Algorithms and Data Structures

Graphs 4: Minimal Spanning Trees

Marius Kloft
Die Energiewende

- Electricity is created in many more places than before
- Electricity is consumed in many places
- Places of production are not evenly distributed across the country
- Many say we need to build new electricity highways

Source: http://www.deutsche-mittelgebirge.de/
Die Energiewende

- How can we do this as cheap as possible?
- Not all connections are possible
  - Mountains, rivers, ...
- Different connections have different costs
Die Energiewende

- Requirement for a solution: Every city and every plant must be connected to the network
Abstraction

- Given an undirected, positively weighted, connected \( G=(V,E) \)
- Find a subset \( E' \subseteq E \) such that cost(\( E' \)) is minimal and \( G'=(V, E') \) is connected
  - cost(\( E' \)): Sum of the edge weights
- \( E' \) (or \( G' \)) is called a minimum spanning tree (MST) for \( G \)
Example 1

• Cost = 62
Example 2

- Cost = 61
First Algorithm

- Let’s try greedy
  - Sort edges by weight
  - Add edges to $E'$ whenever it connects a new node to something

- Hmm
Second Algorithm

- Let’s try greedy – another way
  - Sort edges by weight
  - Add cheapest edge to $E'$
  - Add edges to $E'$ in ascending order such that every new edge connects a new node with the graph induced by $E'$
  - Repeat until all nodes are connected

- Cost = 42
  - Is this optimal?
  - Does this always work?
  - How can we implement this algorithm efficiently?
Overview

- First algorithms for computing MST date back to the 1920s
- Algorithms are not very difficult; much research went into efficient implementations
- Actually, MSTs can be computed in a greedy manner
- Algorithms need not grow only one component; in general, we may have “connected islands” that all get connected to one component in the end
- In each step, one needs to decide which edge to add next to which island (or which edges not to add)
- What are criteria for adding / not adding edges?
Content of this Lecture

• Minimal Spanning Trees
• Basic Properties
  – Tree
  – Cuts
  – Cycles
• Algorithms
• Implementation
Minimal Spanning Tree

- **Lemma**
  
  $G = (V, E)$ and let $E' \subseteq E$ be the subset of $E$ with minimal cost such that $G'$, the graph induced by $E'$, is connected. Then $G'$ is a tree (called “minimal spanning tree”, MST).

- **Proof**
  
  - Recall: A (undirected) tree is a undirected, connected acyclic graph
  - By definition, $G'$ is connected and undirected
  - Need to show that $G'$ contains no cycle
Proof: MST is a Tree

- Imagine $G'$ had a cycle. Then $G'$ cannot have minimal cost
  - because removing any of the edges of the cycle from $E'$ would create a subset $E''$ that has less cost (since we assumed all edge weights to be positive), and the induced subgraph would still be connected
- Contradiction

- Remark: If all edge weights are distinct, the MST is unique
Cuts & Crossing Bridges

- Definition
  Let $G=(V, E)$. A cut is a binary partition of $V$ into sets $V_1$, $V_2$ such that $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$. 
Cuts & Crossing Bridges

- Definition
  Let $G=(V, E)$. A cut is a binary partition of $V$ into sets $V_1$, $V_2$ such that $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$.

- Definition
  Let $G=(V, E)$ and $V_1$, $V_2$ be a cut of $V$. Any edge connecting a node in $V_1$ to a node in $V_2$ is called crossing bridge. We denote the set of all crossing bridges by $F$. 
Cut Property on Minimal Crossing Bridges

• Lemma (Cut Property)
  Let $G=(V, E)$, let $V_1, V_2$ be a cut of $V$ with crossing bridges $F$. Let $F'$ be those edges of $F$ with minimal weight. Then:
  1) Any MST $G'$ of $G$ must contain at least one $f' \in F'$
  2) Every $f' \in F'$ is contained in at least one MST of $G$

• Remarks
  – This holds for arbitrary cuts – a very powerful statement
Proof, 1a)

