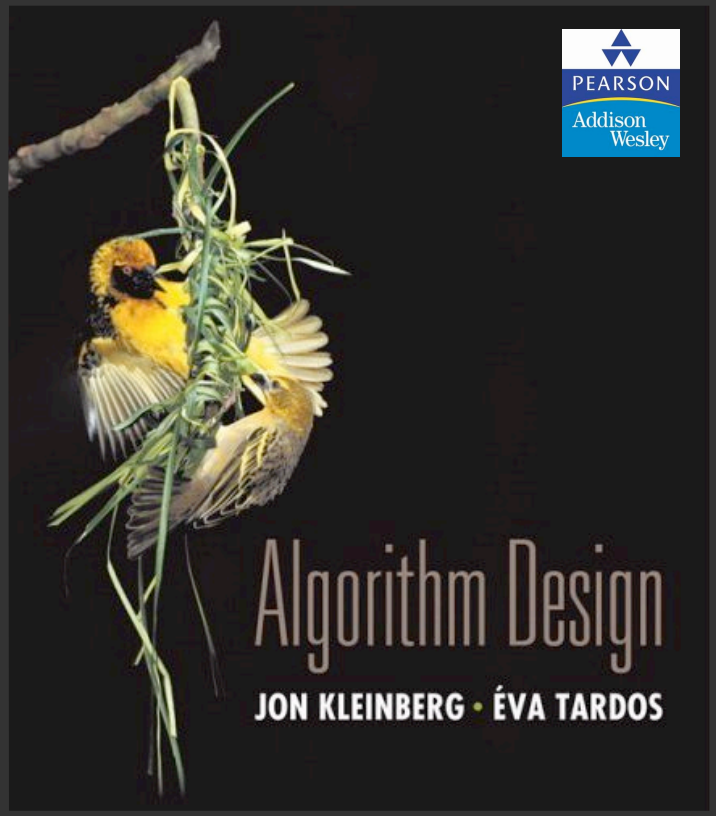


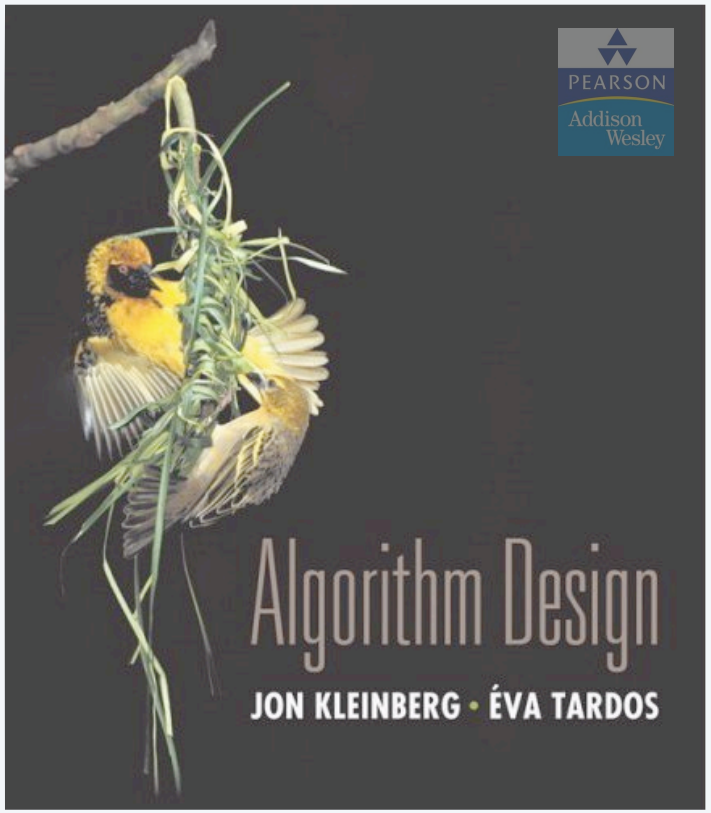
4. Greedy Algorithms II

- *Dijkstra's algorithm*
- *minimum spanning trees*
- *Prim, Kruskal,*
- *single-link clustering*

