Clustering with MST Shortest Path Problems

CSCI 3100

MST in Clustering Problems

Clustering – a problem of grouping input into a number of components, based on how similar inputs are to each other.

- Define DISTANCE function between inputs

Example 1 – Cluster points on a 2D plot into K sets

- Define DISTANCE(u, v) as the Euclidean distance

Example 2 – Cluster images into K sets based on their similarity

- Define DISTANCE(u, v) as a function of pixel color, intensity, etcc

Build a graph G

- Each input is a vertex on the graph
- There is an edge between each pair of vertices
- The weight of edge (v, w) is DISTANCE(v, w)

K-Clustering of a set of objects U is partitioning of U into k non-empty subsets, such that:

The subsets are as far apart from each other as possible

Define: ***spacing*** of a k-clustering as the minimum distance between any pair of points in different clusters

In k-clustering, we want to maximize the spacing

Solution:

- Find MST of the graph
- Delete k-1 edges of the resulting tree

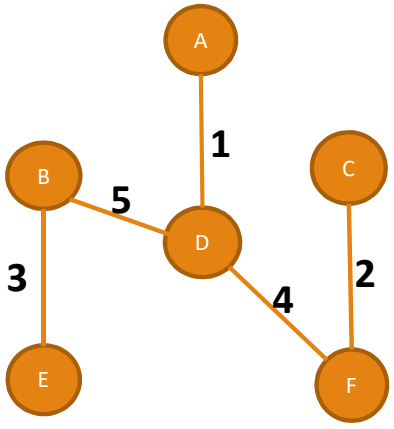
Example: k-clustering

Suppose the tree on the right is the MST of the graph

To find 3-clustering, we will remove 2 largest edges: (B, D) and (D, F))

The three clusters are:
{A, D}, {B, E}, {C, F}

Then the spacing of this clustering is 5



Prove that removing k-1 edges from MST of a graph results in maximum spacing

Which option is **NOT** a good start of this proof?

A. Let d be the spacing produced by the algorithm. Show that all other solutions have smaller spacing.

B. Assume there exists a solution with larger spacing. Show that this leads to a contradiction

C. Assume that our algorithm results in the maximum spacing. Show that adding k-1 edges to our solution results in an MST.

D. All above approaches are valid.

Shortest Path

Suppose you have a job interview

- You've never been to this location before
- You want to be on time
- So you look up directions on your phone
- There are several suggestions with time estimates

This is an example of the "shortest path" prooblem

Formal definition

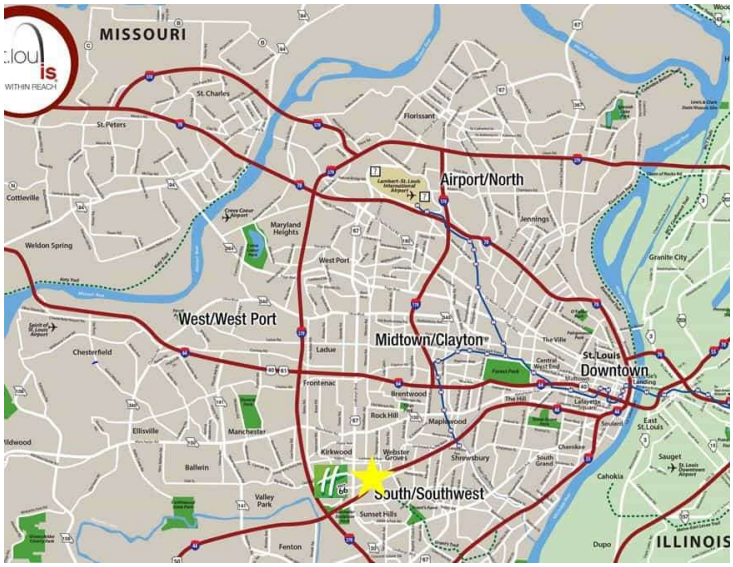
In a shortest path problem, we are given:

- A weighted directed graph G = (V, E)
- With a weight function w: E-> $\mathbb{R}$ mapping edges to real-valued weights

