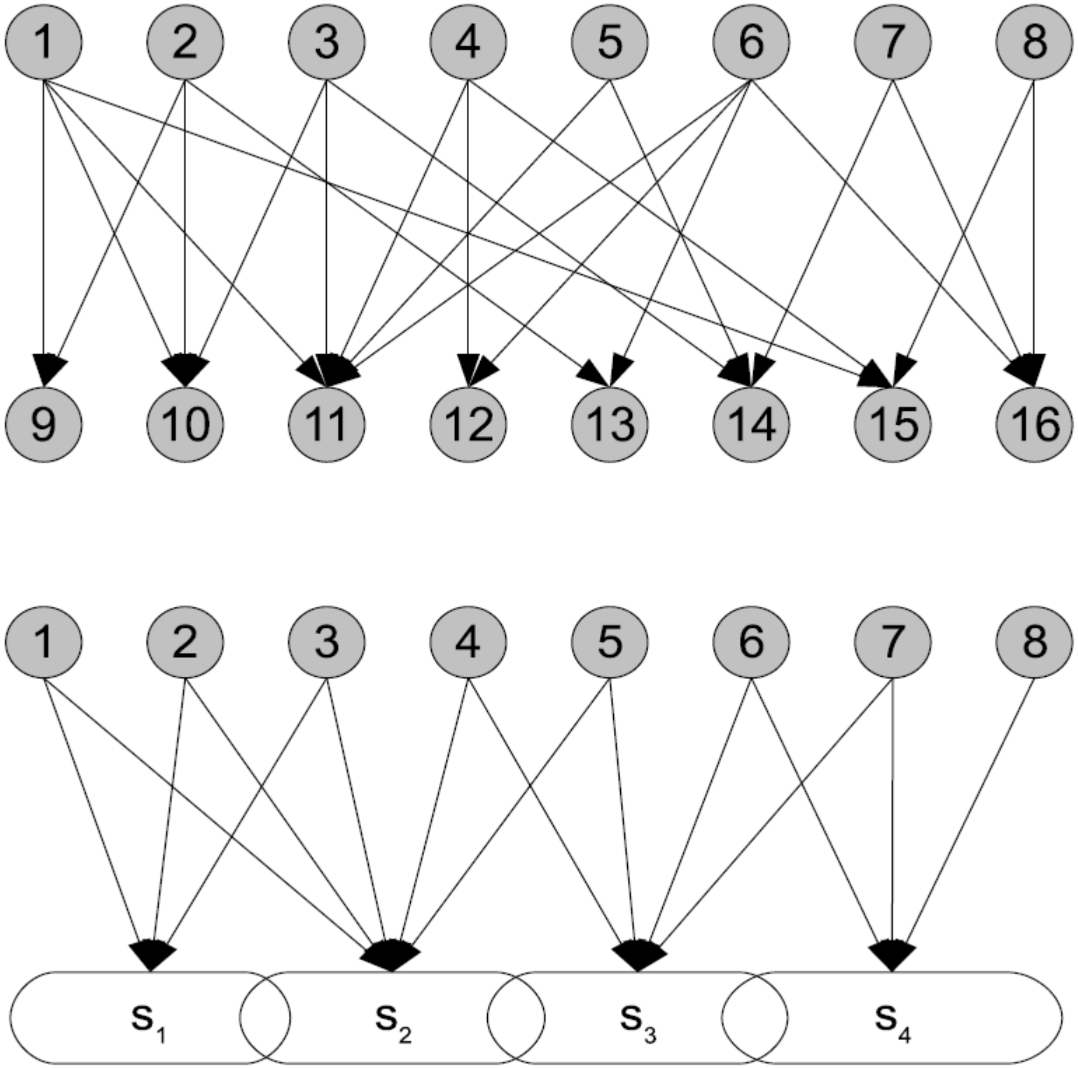
Advantages of shingling

- A and B can be huge
  - but the shingle vector is of fixed size!
  - comparisons of shingles are much faster
- *How to apply this to finding dense sub-graphs?*
  - We are going to use procedure shingle(list), which computes a shingle of size c of a list

Algorithm

- Let e(v) be the edges of v
- Start with lists <v, e(v)>
- Compute <v, shingles(e(v))>
- Invert this list to obtain <shingle, list of v> = S1
- Cluster this list, how?
  - Compute <shingle, shingles(list of v)> = S2
  - Cluster S2 using any clustering method
- Output = list of shingles, and list of vertices sharing those shingles

Algorithm (visually)



http://charuaggarwal.net/dense-survey