Minimum Spanning Trees

Birgit Vogtenhuber

Outline

- Introduction and Definitions
- A general idea for algorithms
- A characterization of “good” edges
- Prim’s algorithm
- Kruskal’s algorithm

Trees in (un)weighted Graphs

- Given an unweighted connected graph $G = (V, E)$, we can compute a tree T with all shortest paths from a root s to the other vertices using breadth first search.

  $\Rightarrow$ This does not work if G is a weighted graph.

- Given an unweighted connected graph $G = (V, E)$ with n vertices, every subtree with n vertices has the **same** total edge length $n - 1$.

  $\Rightarrow$ This is not true if G is a weighted graph.

This topic: Trees in weighted graphs with minimum total edge length (edge weight / edge cost).

Problem Definition

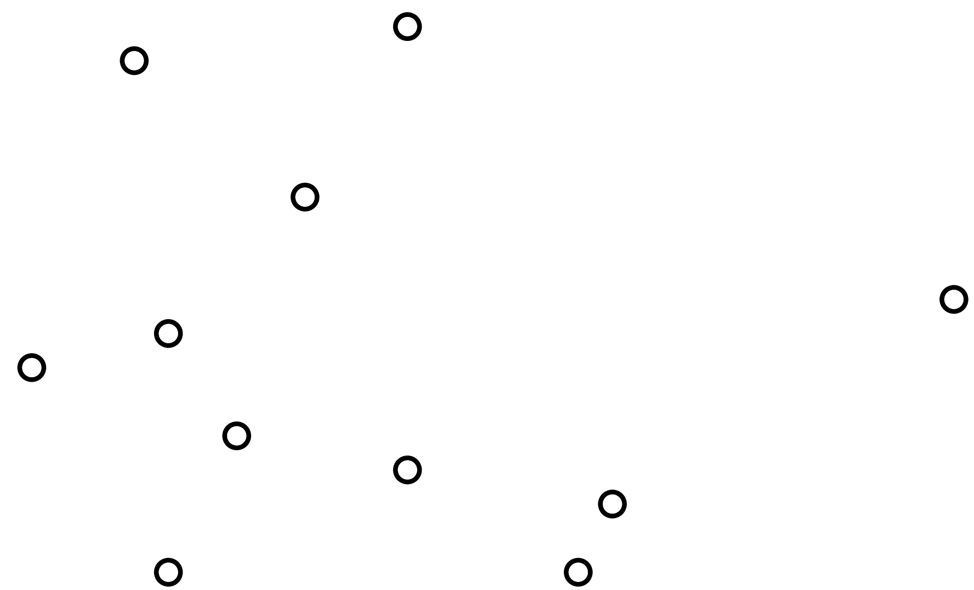
Basic task:

Create connections between n locations with minimal cost.

Definition:

A Euclidean minimum spanning tree of a set S of points is a tree that connects all points and minimizes the total edge length among all trees on S.

Example:



Problem Definition

Basic task:

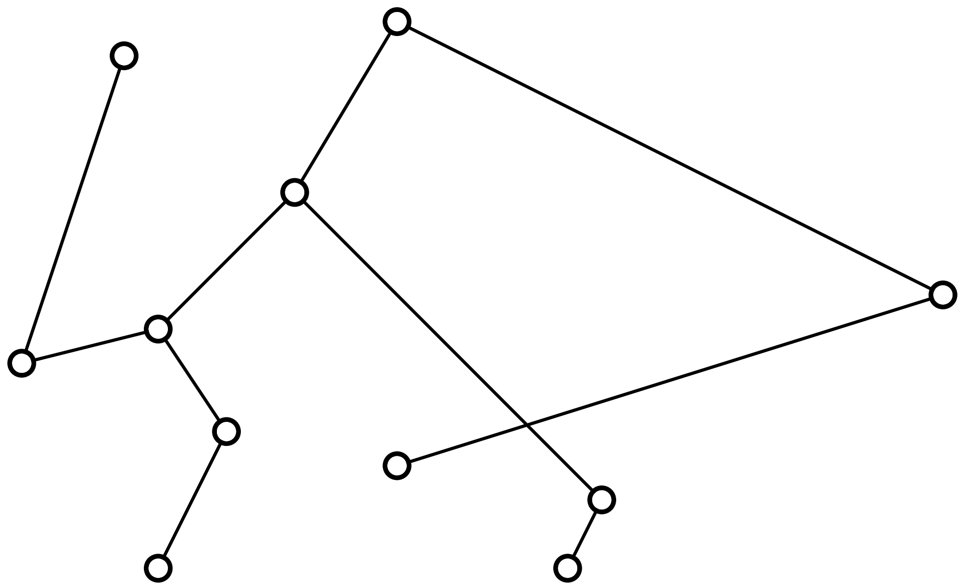
Create connections between n locations with minimal cost.