The weight *w(p)* of path $p = \langle v_0, v_1, \ldots, v_k \rangle$ is the sum of weights of its constituent edges:

- $w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$

The shortest path from *u* to *v* is a with the smallest *w(p)*



Airport to Downtown

Multiple paths

Not all paths have the same travel time/cost/distance

Define a graph where vertices are intersections of roads, edges are roads

Define weight as:

- Travel time
- Distance
- Cost (toll roads)

Run shortest path algorithm

Variations of the problem

Single-source shortest path: shortest path from the starting vertex to all other vertices

Single-destination shortest path: shortest path to a given destination vertex from all other vertices

Single-pair shortest path: shortest path from *u* to *v*, for given vertices *u* and *v*.

All-pairs shortest path: shortest path from *u* to *v* for every pair of vertices *u* and *v*.

Given a shortest path $p=\langle v_1, v_2, \ldots, v_k \rangle$ from v_1 to v_k in a directed graph G, the following is true:

A. The path $\langle v_k, v_{k-1}, \ldots, v_2, v_1 \rangle$ is the shortest path from v_k to v_1
B. All other paths from v_1 to v_k have a weight greater than w(p)
C. All edges of p are in a minimum spanning tree of G
D. If path $\langle v_i, v_{i+1}, \ldots, v_j \rangle$ is a sub-path of p, then it is the shortest path from vi to vj
E. All of the above statements are true

Optimal Substructure of the Shortest Path

Shortest path problem exhibits optimal substructure

Can apply dynamic programming and (possibly) greedy algorithms

Given a weighted graph G=(V, E), where some edge weights can be negative, and shortest path $p=\langle v_1, v_2, \ldots, v_k \rangle$ from v_1 to v_k, the following is true:

A. The path p cannot contain a cycle
B. The path p can contain a cycle, only if the sum of weights of edges in the cycle is less than or equal to 0.
C. The path p can contain a cycle, only if the sum of weights of edges in the cycle is equal to 0.
D. None of the above

Negative Weights

Some edges may include negative weights

Shortest path from s to other vertices is **well-defined** if there are no negative weight cycles reachable from source *s.*

Example:

Restrict our attention to shortest paths with at most |V|-1 edges

Properties of shortest paths

Define δ(s, u) be the weight of a shortest path from s to u.

Triangle inequality:
For any edge (u, v), $\delta(s, v) \leq \delta(s, u) + w(u, v)$

No-path property:
If there is no path from s to v, then $\delta(s, v) = \infty$

Shortest Path *estimate*

Given a graph G and a starting vertex s,

- Define v.d for all vertices of G as the *estimate* of the shortest path from s to v
- $v.d \geq \delta(s, v)$

Representing a path

Use "predecessor" notation to represent a path.

Given a path $p=<v_1, v_2, \ldots, v_k>$, for all $i>1$, the predecessor of v_i is v_{i-1}

The predecessor of the starting vertex is NULL

$v.\pi$ is the predecessor of v in the path.

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

Relaxation (compute estimates)

The Bellman-Ford algorithm idea

Given a graph G and a starting vertex s

Apply "relaxation" to each vertex, v, until v.d = δ(s, v)

Whenever a smaller v.d is found, update the predecessor of v, v.π

Question: how do we know when v.d = δ(s, v)?

Relaxation properties

Convergence property

- If p is a shortest path from s to v using an edge (u, v), and
- if u.d = δ(s, u) at any time prior to relaxing edge (u, v),
- then v.d= δ(s, v) after edge (u, v) has been relaxed.

Path relaxation property

- If p=<v_1, v_2, …, v_k> is a shortest path from v_1 to v_k, and we relax the edges of p in order (v_1, v_2), (v_2, v_3), …, (v_{k-1}, v_k), then v_k.d = δ(v_1, v_k)
- This property holds even if other relaxations are intermixed with the relaxation of edges of p

Bellman-Ford Algorithm

Line 1

Lines 2 – 4

Lines 5 - 7

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3      for each edge (u, v) ∈ G.E
4          RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6      if v.d > u.d + w(u, v)
7          return FALSE
8  return TRUE
```