Minimum Spanning Tree

Given connected graph $G$ with positive edge weights, find a minimum weight set of edges that connects all of the vertices.

Cayley’s Theorem (1889). There are $V^{V-2}$ spanning trees on the complete graph on $V$ vertices.

- Can’t solve MST by brute force.

Otakar Boruvka (1926).
- Electrical Power Company of Western Moravia in Brno.
- Most economical construction of electrical power network.
- Concrete engineering problem is now a cornerstone problem in combinatorial optimization.

Applications

MST is fundamental problem with diverse applications.

- Network design.
  - telephone, electrical, hydraulic, TV cable, computer, road
- Cluster analysis.
  - analyzing fungal spore spatial patterns
  - microarray gene expression data clustering
  - finding clusters of quasars and Seyfert galaxies
- Approximation algorithms for NP-hard problems.
  - traveling salesperson problem, Steiner tree
- Indirect applications.
  - max bottleneck paths
  - LDPC codes for error correction
  - image registration with Renyi entropy
  - learning salient features for real-time face verification
  - reducing data storage in sequencing amino acids in a protein
  - model locality of particle interactions in turbulent fluid flows
  - autoconfig protocol for Ethernet bridging to avoid cycles in a network
**Medical Image Processing**

Arrangement of nuclei in skin cell for cancer research.

---

**Prim’s Algorithm**

Prim’s algorithm.  (Jarník 1930, Dijkstra 1957, Prim 1959)

- Initialize $T = \emptyset$, $S = \{s\}$ for some arbitrary vertex $s$.
- Grow $S$ until it contains all of the vertices:
  - let $f$ be smallest edge with exactly one endpoint in $S$
  - add edge $f$ to $T$
  - add other endpoint to $S$

<table>
<thead>
<tr>
<th>$S$</th>
<th>$T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>0-1</td>
</tr>
<tr>
<td>5</td>
<td>0-5</td>
</tr>
<tr>
<td>4</td>
<td>5-4</td>
</tr>
<tr>
<td>3</td>
<td>4-3</td>
</tr>
</tbody>
</table>

---

**Prim’s Algorithm: Example**

- Graph showing the growth of $T$ and $S$ with edges and vertices.

---
Observation. Given a spanning tree $T$. Let $f$ be an edge not in $T$. Adding $f$ to $T$ creates a unique cycle. If $c_f < c_e$ for some edge $e$ of cycle, then $T \cup \{f\} - \{e\}$ is a tree of lower cost.

Theorem. Upon termination of Prim's algorithm, $T$ is a MST.

Proof. (by induction on number of iterations)

Invariant: There exists a MST $T^*$ containing all of the edges in $T$.

Base case: $T = \emptyset \Rightarrow$ every MST satisfies invariant.

Induction step: invariant true at beginning of iteration $i$.
- Let $f$ be the edge that Prim's algorithm chooses.
- If $f \in T^*$, $T^*$ still satisfies invariant.
- Otherwise, consider cycle $C$ formed by adding $f$ to $T^*$
  - let $e \in C$ be another arc with exactly one endpoint in $S$
  - $c_f < c_e$ since algorithm chooses $f$ instead of $e$
  - $T^* \cup \{f\} - \{e\}$ satisfies invariant

How to represent a spanning tree?
- List of edges: 0-1 0-5 2-3 3-4 3-7 4-5 5-6 7-8
- Parent-link representation: vertex indexed array $\text{pred}[v]$.

<table>
<thead>
<tr>
<th>$v$</th>
<th>$\text{pred}[v]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>
Prim's Algorithm

Maintain $S$ = set of vertices in current tree.

- For each vertex not in $S$, maintain vertex in $S$ to which it is closest.
- Choose next vertex $v$ to add to $S$ with min $\text{dist}[v]$.
- For each neighbor $w$ of $v$, if $w$ is closer to $v$ than current neighbor in $S$, update $\text{dist}[w]$.

<table>
<thead>
<tr>
<th>v</th>
<th>pred</th>
<th>dist</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>-</td>
</tr>
<tr>
<td>B</td>
<td>E</td>
<td>15</td>
</tr>
<tr>
<td>C</td>
<td>A</td>
<td>-</td>
</tr>
<tr>
<td>D</td>
<td>E</td>
<td>9</td>
</tr>
<tr>
<td>E</td>
<td>F</td>
<td>-</td>
</tr>
<tr>
<td>F</td>
<td>G</td>
<td>-</td>
</tr>
<tr>
<td>G</td>
<td>A</td>
<td>-</td>
</tr>
<tr>
<td>H</td>
<td>C</td>
<td>23</td>
</tr>
<tr>
<td>I</td>
<td>E</td>
<td>11</td>
</tr>
</tbody>
</table>

Weighted Graphs

Weights.