Lecture slides by Kevin Wayne

http://www.cs.princeton.edu/~wayne/kleinberg-tardos

Last updated on Sep 8, 2013 6:30 AM



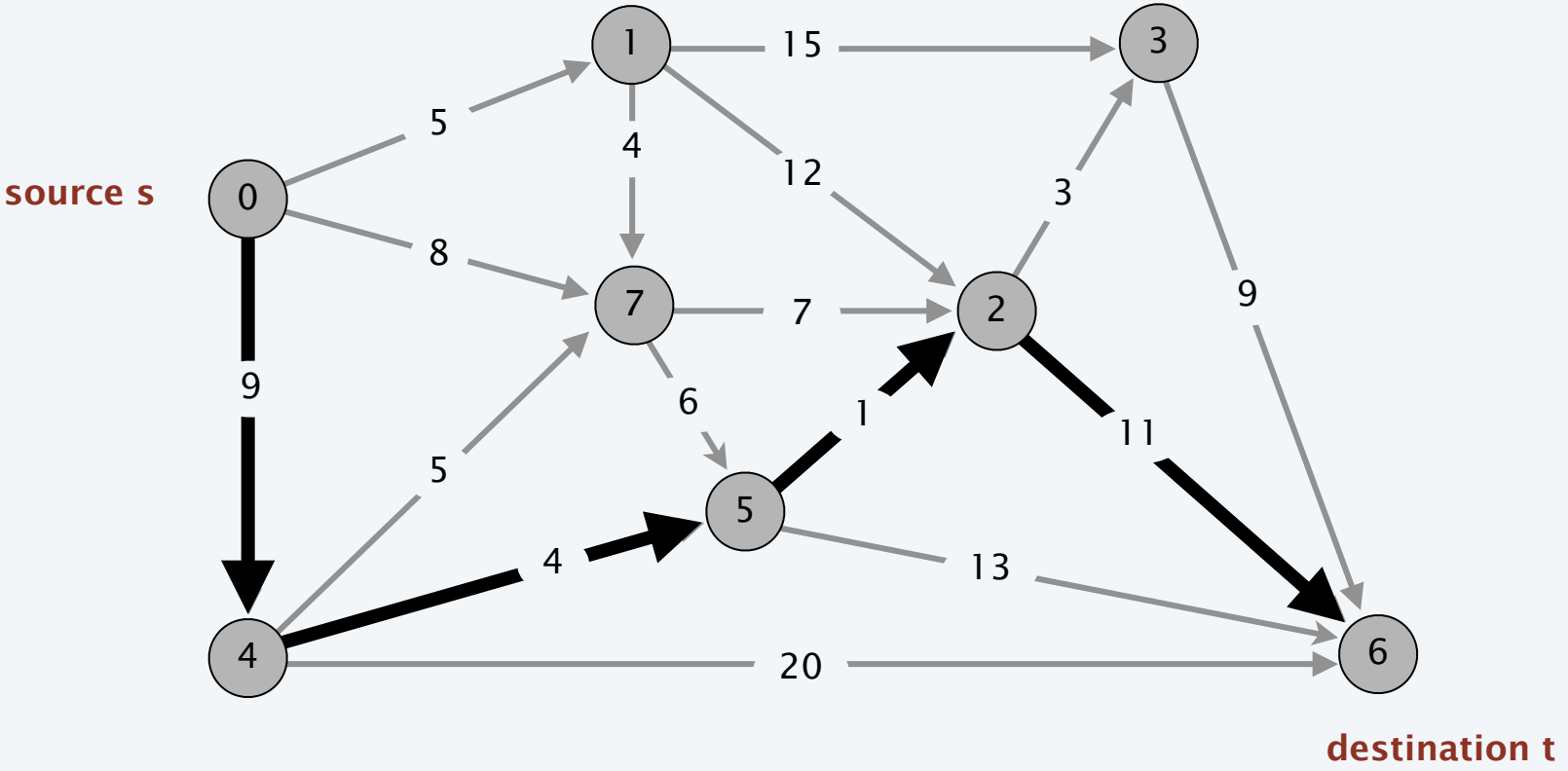
SECTION 4.4

4. GREEDY ALGORITHMS II

- *Dijkstra's algorithm*
- *minimum spanning trees*
- *Prim, Kruskal*
- *single-link clustering*

Shortest-paths problem

Problem. Given a digraph $G = (V, E)$, edge lengths $\ell_e \geq 0$, source $s \in V$, and destination $t \in V$, find the shortest directed path from s to t.



length of path = 9 + 4 + 1 + 11 = 25

Car navigation

Shortest path applications

- PERT/CPM.
- Map routing.
- Seam carving.
- Robot navigation.
- Texture mapping.
- Typesetting in LaTeX.
- Urban traffic planning.
- Telemarketer operator scheduling.
- Routing of telecommunications messages.
- Network routing protocols (OSPF, BGP, RIP).
- Optimal truck routing through given traffic congestion pattern.

Reference: Network Flows: Theory, Algorithms, and Applications, R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, Prentice Hall, 1993.

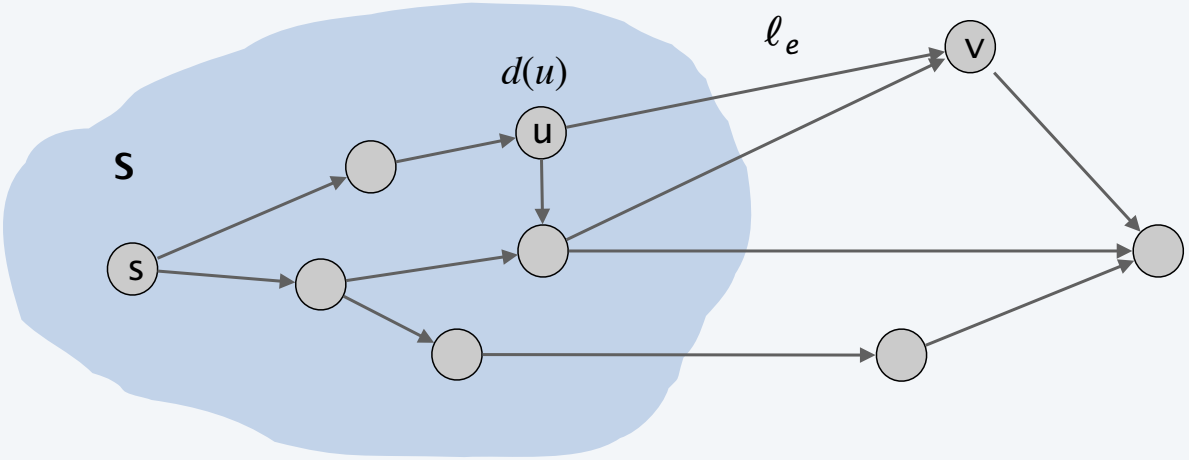
Dijkstra's algorithm

Greedy approach. Maintain a set of explored nodes S for which algorithm has determined the shortest path distance $d(u)$ from s to u.

- Initialize $S = \{ s \}$, $d(s) = 0$.
- Repeatedly choose unexplored node v which minimizes

$$\pi(v) = \min_{e = (u,v) : u \in S} d(u) + \ell_e ,$$

shortest path to some node u in explored part, followed by a single edge (u, v)



Dijkstra's algorithm

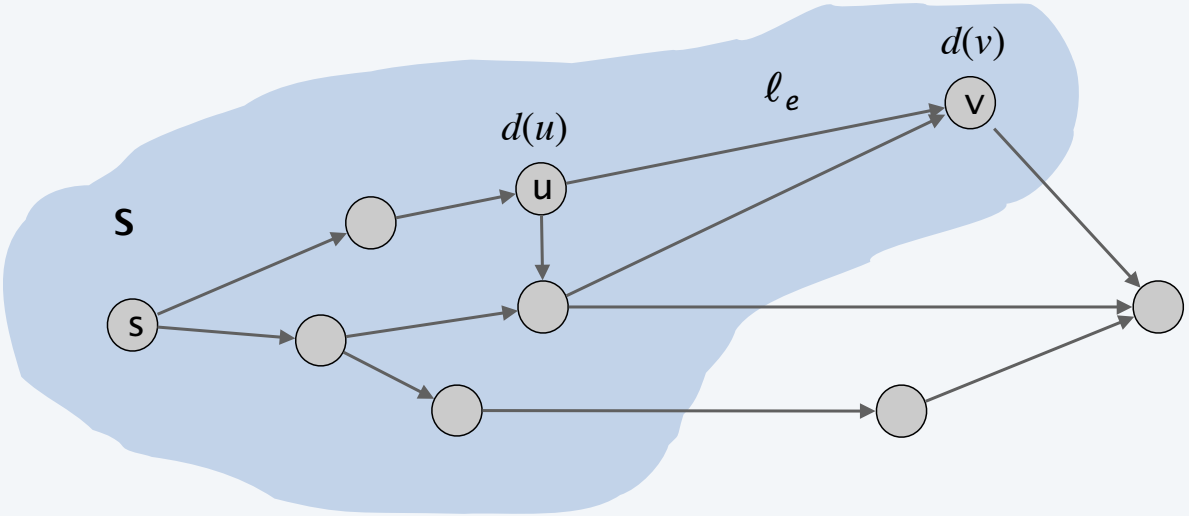
Greedy approach. Maintain a set of explored nodes S for which algorithm has determined the shortest path distance $d(u)$ from s to u.

- Initialize $S = \{ s \}$, $d(s) = 0$.
- Repeatedly choose unexplored node v which minimizes

$$\pi(v) = \min_{e = (u,v) : u \in S} d(u) + \ell_e ,$$

add v to S, and set $d(v) = \pi(v)$.

shortest path to some node u in explored part, followed by a single edge (u, v)



Dijkstra's algorithm: proof of correctness

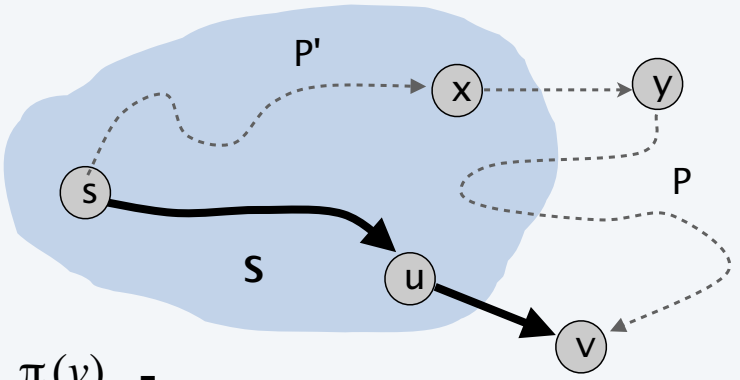
Invariant. For each node $u \in S$, $d(u)$ is the length of the shortest $s \rightarrow u$ path.