Definition:

A Euclidean minimum spanning tree of a set S of points is a tree that connects all points and minimizes the total edge length among all trees on S.

Example:

spanning tree



Problem Definition

Basic task:

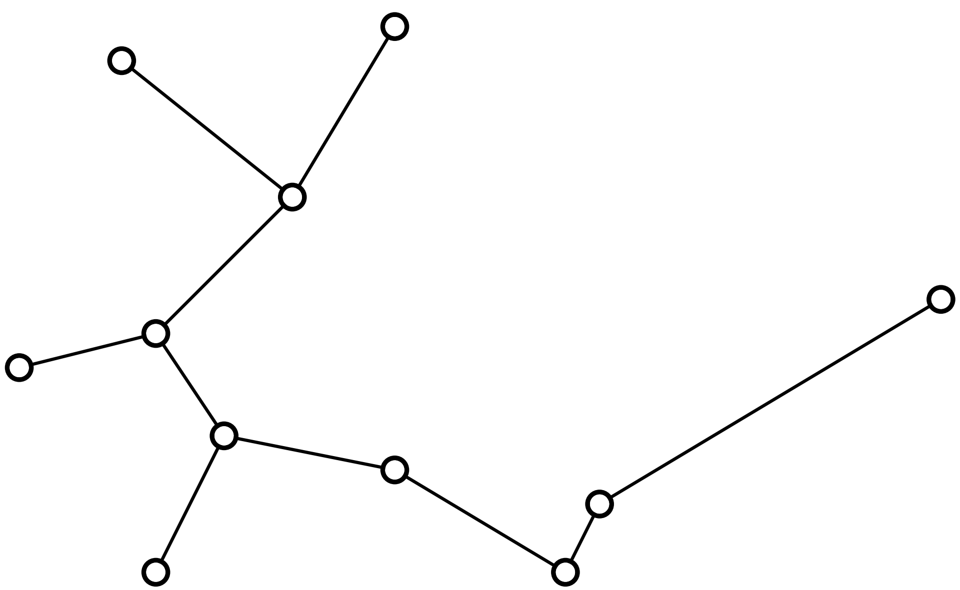
Create connections between n locations with minimal cost.

Definition:

A Euclidean minimum spanning tree of a set S of points is a tree that connects all points and minimizes the total edge length among all trees on S.

Example:

minimum spanning tree



Observation:

every Euclidean minimum spanning tree is crossing-free

Problem Definition

Basic task:

Create connections between n locations with minimal cost.

Definition:

A Euclidean minimum spanning tree of a set S of points is a tree that connects all points and minimizes the total edge length among all trees on S.

Example applications:

bicycle path network, electrical circuits, telephone network.

Possible problem:

Direct connection from A to B not always possible or not proportional to the distance (example: mountain road).

$\Rightarrow$ Consider weighted graphs instead.

Problem Definition

Basic task:

Create connections between n locations with minimal cost.

Definition:

A minimum spanning tree of a weighted graph $G = (V, E, w)$ is a tree $T = (V, E')$ with $E' \subseteq E$ and with minimal total edge length among all spanning trees in G:

$$w(T) = \sum_{e \in E'} w(e)$$

is minimized over all trees in G with vertex set V.

