MAT344 Lecture 21

2019/July/30

1 Announcements

2 This week

This week, we are talking about

1. Minimum weight spanning trees

3 Recap

Last time we talked about

1. Probability

4 Minimum weight spanning trees (Ch.12.1 in [KT17])

We have looked at graphs as representing many real-world scenarios before. However, in many cases, a graph does not contain all the necessary information that would model the problem accurately. For example, if we have a graph that represnts nodes in a future network, and we want to establish connections between these nodes, it is likely that the cost of establishing connections between different nodes are different.

For example, consider the graph in 1. The nodes may represent locations of computers, and the labels (weights) on the edges represent the cost of laying the cables necessary for the connection between two given locations. We would like all computers to be able to communicate with each other, and we want to minimize the cost of establishing the connection.

Definition 4.1. *Let $G = (V_G, E_G)$ be a graph, and let $w : E_G \to \mathbb{R}_{\geq 0}$ be a function. For any edge $e \in E_G$, the quantity $w(e)$ is called the **weight** of e. If $H = (V_H, E_H) \subseteq G$ is a subgraph of G, then $w(H) = \sum_{e \in E_H} w(e)$.*

In mathematical terms, we are looking for a spanning tree in this graph that has a minimal sum of weights. Recall that a labeled graph with n vertices can have as many as n^{n-2} spanning trees, so enumerating over all cases is not really an option, we need a more clever algorithm.

This question is one that can be solved by a *greedy algorithm*. A greedy algorithm is one that proceeds with a step that seems most optimal immediately.

5 Prim's algorithm (Ch. 5.6 in [Gui18])

Prim's algorithm was developed in 1930 by Vojtěch Jarník, then it was rediscovered by Robert C. Prim in 1957.

Given a weighted connected graph G, we construct a minimum weight spanning tree T as follows.

Choose any vertex v_0 in G and include it in T . If vertices $S = \{v_0, v_1, \ldots, v_k\}$ have been chosen, choose an edge with one endpoint in S and one endpoint not in S and with smallest weight among all such edges. Let v_{k+1} be the endpoint of this edge not in S, and add it and the associated edge to T. Continue until all vertices of G are in T.

Theorem 5.1 (Theorem 5.6.2 in [Gui18])**.** *Prim's algorithm produces a minimum weight spanning tree*

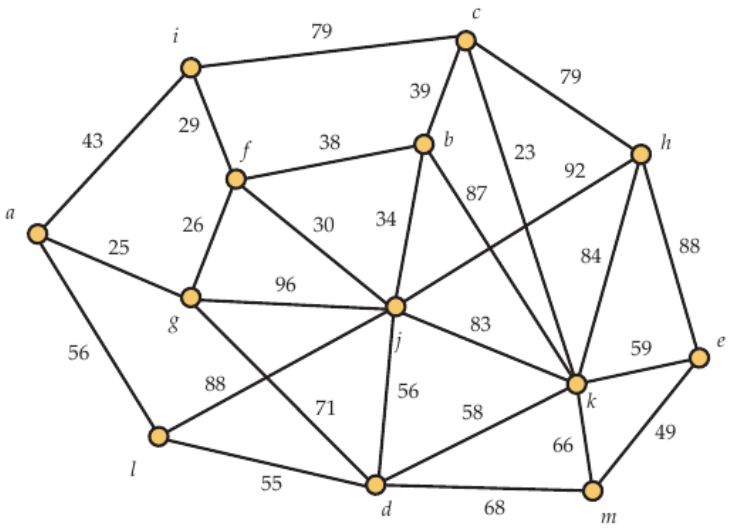

Figure 1: A proposed network

Proof. Suppose G is connected on n vertices. Let T be the spanning tree produced by the algorithm, and T_{min} a minimum cost spanning tree. We prove that $c(T) = c(T_{min})$.

Let $e_1, e_2, \ldots, e_{n-1}$ be the edges of T in the order in which they were added to T; we label the vertices in a way that one endpoint of e_i is v_i, the other is in $\{v_0, \ldots, v_{i-1}\}$. We construct a sequence of trees $T_{min} = T_0, T_1, \ldots, T_{n-1} = T$ such that for each i, $c(T_i) = c(T_{i+1})$.

Suppose we have constructed tree T_i. If e_{i+1} is in T_i, let $T_{i+1} = T_i$. Otherwise, add edge e_{i+1} to T_i. This creates a cycle, one of whose edges, call it f_{i+1}, is not in $e_1, e_2, \ldots, e_i$ and has exactly one endpoint in $\{v_0, \ldots, v_i\}$. Remove f_{i+1} to create T_{i+1}. Since the algorithm added e_{i+1}, $c(e_{i+1}) \leq c(f_{i+1})$. If $c(e_{i+1}) < c(f_{i+1})$, then $c(T_{i+1}) < c(T_i) = c(T_m)$, a contradiction, so $c(e_{i+1}) = c(f_{i+1})$ and $c(T_{i+1}) = c(T_i)$.

Therefore $c(T) = c(T_{n-1}) = c(T_0) = c(T_{min})$.

Q.E.D.

Remark 5.2. *There is another commonly used algorithm, known as Kruskal's algorithm. See [KT17], Ch. 12.2.1 for details.*

References

[Gui18] David Guichard. *Combinatorics and Graph Theory.* Open access, 2018. Available at https://www.whitman.edu/mathematics/cgt_online/book/. 1

[KT17] Mitchel T. Keller and William T. Trotter. *Applied Combinatorics.* Open access, 2017. Available at http://www.rellek.net/appcomb/. 1, 2