1) Every MST $G'$ contains at least one $f' \in F'$
   - Assume the contrary ($G'$ has no such $f'$) and let $f' \in F'$
   - Still, $G'$ is connected, so it must contain at least one of the crossing edges from $F$
     (a) Assume $G'$ contains only one $f \in F$
       • $f$ must have a higher weight than $f'$ because – by assumption – $f \notin F'$
       • Furthermore, because – by assumption – $f$ is the only crossing edge, $V_1$ and $V_2$ must be connected in themselves
       • Thus, removing $f$ and adding some $f' \in F'$ creates a cheaper MST, so $G'$ cannot be minimal – contradiction.
Proof, 1b)

1) Every MST $G'$ contains at least one $f' \in F'$

(b) The proof is similar if $G'$ contains multiple $f_i \in F$

- Write $f' = (v, v')$
- Since $G'$ is connected there exists a path $p$ in $G'$ from $v$ to $v'$
- Since $f'$ is a crossing bridge, $v$ and $v'$ must lie on opposite sides of the cut
  - So the path $p$ contains a crossing bridge $f_i \in F$
- Removing $f_i$ from MST $G'$ breaks $G'$ into two components, and adding $f'$ re-connects them
  - resulting in cheaper MST (since $f'$ has smaller weight than $f_i$ because $f_i \notin F'$)
  - Contradiction
Proof, 2)

(2) Every $f' \in F'$ is contained in at least one MST of $G$

- Imagine $f'$ is not contained in any MST
- Let $G'$ be such an MST
- Proof uses analogue argument as in (1):
  - Consider $f \in F$ connecting $V_1$ and $V_2$
  - Removing $f_i$ from $G'$ breaks $G'$ into two components, and adding $f'$ re-connects them, resulting in $G''$ with equal or cheaper cost as $G'$
  - Thus $G''$ is an MST - Contradiction
Beware

- For a cut $V_1, V_2$, an MST $G'$ may (have to) contain more than one crossing edge (but one must have minimal weight)
Content of this Lecture

- Minimal Spanning Trees
- Basic Properties
  - Tree
  - Cuts
  - Cycles
- Algorithms
- Implementation
Cycles

• Lemma (cycle property)
  Let \( G = (V, E) \) and \( G' = (V, E') \) with \( E' = E \setminus e \) for some edge \( e \) such that \( G' \) still is connected. Let \( T' \) be an MST for \( G' \). When we add \( e \) to \( T' \) and remove the edge with the highest weight on the then introduced cycle in \( T' \), forming \( T \), then \( T \) is an MST for \( G \).

• Proof idea
  – Adding \( e \) must build a cycle because \( T' \) is MST over the same \( V \)
  – Removing any of the edges on the cycle still leaves a connected tree
  – Removing the most expensive one leaves the minimal tree
Cycle Property

Remove e

MST of G'

Add e

MST for G

Remove highest weight on cycle
Implications

• Note that $T'$ is an MST for $G$ without $e$
• Imagine we would enumerate edges by some order
• Taking into account a new $e$ allows us to replace an edge in $T'$ with a cheaper one, creating a “better” MST for $G$
  – If $e$ is not the edge with the highest weight on the cycle
• This means that an edge with maximal weight on a cycle in $G$ cannot be part of any MST of $G$
Content of this Lecture

- Minimal Spanning Trees
- Basic Properties
- **Algorithms**
    - Also Jarnik, Prim, Dijkstra: Jarník, 1930 – Prim, 1957 – Dijkstra, 1959
  - **Otakar Borůvka**: O jistém problému minimálním (Über ein gewisses Minimierungsproblem), 1926
  - [Wikipedia, OW93, Sed04, Cor03]
- Implementation
• Prim’s Algorithm

Start with an empty tree $T$. Continue adding the edge $e$ with the **lowest cost to $T$** such that $e$ connects $T$ with a new node until all nodes of $G$ are in $T$. Then $T$ is an MST

• Proof

  – Consider, at each stage, nodes in $T$ as one partition $V_1$ and all other nodes as the other partition $V_2$
  – By cut-property lemma, the cheapest crossing-edge between $V_1$ and $V_2$ must be in an MST of $G$
  – Since we only add those edges, $T$ finally must be an MST

Greedy; we never make mistakes
Prim's Algorithm: Example
Prim’s Algorithm: Example
Kruskal’s Algorithm