Problem Definition

Basic task:

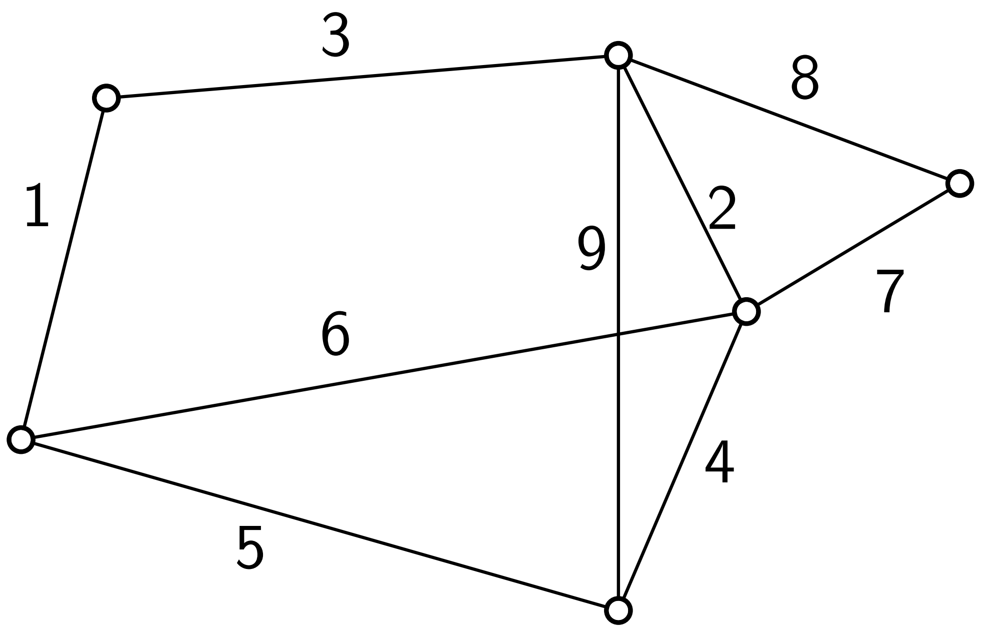
Create connections between n locations with minimal cost.

Definition:

A minimum spanning tree of a weighted graph $G = (V, E, w)$ is a tree $T = (V, E')$ with $E' \subseteq E$ and with minimal total edge length among all spanning trees in G:

Example:

weighted graph G



Problem Definition

Basic task:

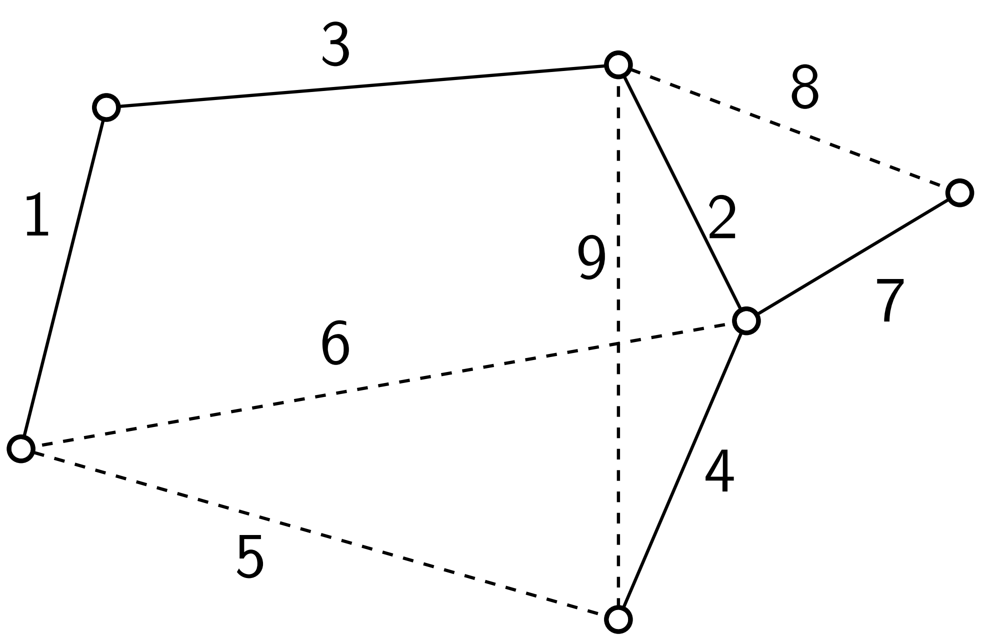
Create connections between n locations with minimal cost.

Definition:

A minimum spanning tree of a weighted graph $G = (V, E, w)$ is a tree $T = (V, E')$ with $E' \subseteq E$ and with minimal total edge length among all spanning trees in G:

Example:

minimum spanning tree of G



Problem Definition

Basic task:

Create connections between n locations with minimal cost.

