

Lecture 3: More Tree Properties and Applications

Claim 1 *Characterization of trees*

If a graph $G(V, E)$ has any two of the following three properties, it has all three

1. G is connected
2. G has no cycles
3. $|E| = |V| - 1$

Therefore, any graph with any two of these properties is a tree.

Proof: 1 and 2 define a tree and we've already shown that a tree has $|V| - 1$ edges.

First we'll prove 1 and 3 imply 2. Assume 1 and 3 are true. Assume also that 2 is not true, i.e. there is at least one cycle in the graph. Choose one of the cycles and delete an edge from it. It is easy to see that the graph remains connected. Repeat until no cycles remain. Now condition 2 is true and together with condition 1 we have a tree. Hence condition 3 must hold. But since we deleted at least one edge after starting with $|E| = |V| - 1$ we get a contradiction.

Finally, we'll prove 2 and 3 imply 1. Assume 1 is false, i.e. that G is disconnected. We can then partition V into k ($k > 1$) subsets V_1, \dots, V_k such that, for all i , the graph induced on V_i is connected and there is no edge between V_i and V_j for $i \neq j$. Call such graphs G_i . Then each G_i is a tree since there are no cycles and we have $|E| = \sum |E(G_i)| = |V| - k < |V| - 1 = |E|$, which is a contradiction. ■

Minimum Spanning Trees

Suppose we have n nodes and for each pair of nodes i and j , we know the cost to connect those two nodes $c(i, j) \geq 0$. What is cheapest connected subgraph one can get?

Clearly, there is a solution that is a tree. Otherwise, suppose the solution has at least one cycle. Pick an edge from that cycle and delete. The resulting graph is still connected and has a cost of at most that of the previous graph. So how can one find such tree?

This is the Minimum Spanning Tree problem (MST). To solve this problem, one way would be to use Prüfer codes to enumerate each possible tree, and compute the tree with smallest

cost. It is important to realize that Prüfer codes define an ordering on trees of size n , as well as a method for uniform sampling of labelled trees of size n .

However, computing each of the n^{n-2} labelled trees can take a long time. If $n = 100$, there are 100^{98} trees, which is much more than the number of atoms in the observable universe! We need to come with a more efficient way to find an MST. To do so, we will exploit the structure of the MST problem, namely that spanning trees on a graph create a matroid, and greedy algorithms tend to work well on matroids.

Given a simple graph $G(V, E)$ and an edge cost function $c(\cdot) \geq 0$, the following algorithm produces a spanning tree T which minimizes $\sum_{e \in E(T)} c(e)$ and is due to Joseph Kruskal.

The Greedy Algorithm: Greedy-MST

1. Initialize $E(T) = \{\}$ and $V(T) = V$.
 2. Sort the edges e_1, \dots, e_m so that $c(e_{i_1}) \leq c(e_{i_2}) \leq \dots \leq c(e_{i_m})$
 3. While $|E(T)| < |V| - 1$, add cheapest unused e_{i_k} so that T has no cycle.
-

Claim 2 *Greedy-MST produces a MST*

Proof: For simplicity, we assume that all of the edge costs are distinct, i.e. $c(e_{i_1}) < c(e_{i_2}) < \dots < c(e_{i_m})$. Let's first prove that e_{i_1} is in *MST*. Call T the tree Greedy-MST outputs.

Assume e_{i_1} does not belong to *MST*. Add e_{i_1} to *MST*. This creates a cycle in the new constructed graph. Remove any edge from such cycle (except e_{i_1}). We have a new spanning tree whose weight is strictly smaller than that of *MST*, which is a contradiction.

We prove the rest by induction. Let e_{i_1}, \dots, e_{i_k} be the first k edges Greedy-MST picks. Assume they all belong to *MST*. Suppose $e_{i_{k+1}}$ does not belong to *MST*. Add it; the resulting graph has a cycle. Given that $e_{i_{k+1}} \in E(T)$, such cycle has at least one edge whose cost is strictly greater than that of $e_{i_{k+1}}$. Remove that edge, thus creating a new spanning tree of smaller weight. Contradiction. ■

Application: Clustering in Bio-informatics

Biologists commonly use DNA arrays to analyze gene function. This technique allows the measurement of "expression levels" (amounts of mRNA produced) in genes. The result is a $n \times m$ matrix recording in each row the expression levels of a gene.

Clustering algorithms allow for genes with similar expression levels to be linked together in hopes that genes placed in common clusters share common functions. To achieve this an $n \times n$ distance matrix D is constructed which records in each entry how close, or similar, two genes are based on their expression levels.

The goal of clustering is to classify nodes, i.e. we want to identify clusters of nodes that are close to each other and those clusters that are far from each other as well. Many clustering algorithms exist to achieve this goal but the biggest difference usually amounts to how to define a distance in between groups of nodes. Some common heuristics are based on separation or average distance between nodes.

One possibility is to use the smallest distance between any pair of nodes. More formally, let A and B be two disjoint sets. We define the distance between A and B as the minimal distance between any two nodes, one in A and the other in B . Given this definition, if we want to cluster n points into k clusters so that the distance between clusters is maximized, we just run Greedy-MST and stop $k - 1$ steps before it ends. This is equivalent to removing $k - 1$ edges from a fully constructed MST. Clearly this will not always produce k clusters and some modifications are usually imposed to achieve good performance on different kinds of data sets.

We can also use MST to create a hierarchical clustering of a set. In biology, this tool is used to classify species, i.e. to build the “tree of life” or the phylogenetic tree. Solving this problem exactly is very hard but several approximations heuristics are used, one of which relies on minimum spanning trees. To build a “maximum likelihood” phylogenetic tree in this way, one can use the “closeness” of DNA sequences between different species as weights of a graph and build the MST of this graph. The phylogenetic tree can then be reconstructed by looking at where the nodes are located in the MST.

References

- [1] Neil C. Jones, Pavel A. Pevzner, *An Introduction To Bioinformatics Algorithms*.