- **Start with an empty forest F.** Continue “adding” edges e to F in order of increasing cost until F becomes a tree. Adding an edge e=(v, w) to F proceeds as follows:
  - **Case 1:** If F already contains a tree containing both v and w, then e is dropped.
  - **Case 2:** If no tree in F contains either v or w, then a new tree formed by e is added to F.
  - **Case 3:** If F contains a tree T containing either v or w and neither T nor any other tree in F contains the other node, then e is added to T.
  - **Case 4:** If F contains a tree T containing either v or w and a tree T’ containing the other node, then T, T’ and e are merged into one tree.
Kruskal’s Algorithm: Example
Proof by Induction (Only Central Idea)

- We show that each of the trees in F is an MST of a subgraph of G
- Claim is true at the beginning (F empty)
- Assume claim holds before we consider next edge e=(v, w)
- Case 1: Claim holds, because e would introduce a cycle, and e has the highest cost on this cycle (all cheaper edges were considered before). Thus, e cannot be in an MST of G
- Case 2: Claim holds because e is the cheapest edge connecting v and w, and thus the new tree is an MST (for subgraph induced by {v,w})
- Case 3: Claim holds because e is the cheapest edge connecting v (or w) and T, and thus the new tree is an MST
- Case 4: Claim holds because e is the cheapest edge connecting T and T’, and thus the new tree is an MST
Boruvka’s Algorithm

- Boruvka’s Algorithm
  
  *Start with an empty forest F. Add all edges (at once) that connect a node with its “cheapest” neighbor (edge with least cost) – taking care of not introducing cycles. Then consider each pair of trees in F and add cheapest crossing-edge until F becomes a unique tree.*

- Proof (and details) omitted; see [Sed04]
Boruvka’s Algorithm: Example
Communalities

- All three algorithms iteratively choose an edge by the cut property or reject an edge by the cycle property
  - Prim: Growing T is one partition, all other nodes the other (isolated nodes)
  - Kruskal: Each T that grows is one partition, all other nodes the other (islands of mini-MSTs)
  - Boruvka: Each T that grows is one partition, all other nodes the other (islands of mini-MSTs)

- Difference is the order in which edges are chosen – there are always many candidates

- Differences are the data structures that these algorithms need to maintain
Content of this Lecture

- Minimal Spanning Trees
- Basic Properties
- Algorithms
- Implementation
  - Prim’s, Kruskal’s
Implementing Prim’s Algorithm

- **ChooseCheapest**: Choose cheapest edge connecting a node in T to a node not yet in T
- **Brute force**: Search all such edges in every step
- **More clever**
  - Maintain a PQ of nodes reachable by one edge from T sorted by cost
  - When adding a new node to T, look at its neighbors and add them to the PQ (if not reachable before) or update costs (if now there is a cheaper edge reaching them)

```
G := (V, E);
T := ∅;
R := E;
for i = 1 to |V|-1 do
  e := chooseCheapest( T, R);
  T := T ∪ e;
  R := R \ e;
end for;
```
Example

- \( T = \{A, F, E, B, G\} \)
- \( PQ = \{(D, 6), (I, 6), (C, 7)\} \)
- Choose (A-D, 6)
Example

- $T = \{A, F, E, B, G\}$
- $PQ = \{(D,6), (I, 6), (C, 7)\}$
- Choose (A-D, 6)
- New $T$: \{A, F, E, B, G, D\}
- $PQ = \{(C,4), (I, 6), (H, 18)\}$
Complexity

- $n = |V|$, $m = |E|$

- Prim’ algorithm runs in $O((n+m) \cdot \log(n))$
  - $n$ times through the loop, performing altogether at most $m$ PQ-operations in $\log(n)$
Implementing Kruskal’s Algorithm

- **ChooseCheapest**: Simply choose cheapest edge in $E$
  - I.e., sort $E$ at the beginning
- **UNION-FIND** data structure
  - Maintains a set of sets (all trees $T$)
  - Needs a method for quickly finding the set containing a given element (find)
  - Needs a method for quickly merging two sets (union)
- Can be implemented in $O(m\times\log(n))$