Definition:

A minimum spanning tree of a weighted graph $G = (V, E, w)$ is a tree $T = (V, E')$ with $E' \subseteq E$ and with minimal total edge length among all spanning trees in G:

$$w(T) = \sum_{e \in E'} w(e)$$

is minimized over all trees in G with vertex set V.

Connections between n locations with minimal total cost:

Consider the complete weighted graph whose vertices are the locations.

How Many Different Trees?

Questions:

- How many different spanning trees for a graph G?
- How many different plane spanning trees for n points?

Answers:

- Complete graph K_n: n^{n-2} different spanning trees (!!)
- Plane graphs on n vertices: $O(5.2852^n)$ spanning trees
- Number of different plane spanning trees on n points: depends on point set; bounds: $\Omega(6.75^n)$, $O(229.33^n)$
- Point sets known with $\Omega(12.52^n)$ plane spanning trees

$\Rightarrow$ Trying them all is infeasible.

Iterative Algorithm Idea

Idea:

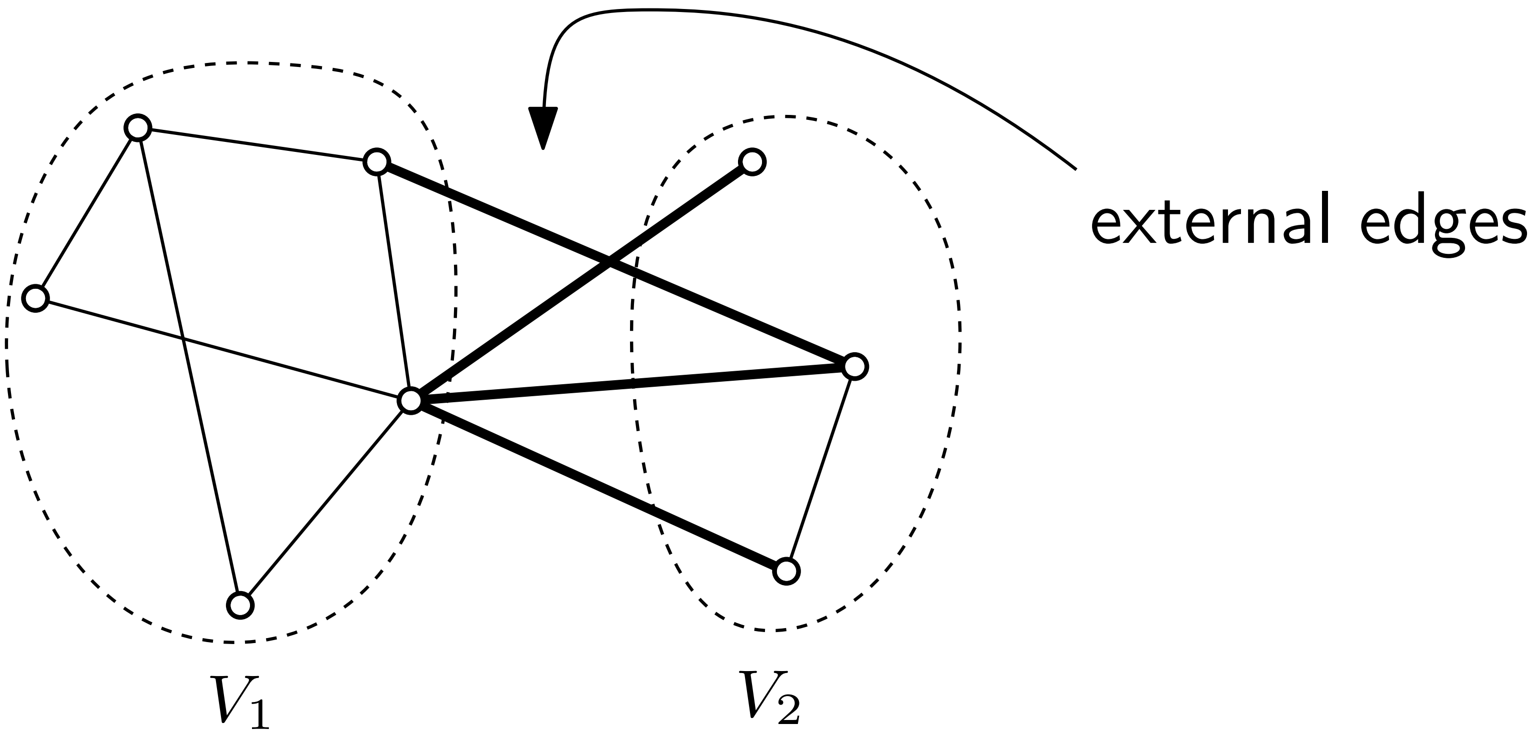
Build a minimum spanning tree (MST) for a graph $G = (V, E, w)$ by iteratively inserting edges:

$E' = 0$
<u>while</u> $|E'| < n - 1$ **<u>do</u>**
  select an edge $e \in E \setminus E'$ which is **'good'** for E'
  $E' = E' \cup e$
<u>od</u>
write E'

Edge $e \in E \setminus E'$ is **'good'** for E' if $E' \cup e$ is a subset of an MST of G (there can be more than one MST of G).

Cuts in Graphs

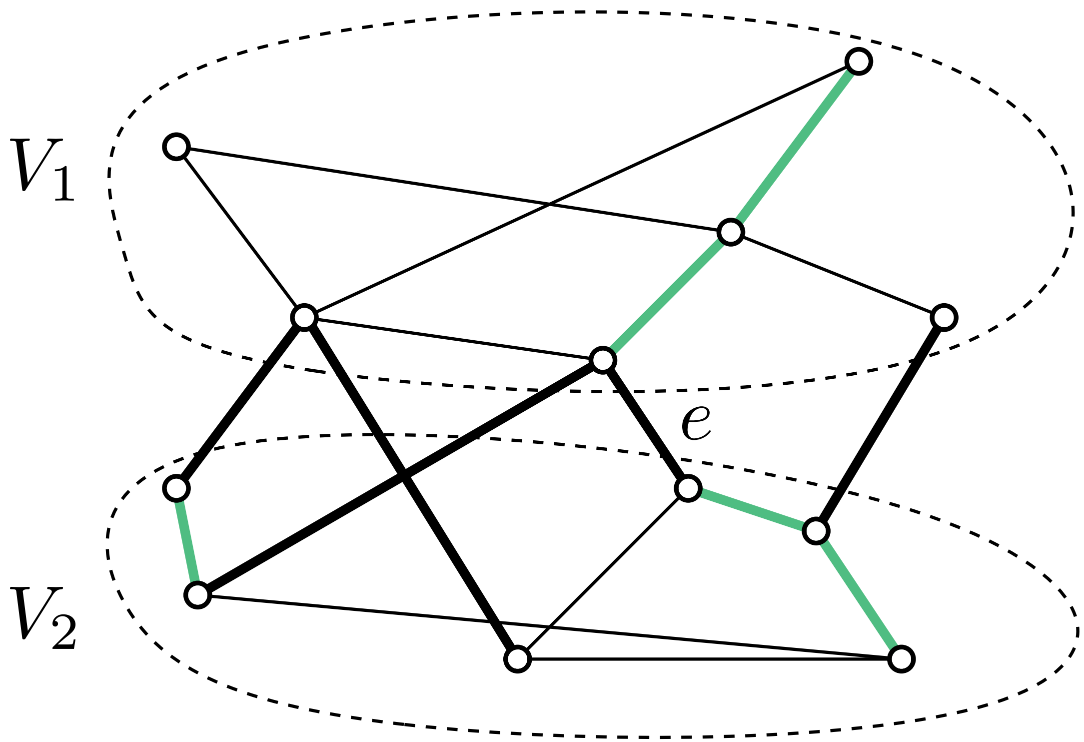
A **cut** of a graph $G = (V, E)$ is a partition of V into V_1, V_2.



An edge e is called **external** for the cut (V_1, V_2) if it has one endpoint in V_1 and one in V_2; otherwise e is called **internal**.

Characterization of Good Edges

Theorem: Let E' be a subset of edges of an MST of $G = (V, E, w)$. Let (V_1, V_2) be a cut of G for which all edges of E' are internal. Then the external edge of the cut with **minimum weight** is a good edge for E'.



