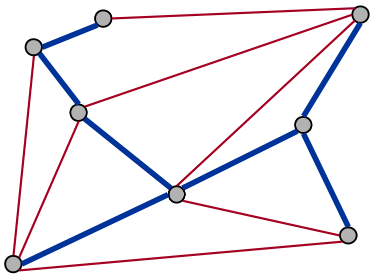
MST: Red Rule, Blue Rule

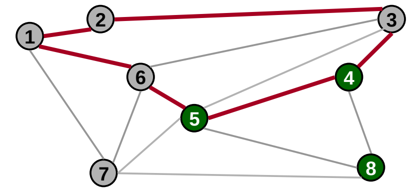


Some of these lecture slides are adapted from material in:
- *Data Structures and Algorithms*, R. E. Tarjan.
- *Randomized Algorithms*, R. Motwani and P. Raghavan.

Princeton University • COS 423 • Theory of Algorithms • Spring 2002 • Kevin Wayne

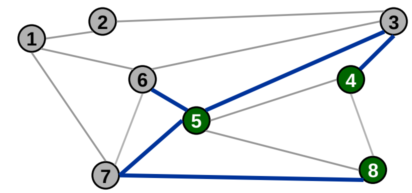
Cycles and Cuts

Cycle.
- A cycle is a set of arcs of the form {a,b}, {b,c}, {c,d}, . . ., {z,a}.



Path = 1-2-3-4-5-6-1
Cycle = {1, 2}, {2, 3}, {3, 4}, {4, 5}, {5, 6}, {6, 1}

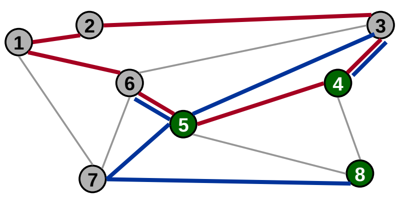
Cut.
- The cut induced by a subset of nodes S is the set of all arcs with exactly one endpoint in S.



S = {4, 5, 6}
Cut = {5, 6}, {5, 7}, {3, 4}, {3, 5}, {7, 8}

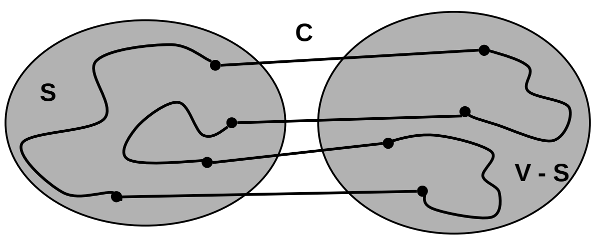
Cycle-Cut Intersection

A cycle and a cut intersect in an even number of arcs.



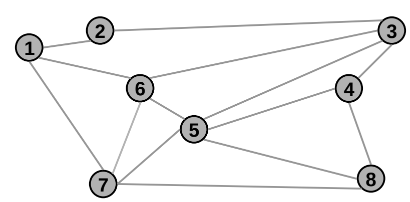
Intersection = {3, 4}, {5, 6}

Proof.

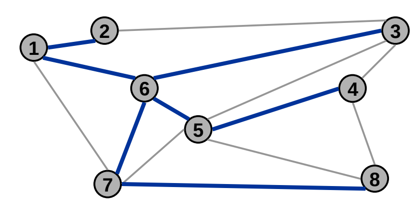


Spanning Tree

Spanning tree. **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 |V| - 1 arcs.
- T is acyclic and has |V| - 1 arcs.
- T is minimally connected: removal of any arc disconnects it.
- T is maximally acyclic: addition of any arc creates a cycle.
- T has a unique simple path between every pair of vertices.



