COMP251: Minimum Spanning Trees

Jérôme Waldispühl
School of Computer Science
McGill University

Based on (Cormen et al., 2002)

Based on slides from D. Plaisted (UNC)
Minimum Spanning Tree (Example)

- A town has a set of houses and a set of roads.
- A road connects 2 and only 2 houses.
- A road connecting houses $u$ and $v$ has a repair cost $w(u, v)$.

**Goal:** Repair enough (and no more) roads such that:

1. everyone stays connected: can reach every house from all other houses, and
2. total repair cost is minimum.
• Undirected graph $G = (V, E)$.
• **Weight** $w(u, v)$ on each edge $(u, v) \in E$.
• Find $T \subseteq E$ such that:
  1. $T$ connects all vertices ($T$ is a *spanning tree*),
  2. $w(T) = \sum_{(u,v)\in T} w(u,v)$ is minimized.
Minimum Spanning Tree (MST)

- It has $|V| - 1$ edges.
- It has no cycles.
- It might not be unique.
Generic Algorithm

- Initially, A has no edges.
- Add edges to A and maintain the **loop invariant**: “A is a subset of some MST”.

```plaintext
A ← ∅;
while A is not a spanning tree do
    find a edge (u, v) that is safe for A;
    A ← A ∪ {(u, v)}
return A
```

- **Initialization**: The empty set trivially satisfies the loop invariant.
- **Maintenance**: We add only safe edges, A remains a subset of some MST.
- **Termination**: All edges added to A are in an MST, so when we stop, A is a spanning tree that is also an MST.
A cut respects $A$ if and only if no edge in $A$ crosses the cut.

A light edge crossing cut (may not be unique)

This edge crosses the cut. (one endpoint is in $S$ and the other is in $V - S$.)
What is a safe edge?

Intuitively: Is (c,f) safe when A=∅?

• Let S be any set of vertices including c but not f.
• There has to be one edge (at least) that connects S with V − S.
• Why not choosing the one with the minimum weight?
Safe edge

**Theorem 1:** Let \((S, V-S)\) be any cut that respects \(A\), and let \((u, v)\) be a light edge crossing \((S, V-S)\). Then, \((u, v)\) is safe for \(A\).

**Proof:**
Let \(T\) be a MST that includes \(A\).

**Case 1:** \((u, v)\) in \(T\). We’re done.

**Case 2:** \((u, v)\) not in \(T\). We have the following:

We show edges in \(T\)

- \((x, y)\) crosses the cut.
- Let \(T' = T - \{(x, y)\} \cup \{(u, v)\}\).
- Because \((u, v)\) is light for cut, \(w(u, v) \leq w(x, y)\). Thus,
- \(w(T') = w(T) - w(x, y) + w(u, v) \leq w(T)\). Hence, \(T'\) is also a MST.
- So, \((u, v)\) is safe for \(A\).
Corollary

In general, A will consist of several connected components.

**Corollary:** If \((u, v)\) is a light edge connecting one CC in \((V, A)\) to another CC in \((V, A)\), then \((u, v)\) is safe for A.
Kruskal’s Algorithm

1. Starts with each vertex in its own component.
2. Repeatedly merges two components into one by choosing a light edge that connects them (i.e., a light edge crossing the cut between them).
3. Scans the set of edges in monotonically increasing order by weight.
4. Uses a disjoint-set data structure to determine whether an edge connects vertices in different components.
Example
Example

```
Example

```

```
Example

```

```
Example

```

```
Example

```

```
Example

```

```
Example

```

```
Example

```

```
Example

```

```
Example

```

```
Example

```

```
Example

```
Example

Reject!
Example
Example
Example
Example

Graph with nodes labeled a, b, c, d, e, f, g, h, and i. Edges connect the nodes with weights indicated. For example, the edge between nodes a and c has a weight of 9, and the edge between nodes d and g has a weight of 8. The graph includes a dashed outlined region and a blue vertical line segment.
Example
Kruskal’s complexity

- Initialize $A$: $O(1)$
- First for loop: $|V|$ MAKE-SETs
- Sort $E$: $O(E \lg E)$
- Second for loop: $O(E)$ FIND-SETs and UNIONs

Assuming union by rank and path compression:
$O((V + E)\alpha(V)) + O(E \lg E)$

- Since $G$ is connected, $|E| \geq |V| - 1 \Rightarrow O(E \alpha(V)) + O(E \lg E)$.
- $\alpha(|V|) = O(\lg V) = O(\lg E)$.
- Therefore, total time is $O(E \lg E)$.