Pf. [by induction on $|S|$]

Base case: $|S| = 1$ is easy since $S = \{ s \}$ and $d(s) = 0$.

Inductive hypothesis: Assume true for $|S| = k \geq 1$.

- Let v be next node added to S, and let (u, v) be the final edge.
- The shortest $s \rightarrow u$ path plus (u, v) is an $s \rightarrow v$ path of length $\pi(v)$.
- Consider any $s \rightarrow v$ path P. We show that it is no shorter than $\pi(v)$.
- Let (x, y) be the first edge in P that leaves S, and let P' be the subpath to x.
- P is already too long as soon as it reaches y.



$$\ell(P) \geq \ell(P') + \ell(x, y) \geq d(x) + \ell(x, y) \geq \pi(y) \geq \pi(v) \quad \blacksquare$$

nonnegative lengths

inductive hypothesis

definition of $\pi(y)$

Dijkstra chose v instead of y

Dijkstra's algorithm: efficient implementation

Implementation.

- Algorithm stores $d(v)$ for each explored node v.
- Priority queue stores $\pi(v)$ for each unexplored node v.
- Recall: $d(u) = \pi(u)$ when u is deleted from priority queue.

DIJKSTRA (V, E, s)

Create an empty priority queue.

FOR EACH $v \neq s$: $d(v) \leftarrow \infty$; $d(s) \leftarrow 0$.

FOR EACH $v \in V$: *insert* v with key $d(v)$ into priority queue.

WHILE (the priority queue *is not empty*)

  $u \leftarrow$ *delete-min* from priority queue.

  FOR EACH edge $(u, v) \in E$ leaving u:

    IF $d(v) > d(u) + \ell(u, v)$

      decrease-key of v to $d(u) + \ell(u, v)$ in priority queue.

      $d(v) \leftarrow d(u) + \ell(u, v)$.

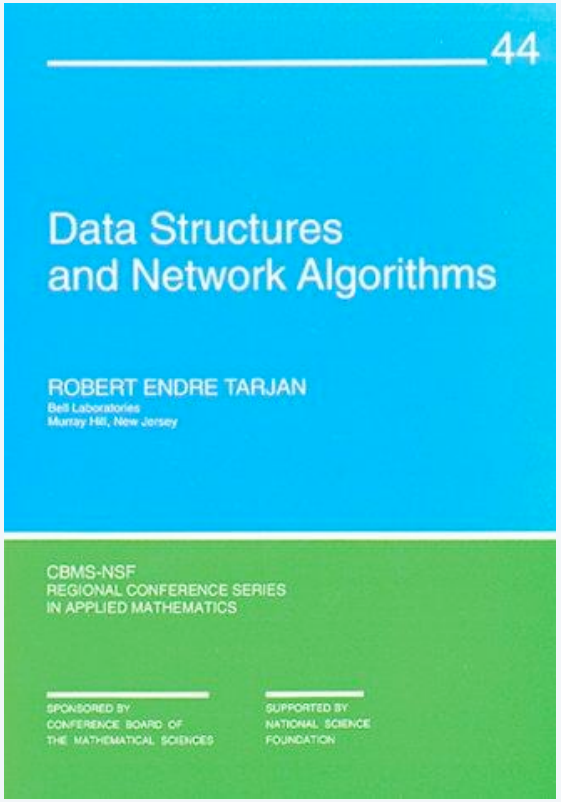
Dijkstra's algorithm: which priority queue?

Performance. Depends on PQ: n insert, n delete-min, m decrease-key.

- Array implementation optimal for dense graphs.
- Binary heap much faster for sparse graphs.
- 4-way heap worth the trouble in performance-critical situations.
- Fibonacci/Brodal best in theory, but not worth implementing.

PQ implementation	insert	delete-min	decrease-key	total
unordered array	$O(1)$	$O(n)$	$O(1)$	$O(n^2)$
binary heap	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(m \log n)$
d-way heap (Johnson 1975)	$O(d \log_d n)$	$O(d \log_d n)$	$O(\log_d n)$	$O(m \log_{m/n} n)$
Fibonacci heap (Fredman-Tarjan 1984)	$O(1)$	$O(\log n)$ †	$O(1)$ †	$O(m + n \log n)$
Brodal queue (Brodal 1996)	$O(1)$	$O(\log n)$	$O(1)$	$O(m + n \log n)$

† amortized



SECTION 6.1

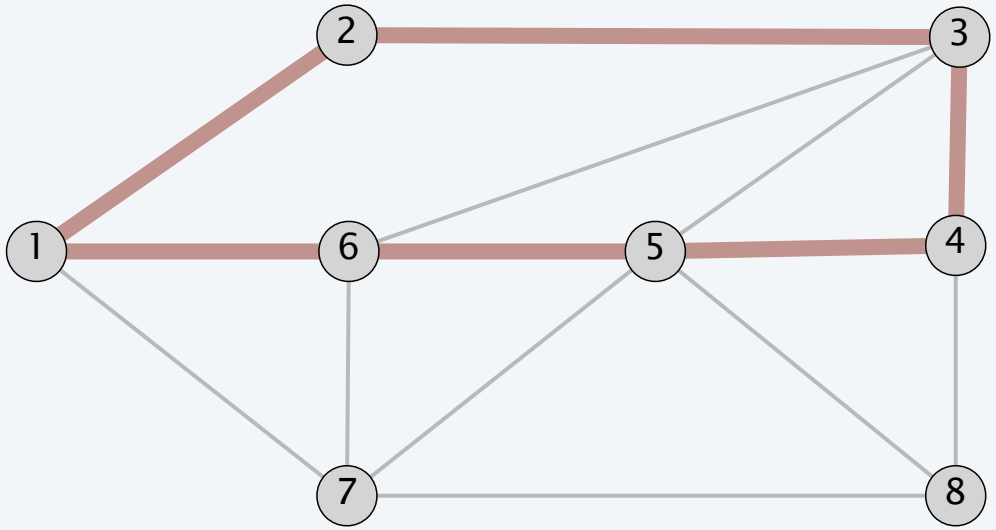
4. GREEDY ALGORITHMS II

- *Dijkstra's algorithm*
- *minimum spanning trees*
- *Prim, Kruskal*
- *single-link clustering*

Cycles and cuts

Def. A path is a sequence of edges which connects a sequence of nodes.

Def. A cycle is a path with no repeated nodes or edges other than the starting and ending nodes.

