Recap: Greedy Algorithms

- **Minimizing Lateness**
  - **Input**: list of jobs \((t_1, d_1), \ldots, (t_n, d_n)\) where job \(j\)
    - requires \(t_j\) units of processing time and
    - is due at time \(d_j\).
  - **Goal**: Find schedule to minimize maximum late time.
  - **Greedy Algorithm**: Sort jobs by earliest deadline
  - **Running Time**: \(O(n \log n)\)

- **Offline Cache Eviction Problem**
  - **Input**: list of page requests, cache size \(m\)
  - **Goal**: Find eviction schedule that minimizes number of cache misses
  - **Solution**: Evict the item that will be requested furthest in the future.

4.5 Minimum Spanning Tree

**Minimum Spanning Tree**

Given a connected graph \(G = (V, E)\) with real-valued edge weights \(c_e\), an MST is a subset of the edges \(T \subseteq E\) such that \(T\) is a spanning tree whose sum of edge weights is minimized.

**Cayley’s Formula**

There are \(n^n\) spanning trees of \(K_n\).

**Applications**

- **Network design**
  - telephone, electrical, hydraulic, TV cable, computer, road
- **Approximation algorithms for NP-hard problems**
  - traveling salesman 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
- **Cluster analysis**
**Greedy Algorithms**

**Kruskal’s algorithm.** Start with \( T = \emptyset \). Consider edges in ascending order of cost. Insert edge \( e \) in \( T \) unless doing so would create a cycle.

**Reverse-Delete algorithm.** Start with \( T = E \). Consider edges in descending order of cost. Delete edge \( e \) from \( T \) unless doing so would disconnect \( T \).

**Prim’s algorithm.** Start with some root node \( s \) and greedily grow a tree \( T \) from \( s \) outward. At each step, add the cheapest edge \( e \) to \( T \) that has exactly one endpoint in \( T \) that has exactly one endpoint in \( T \).

**Remark.** All three algorithms produce an MST.

---

**Greedy Algorithms**

**Simplifying assumption.** All edge costs \( c_e \) are distinct.

**Cut property.** Let \( S \) be any subset of nodes, and let \( e \) be the min cost edge with exactly one endpoint in \( S \). Then the MST \( T^* \) contains \( e \).

**Pf.** (exchange argument)

\[ \text{Suppose } e \text{ does not belong to } T^*, \text{ and let's see what happens.} \]

Adding \( e \) to \( T^* \) creates a cycle \( C \) in \( T^* \).

Edge \( e \) is both in the cycle \( C \) and in the cutset \( D \) corresponding to \( S \) \( \Rightarrow \) there exists another edge, say \( f \), that is in both \( C \) and \( D \).

\[ T' = T^* \cup \{ e \} - \{ f \} \text{ is also a spanning tree.} \]

Since \( c_e < c_f \), cost\( (T') < \text{cost}(T^*) \).

This is a contradiction.

---

**Greedy Algorithms**

**Cycles and Cuts**

**Cycle.** Set of edges in the form \( a-b, b-c, \ldots, y-x, x-a \).

**Cutset.** A cut is a subset of nodes \( S \). The corresponding cutset \( D \) is the subset of edges with exactly one endpoint in \( S \).

**Greedy Algorithms**

**Cycle-Cut Intersection**

**Claim.** A cycle and a cutset intersect in an even number of edges.

**Pf.** (by picture)

\[ \text{Suppose } f \text{ belongs to } T^*, \text{ and let's see what happens.} \]

Deleting \( f \) from \( T^* \) creates a cut \( S \) in \( T^* \).

Edge \( f \) is both in the cycle \( C \) and in the cutset \( D \) corresponding to \( S \) \( \Rightarrow \) there exists another edge, say \( e \), that is in both \( C \) and \( D \).

\[ T' = T^* - \{ f \} \text{ is also a spanning tree.} \]

Since \( c_e < c_f \), cost\( (T') < \text{cost}(T^*) \).

This is a contradiction.
Prim’s Algorithm: Proof of Correctness

Prim’s algorithm. [Jarník 1930, Dijkstra 1959, Prim 1957]
- Initialize S = any node.
- Apply cut property to S.
- Add min cost edge in cutset corresponding to S to T, and add one new explored node u to S.

Implementation: Prim’s Algorithm

Implementation. Use a priority queue a la Dijkstra.
- Maintain set of explored nodes S.
- For each unexplored node v, maintain attachment cost a[v] = cost of cheapest edge v to a node in S.
- O(n^2) with an array; O(m log n) with a binary heap; O(m + n log n) with Fibonacci Heap

Prim() {
    foreach (v ∈ V) a[v] = ∞
    Initialize an empty priority queue Q
    foreach (v ∈ V) insert v onto Q
    Initialize set of explored nodes S = ∅
    while (Q is not empty) {
        u — delete min element from Q
        S = S ∪ {u}
        foreach (edge e = (u, v) incident to u)
            if ((v ∈ S) and (ce < a[v]))
                decrease priority a[v] to ce
    }
}

Implementation: Kruskal’s Algorithm

Implementation. Use the union-find data structure.
- Build set T of edges in the MST.
- Maintain set for each connected component
- O(m log n) for sorting and O(m log n) for union-find.

