Foundations of Operations Research

Master of Science in Computer Engineering

Roberto Cordone
roberto.cordone@unimi.it

Tuesday 13.15 - 15.15
Thursday 10.15 - 13.15

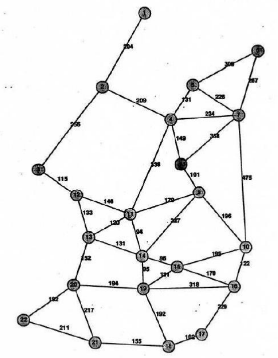
http://homes.di.unimi.it/~cordone/courses/2014-for/2014-for.html

A network design model

A telecommunication company wants to build a new fiberoptic network between some major European cities.

All cities should be connected to each other, directly or indirectly. A set of potential connections and the cost of building each link (proportional to the distance) are known.

Design the fiberoptic network of minimum total cost.



A natural combinatorial model is given by an edge-weighted undirected graph (V, E, c)

- V includes the cities
- E includes the potential links
- $c : E \to \mathbb{R}^+$ provides the cost of a link

We are looking for a subgraph $T = (U, X)$

What kind of subgraph?

A network design model

Let us denote by $T = (V, X)$ any feasible subgraph:

- it must include all vertices: it must be spanning
- it must include a path between any pair of vertices: it must be connected
- its total cost must be minimum: $X = \arg\min \sum_{e \in X} c_e$

Can it include cycles?

Given a cyclic connected subgraph, remove one edge e from a cycle:

- the result is connected
- if all cycles include an edge with $c_e \geq 0$, the result is not more expensive

The optimal solution includes no cycle: $T = (V, X)$ is an acyclic subgraph

Definitions

- a forest is an acyclic graph
- a tree is an acyclic connected graph
- a spanning tree is an acyclic connected spanning subgraph

If all cycles of G include an edge e with $c_e \geq 0$, T is a minimum spanning tree

Minimum spanning tree problem

Given

- an undirected connected graph $G = (V, E)$
  with $n = |V|$ vertices and $m = |E|$ edges
- a cost function $c : E \rightarrow \mathbb{R}$

find a subgraph $T^* = (U^*, X^*)$

1. spanning: U^* contains all vertices ($U^* = V$)
2. connected: X^* includes a path between each pair of vertices u and v
3. acyclic: X^* does not contain any cycle
4. of minimum total cost:

$$c_{X^*} \leq c_X \text{ for all } T = (U, X) \text{ enjoying properties 1, 2 e 3}$$

where $c_X = \sum_{e \in X} c_e$

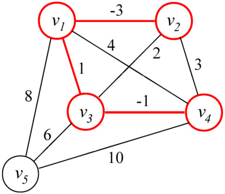
If G includes a cycle with all edges of negative cost, the minimum spanning tree problem is not a good model for the previous problem

But you can apply a simple adaptation: which one?

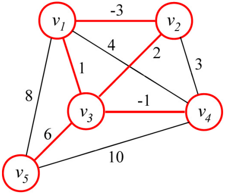
Examples

T is not an optimal solution because it is. . .

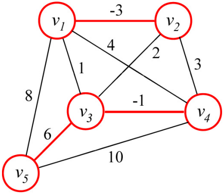
. . . nonspanning



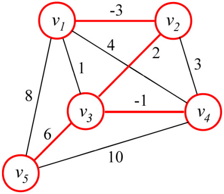
. . . cyclic



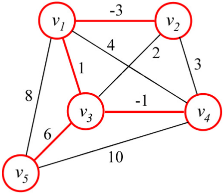
. . . nonconnected



. . . nonminimal ($c_X = 4$)



T^* is optimal ($c_{X^*} = 3$)



A second model: secure message transmission

Broadcast to all stations of a communication network a secret message, minimizing the probability of interception at the links.

We model the network as a graph $G = (V, E)$:

- vertices for the stations
- edges for the links
- a probability of interception $p_e \in [0; 1)$ for each edge

What is the probability of interception using a subset X of the edges?

$$f(X) = 1 - \prod_{e \in X} (1 - p_e)$$

i. e. the complement of the probability not to be intercepted at any link

$$\min_X f(X) \Leftrightarrow \max_X \log \prod_{e \in X} (1 - p_e) = \sum_{e \in X} \log (1 - p_e) \Leftrightarrow \min_X \sum_{e \in X} \log \frac{1}{(1 - p_e)}$$

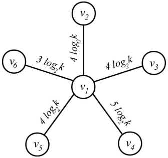
We are looking for a connected spanning subgraph (V, X) (nonnegative costs: the optimal subgraph is also acyclic)

A third model: compact binary sequence representation

You have a large number n of binary sequences of huge length k, and you want to represent them in a compact way

s_1 : [011100011101]	s_2 : [101101011001]	s_3 : [110100111001]
s_4 : [101001111101]	s_5 : [100100111101]	s_6 : [010101011100]