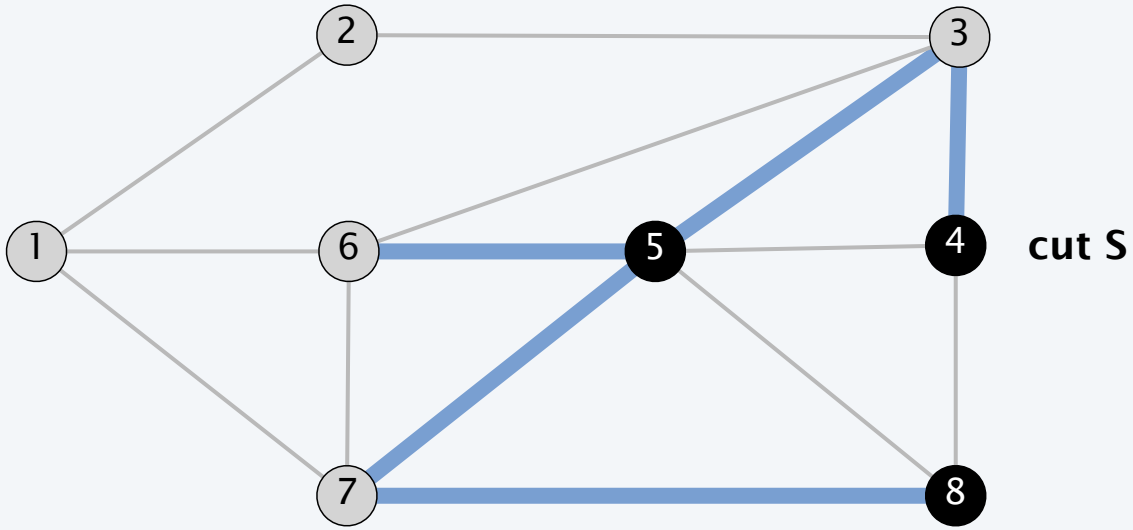


cycle C = { (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1) }

Cycles and cuts

Def. A cut is a partition of the nodes into two nonempty subsets S and $V - S$.

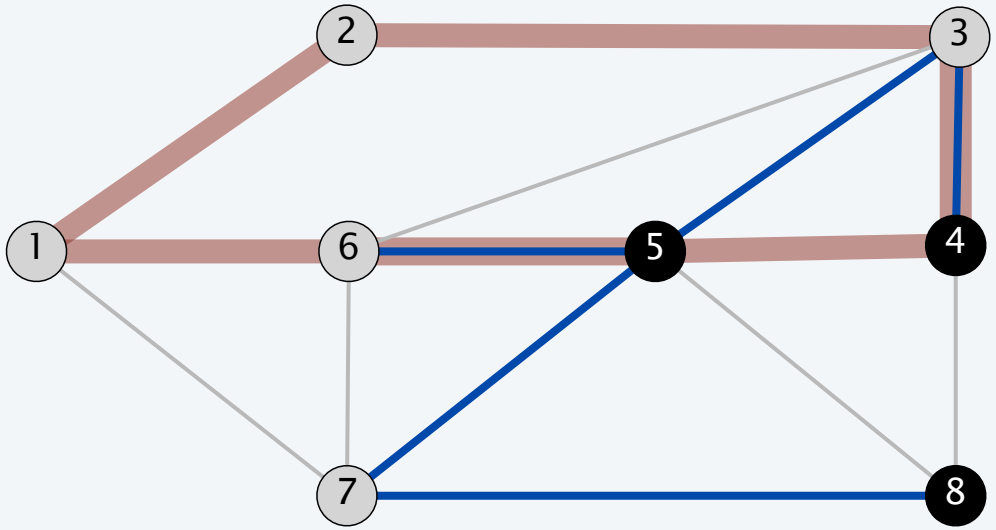
Def. The cutset of a cut S is the set of edges with exactly one endpoint in S.



cutset D = { (3, 4), (3, 5), (5, 6), (5, 7), (8, 7) }

Cycle-cut intersection

Proposition. A cycle and a cutset intersect in an even number of edges.



cutset D = { (3, 4), (3, 5), (5, 6), (5, 7), (8, 7) }

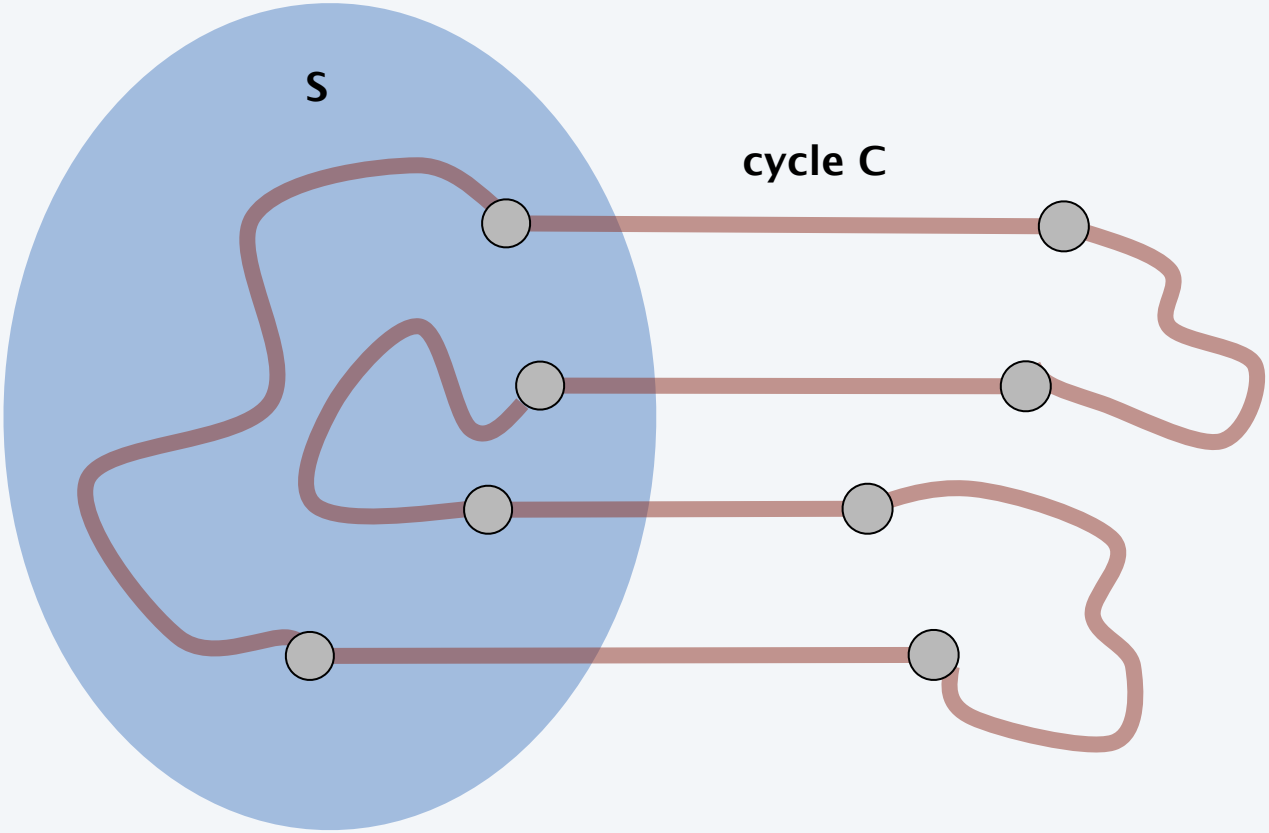
cycle C = { (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1) }

intersection C ∩ D = { (3, 4), (5, 6) }

Cycle-cut intersection

Proposition. A cycle and a cutset intersect in an even number of edges.