Kruskal(G, c) {
    Sort edges weights so that c1 ≤ c2 ≤ ... ≤ cm.
    T = Ø
    foreach (u ∈ V) make a set containing singleton u
    for i = 1 to m
        (u, v) = ei
        if (u and v are in different sets) {
            T = T ∪ {ei}
            merge the sets containing u and v
        }
    return T
}

Lexicographic Tiebreaking

To remove the assumption that all edge costs are distinct: perturb all edge costs by tiny amounts to break any ties.

Impact. Kruskal and Prim only interact with costs via pairwise comparisons. If perturbations are sufficiently small, MST with perturbed costs is MST with original costs.

Implementation. Can handle arbitrarily small perturbations implicitly by breaking ties lexicographically, according to index.
### 4.7 Clustering

**MST Algorithms: Theory**

- Deterministic comparison based algorithms:
  - \( O(m \log n) \) [Janák, Prim, Dijkstra, Kruskal, Boruvka]
  - \( O(m \log \log n) \) [Cheriton-Tarjan 1976, Yao 1975]
  - \( O(m \log (m, n)) \) [Fredman-Tarjan 1987]
  - \( O(m = (m, n)) \) [Chazelle 2000]

- Holy grail: \( O(m) \)

- **Notable**:
  - \( O(m) \) randomized. [Karger-Klein-Tarjan 1995]
  - \( O(n) \) verification. [Dixon-Rauch-Tarjan 1992]

**Euclidean**

- 2-d: \( O(n \log n) \) compute MST of edges in Delaunay
- k-d: \( O(k n^2) \) dense Prim

**Clustering**

- **Clustering**: Given a set \( U \) of \( n \) objects labeled \( p_1, \ldots, p_n \), classify into coherent groups.

- **Distance function**: Numeric value specifying "closeness" of two objects.

- **Fundamental problem**: Divide into clusters so that points in different clusters are far apart.
  - Routing in mobile ad hoc networks
  - Identity patterns in gene expression
  - Document categorization for web search
  - Similarity searching in medical image databases
  - Skycat: cluster 109 sky objects into stars, quasars, galaxies

- **Clustering of Maximum Spacing**
  - **k-clustering**: Divide objects into \( k \) non-empty groups.

  - **Distance function**: Assume it satisfies several natural properties.
    - \( d(p, p_j) = 0 \) iff \( p_i = p_j \) (identity of indiscernibles)
    - \( d(p, p_j) \geq 0 \) (nonnegativity)
    - \( d(p, p_j) = d(p_j, p) \) (symmetry)

  - **Spacing**: Min distance between any pair of points in different clusters.

  - **Clustering of maximum spacing**: Given an integer \( k \), find a \( k \)-clustering of maximum spacing.

- **Greedy Clustering Algorithm**

  - **Single-link k-clustering algorithm**:
    - Form a graph on the vertex set \( U \), corresponding to \( n \) clusters.
    - Find the closest pair of objects such that each object is in a different cluster, and add an edge between them.
    - Repeat \( n-k \) times until there are exactly \( k \) clusters.

  - **Key observation**: This procedure is precisely Kruskal’s algorithm (except we stop when there are \( k \) connected components).

  - **Remark**: Equivalent to finding an MST and deleting the \( k-1 \) most expensive edges.

- **Greedy Clustering Algorithm: Analysis**

  - **Theorem**: Let \( C^* \) denote the clustering \( C^*_1, \ldots, C^*_k \) formed by deleting the \( k-1 \) most expensive edges of an MST; \( C^* \) is a \( k \)-clustering of max spacing.

  - **Proof**:
    - Let \( C \) denote some other clustering \( C_1, \ldots, C_k \).
    - The spacing of \( C^* \) is the length \( d^* \) of the \((k-1)\)th most expensive edge.
    - Let \( p, q \) be in the same cluster in \( C^* \), any \( C^*_i \), but different clusters in \( C \) by \( C_i \) and \( C_j \).
    - Some edge \((p, q)\) on \( p-q \) path in \( C^* \) spans two different clusters in \( C \).
    - All edges on \( p-q \) path have length \( \leq d^* \) since Kruskal chose them.
    - Spacing of \( C \) is \( \leq d^* \) since \( p \) and \( q \) are in different clusters.

- **Clustering of Maximum Spacing**: Given an integer \( k \), find a \( k \)-clustering of maximum spacing.
MST Algorithms: Theory

Deterministic comparison based algorithms:
- $O(m \log n)$ [Jarník, Prim, Dijkstra, Kruskal, Boruvka]
- $O(m \log \log n)$ [Cheriton-Tarjan 1976, Yao 1975]
- $O(m \log\log n)$ [Fredman-Tarjan 1987]
- $O(m \log\log\log n)$ [Gabow-Goll-Branch-Tarjan 1986]
- $O(m \log (m, n))$ [Chazelle 2000]

Holy grail: $O(m)$.

Notable:
- $O(m)$ randomized. [Karger-Klein-Tarjan 1995]
- $O(m)$ verification. [Dixon-Rauch-Tarjan 1992]

Euclidean:
- 2-d: $O(n \log n)$, compute MST of edges in Delaunay
- k-d: $O(kn^2)$, dense Prim

Dendrogram of Cancers in Human

Tumors in similar tissues cluster together.

Extra Slides