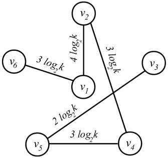
An idea is to select a reference sequence and provide the differences ("bit flips") between the other ones and it



s_1 : [011100011101]	$s_2 - s_1$: [1 2 6 10]	$s_3 - s_1$: [1 3 7 10]
$s_4 - s_1$: [1 2 4 6 7]	$s_5 - s_1$: [1 2 3 7]	$s_6 - s_1$: [3 6 12]

This pays if many sequences are similar to the reference

A better idea is to allow a connected set of differences



s_6 : [010101011100]	$s_1 - s_6$: [3 6 12]	$s_2 - s_1$: [1 2 6 10]
$s_4 - s_2$: [4 7 10]	$s_5 - s_4$: [3 4 6]	$s_3 - s_5$: [2 10]

A third model: compact binary sequence representation

Consider a complete undirected weighted graph:

- the vertices represent sequences
- the edges represent pairs of sequences
- the cost function is the number of bit flips between two sequences

c_{uv}	1	2	3	4	5	6
1	0	4	4	5	4	3
2	4	0	4	3	4	5
3	4	4	0	5	2	5
4	5	3	5	0	3	6
5	4	4	2	3	0	5
6	3	5	5	6	5	0

We look for a subgraph, which must be

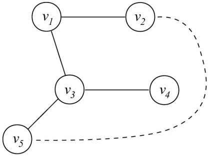
- spanning, to represent all sequences
- connected, to allow reconstructing any sequence from the reference
- of minimum cost, to save memory space

Since the costs are nonnegative, the subgraph is acyclic

Useful properties on trees

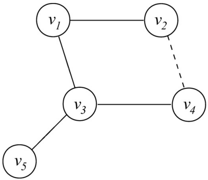
A tree contains exactly one path P_{uv} between any pair of vertices u and v

- *a tree is connected $\Rightarrow$ there is at least one path*
- *two paths form a cycle, but a tree is acyclic $\Rightarrow$ there is at most one path*



Adding an edge $[u, v]$ to a spanning tree yields exactly one cycle

- *the tree spans u and v and contains a path P_{uv} $\Rightarrow$ $[u, v] \cup P_{uv}$ is a cycle $\Rightarrow$ there is at least one cycle*
- *if adding $[u, v]$ yields at least two cycles, the original tree had two different paths between u and v (contrary to the previous thesis) adding $\Rightarrow$ $[u, v]$ yields at most one cycle*



A general scheme

The vertex set of an optimal spanning tree is obviously V

We want to build the edge set with a scheme of this kind:

1. Find a set of edges X certainly included in the edge set of an optimal solution
2. If (V, X) is an optimal solution, terminate
3. Otherwise, find an edge e^* such that $X \cup \{e^*\}$ is still included in the edge set of an optimal solution and go back to point 2

The scheme provides an optimal solution in a finite number of steps, provided that we can always find e^*

The optimal spanning tree problem is one of the few problems which admits such a scheme

How is it possible, and why?

A fundamental theorem

Given the following assumptions:

1. $S \subset V$ is a nonempty proper subset of vertices and
   $\Delta_S = \{[u, v] \in E : |\{u, v\}| \cap S = 1\}$ is its induced cut
2. $e^* = \arg \min_{e \in \Delta(S)} c_e$ is one of the edges of minimum cost in $\Delta(S)$

there exists an optimal spanning tree whose edge set includes e^*

3. $T^* = (V, X^*)$ is an optimal spanning tree and
   $X \subseteq X^*$ is a subset of its edges
4. $\Delta_S \cap X = \emptyset$

there exists an optimal spanning tree whose edge set includes $X \cup \{e^*\}$

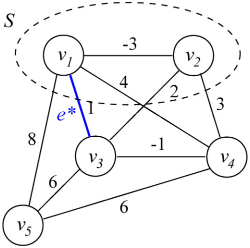
Such a tree can be different from T^*!

One can always enrich a subset of the edges of an optimal spanning tree with a minimum cost edge of a cut not intersecting the subset

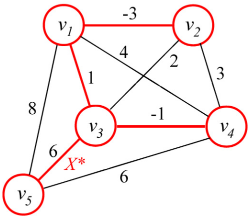
The only condition is that the graph be connected

Examples (1)

S, Δ_S, e^*, X



T^*



$S = \{v_1, v_2\}$

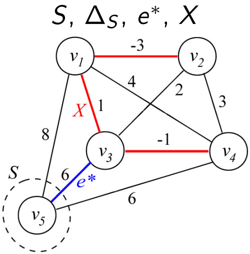
$\Delta_S = \{(v_1, v_3), (v_1, v_4), (v_1, v_5), (v_2, v_3), (v_2, v_4)\}$

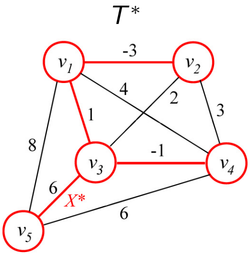
$e^* = (v_1, v_3)$

$X = \emptyset$

$\Rightarrow X \cup \{e^*\} = \{(v_1, v_3)\} \subseteq X^*$

Examples (2)