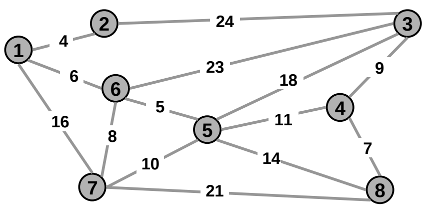
G = (V, E)



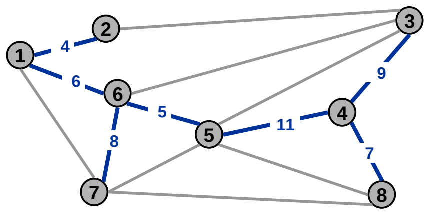
T = (V, F)

Minimum Spanning Tree

Minimum spanning tree. Given connected graph G with real-valued arc weights c_e, an *MST* is a spanning tree of G whose sum of arc weights is minimized.



G = (V, E)



T = (V, F)

w(T) = 50

Cayley's Theorem (1889). There are n^{n-2} spanning trees of K_n.

- n = |V|, m = |E|.
- Can't solve MST by brute force.

Applications

MST is central combinatorial problem with divserse applications.

- Designing physical networks.
  - telephone, electrical, hydraulic, TV cable, computer, road
- Cluster analysis.
  - delete long edges leaves connected components
  - finding clusters of quasars and Seyfert galaxies
  - analyzing fungal spore spatial patterns
- Approximate solutions to NP-hard problems.
  - metric TSP, Steiner tree
- Indirect applications.
  - describing arrangements of nuclei in skin cells for cancer research
  - learning salient features for real-time face verification
  - modeling locality of particle interactions in turbulent fluid flow
  - reducing data storage in sequencing amino acids in a protein

Optimal Message Passing

Optimal message passing.

- Distribute message to N agents.
- Each agent can communicate with some of the other agents, but their communication is (independently) detected with probability p_{ij}.
- Group leader wants to transmit message (e.g., Divx movie) to all agents so as to minimize the total probability that message is detected.

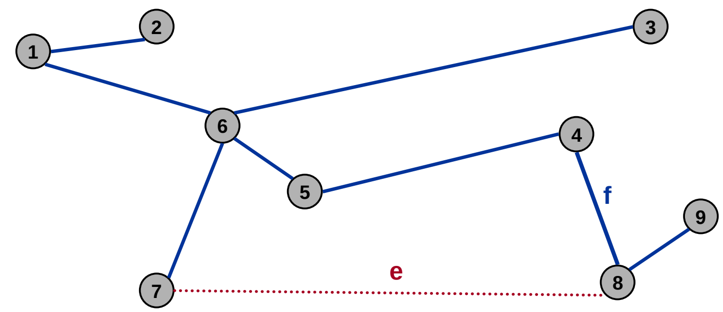
Objective.

- Find tree T that minimizes: $1 - \prod_{(i,j)\in T} (1 - p_{ij})$
- Or equivalently, that maximizes: $\prod_{(i,j)\in T} (1 - p_{ij})$
- Or equivalently, that maximizes: $\sum_{(i,j)\in T} \log(1 - p_{ij})$
- Or equivalently, MST with weights p_{ij}.

Fundamental Cycle

Fundamental cycle.

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

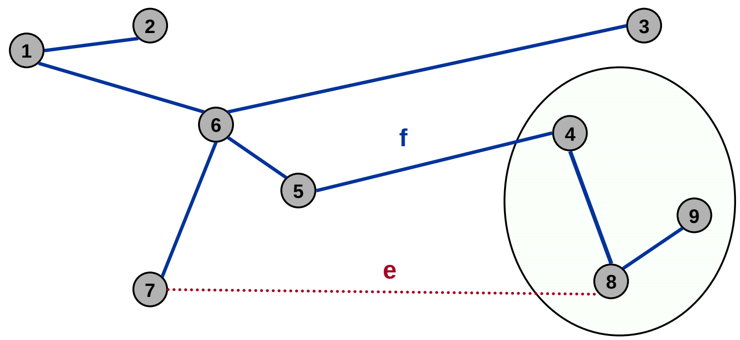


Cycle optimality conditions: For every non-tree arc e, and for every tree arc f in its fundamental cycle: $c_f \le c_e$.

Observation: If $c_f > c_e$ then T is not a MST.