Example:

$w =$ Eucl. distance

set E': green

external edges: black

good edge for E': e

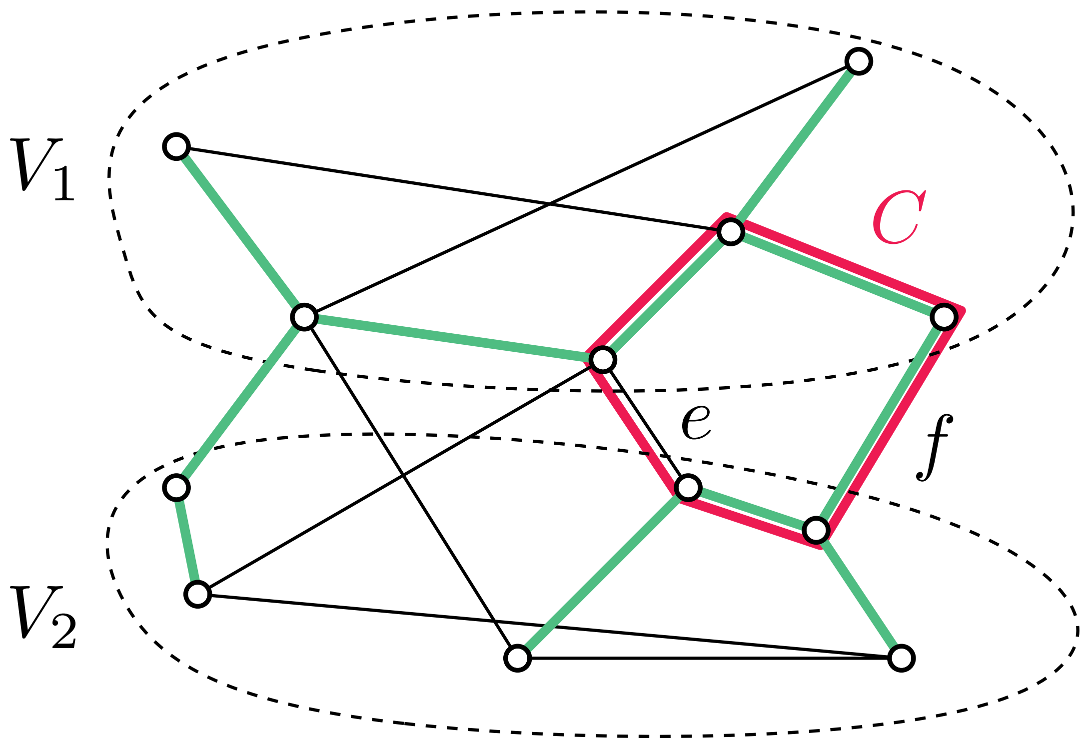
Characterization of Good Edges

Proof:

Assume there is an MST T with E' and without e.

$\Rightarrow$ e closes a cycle C in T.

The cycle C contains at least one edge f of the tree T that is external for the cut.



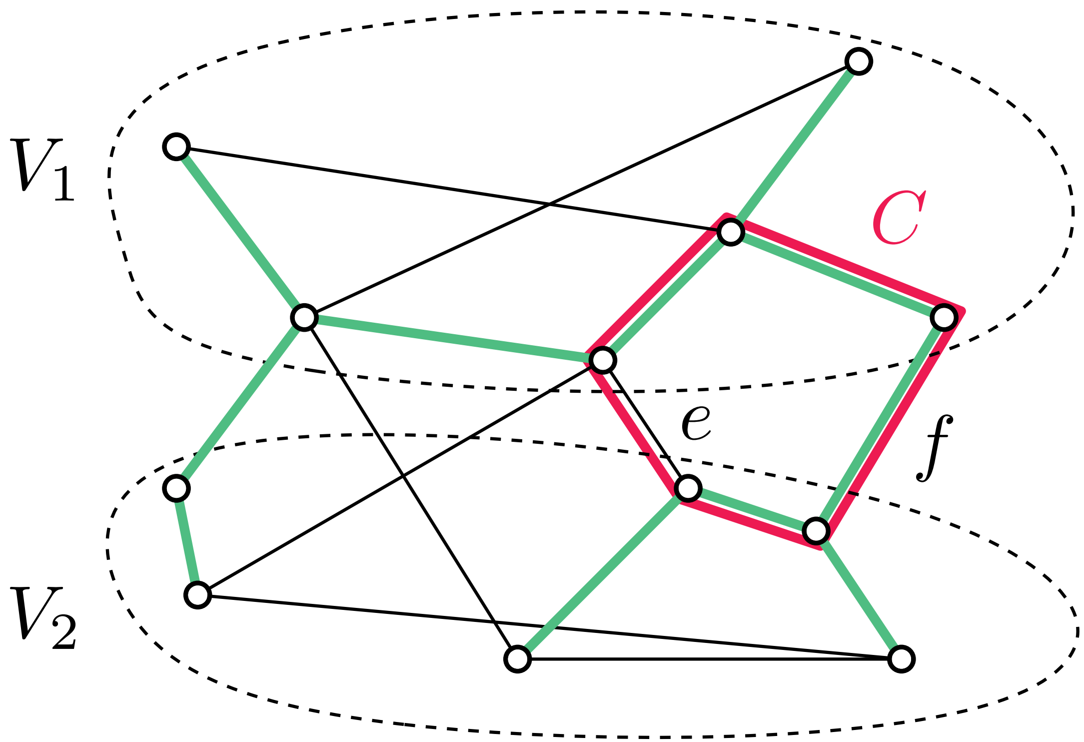
By definition of e, the weight $w(f) \geq w(e)$.