$S = \{v_5\}$

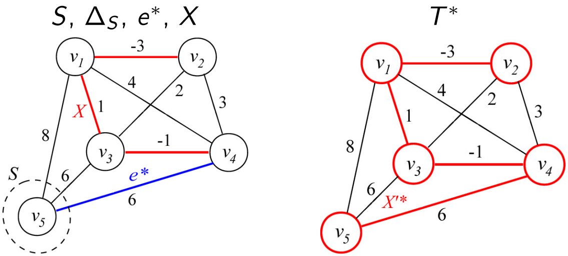
$\Delta_S = \{(v_1, v_5), (v_3, v_5), (v_4, v_5)\}$

$e^* = (v_3, v_5)$

$X = \{(v_1, v_2), (v_1, v_3), (v_3, v_4)\}$

$$\Rightarrow X \cup \{e^*\} = \{(v_1, v_2), (v_1, v_3), (v_3, v_4), (v_3, v_5)\} \subseteq X^*$$

Examples (3)



$S = \{v_5\}$

$\Delta_S = \{(v_1, v_5), (v_3, v_5), (v_4, v_5)\}$

$e^* = (v_4, v_5)$

$X = \{(v_1, v_2), (v_1, v_3), (v_3, v_4)\}$

$$\Rightarrow X \cup \{e^*\} = \{(v_1, v_2), (v_1, v_3), (v_3, v_4), (v_4, v_5)\}$$

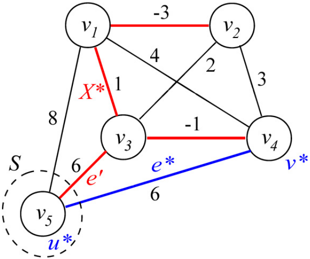
$X \cup \{e^*\} \not\subseteq X^*$, but it is included in another optimal spanning tree X'^*

But what if you consider $S = \{v_5\}$ and $X = \{(v_3, v_4), (v_3, v_5)\}$?

Proof (1)

There are two possible cases

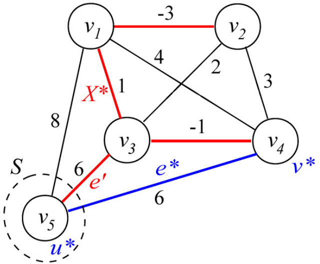
1. $e^* \in X^*$: since $X \subseteq X^*$, then $X \cup \{e^*\} \subseteq X^*$ and the thesis follows
2. $e^* = [u^*, v^*] \notin X^*$:
   the optimal solution $T = (V, X^*)$ is spanning and connected
   X^* includes a path $P_{u^* v^*}$ between u^* and v^*
   $P_{u^* v^*}$ intersects Δ_S in at least one edge e'



adding e^* to X^* produces a cycle
removing e' from this cycle yields another spanning tree
(the extreme vertices of e' are now connected through e^*)

Proof (2)

$(V, X^* \cup \{e^*\} \setminus \{e'\})$ is another spanning tree
and its cost is $c_{X^*} + c_{e^*} - c_{e'}$ (where $c_{X^*} = \sum_{e \in X^*} c_e$)



Notice that

- $T = (V, X^*)$ is optimal $\Rightarrow c_{X^*} + c_{e^*} - c_{e'} \geq c_{X^*} \Rightarrow c_{e^*} \geq c_{e'}$
- $e^* = \arg\min_{e \in \Delta_S} c_e$ and $e' \in \Delta_S \Rightarrow c_{e^*} \leq c_{e'}$

which implies that $c_{e^*} = c_{e'}$ (the two edges have the same cost)

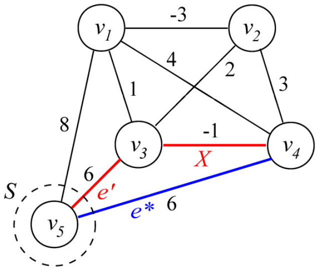
The two spanning trees have equal cost: the new spanning tree is optimal

Proof (3)

Given a partial optimal solution (V, X), if we find a vertex set $S \subset V$ whose induced cut Δ_S does not intersect X, we can augment X obtaining a partial optimal solution $(V, X \cup \{e^*\})$

Sooner or later, we will obtain a complete optimal solution

If $\Delta_S \cap X \neq \emptyset$, one cannot correctly enlarge set X:
either $e^* \in X$ (and X does not grow) or e^* closes a cycle with X (and the new tree includes e^* and $X \setminus \{e'\}$, but not X)



A general scheme (revisited)

1. Set $X := \emptyset$ (to be included in an optimal solution)
2. Find a cut Δ_S not intersecting X; if there is none, terminate
3. Otherwise, set $X := X \cup \arg\min_{e \in \Delta_S} c_e$ and go to step 2