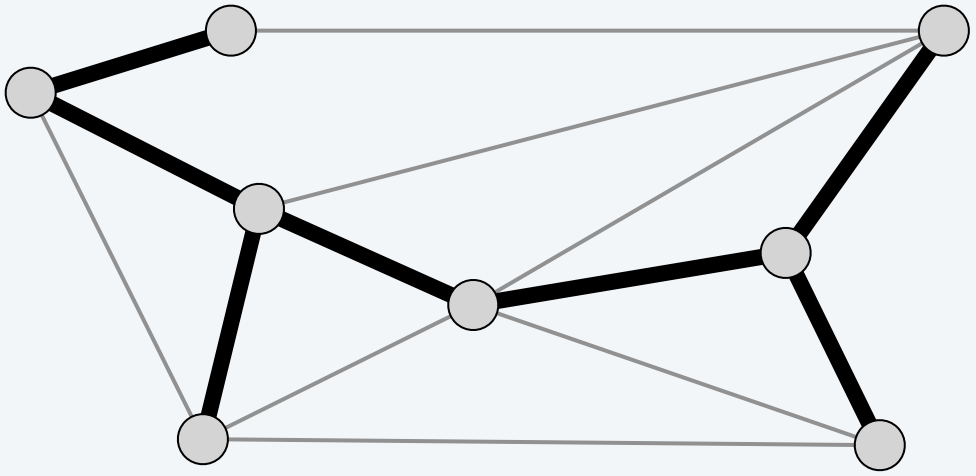
Pf. [by picture]



Spanning tree properties

Proposition. Let $T = (V, F)$ be a subgraph of $G = (V, E)$. TFAE:

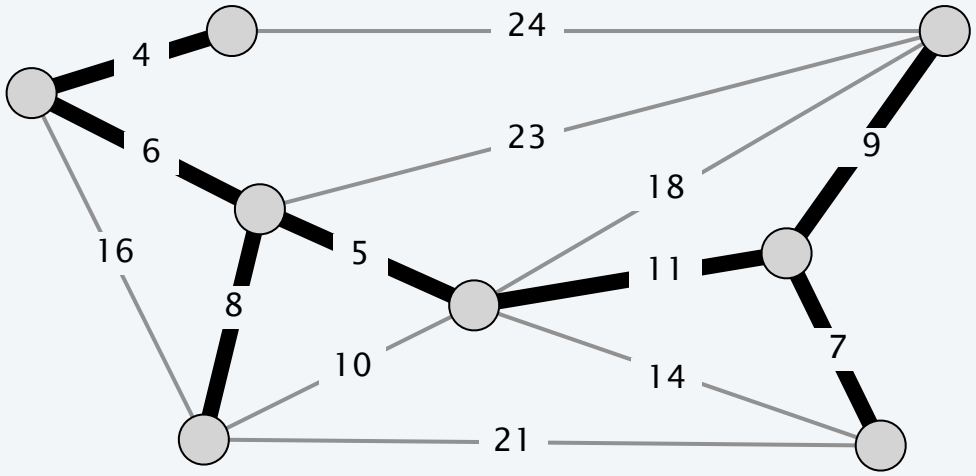
- T is a spanning tree of G.
- T is acyclic and connected.
- T is connected and has $n - 1$ edges.
- T is acyclic and has $n - 1$ edges.
- T is minimally connected: removal of any edge disconnects it.
- T is maximally acyclic: addition of any edge creates a cycle.
- T has a unique simple path between every pair of nodes.



T = (V, F)

Minimum spanning tree

Given a connected graph $G = (V, E)$ with edge costs c_e, an MST is a subset of the edges $T \subseteq E$ such that T is a spanning tree whose sum of edge costs is minimized.



MST cost = 50 = 4 + 6 + 8 + 5 + 11 + 9 + 7

Cayley's theorem. There are n^{n-2} spanning trees of K_n. ← can't solve by brute force

Applications

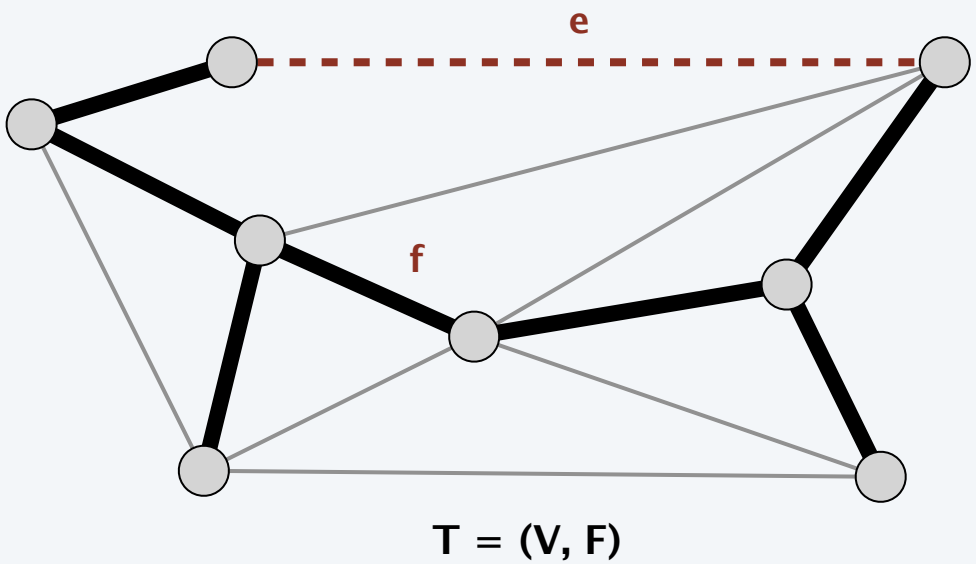
MST is fundamental problem with diverse applications.

- Dithering.
- Cluster analysis.
- Max bottleneck paths.
- Real-time face verification.
- LDPC codes for error correction.
- Image registration with Renyi entropy.
- Find road networks in satellite and aerial imagery.
- Reducing data storage in sequencing amino acids in a protein.
- Model locality of particle interactions in turbulent fluid flows.
- Autoconfig protocol for Ethernet bridging to avoid cycles in a network.
- Approximation algorithms for NP-hard problems (e.g., TSP, Steiner tree).
- Network design (communication, electrical, hydraulic, computer, road).

Fundamental cycle

Fundamental cycle.

- Adding any non-tree edge e to a spanning tree T forms unique cycle C.
- Deleting any edge $f \in C$ from $T \cup \{ e \}$ results in new spanning tree.