- Method 1: graph access function $G$.cost($v$, $w$).
- Method 2: modify adjacency list iterator to return $\text{Edge}$.

Tradeoffs.

- Method 1 is easier with adjacency matrix or Euclidean weights.
- Method 2 is more general.

Adjacency list implementation.

- Initialize, $\text{dist}[v] = \infty$ and $\text{dist}[s] = 0$.
- Insert all vertices onto PQ.
- Repeatedly delete vertex $v$ from PQ with min $\text{dist}[v]$.
- For each $v$-$w$, if ($\text{dist}[w] > G$.cost($v$, $w$)), update $\text{dist}[w]$

```java
while (!pq.isEmpty()) {
    int v = pq.delMin();
    IntIterator i = G.neighbors(v);
    while (i.hasNext()) {
        int w = i.next();
        if (dist[w] > G.cost(v, w)) {
            dist[w] = G.cost(v, w);
            pq.decrease(w, dist[w]);
            pred[w] = v;
        }
    }
}
```

Decay key

main loop
Priority Queues for Index Items

Index heap-based priority queue.
- Insert, delete min, test if empty.
- Decrease key.

Brute force array implementation.
- Maintain vertex indexed array `dist[w].`
- Decrease key: change `dist[w].`
- Delete min: scan through `dist[w]` for each vertex `w`.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Prim</th>
<th>Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>delete-min</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>decrease-key</td>
<td>E</td>
<td>1</td>
</tr>
<tr>
<td>is-empty</td>
<td>V</td>
<td>1</td>
</tr>
<tr>
<td>total</td>
<td></td>
<td>$V^2$</td>
</tr>
</tbody>
</table>

How to decrease key of vertex `i`? Bubble it up.
How to know which heap node to bubble up? Maintains an extra array `qp[i]` that stores the heap index of vertex `i`.

Design issues.
- PQ maintains priorities; client accesses through PQ interface.
- Client maintains priorities; PQ accesses through client.
- Both maintain their own copy.

```java
public void insert(int k, double value) {
    N++;  
pq[N] = k;  
qp[k] = N;  
priority[k] = value;  
fixUp(pq, N);  
}

public void decrease(int k, double value) {
    priority[k] = value;  
fixUp(pq, qp[k]);  
}

private void exch(int i, int j) {
    int swap = qp[i];  
qp[i] = qp[j];  
qp[j] = swap;  
pq[qp[i]] = i;  
pq[qp[j]] = j;  
}
```

Prim's Algorithm: Implementation Cost Summary

<table>
<thead>
<tr>
<th>Operation</th>
<th>Prim</th>
<th>Array</th>
<th>Binary heap</th>
<th>d-way Heap</th>
<th>Fibonacci heap †</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert</td>
<td>V</td>
<td>V</td>
<td>$\log V$</td>
<td>$d \log_d V$</td>
<td>1</td>
</tr>
<tr>
<td>delete-min</td>
<td>V</td>
<td>V</td>
<td>$\log V$</td>
<td>$d \log_d V$</td>
<td>$\log V$</td>
</tr>
<tr>
<td>decrease-key</td>
<td>E</td>
<td>1</td>
<td>$\log V$</td>
<td>$\log_d V$</td>
<td>1</td>
</tr>
<tr>
<td>is-empty</td>
<td>V</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>total</td>
<td></td>
<td>$V^2$</td>
<td>$E \log V$</td>
<td>$E \log_{E/V} V$</td>
<td>$E + V \log V$</td>
</tr>
</tbody>
</table>

† Individual ops are amortized bounds
Prim’s Algorithm: Priority Queue Choice

The choice of priority queue matters in Prim implementation.

- Array: \( O(V^2) \).
- Binary heap: \( O(E \log V) \).
- Fibonacci heap: \( O(E + V \log V) \).

Best choice depends on whether graph is SPARSE or DENSE.
- 2,000 vertices, 1 million edges. Heap: 2-3x slower.
- 100,000 vertices, 1 million edges. Heap: 500x faster.
- 1 million vertices, 2 million edges. Heap: 10,000x faster.