Fundamental Cut

Fundamental cut.
- Deleting any tree arc f from T disconnects tree into two components with cut D.
- Adding back any arc e ∈ D to T - {f} results in new spanning tree.

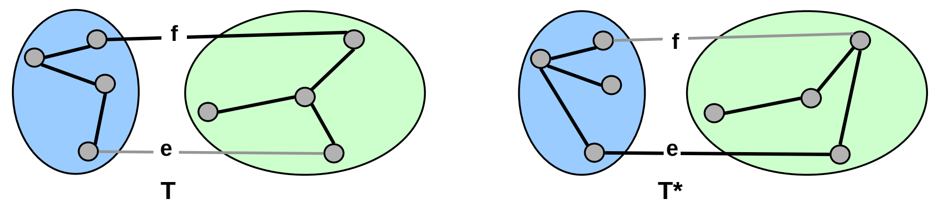


Cut optimality conditions: For every tree arc f, and for every non-tree arc e in its fundamental cut: $c_e \ge c_f$.

Observation: If $c_e < c_f$ then T not a MST.

MST: Cut Optimality Conditions

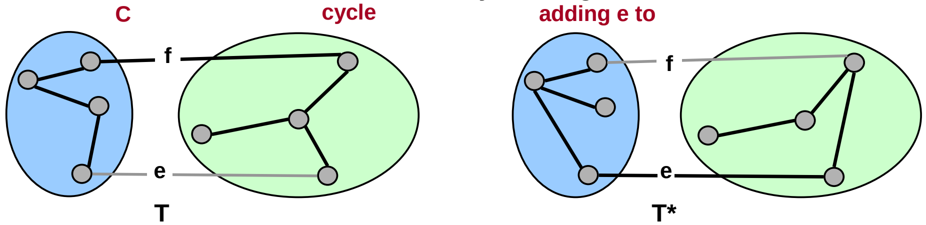
Theorem. Cut optimality ⇒ MST. (proof by contradiction)
- T = spanning tree that satisfies cut optimality conditions.
  T* = MST that has as many arcs in common with T as possible.
- If T = T*, then we are done. Otherwise, let f ∈ T s.t. f ∉ T*.
- Let D be fundamental cut formed by deleting f from T.



- Adding f to T* creates a fund cycle C, which shares (at least) two arcs with cut D. One is f, let e be another. Note: e ∉ T.
- Cut optimality conditions ⇒ $c_f \le c_e$.
- Thus, we can replace e with f in T* without increasing its cost.

MST: Cycle Optimality Conditions

Theorem. ~~Cut~~ Cycle optimality ⇒ MST. (proof by contradiction)
- T = spanning tree that satisfies ~~cut~~ cycle optimality conditions.
  T* = MST that has as many arcs in common with T as possible.
- If T = T*, then we are done. Otherwise, let ~~f ∈ T s.t. f ∉ T*~~. e ∈ T* s.t. e ∉ T
- Let ~~D~~ C be fundamental ~~cut~~ cycle formed by ~~deleting f from~~ adding e to T.

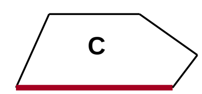


- ~~Adding f to~~ Deleting e from T* creates a fund ~~cycle C~~ cut D, which shares (at least) two arcs with ~~cut D~~ cycle C. One is ~~f~~ e, let ~~e~~ f be another. Note: ~~e ∉ T~~ f ∉ T*.
- ~~Cut~~ Cycle optimality conditions ⇒ $c_f \le c_e$.
- Thus, we can replace e with f in T* without increasing its cost.

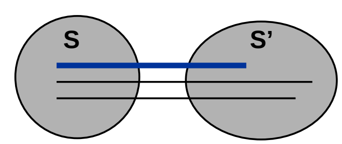
Towards a Generic MST Algorithm

If all arc weights are distinct:

- MST is unique.
- Arc with largest weight in cycle C is not in MST.
  - cycle optimality conditions



- Arc with smallest weight in cutset D is in MST.
  - cut optimality conditions



Generic MST Algorithm

Red rule.