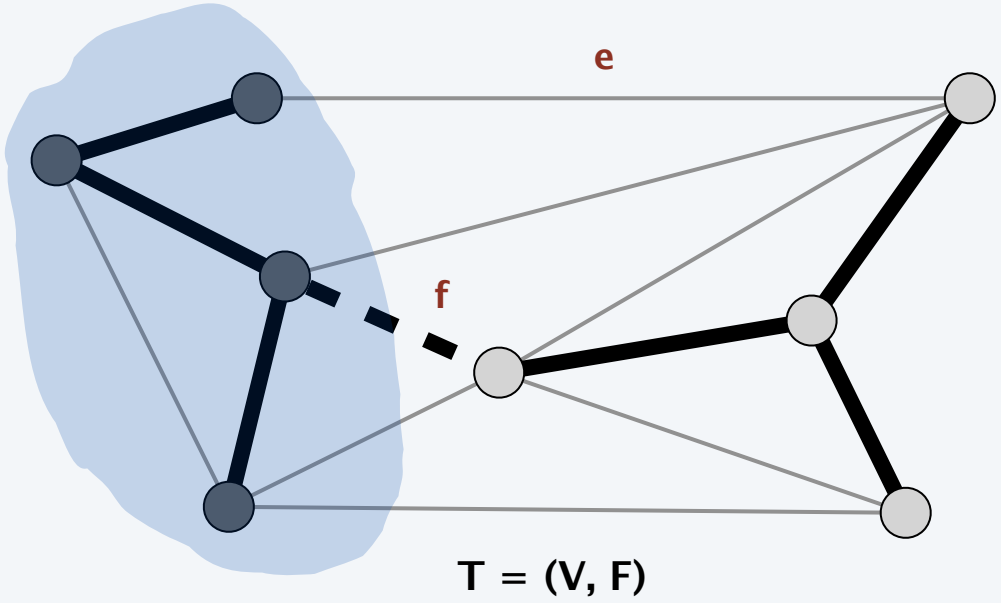


Observation. If $c_e < c_f$, then T is not an MST.

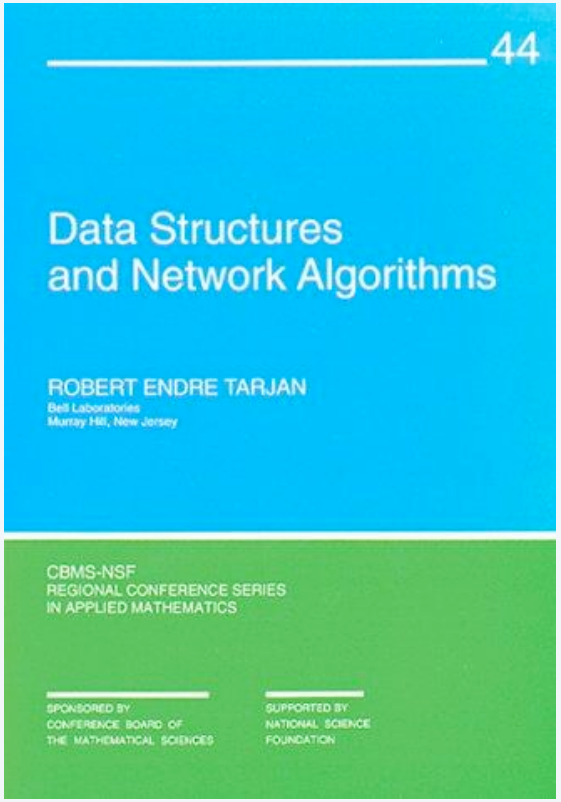
Fundamental cutset

Fundamental cutset.

- Deleting any tree edge f from a spanning tree T divide nodes into two connected components. Let D be cutset.
- Adding any edge $e \in D$ to $T - \{ f \}$ results in new spanning tree.



Observation. If $c_e < c_f$, then T is not an MST.



SECTION 6.2

4. GREEDY ALGORITHMS II

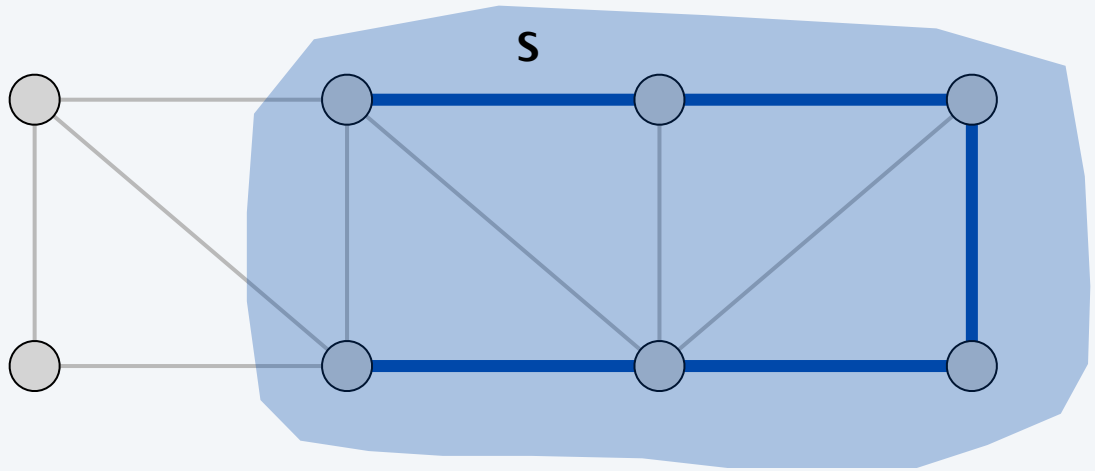
- *Dijkstra's algorithm*
- *minimum spanning trees*
- *Prim, Kruskal*
- *single-link clustering*

Prim's algorithm

Initialize S = any node.

Repeat $n - 1$ times:

- Add to tree the min weight edge with one endpoint in S.
- Add new node to S.



Prim's algorithm: implementation

Theorem. Prim's algorithm can be implemented in $O(m \log n)$ time.

Pf. Implementation almost identical to Dijkstra's algorithm.

[$d(v) =$ weight of cheapest known edge between v and S]

PRIM (V, E, c)

Create an empty priority queue.

$s \leftarrow$ any node in V.

FOR EACH $v \neq s$: $d(v) \leftarrow \infty$; $d(s) \leftarrow 0$.

FOR EACH v : *insert* v with key $d(v)$ into priority queue.

WHILE (the priority queue *is not empty*)

  $u \leftarrow$ *delete-min* from priority queue.