$\Rightarrow$ $T \backslash \{f\} \cup \{e\}$ is an MST of G. $\square$

If $w(f) > w(e)$ then T is not an MST of G.

Characterization of Good Edges

Theorem: Let E' be a subset of edges of an MST of $G = (V, E, w)$. Let (V_1, V_2) be a cut of G for which all edges of E' are internal. Then the external edge of the cut with **minimum weight** is a good edge for E'. $\square$



Next:
Two different greedy algorithms that use this theorem to efficiently compute an MST

Difference:
Choice of the cuts

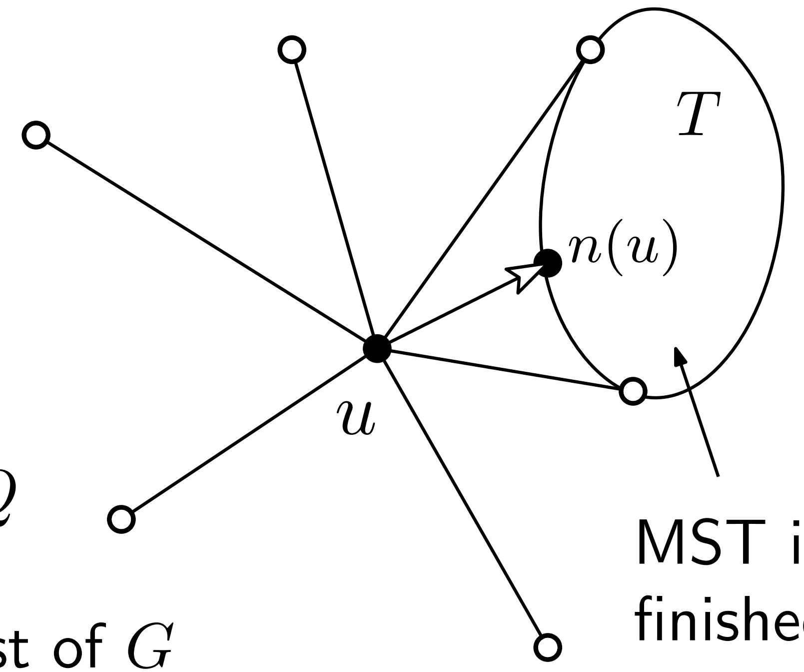
Prim's Algorithm

- Start with an arbitrary vertex s of G and iteratively 'grow' an MST T from s.
- **Iterative step:**
  Choose the 'cheapest' edge with exactly one node in T.
- Cut: $V_1 =$ vertices of T, $V_2 =$ vertices not yet in T.
- For each vertex $v \notin T$ we maintain:
  - Priority $p(v)$: weight of the shortest edge from v to a vertex in T (initially: ∞).
  - Nearest $n(v)$: vertex in T realizing $p(v)$: $w(v, n(v))$ is min. among neighbors of v in T (initially no vertex).
- A **queue** Q contains all vertices not yet in T, organized by priorities (e.g., in a min-heap; initially all vertices).