The scheme works because

- X is always included in an optimal solution (theorem)
- X is augmented step by step (since Δ_S does not intersect X)
- when every cut intersects X, (V, X) is a spanning tree

$\Rightarrow$ in the end, (V, X) is an optimal spanning tree

Algorithms

Different algorithms apply this scheme

Prim's algorithm (1957)

- S collects the extreme vertices of the edges of X
  (*necessarily,* Δ_S *does not intersect* X)
  at first S is a single vertex, chosen *ad libitum*
- $e^* := \arg\min_{e \in \Delta_S} c_e$

Kruskal's algorithm (1956)

- first find the minimum cost edge $e^* := \arg\min_{e \in E \setminus X} c_e$
- if there is a cut including e and not intersecting X, add e^* to X
  (i. e. if the extreme vertices of e^* are disconnected in X),
  otherwise, remove e^* from E

Prim's algorithm

Prim(V, E, c)

$X := \emptyset$; $S := \{\bar{v}\}$;

While $S \subset V$

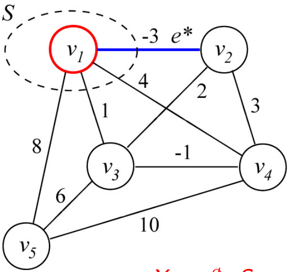
$e^* = [u^*, v^*] := \arg\min_{e \in \Delta_S} c_e$;

$X := X \cup \{e^*\}$;

$S := S \cup \{u^*, v^*\}$; { One of the two extremes is already in S }

Return (S, X);

Application of Prim's algorithm (1)



Prim(V, E, c)

$X := \emptyset$; $S := \{\bar{v}\}$;

While $S \subset V$

$\quad e^* := (u^*, v^*) := \arg\min_{e \in \Delta_S} c_e$;

$\quad X := X \cup \{e^*\}$;

$\quad S := S \cup \{u^*, v^*\}$;

Return (S, X);

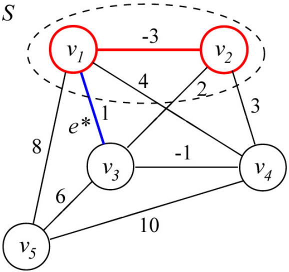
$X := \emptyset$; $S := \{v_1\}$;

$e^* := (v_1, v_2)$;

$X := \{(v_1, v_2)\}$;

$S := \{v_1, v_2\}$;

Application of Prim's algorithm (2)



Prim(V, E, c)

$X := \emptyset$; $S := \{\bar{v}\}$;

While $S \subset V$

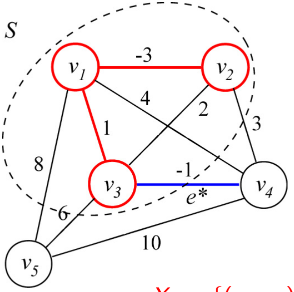
$e^* := (u^*, v^*) := \arg\min_{e \in \Delta_S} c_e$;

$X := X \cup \{e^*\}$;
$S := S \cup \{u^*, v^*\}$;

Return (S, X);

$X := \{(v_1, v_2)\}$; $S := \{v_1, v_2\}$;
$e^* := (v_1, v_3)$;
$X := \{(v_1, v_2), (v_1, v_3)\}$;
$S := \{v_1, v_2, v_3\}$;

Application of Prim's algorithm (3)



Prim(V, E, c)

$X := \emptyset$; $S := \{\bar{v}\}$;

While $S \subset V$

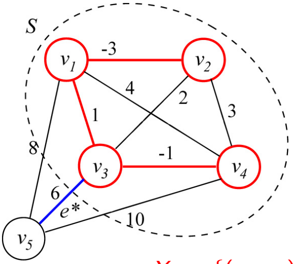
$e^* := (u^*, v^*) := \arg\min_{e \in \Delta_S} c_e$;

$X := X \cup \{e^*\}$;
$S := S \cup \{u^*, v^*\}$;

Return (S, X);

$X := \{(v_1, v_2), (v_1, v_3)\}$;
$S := \{v_1, v_2, v_3\}$;
$e^* := (v_3, v_4)$;
$X := \{(v_1, v_2), (v_1, v_3), (v_3, v_4)\}$;
$S := \{v_1, v_2, v_3, v_4\}$;

Application of Prim's algorithm (4)



Prim(V, E, c)

$X := \emptyset$; $S := \{\bar{v}\}$;

While $S \subset V$

$e^* := (u^*, v^*) := \arg\min_{e \in \Delta_S} c_e$;

$X := X \cup \{e^*\}$;

$S := S \cup \{u^*, v^*\}$;

Return (S, X);

$X := \{(v_1, v_2), (v_1, v_3, (v_3, v_4))\}$;

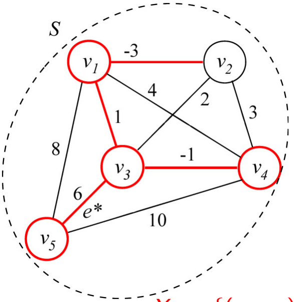
$S := \{v_1, v_2, v_3, v_4\}$;