- Let *C* be a cycle with no red arcs. Select an uncolored arc of C of max weight and color it red.

Blue rule.

- Let D be a cut with no blue arcs. Select an uncolored arc in D of min weight and color it blue.

Greedy algorithm.

- Apply the red and blue rules (non-deterministically!) until all arcs are colored. The blue arcs form a MST.
- Note: can stop once n-1 arcs colored blue.

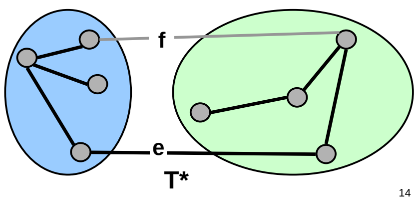
Greedy Algorithm: Proof of Correctness

Theorem. The greedy algorithm terminates. Blue edges form a MST.

Proof. (by induction on number of iterations)

> Color Invariant: There exists a MST T* containing all the blue arcs and none of the red ones.

- Base case: no arcs colored $\Rightarrow$ every MST satisfies invariant.
- Induction step: suppose color invariant true before blue rule.
  - let D be chosen cut, and let f be arc colored blue
  - if f $\in$ T*, T* still satisfies invariant
  - o/w, consider fundamental cycle C by adding f to T*
  - let e $\in$ C be another arc in D
  - e is uncolored and $c_e \geq c_f$ since
    - e $\in$ T* $\Rightarrow$ not red
    - blue rule $\Rightarrow$ not blue, $c_e \geq c_f$
  - T* $\cup$ { f } - { e } satisfies invariant



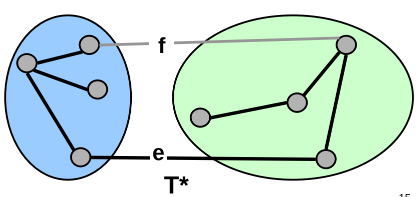
Greedy Algorithm: Proof of Correctness

Theorem. The greedy algorithm terminates. Blue edges form a MST.

Proof. (by induction on number of iterations)

> Color Invariant: There exists a MST T* containing all the blue arcs and none of the red ones.

- Base case: no arcs colored $\Rightarrow$ every MST satisfies invariant.
- Induction step: suppose color invariant true before ~~blue~~ red rule.
  - let ~~D~~ C be chosen ~~cut~~ cycle, and let ~~f~~ e be arc colored ~~blue~~ red
  - if ~~f $\in$ T*~~ e $\notin$ T*, T* still satisfies invariant
  - o/w, consider fundamental ~~cycle C~~ cut D ~~by adding f~~ deleting e from to T*
  - let ~~e $\in$ C~~ f $\in$ D be another arc in ~~D~~ C
  - ~~e~~ f is uncolored and $c_e \geq c_f$ since
    - ~~e $\in$ T*~~ f $\notin$ T* $\Rightarrow$ not ~~red~~ blue
    - ~~blue rule~~ red rule $\Rightarrow$ ~~not blue~~ f not red, $c_e \geq c_f$
  - T* $\cup$ { f } - { e } satisfies invariant

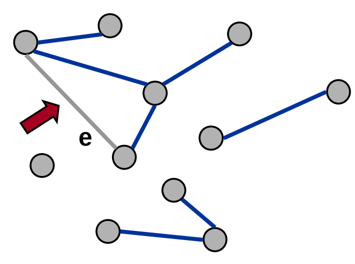


Greedy Algorithm: Proof of Correctness

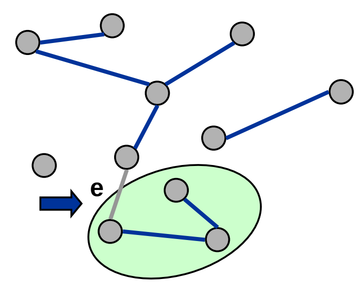
Proof (continued).