Prim's Algorithm

PRIM-MST (G,s)

for all $v \in V$ **do** $p(v) = \infty$ **od**
$p(s) = 0, \;\; n(s) =$ nil
$Q = V$ // build up Q
while $Q \neq 0$ **do**
  $u =$ MIN(Q)
  remove u from Q // reorganize Q
  write $u, n(u)$
  for all $v \in A(u)$ **do** // A: adj. list of G
    if $v \in Q$ and $p(v) > w(u,v)$ **then**
      $p(v) = w(u,v)$ // reorganize Q
      $n(v) = u$
    fi
  od
od



MST is finished here

Prim's Algorithm

Run-time-analysis:

n ... number of vertices of G
m ... number of edges of G
$d(v)$ ... Degree of vertex v in G

- Initialization, construction of the heap Q: $\Theta(n)$
- n times removing the minimum from Q: $O(n \log n)$
- report MST edges: $\Theta(n)$
- Update priorities for all neighbors of v: $O(d(v) \cdot \log n)$

$\Rightarrow$ Alltogether:

$$O(n + n \log n + \textstyle\sum_{v \in V} d(v) \log n) = O(m \log n)$$

Memory requirements: $\Theta(m + n) = \Theta(m)$
Graph + queue + priorities + nearests + constant additional

Prim's Algorithm

Remarks:

- MST always begins at the start vertex s and grows from there as a connected tree.
- Shrinking $p(v)$ causes v to move up in the heap.
  $\Rightarrow O(\log n)$ time.
- Test for $v \in Q$ in $O(1)$ time when bit vector is used to store which vertices are already in T.
- Runtime can be changed to $O(n^2)$. This is useful for dense graphs (see notes on Dijkstra's algorithm in the next chapter).

Kruskal's Algorithm

- Start with empty edge set E'.
- Sort edge set E of $G = (V, E, w)$ in increasing order of their weights (edges will be considered in this order): $e_1, e_2, ...e_m$ with $w(e_1) < w(e_2) < ... < w(e_m)$.
- **Iterative step**:
  - E' forms a forest F (= set of disjoint subtrees, acyclic) in G and in the MST to be constructed.
  - Edge e that is added to E' is the shortest edge in $E \setminus E'$ that does not form a cycle with edges from E'.

Kruskal's Algorithm

Use a **UNION-FIND** data structure on V for the components (subtrees) $M_1, M_2, ...M_t$ of F:

- Label the vertex set of G as $v_1, v_2, ...v_n$ (arbitrary)
- Initially there are n disjoint sets $M_1, M_2, ...M_n$ (each with one vertex)
- FIND(v): returns index i if vertex v is in M_i
- UNION(i,j): join sets M_i and M_j: $M_i = M_i \cup M_j$ (index of resulting set: minimum of i and j)
- End of the algorithm: one component M_1 with all vertices of G.

Kruskal's Algorithm

There are many different **UNION-FIND** data structures, with different runtime- and memory requirements.

Here we use one with the following properties:

- Creating a 1-element set needs $\Theta(1)$ time.
- f FIND and u UNION operations need $O(f + u \log u)$ time in total.
- The total memory requirement of the data structure is linear in the number of initial 1-element sets.

Kruskal's Algorithm

KRUSKAL-MST(G)

sort edges by weight: $\{e_1, e_2, \ldots e_m\}$

for $i = 1$ **to** n **do** $M_i = \{v_i\}$ **od**

for $k = 1$ **to** m **do**

    $(u, v) = e_k$

    $i =$FIND(u)

    $j =$FIND(v)

    if $i \neq j$ **then**

        write e_k

        UNION(i, j)

    fi

od

Kruskal's Algorithm

Runtime analysis:

- Sorting of the edges: $O(m \log m)$
- Initialize UNION-FIND data structure for vertices: $\Theta(n)$
- In total $2m$ FIND operations and $n-1$ UNION operations: $O(m + n \log n)$
- Extract edges + write MST edges: $\Theta(m)$

$\Rightarrow$ Altogether $O(m \log m)$ time.

$\Rightarrow$ Sorting of the edges dominates the runtime.

Memory requirements:

$\Theta(n+m) = \Theta(m)$ in total.

Concluding Remarks

- Both algorithms also work if edge weights can be negative.
- If the calculation of the MST for a point set in a plane with Euclidean distance function is to be carried out (geometric version), this is possible in $O(n \log n)$ time. The main observation is that the MST is a subgraph of the Delaunay triangulation of the point set, which can be computed in $O(n \log n)$ time.
- For both algorithms, animated versions are available (see course webpage).

Thank you for your attention.