  FOR EACH edge $(u, v) \in E$ incident to u:

    IF $d(v) > c(u, v)$

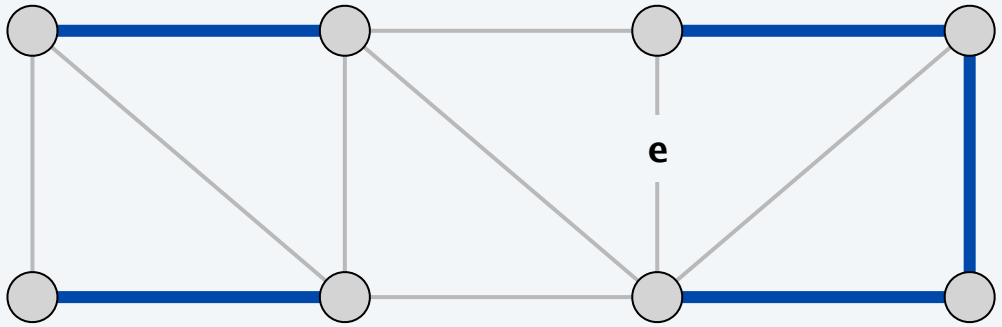
      decrease-key of v to $c(u, v)$ in priority queue.

      $d(v) \leftarrow c(u, v)$.

Kruskal's algorithm

Consider edges in ascending order of weight:

- Add to tree unless it would create a cycle.



Kruskal's algorithm: implementation

Theorem. Kruskal's algorithm can be implemented in $O(m \log m)$ time.

- Sort edges by weight.
- Use union-find data structure to dynamically maintain connected components.

KRUSKAL (V, E, c)

SORT m edges by weight so that $c(e_1) \leq c(e_2) \leq \ldots \leq c(e_m)$

$S \leftarrow \phi$

FOREACH $v \in V$: MAKESET(v).

FOR $i = 1$ TO m

  $(u, v) \leftarrow e_i$

  IF FINDSET$(u) \neq$ FINDSET(v) ← are u and v in same component?

    $S \leftarrow S \cup \{ e_i \}$

    UNION(u, v). ← make u and v in same component

RETURN S

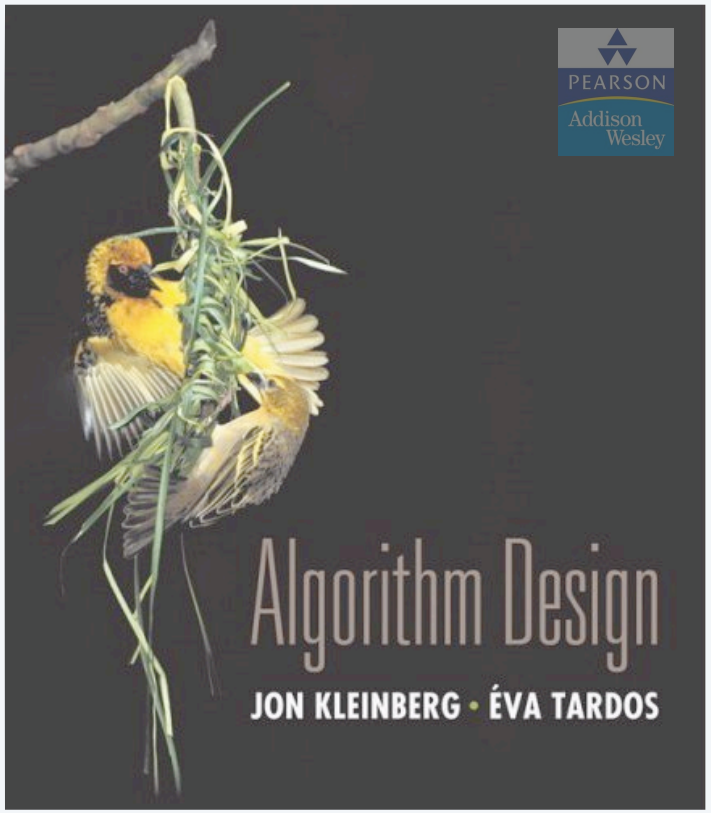
Does a linear-time MST algorithm exist?

deterministic compare-based MST algorithms

year	worst case	discovered by
1975	$O(m \log \log n)$	Yao
1976	$O(m \log \log n)$	Cheriton-Tarjan
1984	$O(m \log^* n)$ $O(m + n \log n)$	Fredman-Tarjan
1986	$O(m \log (\log^* n))$	Gabow-Galil-Spencer-Tarjan
1997	$O(m \, \alpha(n) \log \alpha(n))$	Chazelle
2000	$O(m \, \alpha(n))$	Chazelle
2002	*optimal*	Pettie-Ramachandran
20xx	$O(m)$	???

Remark 1. $O(m)$ randomized MST algorithm. [Karger-Klein-Tarjan 1995]

Remark 2. $O(m)$ MST verification algorithm. [Dixon-Rauch-Tarjan 1992]



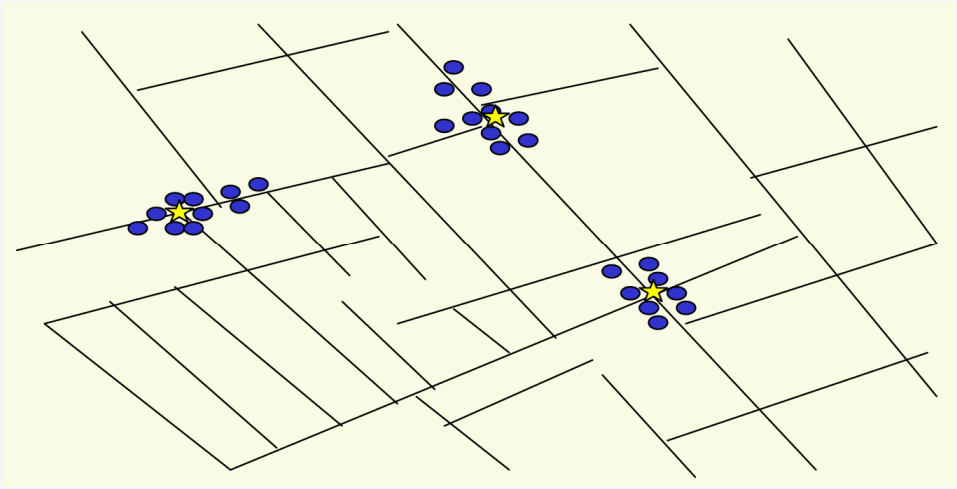
SECTION 4.7

4. GREEDY ALGORITHMS II

- *Dijkstra's algorithm*
- *minimum spanning trees*
- *Prim, Kruskal*
- *single-link clustering*

Clustering

Goal. Given a set U of n objects labeled $p_1, \ldots, p_n$, partition into clusters so that objects in different clusters are far apart.



outbreak of cholera deaths in London in 1850s (Nina Mishra)

Applications.

- Routing in mobile ad hoc networks.
- Document categorization for web search.
- Similarity searching in medical image databases
- Skycat: cluster 10^9 sky objects into stars, quasars, galaxies.
- ...