$e^* := (v_3, v_5)$;

$X := \{(v_1, v_2), (v_1, v_3), (v_3, v_4), (v_3, v_5)\}$;

$S := \{v_1, v_2, v_3, v_4, v_5\}$;

Application of Prim's algorithm (5)



Prim(V, E, c)

$X := \emptyset;\ S := \{\bar{v}\};$

While $\ S \subset V$

$\quad e^* = (u^*, v^*) := \arg\min_{e \in \Delta_S} c_e;$

$\quad X := X \cup \{e^*\};$
$\quad S := S \cup \{u^*, v^*\};$

Return $\ (S, X);$

$X := \{(v_1, v_2), (v_1, v_3, (v_3, v_4)), (v_3, v_5)\};$
$S := \{v_1, v_2, v_3, v_4, v_5\} = V;$

Complexity of Prim's algorithm

Prim's algorithm consists of an initial step of complexity T_{in}
and a certain number $i_{\max}$ of iterations of complexity $T^{(i)}_{\text{iter}}$

$$T = T_{\text{in}} + \sum_{i=1}^{i_{\max}} T^{(i)}_{\text{iter}}$$

Prim(V, E, c)
$X := \emptyset$; $S := \{\bar{v}\}$;
While $S \subset V$
  $e^* := (u^*, v^*) := \arg\min_{e \in \Delta_S} c_e$;
  $X := X \cup \{e^*\}$;
  $S := S \cup \{u^*, v^*\}$;
Return (S, X);

$T_{\text{in}} \in O(1)$
$i_{\max} = n - 1$ (*one vertex at a time*)
$T^{(i)}_{\text{iter}} = \alpha$ (*to be determined*)

Overall $T \in O(\alpha n)$

Minimum cost edge identification (1)

Possible implementations

1. Scan all the edges and verify which ones belong to Δ_S: $O(m)$
2. Maintain subset Δ_S
   - build it: $O(n)$
   - find the minimum cost element: $O(m)$
   - update it: $O(n)$
3. Maintain for each $v \in V \setminus S$ the cheapest edge in $\Delta_S \cap \Delta_{\{v\}}$

$$\tilde{e}_v = \arg \min_{[u,v] \in \Delta_S \cap \Delta_{\{v\}}} c_e$$

   - build $\tilde{e}_v$: $O(n)$
   - find the minimum $\tilde{e}_v$: $O(n)$
   - update $\tilde{e}_v$: $O(n)$

Prim(V, E, c)

$X := \emptyset$; $S := \{\bar{v}\}$;

While $S \subset V$

$e^* = (u^*, v^*) := \arg \min_{e \in \Delta_S} c_e$;

$X := X \cup \{e^*\}$;

$S := S \cup \{u^*, v^*\}$;

Return (S, X);

$\rightarrow$ the complexity of the first implementation is $T \in O(mn)$

Minimum cost edge identification (2)

Possible implementations

1. Scan all the edges and verify which ones belong to Δ_S: $O(m)$
2. Maintain subset Δ_S
   - build it: $O(n)$
   - find the minimum cost element: $O(m)$
   - update it: $O(n)$
3. Maintain for each $v \in V \setminus S$ the cheapest edge in $\Delta_S \cap \Delta_{\{v\}}$

$$\tilde{e}_v = \arg \min_{[u,v] \in \Delta_S \cap \Delta_{\{v\}}} c_e$$

   - build $\tilde{e}_v$: $O(n)$
   - find the minimum $\tilde{e}_v$: $O(n)$
   - update $\tilde{e}_v$: $O(n)$

Prim(V, E, c)

$X := \emptyset$; $S := \{\bar{v}\}$; $D := \Delta_{\bar{v}}$;

While $S \subset V$

  $e^* = (u^*, v^*) := \arg\min_{e \in D} c_e$;

  $X := X \cup \{e^*\}$;

  $S := S \cup \{u^*, v^*\}$;

  For each $w \in S$

    $D := D \setminus \{(w, v^*)\}$;

  For each $w \in V \setminus S$

    $D := D \cup \{(w, v^*)\}$;

Return (S, X);

$\rightarrow$ the complexity of the second implementation is $T \in O(mn)$

Minimum cost edge identification (3)

Possible implementations

1. Scan all the edges and verify which ones belong to Δ_S: $O(m)$
2. Maintain subset Δ_S
   - build it: $O(n)$
   - find the minimum cost element: $O(m)$
   - update it: $O(n)$
3. Maintain for each $v \in V \setminus S$ the cheapest edge in $\Delta_S \cap \Delta_{\{v\}}$

$$\tilde{e}_v = \arg \min_{[u,v] \in \Delta_S \cap \Delta_{\{v\}}} c_e$$

   - build $\tilde{e}_v$: $O(n)$
   - find the minimum $\tilde{e}_v$: $O(n)$
   - update $\tilde{e}_v$: $O(n)$