- Induction step: suppose color invariant true before red rule.
  - cut-and-paste

- Either the red or blue rule (or both) applies.
  - suppose arc e is left uncolored
  - blue edges form a forest



Case 1

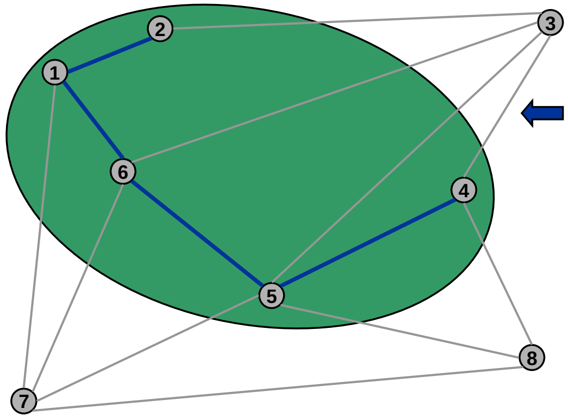


Case 2

Special Case: Prim's Algorithm

Prim's algorithm. (Jarník 1930, Dijkstra 1957, Prim 1959)

- S = vertices in tree connected by blue arcs.
- Initialize S = any vertex.
- Apply blue rule to cut induced by S.



Implementing Prim's Algorithm

Prim's Algorithm

```
Q ← PQinit()
for each v ∈ V
   key(v) ← ∞
   pred(v) ← nil
   PQinsert(v, Q)

key(s) ← 0
while (!PQisempty(Q))
   v = PQdelmin(Q)
   for each w ∈ Q s.t {v,w} ∈ E
      if key(w) > c(v,w)
         PQdeckey(w, c(v,w))
         pred(w) ← v
```

O(m + n log n)	Fib. heap
$O(n^2)$	array

Dijkstra's Shortest Path Algorithm

Dijkstra's ~~Prim's~~ Algorithm

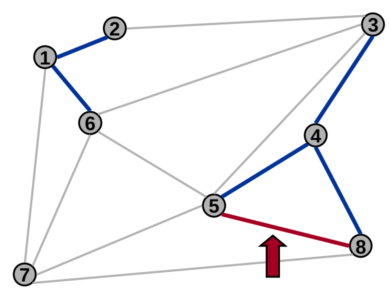
```
Q ← PQinit()
for each v ∈ V
   key(v) ← ∞
   pred(v) ← nil
   PQinsert(v, Q)

key(s) ← 0
while (!PQisempty(Q))
   v = PQdelmin(Q)
   for each w ∈ Q s.t {v,w} ∈ E
      if key(w) > ~~c(v,w)~~
         PQdeckey(w, ~~c(v,w)~~) c(v,w) + key(v)
         pred(w) ← v
```

O(m + n log n)	Fib. heap
$O(n^2)$	array

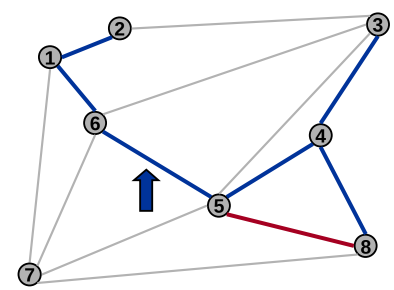
Special Case: Kruskal's Algorithm

Kruskal's algorithm (1956).

- Consider arcs in ascending order of weight.
  - if both endpoints of e in same blue tree, color red by applying red rule to unique cycle
  - else color e blue by applying blue rule to cut consisting of all vertices in blue tree of one endpoint



