CPSC 320 Notes, Clustering Completed

February 2, 2017

AS BEFORE: We're given a complete, weighted, undirected graph $G = (V, E)$ represented as an adjacency list, where the weights are all between 0 and 1 and represent similarities—the higher the more similar—and a desired number $1 \leq k \leq |V|$ of categories.

We define the similarity between two categories C_1 and C_2 to be the maximum similarity between any pair of nodes $p_1 \in C_1$ and $p_2 \in C_2$. We must produce the categorization—partition into k (non-empty) sets—that minimizes the maximum similarity between categories.

Now, we'll prove this greedy approach optimal.

1. Sort a list of the edges E in decreasing order by similarity.
2. Initialize each node as its own category.
3. Initialize the category count to $|V|$.
4. While we have more than k categories:
   (a) Remove the highest similarity edge (u, v) from the list.
   (b) If u and v are not in the same category: Merge u's and v's categories, and reduce the category count by 1.

1 Greedy is at least as good as Optimal

We'll start by noting that any solution to this problem partitions the edges into the "intra-category" edges (those that connect nodes within a category) and the "inter-category" edges (those that cross categories).

1. **Getting to know the terminology:** Imagine we're looking at a categorization produced by our algorithm in which the inter-category edge with maximum similarity is e.

   Can our greedy algorithm's solution have an intra-category edge with **lower** weight than e? Either draw an example in which this can happen, or sketch a proof that it cannot.

2. Give a bound—indicating whether it's an upper- or lower-bound—on the maximum similarity of an arbitrary solution in terms of any one of its inter-category edge weights. (That is, I tell you that the solution has an inter-category edge with weight w. How much can you tell me so far about the solution's overall "goodness"?)

3. Give a bound on the maximum similarity of a solution produced by the greedy algorithm in terms of the weight of any one of the edges it merged on (in step 4(b)).

4. Consider an optimal solution $\mathcal{O}$ to an instance of the problem. Prove that its intra-category edges cannot be a proper superset of greedy's intra-category edges (i.e., cannot be the same plus at least one more intra-category edge).

   You should **assume** that both $\mathcal{O}$ and the greedy algorithm produce valid solutions, i.e., partitions of V into exactly k subsets. (That's clearly true for $\mathcal{O}$ since it's an optimal solution and not too hard to prove for the greedy approach.)

5. If $\mathcal{O}$'s set of intra-category edges is not a superset of greedy's and it's not the same solution (i.e., the edge sets of the two are not the same), prove that at least one edge that greedy merged on in step 4(b) is an inter-category edge in $\mathcal{O}$.

6. Prove that if $\mathcal{O}$'s set of intra-category edges is neither equal to nor a superset of greedy's, then greedy's solution is optimal. (Remember: optimal doesn't mean "better than all other solutions", just "at least as good as all other solutions".)

2 Challenge

1. We can also give an "exchange" argument starting: "Consider a greedy solution g and an optimal solution $\mathcal{O}$. If they're different, then some edge (u, v) must be inter-category in $\mathcal{O}$ but intra-category in g. In that case, we can make $\mathcal{O}$ more similar to g without decreasing $\mathcal{O}$'s goodness by..."

   Finish the proof. (Warning: even if you select (u, v) more carefully, you may not **just** want to move u into v's category or vice versa.)

2. Switching to MSTs:

   (a) What does the "cut" property tell us if we partition the graph into a "cluster" of: an already connected set of nodes, and everything else?

   (b) Use this insight to create an efficient algorithm somewhat similar to Kruskal's algorithm except that it adds **many** edges to the MST-so-far at each step of the outer loop.

   (c) Analyse the performance of your algorithm, including determining what data structures to use. (Make each data structure as simple as possible while still obtaining the best bound rather than making each as efficient as possible.)