Bottom line.
- Array implementation optimal for dense graphs.
- Binary heap far better for sparse graphs.
- Fibonacci heap best in theory, but not in practice.

Kruskal’s Algorithm

Kruskal’s algorithm (1956).

- Initialize forest \( F = \emptyset \).
- Consider edges in ascending order of weight.
- If adding edge \( e \) to forest \( F \) does not create a cycle, then add it. Otherwise, discard \( e \).

Kruskal’s Algorithm: Example

Kruskal’s Algorithm: Proof of Correctness

Theorem. Upon termination of Kruskal’s algorithm, \( F \) is a MST.

Proof. Identical to proof of correctness for Prim’s algorithm except that you let \( S \) be the set of nodes in component of \( F \) containing \( v \).

Greed is good. Greed is right. Greed works. Greed clarifies, cuts through, and captures the essence of the evolutionary spirit.” – Gordon Gecko
Kruskal’s Algorithm: Implementation

How to check if adding an edge to F would create a cycle?

- Naïve solution: DFS in O(V) time.
- Clever solution: union-find in \(O(\log^* V)\) amortized time.
  - each tree in forest F corresponds to a set
  - adding \(v-w\) creates a cycle if \(v\) and \(w\) are in same component
  - when adding \(v-w\) to forest F, merge sets containing \(v\) and \(w\)

Kruskal’s Algorithm: Running Time

<table>
<thead>
<tr>
<th>Operation</th>
<th>Frequency</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>sort</td>
<td>1</td>
<td>(E \log E)</td>
</tr>
<tr>
<td>union</td>
<td>(V - 1)</td>
<td>(\log^* V) †</td>
</tr>
<tr>
<td>find</td>
<td>(E)</td>
<td>(\log^* V) †</td>
</tr>
</tbody>
</table>

† Amortized bound using weighted quick union with path compression.

Kruskal running time: \(O(E \log V)\).

If edges already sorted. \(O(E \log^* V)\) time.

Related Problems

<table>
<thead>
<tr>
<th>Year</th>
<th>Problem</th>
<th>Time</th>
<th>Discovered By</th>
</tr>
</thead>
<tbody>
<tr>
<td>1976</td>
<td>Planar MST</td>
<td>(E)</td>
<td>Cheriton-Tarjan</td>
</tr>
<tr>
<td>1992</td>
<td>MST Verification</td>
<td>(E)</td>
<td>Dixon-Rauch-Tarjan</td>
</tr>
<tr>
<td>1995</td>
<td>Randomized MST</td>
<td>(E)</td>
<td>Karger-Klein-Tarjan</td>
</tr>
</tbody>
</table>
Euclidean MST

Given N points in the plane, find MST connecting them.
- Distances between point pairs are Euclidean distances.

Brute force: compute $O(N^2)$ distances and run Prim’s algorithm.
- Memory and running time are $O(N^2)$, which is quadratic in input size.
- Can use squares of distances to avoid taking square roots.

Is it possible to do better by exploiting the geometry?

Euclidean MST

Key geometric fact. Edges of the Euclidean MST are edges of the Delaunay triangulation.

Euclidean MST algorithm.
- Compute Voronoi diagram to get Delaunay triangulation.
- Run Kruskal’s MST algorithm on Delaunay edges.

Running time: $O(N \log N)$.
- Fact: $\leq 3N - 6$ Delaunay edges since it’s planar.
- $O(N \log N)$ for Voronoi.
- $O(N \log N)$ for Kruskal.

Lower bound. Any comparison-based Euclidean MST algorithm requires $\Omega(N \log N)$ comparisons.

Optimal Message Passing

Optimal message passing.
- Distribute message to N agents.
- Each agent $i$ can communicate with some of the other agents $j$, but their communication is (independently) detected with probability $p_{ij}$.
- Group leader wants to transmit message to all agents so as to minimize overall probability of detected.

Objective.
- Find tree $T$ that minimizes: $1 - \prod_{(i,j) \in T} (1 - p_{ij})$
- Or equivalently, that maximizes: $\prod_{(i,j) \in T} (1 - p_{ij})$
- Or equivalently, that maximizes: $\sum_{(i,j) \in T} \log (1 - p_{ij})$

Algorithm. MST with weights $= -\log (1 - p_{ij})$. Weights $p_{ij}$ also work!