Case 1: {5, 8}



Case 2: {5, 6}

Implementing Kruskal's Algorithm

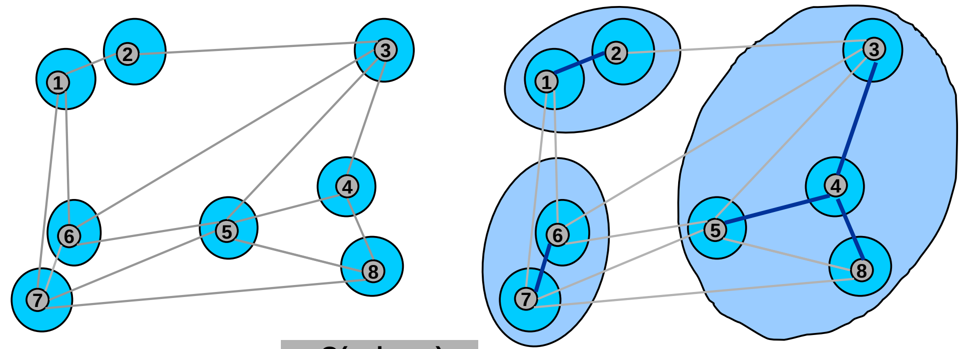
Kruskal's Algorithm

```
Sort edges weights in ascending order
c1 ≤ c2 ≤ ... ≤ cm.

S = φ
for each v ∈ V
   UFmake-set(v)

for i = 1 to m
   (v,w) = ei
   if (UFfind-set(v) ≠ UFfind-set(w))
      S ← S ∪ {i}
      UFunion(v, w)
```

O(n log n)	O(m α (m, n))
sorting	union-find

Special Case: Boruvka's Algorithm

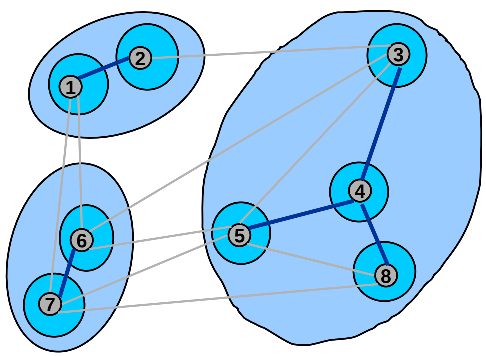
Boruvka's algorithm (1926).
- Apply blue rule to cut corresponding to each blue tree.
- Color all selected arcs blue.
- O(log n) phases since each phase halves total # nodes.



O(m log n)

Implementing Boruvka's Algorithm

Boruvka implementation.
- Contract blue trees, deleting loops and parallel arcs.
- Remember which edges were contracted in each super-node.



Advanced MST Algorithms

Deterministic comparison based algorithms.
- O(m log n) — Jarník, Prim, Dijkstra, Kruskal, Boruvka
- O(m log log n). — Cheriton-Tarjan (1976), Yao (1975)
- O(m β(m, n)). — Fredman-Tarjan (1987)
- O(m log β(m, n)). — Gabow-Galil-Spencer-Tarjan (1986)
- O(m α (m, n)). — Chazelle (2000)
- O(m). — Holy grail.

Worth noting.
- O(m) randomized. — Karger-Klein-Tarjan (1995)
- O(m) verification. — Dixon-Rauch-Tarjan (1992)

Linear Expected Time MST

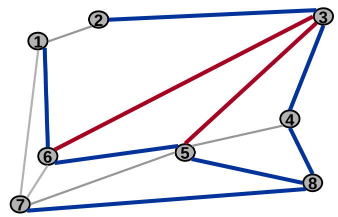
Random sampling algorithm. (Karger, Klein, Tarjan, 1995)

- If lots of nodes, use Boruvka.
  - decreases number of nodes by factor of 2
- If lots of edges, delete useless ones.
  - use random sampling to decrease by factor of 2
- Expected running time is O(m + n).

Filtering Out F-Heavy Edges

Definition. Given graph G and forest F, an edge e is F-heavy if both endpoints lie in the same component and $c_e > c_f$ for all edges f on fundamental cycle.

