Minimum Spanning Trees

Kruskal’s Algorithm
Prim’s Algorithm
Comparison
Other Algorithms
Appendix

COMP2521 24T1
Graphs (VII)
Minimum Spanning Trees

Kevin Luxa
cs2521@cse.unsw.edu.au

minimum spanning trees
kruskal’s algorithm
prim’s algorithm
A spanning tree of an undirected graph $G$ is a subgraph of $G$ that contains all vertices of $G$, that is connected and contains no cycles.

A minimum spanning tree of an undirected weighted graph $G$ is a spanning tree of $G$ that has minimum total edge weight among all spanning trees of $G$.

Applications:
- Electrical grids, networks
- Any situation where we want to connect nodes as cheaply as possible
Minimum Spanning Trees

Example

Original graph

Spanning tree

Minimum spanning tree
Basic minimum spanning tree algorithms:

- Kruskal’s algorithm
- Prim’s algorithm
Invented by
American mathematician, statistician, computer scientist
Joseph Kruskal in 1956
Algorithm:

1. Start with an empty graph
   - With same vertices as original graph
2. Consider edges in increasing weight order
   - Add edge if it does not form a cycle in the MST
3. Repeat until $V - 1$ edges have been added

Critical operations:

- Iterating over edges in weight order
- Checking if adding an edge would form a cycle
Run Kruskal’s algorithm on this graph:
Kruskal’s Algorithm

Example

Minimum Spanning Trees

Kruskal’s Algorithm

Example

Pseudocode

Analysis

Prim’s Algorithm

Comparison

Other Algorithms

Appendix

Add 0-1

Add 3-4

Add 0-3

Don’t add 0-4

Don’t add 1-4

Add 2-3
Kruskal’s Algorithm

Example

Minimum Spanning Trees
Kruskal’s Algorithm
Pseudocode
Analysis
Prim’s Algorithm
Comparison
Other Algorithms
Appendix

MST:

0 — 1
3 — 4
4 — 1
3 — 6
2 — 1
2 — 3
2 — 4
2 — 3
3 — 4
4 — 1
1 — 0

Graph showing the minimum spanning tree (MST) of a given graph.
Kruskal’s Algorithm
Pseudocode (Version 1)

kruskalMst(G):

\textbf{Input:} graph $G$ with $V$ vertices
\textbf{Output:} minimum spanning tree of $G$

$mst = \text{empty graph with } V \text{ vertices}$

$\text{sortedEdges} = \text{sort edges of } G \text{ by weight}$

\textbf{for each} edge $e$ in $\text{sortedEdges}$:
\hspace{1em} add $e$ to $mst$
\hspace{1em} \textbf{if} $mst$ has a cycle:
\hspace{2em} remove $e$ from $mst$

\textbf{if} $mst$ has $V-1$ edges:
\hspace{1em} \textbf{return} $mst$
**Kruskal’s Algorithm**

**Pseudocode (Version 2)**

```plaintext
kruskalMst(G):

**Input:** graph G with V vertices
**Output:** minimum spanning tree of G

mst = empty graph with V vertices

sortedEdges = sort edges of G by weight

for each edge (v, w, weight) in sortedEdges:
    if there is no path between v and w in mst:
        add edge (v, w, weight) to mst

if mst has V−1 edges:
    return mst
```
Proof by exchange argument.

Idea:

- Suppose there exists another algorithm $A$ which makes a different set of choices
  - In this case, chooses a different set of edges for the MST
- Identify one choice made by $A$ which is not made by our algorithm
- Show that by exchanging that choice with one of the choices made by our algorithm, the solution does not become worse or less optimal
  - In this case, the “solution” is the spanning tree produced
  - In this case, an “optimal” solution is a spanning tree that costs as little as possible
Sort the edges of $G$ in increasing order.

Let $K$ be the set of edges selected by Kruskal’s algorithm. Let $A$ be the set of edges selected by a different algorithm.

```plaintext
edges of $G$  e_1  e_2  e_3  e_4  e_5  e_6  e_7  e_8  e_9  \ldots

edges of $K$  e_1  e_2  e_4  e_5  e_7  e_9  \ldots

edges of $A$  e_1  e_2  e_4  e_7  e_8  e_9  \ldots
```

Graph $G$, $K$, and $A$ are shown with their respective edges.
Consider the first edge that is chosen by $K$ but not by $A$. 

\begin{align*}
\text{edges of } G & \quad e_1 \quad e_2 \quad e_3 \quad e_4 \quad e_5 \quad e_6 \quad e_7 \quad e_8 \quad e_9 \quad \cdots \\
\text{edges of } K & \quad e_1 \quad e_2 \quad e_4 \quad e_5 \quad e_7 \quad e_8 \quad e_9 \quad \cdots \\
\text{edges of } A & \quad e_1 \quad e_2 \quad e_4 \quad e_7 \quad e_8 \quad e_9 \quad \cdots \\
\end{align*}
Consider the first edge that is chosen by $K$ but not by $A$. Add this edge to a copy of $A$ (call it $A'$). This forms a cycle in $A'$.
Kruskal’s Algorithm
Analysis - Correctness