Prim(V, E, c)

$X := \emptyset$; $S := \{\bar{v}\}$;

For each $w \in V \setminus \{\bar{v}\}$

$\quad \tilde{e}_w := [\bar{v}, w]$;

While $S \subset V$

$\quad e^* = (u^*, v^*) := \arg \min_{w \in V \setminus S} \tilde{e}_w$;

$\quad X := X \cup \{e^*\}$;

$\quad S := S \cup \{u^*, v^*\}$;

$\quad$ **For each** $w \in V \setminus S$

$\quad\quad$ **If** $c_{wv^*} < c_{\tilde{e}_w}$

$\quad\quad$ **then** $\tilde{e}_w := (w, v^*)$;

Return (S, X);

$\rightarrow$ the complexity of the third implementation is $T \in O(n^2)$

Kruskal's algorithm

Start with $X = \emptyset$

Find the minimum cost edge e^* not in X and not discarded

- if there is a cut Δ_S including e^* and not intersecting X add e^* to X

  Notice that it is not required to determine S, because

  $$\exists S \subset V : e^* \in \Delta_S \text{ and } \Delta_S \cap X = \emptyset \Leftrightarrow X \cup \{e^*\} \text{ is acyclic}$$

- if $\nexists S$, it will not exist for any larger $X \Rightarrow$ discard e^* permanently

Kruskal(V, E, c)

$X := \emptyset$; $S := V$;

$E' := E$; { Not yet discarded edges }

While $E' \neq \emptyset$

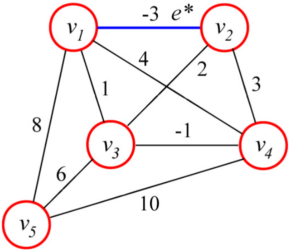
  $e^* := \arg\min_{e \in E'} c_e$;

  $E' := E' \setminus \{e^*\}$;

  If Acyclic$(X \cup \{e^*\})$ **then** $X := X \cup \{e^*\}$;

Return (S, X);

Application of Kruskal's algorithm (1)



Kruskal(V, E, c)

$X := \emptyset;\ S := V;$

$E' := E;$

While $E' \neq \emptyset$

$\quad e^* := \arg\min_{e \in E'} c_e$

$\quad E' := E' \setminus \{e^*\};$

$\quad$ **If** $\text{Acyclic}(X \cup \{e^*\})$

$\quad\quad$ **then** $X := X \cup \{e^*\};$

Return $(S, X);$

$X := \emptyset;$

$|E'| = 9;$

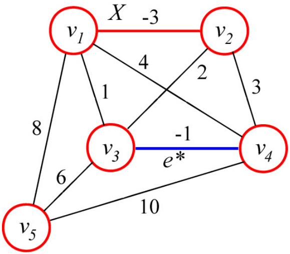
$e^* := (v_1, v_2);$

$|E'| = 8;$

$X \cup \{e^*\} := \{(v_1, v_2)\}$ acyclic

$X := \{(v_1, v_2)\};$

Application of Kruskal's algorithm (2)



Kruskal(V, E, c)

$X := \emptyset$; $S := V$;

$E' := E$; $\quad |E'| = 8$;

While $E' \neq \emptyset$

$\quad e^* := \arg\min_{e \in E'} c_e \qquad e^* := (v_3, v_4)$;

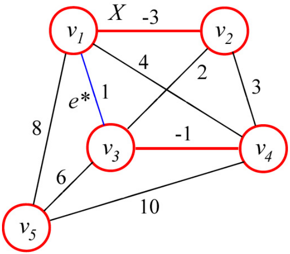
$\quad E' := E' \setminus \{e^*\}$; $\qquad |E'| = 7$;

$\quad$ **If** Acyclic$(X \cup \{e^*\})$ $\qquad X \cup \{e^*\} := \{(v_1, v_2), (v_3, v_4)\}$ acyclic

$\qquad$ **then** $X := X \cup \{e^*\}$; $\qquad X := \{(v_1, v_2), (v_3, v_4)\}$;

Return (S, X);

Application of Kruskal's algorithm (3)



Kruskal(V, E, c)

$X := \emptyset$; $S := V$;

$E' := E$; $|E'| = 7$;

While $E' \neq \emptyset$

$e^* := \arg\min_{e \in E'} c_e$ $e^* := (v_1, v_3)$;

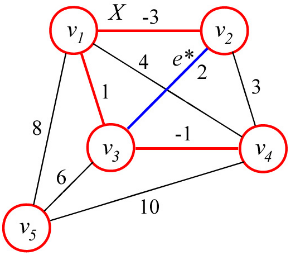
$E' := E' \setminus \{e^*\}$; $|E'| = 6$;

If Acyclic$(X \cup \{e^*\})$ $X \cup \{e^*\}$ acyclic

then $X := X \cup \{e^*\}$; $X := \{(v_1, v_2), (v_3, v_4), (v_1, v_3)\}$;