Clustering of maximum spacing

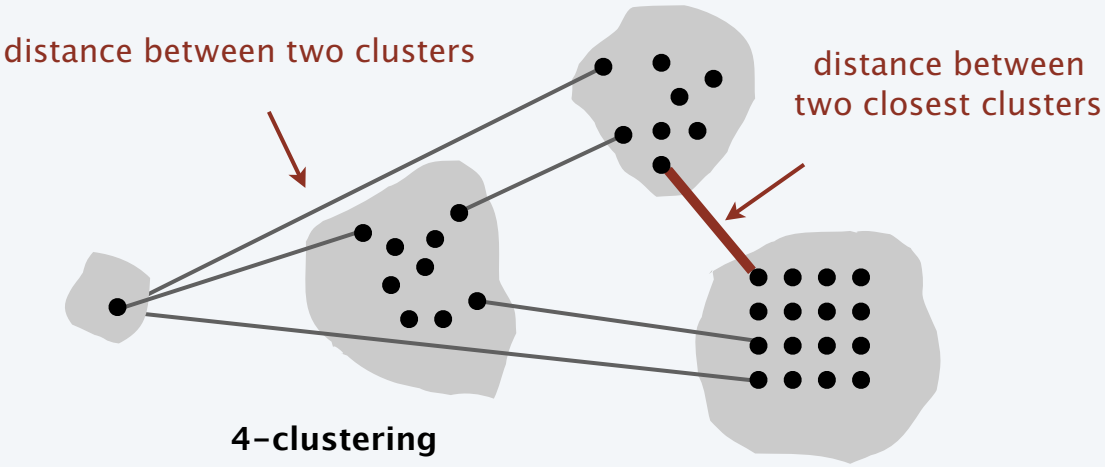
k-clustering. Divide objects into k non-empty groups.

Distance function. Numeric value specifying "closeness" of two objects.

- $d(p_i, p_j) = 0$ iff $p_i = p_j$ [identity of indiscernibles]
- $d(p_i, p_j) \geq 0$ [nonnegativity]
- $d(p_i, p_j) = d(p_j, p_i)$ [symmetry]

Spacing. Min distance between any pair of points in different clusters.

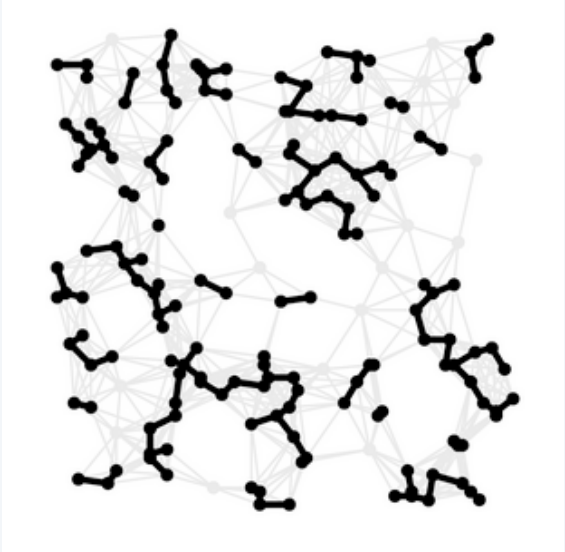
Goal. Given an integer k, find a k-clustering of maximum spacing.



Greedy clustering algorithm

"Well-known" algorithm in science literature for single-linkage k-clustering:

- Form a graph on the node set U, corresponding to n clusters.
- Find the closest pair of objects such that each object is in a different cluster, and add an edge between them.
- Repeat $n - k$ times until there are exactly k clusters.

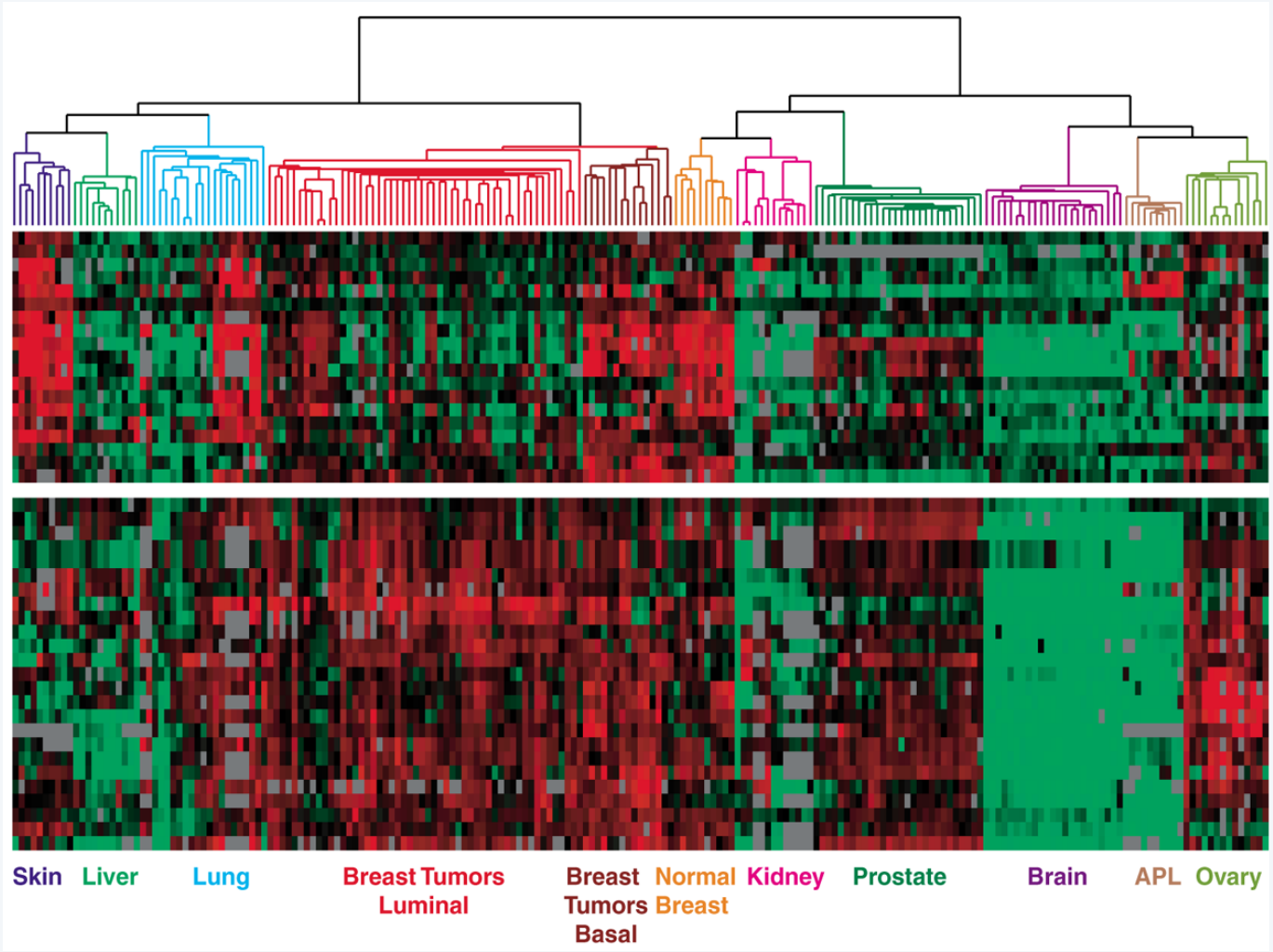


Key observation. This procedure is precisely Kruskal's algorithm (except we stop when there are k connected components).

Alternative. Find an MST and delete the $k - 1$ longest edges.

Dendrogram of cancers in human

Tumors in similar tissues cluster together.



Reference: Botstein & Brown group

gene expressed

gene not expressed