- $|E| \leq |V|^2 \Rightarrow \lg |E| = O(2\lg V) = O(\lg V) \Rightarrow O(E \lg V)$ time
1. Builds **one tree**, so $A$ is always a tree.
2. Starts from an arbitrary “root” $r$.
3. At each step, **adds a light edge** crossing cut $(V_A, V - V_A)$ to $A$.
   – Where $V_A =$ vertices that $A$ is incident on.
Intuition behind Prim’s Algorithm

• Consider the set of vertices $S$ currently part of the tree, and its complement ($V - S$). We have a cut of the graph and the current set of tree edges $A$ is respected by this cut.

• Which edge should we add next? *Light edge!* 
Finding a light edge

1. Uses a **priority queue** `Q` to find a light edge quickly.
2. Each object in `Q` is a vertex in `V - V_A`.
3. Key of `v` has minimum weight of any edge `(u, v)`, where `u ∈ V_A`.
4. Then the vertex returned by `Extract-Min` is `v` such that there exists `u ∈ V_A` and `(u, v)` is light edge crossing `(V_A, V - V_A)`.
5. Key of `v` is `∞` if `v` is not adjacent to any vertex in `V_A`. 
Basics of Prim ’s Algorithm

• It works by adding leaves on at a time to the current tree.
  – Start with the root vertex $r$ (it can be any vertex). At any time, the subset of edges $A$ forms a single tree. $S = \text{vertices of } A$.
  – At each step, a light edge connecting a vertex in $S$ to a vertex in $V - S$ is added to the tree.
  – The tree grows until it spans all the vertices in $V$.

• Implementation Issues:
  – How to update the cut efficiently?
  – How to determine the light edge quickly?
Implementation: Priority Queue

• Priority queue implemented using heap can support the following operations in $O(lg\ n)$ time:
  – Insert $(Q, u, key)$: Insert $u$ with the key value $key$ in $Q$
  – $u = Extract\_Min(Q)$: Extract the item with minimum key value in $Q$
  – Decrease\_Key$(Q, u, new\_key)$: Decrease the value of $u$’s key value to $new\_key$

• All the vertices that are not in $S$ (the vertices of the edges in $A$) reside in a priority queue $Q$ based on a $key$ field. When the algorithm terminates, $Q$ is empty. $A = \{(v, \pi[v]): v \in V - \{r\}\}$
Prim’s Algorithm

\[ Q := V[G]; \]
\[ \text{for each } u \in Q \text{ do} \]
\[ \text{key}[u] := \infty \]
\[ \pi[u] := \text{Nil}; \]
\[ \text{Insert}(Q,u) \]
\[ \text{Decrease-Key}(Q,r,0); \]
\[ \text{while } Q \neq \emptyset \text{ do} \]
\[ u := \text{Extract-Min}(Q); \]
\[ \text{for each } v \in \text{Adj}[u] \text{ do} \]
\[ \text{if } v \in Q \wedge w(u,v) < \text{key}[v] : \]
\[ \pi[v] := u; \]
\[ \text{Decrease-Key}(Q,v,w(u,v)); \]

**Complexity:**

Using binary heaps: \( O(E \lg V) \).
- Initialization: \( O(V) \).
- Building initial queue: \( O(V) \).
- \( V \) Extract-Min: \( O(V \lg V) \).
- \( E \) Decrease-Key: \( O(E \lg V) \).

Using Fibonacci heaps:
\( O(E + V \lg V) \).

**Notes:**
(i) \( A = \{(v, \pi[v]) : v \in V - \{r\} - Q\} \).
(ii) \( r \) is the root.
Example of Prim’s Algorithm

\[ Q = a \ b \ c \ d \ e \ f \]

\[ 0 \ \infty \ \infty \ \infty \ \infty \ \infty \]

Not in tree
Example of Prim’s Algorithm

\[ Q = b \ d \ c \ e \ f \]

\[
\begin{array}{c}
5 & 11 & \infty & \infty & \infty \\
\end{array}
\]
Example of Prim’s Algorithm
Example of Prim’s Algorithm

Q = d c f
0 1 2
Example of Prim’s Algorithm

Q = c f
1 2
Example of Prim’s Algorithm

\[ Q = f -3 \]
Example of Prim’s Algorithm

Q = ∅
Example of Prim’s Algorithm

```
 a/0  5  b/5  
    3  1     
 d/0  0  e/3  c/1
    3     -3
 f/-3
```
Correctness of Prim

• Again, show that every edge added is a safe edge for $A$
• Assume $(u, v)$ is next edge to be added to $A$.
• Consider the cut $(A, V - A)$.
  – This cut respects $A$
  – and $(u, v)$ is the light edge across the cut
• Thus, by the Theorem 1, $(u,v)$ is safe.