Return (S, X);

Application of Kruskal's algorithm (4)



Kruskal(V, E, c)

$X := \emptyset$; $S := V$;

$E' := E$;

While $E' \neq \emptyset$

$e^* := \arg\min_{e \in E'} c_e$

$E' := E' \setminus \{e^*\}$;

If Acyclic$(X \cup \{e^*\})$

then $X := X \cup \{e^*\}$;

Return (S, X);

$|E'| = 6$;

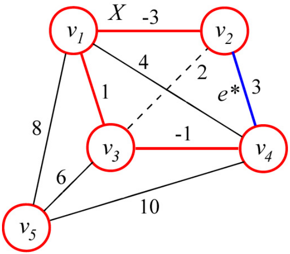
$e^* := (v_2, v_3)$;

$|E'| = 5$;

$X \cup \{e^*\}$ cyclic: discard e^*

$X := \{(v_1, v_2), (v_3, v_4), (v_1, v_3)\}$;

Application of Kruskal's algorithm (5)



Kruskal(V, E, c)

$X := \emptyset$; $S := V$;

$E' := E$;

While $E' \neq \emptyset$

$e^* := \arg\min_{e \in E'} c_e$

$E' := E' \setminus \{e^*\}$;

If Acyclic$(X \cup \{e^*\})$

then $X := X \cup \{e^*\}$;

Return (S, X);

$|E'| = 5$;

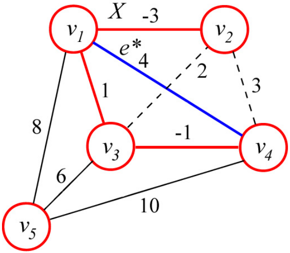
$e^* := (v_2, v_4)$;

$|E'| = 4$;

$X \cup \{e^*\}$ cyclic: discard e^*

$X := \{(v_1, v_2), (v_3, v_4), (v_1, v_3)\}$;

Application of Kruskal's algorithm (6)



Kruskal(V, E, c)

$X := \emptyset$; $S := V$;

$E' := E$;

$|E'| = 4$;

While $E' \neq \emptyset$

$e^* := \arg\min_{e \in E'} c_e$

$E' := E' \setminus \{e^*\}$;

If Acyclic$(X \cup \{e^*\})$

then $X := X \cup \{e^*\}$;

Return (S, X);

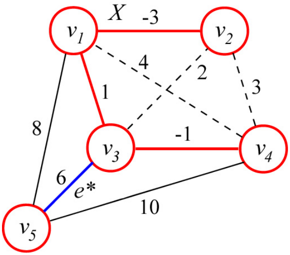
$e^* := (v_1, v_4)$;

$E' = \{(v_1, v_5), (v_3, v_5), (v_4, v_5)\}$;

$X \cup \{e^*\}$ cyclic: discard e^*

$X := \{(v_1, v_2), (v_3, v_4), (v_1, v_3)\}$;

Application of Kruskal's algorithm (7)



Kruskal(V, E, c)

$X := \emptyset$; $S := V$;

$E' := E$;

While $E' \neq \emptyset$

$e^* := \arg\min_{e \in E'} c_e$

$E' := E' \setminus \{e^*\}$;

If Acyclic$(X \cup \{e^*\})$

then $X := X \cup \{e^*\}$;

Return (S, X);

$E' = \{(v_1, v_5), (v_3, v_5), (v_4, v_5)\}$;

$e^* := (v_3, v_5)$;

$E' = \{(v_1, v_5), (v_4, v_5)\}$;

$X \cup \{e^*\}$ acyclic

$X := \{(v_1, v_2), (v_3, v_4), (v_1, v_3), (v_3, v_5)\}$;

Anticipated termination

Given a tree, a leaf is a vertex with a single incident arc

- any acyclic graph with $n > 1$ vertices includes at least one leaf

Proof by contradiction: otherwise, the visit of the tree would never terminate...

Consequently

- an acyclic graph with n vertices has $m \leq n - 1$ edges

Proof by induction

- *an acyclic graph with $n = 1$ vertex has $m = 0$ leaves*
- *a generic acyclic graph with $n > 1$ vertices has a leaf;*
  removing it produces an acyclic graph with n' vertices and m' edges (where $n' = n - 1$ and $m' = m - 1$);
  if for that graph $m' \leq n' - 1 \Rightarrow m \leq n - 1$

Therefore Kruskal's algorithm can terminate as soon as $|X| = n - 1$

Complexity of Kruskal's algorithm

Kruskal's algorithm consists of an initial step of complexity $T_{\rm in}$ and a certain number $i_{\max}$ of iterations of complexity $T^{(i)}_{\rm iter}$

$$T = T_{\rm in} + \sum_{i=1}^{i_{\max}} T^{(i)}_{\rm iter}$$

Kruskal(V, E, c)
$X := \emptyset;$
$E' := E;$
While $|X| < |V| - 1$
  $e^* := \arg\min_{e \in E'} c_e$
  $E' := E' \setminus \{e^*\};$
  If $\text{Acyclic}(X \cup \{e^*\})$
  then $X := X \cup \{e^*\};$
Return $(V, X);$