Consider the first edge that is chosen by $K$ but not by $A$.
Add this edge to a copy of $A$ (call it $A'$). This forms a cycle in $A'$.
Now find the highest-weight edge in this cycle and remove it from $A'$.

edges of $G$ \hspace{1cm} e_1 \hspace{0.5cm} e_2 \hspace{0.5cm} e_3 \hspace{0.5cm} e_4 \hspace{1cm} e_5 \hspace{1cm} e_6 \hspace{1cm} e_7 \hspace{1cm} e_8 \hspace{1cm} e_9 \hspace{1cm} \ldots

edges of $K$ \hspace{1cm} e_1 \hspace{0.5cm} e_2 \hspace{0.5cm} e_4 \hspace{1cm} e_5 \hspace{1cm} e_7 \hspace{1cm} e_9 \hspace{1cm} \ldots

edges of $A'$ \hspace{1cm} e_1 \hspace{0.5cm} e_2 \hspace{0.5cm} e_4 \hspace{1cm} e_5 \hspace{1cm} e_7 \hspace{1cm} e_9 \hspace{1cm} \ldots
Now $A'$ is once again a spanning tree, but it is more similar to $K$ than $A$ and it costs no more than $A$. 
Now $A'$ is once again a spanning tree, \textit{but} it is more similar to $K$ than $A$ and it costs no more than $A$.

Repeat until $A'$ is identical to $K$. Each time we perform an exchange, the spanning tree does not increase in cost.

Therefore, $K$ is an optimal spanning tree (MST).
Kruskal’s Algorithm
Analysis - Time complexity

Analysis:

- **Sorting edges is** $O(E \cdot \log E)$
- **Main loop has at most** $E$ **iterations**
- **Different ways to check if adding an edge would form a cycle**
  - Cycle/path checking is $O(V)$ in the worst case (adjacency list)
    \[ \Rightarrow \text{overall cost} = O(E \cdot \log E + E \cdot V) = O(E \cdot V) \]
  - Using union-find data structure is close to $O(1)$ in the worst case
    \[ \Rightarrow \text{overall cost} = O(E \cdot \log E + E) = O(E \cdot \log E) = O(E \cdot \log V) \]
Developed by Vojtěch Jarník in 1930 and rediscovered by Robert C. Prim in 1957.
Prim’s Algorithm

Algorithm:

1. Start with an empty graph
2. Start from any vertex, add it to the MST
3. Choose cheapest edge $s-t$ such that:
   - $s$ has been added to the MST, and
   - $t$ has not been added to the MST
   and add this edge and the vertex $t$ to the MST
4. Repeat previous step until $V-1$ edges have been added
   - Or until all vertices have been added

Critical operations:

- Finding the cheapest edge $s-t$ such that
  $s$ has been added to the MST and $t$ has not been added to the MST
Run Prim’s algorithm on this graph (starting at 0):
Prim’s Algorithm

Example

Minimum Spanning Trees

Kruskal’s Algorithm

Prim’s Algorithm

Example

Pseudocode

Analysis

Comparison

Other Algorithms

Appendix
Prim's Algorithm

Pseudocode

`primMst(G)`:  
    **Input:** graph $G$ with $V$ vertices  
    **Output:** minimum spanning tree of $G$

    mst = empty graph with $V$ vertices  
    usedV = {0}  
    unusedE = edges of $G$

    while $|\text{usedV}| < V$:
        find cheapest edge $e (s, t, \text{weight})$ in unusedE such that $s \in \text{usedV}$ and $t \notin \text{usedV}$

        add $e$ to mst
        add $t$ to usedV
        remove $e$ from unusedE

    return mst
Analysis:

- Algorithm considers at most $E$ edges $\Rightarrow O(E)$
- Loop has $V$ iterations
- In each iteration, finding the minimum-weighted edge:
  - With set of edges is $O(E)$
    $\Rightarrow$ overall cost $= O(E + V \cdot E) = O(V \cdot E)$
  - With Fibonacci heap is $O(\log E) = O(\log V)$
    $\Rightarrow$ overall cost $= O(E + V \cdot \log V)$
Comparison

Kruskal’s algorithm vs Prim’s algorithm

Kruskal’s algorithm...
- is $O(E \cdot \log V)$
- uses array-based data structures
- performs better on sparse graphs

Prim’s algorithm...
- is $O(E + V \cdot \log V)$
- uses complex linked data structures
  - in its most efficient implementation (Fibonacci heap)
- performs better on dense graphs
• **Boruvka’s algorithm**
  • Oldest MST algorithm
  • Start with \( V \) separate components
  • Join components using min cost links
  • Continue until only a single component
  • Worst-case time complexity: \( O(E \cdot \log V) \)