- Cycle optimality conditions: T* is MST $\Leftrightarrow$ no T*-heavy edges.
- If e is F-heavy for any forest F, then safe to discard e.
  - apply red rule to fundamental cycles



Forest F
F-heavy edges

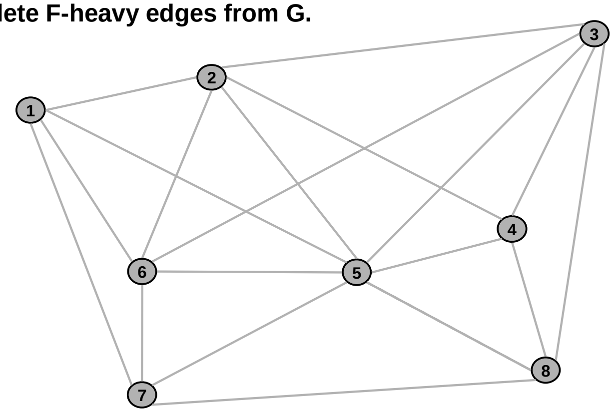
Verification subroutine. (Dixon-Rauch-Tarjan, 1992).

- Given graph G and forest F, is F is a MSF?
- In O(m + n) time, either answers (i) YES or (ii) NO and output all F-heavy edges.

Random Sampling

Random sampling.

- Obtain G(p) by independently including each edge with p = 1/2.
- Let F be MSF in G(p).
- Compute F-heavy edges in G.
- Delete F-heavy edges from G.

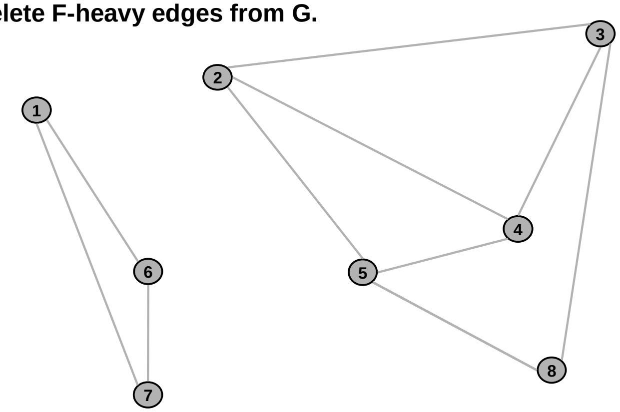


G

Random Sampling

Random sampling.

- Obtain G(p) by independently including each edge with p = 1/2.
- Let F be MSF in G(p).
- Compute F-heavy edges in G.
- Delete F-heavy edges from G.

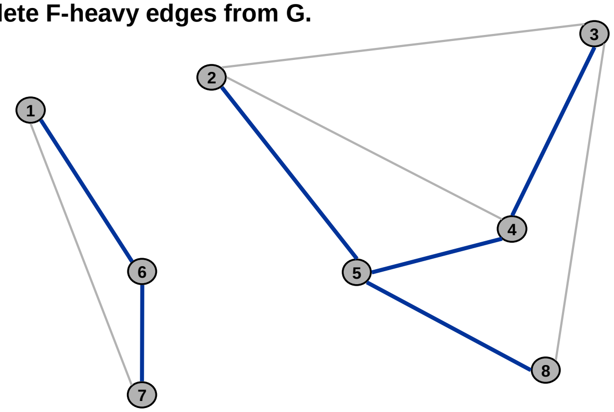


G(1/2)

Random Sampling

Random sampling.

- Obtain G(p) by independently including each edge with p = 1/2.
- Let F be MSF in G(p).
- Compute F-heavy edges in G.
- Delete F-heavy edges from G.

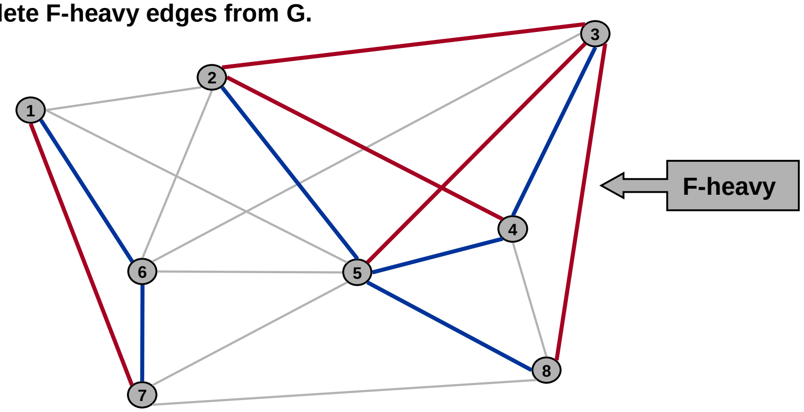


G(1/2) MSF F in G(1/2)

Random Sampling

Random sampling.