$T_{\rm in} \in O(1)$
$i_{\max} \leq m$ (*one edge at a time*)
$T^{(i)}_{\rm iter} \in O(\alpha + \beta)$
(α and β to be determined)

Overall $T \in O((\alpha + \beta)\, m)$

Minimum cost edge identification (1)

Possible implementations

1. Scan all the nondiscarded edges: $O(m)$
2. Sort E' by nondecreasing costs:
   - build it: $O(m \log m)$
   - extract the minimum: $O(1)$
3. Maintain E' as a *min-heap*
   - build it: $O(m)$
   - extract the minimum: $O(1)$
   - update it: $O(\log m)$

Kruskal(V, E, c)

$X := \emptyset$;

$E' := E$;

While $|X| < |V| - 1$

$\quad e^* := \arg\min_{e \in E'} c_e$

$\quad E' := E' \setminus \{e^*\}$;

$\quad$ **If** Acyclic$(X \cup \{e^*\})$

$\quad\quad$ **then** $X := X \cup \{e^*\}$;

Return (V, X);

$\rightarrow$ the complexity is $T \in O\left(m^2 + m\beta\right)$

Minimum cost edge identification (2)

Possible implementations

1. Scan all the nondiscarded edges: $O(m)$
2. Sort E' by nondecreasing costs:
   - build it: $O(m \log m)$
   - extract the minimum: $O(1)$
3. Maintain E' as a *min-heap*
   - build it: $O(m)$
   - extract the minimum: $O(1)$
   - update it: $O(\log m)$

Kruskal(V, E, c)

$X := \emptyset$;

$E' := E$;

Sort(E');

While $|X| < |V| - 1$

$\quad e^* := \mathrm{First}(E')$;
$\quad E' := E' \setminus \{e^*\}$;
$\quad$**If** Acyclic$(X \cup \{e^*\})$
$\quad\quad$**then** $X := X \cup \{e^*\}$;

Return (V, X);

$\rightarrow$ the complexity is $T \in O(m \log m + m\beta)$

Minimum cost edge identification (3)

Possible implementations

1. Scan all the nondiscarded edges: $O(m)$
2. Sort E' by nondecreasing costs:
   - build it: $O(m \log m)$
   - extract the minimum: $O(1)$
3. Maintain E' as a *min-heap*:
   - build it: $O(m)$
   - extract the minimum: $O(1)$
   - update it: $O(\log m)$

Kruskal(V, E, c)

$X := \emptyset$;

$E' := E$;

BuildMinHeap(E');

While $|X| < |V| - 1$

$e^* := \text{ExtractMinimum}(E')$;
$E' := E' \setminus \{e^*\}$;
$\text{Heapify}(E')$;
If Acyclic$(X \cup \{e^*\})$
then $X := X \cup \{e^*\}$;

Return (V, X);

$\rightarrow$ the complexity is $T \in O(m \log m + m\beta)$

Acyclicity test (1)

Possible implementations

1. Visit the graph from u^* and verify whether v^* can be reached: $O(n)$
2. Maintain X as a *merge-find-set*
   - build it: $O(n)$
   - find and compare the components of u^* and v^*: $\approx O(1)$
   - merge the components: $O(1)$

Kruskal(V, E, c)

$X := \emptyset$;

$E' := E$;

BuildMinHeap(E');

While $|X| < |V| - 1$

$\quad e^* := \text{ExtractMinimum}(E')$;
$\quad E' := E' \setminus \{e^*\}$;
$\quad \text{Heapify}(E')$;
$\quad$ **If not** Reachable(u^*, v^*, X)
$\quad\quad$ **then** $X := X \cup \{e^*\}$;

Return (V, X);

$\rightarrow$ the complexity is $T \in O(m \log m + mn)$

Acyclicity test (2)

Possible implementations

1. Visit the graph from u^* and verify whether v^* can be reached: $O(n)$
2. Maintain X as a *merge-find-set*
   - build it: $O(n)$
   - find and compare the components of u^* and v^*: $\approx O(1)$
   - merge the components: $O(1)$

Kruskal(V, E, c)

$X := \emptyset$;

$\mathcal{C} :=$ BuildMFSet(X);

$E' := E$;

BuildMinHeap(E');

While $|X| < |V| - 1$

  $e^* := \mathrm{ExtractMinimum}(E')$;
  $E' := E' \setminus \{e^*\}$;
  $\mathrm{Heapify}(E')$;
  If DiffComponents$(u^*, v^*, \mathcal{C})$
    then $X := X \cup \{e^*\}$;

    Merge$(u^*, v^*, \mathcal{C})$;

Return (V, X);

$\rightarrow$ the complexity is $T \in O(m \log m)$