• **Karger, Klein and Tarjan**
  • Based on Boruvka’s algorithm, but non-deterministic
  • Randomly selects subset of edges to consider
  • Time complexity: \( O(E) \) on average
https://forms.office.com/r/5c0fb4tvMb
Appendix
Kruskal’s Algorithm Example

Original graph

- Adding 0-1 would not create a cycle
- Adding 3-4 would not create a cycle
- Adding 0-3 would not create a cycle
- Adding 0-4 would create a cycle
- Adding 1-4 would create a cycle
- Adding 2-3 would not create a cycle

Done - MST has 4 edges
Kruskal’s Algorithm Example

Adding 0-1 would not create a cycle

Original graph

- Adding 0-1 would not create a cycle
- Adding 3-4 would not create a cycle
- Adding 0-3 would not create a cycle
- Adding 0-4 would create a cycle
- Adding 1-4 would create a cycle
- Adding 2-3 would not create a cycle

Done - MST has 4 edges
Kruskal’s Algorithm Example

Adding 3-4 would not create a cycle

0 --- 1
|     |
3 --- 4 --- 5
|     |
3 --- 6 --- 2

Original graph
Adding 0-1 would not create a cycle
Adding 3-4 would not create a cycle
Adding 0-3 would not create a cycle
Adding 0-4 would create a cycle
Adding 1-4 would create a cycle
Adding 2-3 would not create a cycle
Done - MST has 4 edges
Adding 0-3 would not create a cycle
Kruskal’s Algorithm Example

Adding 0-4 would create a cycle

![Graph Example](image)
Kruskal’s Algorithm Example

Adding 1-4 would create a cycle
Adding 2-3 would not create a cycle

![Graph example](image-url)
Kruskal’s Algorithm Example

Done - MST has 4 edges

Original graph
Adding 0-1 would not create a cycle
Adding 3-4 would not create a cycle
Adding 0-3 would not create a cycle
Adding 0-4 would create a cycle
Adding 1-4 would create a cycle
Adding 2-3 would not create a cycle

0 1
3 4 1
8 5
2 6
3 7 2
4 6

Kruskal’s Algorithm Example
Prim’s Algorithm Example
Prim’s Algorithm Example

Original graph

- Start at vertex 0
- Choose cheapest edge out of these (in red) and add 0-1 to MST
- Choose cheapest edge out of these (in red) and add 0-3 to MST
- Choose cheapest edge out of these (in red) and add 3-4 to MST
- Choose cheapest edge out of these (in red) and add 3-2 to MST
- Done - MST has 4 edges
Prim’s Algorithm Example

Start at vertex 0

Original graph

- Start at vertex 0
- Choose cheapest edge out of these (in red)
  - Add 0-1 to MST
- Choose cheapest edge out of these (in red)
  - Add 0-3 to MST
- Choose cheapest edge out of these (in red)
  - Add 3-4 to MST
- Choose cheapest edge out of these (in red)
  - Add 3-2 to MST

Done - MST has 4 edges
Choose cheapest edge out of these (in red)
Prim's Algorithm Example

Original graph

Start at vertex 0

Choose cheapest edge out of these (in red)

Add 0-1 to MST

Choose cheapest edge out of these (in red)

Add 0-3 to MST

Choose cheapest edge out of these (in red)

Add 3-4 to MST

Choose cheapest edge out of these (in red)

Add 3-2 to MST

Done - MST has 4 edges
Prim’s Algorithm Example

Choose cheapest edge out of these (in red)

Original graph

Start at vertex 0

Choose cheapest edge out of these (in red)

Add 0-1 to MST

Choose cheapest edge out of these (in red)

Add 0-3 to MST

Choose cheapest edge out of these (in red)

Add 3-4 to MST

Choose cheapest edge out of these (in red)

Add 3-2 to MST

Done - MST has 4 edges
Prim’s Algorithm Example

Add 0-3 to MST
Choose cheapest edge out of these (in red)

Original graph
Start at vertex 0
Choose cheapest edge out of these (in red)
Add 0-1 to MST
Choose cheapest edge out of these (in red)
Add 0-3 to MST
Choose cheapest edge out of these (in red)
Add 3-4 to MST
Choose cheapest edge out of these (in red)
Add 3-2 to MST
Done - MST has 4 edges
Prim’s Algorithm Example

Add 3-4 to MST

Original graph

Start at vertex 0

Choose cheapest edge out of these (in red)

Add 0-1 to MST

Choose cheapest edge out of these (in red)

Add 0-3 to MST

Choose cheapest edge out of these (in red)

Add 3-4 to MST

Done - MST has 4 edges
Choose cheapest edge out of these (in red)
Prim’s Algorithm Example

Original graph

Start at vertex 0

Choose cheapest edge out of these (in red)

Add 0-1 to MST

Choose cheapest edge out of these (in red)

Add 0-3 to MST

Choose cheapest edge out of these (in red)

Add 3-4 to MST

Add 3-2 to MST

Done - MST has 4 edges
Done - MST has 4 edges