- Obtain G(p) by independently including each edge with p = 1/2.
- Let F be MSF in G(p).
- Compute F-heavy edges in G.
- Delete F-heavy edges from G.

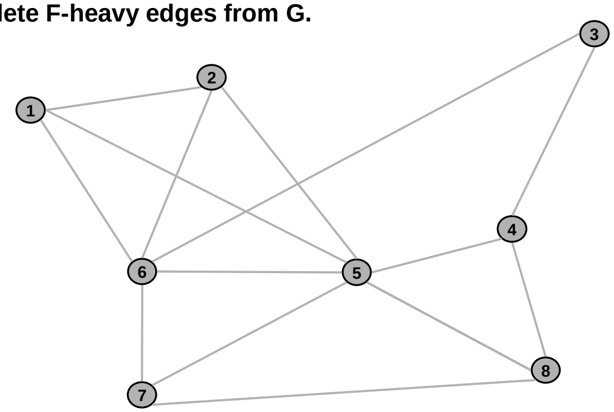


G MSF F in G(1/2)

Random Sampling

Random sampling.

- Obtain G(p) by independently including each edge with p = 1/2.
- Let F be MSF in G(p).
- Compute F-heavy edges in G.
- Delete F-heavy edges from G.



G

Random Sampling Lemma

Random sampling lemma. Given graph G, let F be a MSF in G(p). Then the expected number of F-light edges is $\le$ n / p.

Proof.

- WMA $c_1 \le c_2 \le \ldots \le c_m$, and that G(p) is constructed by flipping coin m times and including edge e_i if i^{th} coin flip is heads.
- Construct MSF F at same time using Kruskal's algorithm.
  - edge e_i added to F $\Leftrightarrow$ e_i is F-light
  - F-lightness of edge e_i depends only on first i-1 coin flips and does not change after phase i
- Phase k = period between when |F| = k-1 and |F| = k.
  - F-light edge has probability p of being added to F
  - # F-light edges in phase k ~ Geometric(p)
- Total # F-light edges $\prec$ NegativeBinomial(n, p).

Random Sampling Algorithm

Random Sampling Algorithm(G, m, n)

```
Run 3 phases of Boruvka's algorithm on G. Let G1 be
resulting graph, and let C be set of contracted edges.

IF G1 has no edges RETURN F ← C

G2 ← G1(1/2)
Compute MSF F2 of G2 recursively.

Compute all F2-heavy edges in G1, remove these
edges from G1, and let G' be resulting graph.

Compute MSF F' of G' recursively.

Return F ← C ∪ F'
```


Analysis of Random Sampling Algorithm

Theorem. The algorithm computes an MST in O(m+n) expected time.

Proof.

- Correctness: red-rule, blue-rule.
- Let T(m, n) denote expected running time to find MST on graph with n vertices and m arcs.
- G_1 has ≤ m arcs and ≤ n/8 vertices.
  - each Boruvka phase decreases n by factor of 2
- G_2 has ≤ n/8 vertices and expected # arcs ≤ m/2
  - each edge deleted with probability 1/2
- G' has ≤ n/8 vertices and expected # arcs ≤ n/4
  - random sampling lemma

$$T(m,n) \le \begin{cases} c(m+n) & \text{if } m \le 1 \text{ or } n \le 1 \\ \underbrace{T(m/2, n/8)}_{\text{MSF of } G_2} + \underbrace{T(n/4, n/8)}_{\text{MSF of } G'} + \underbrace{c(m+n)}_{\text{everything else}} & \text{otherwise} \end{cases}$$

$$\Rightarrow \quad T(m,n) \le 2c(m+n)$$