CPSC 320 Notes, Clustering

You're working on software to manage people's photos. Your algorithm receives as input:

- a bunch of uncategorized photos,
- the number of categories to group them into (i.e., how many categories to use), and
- a *similarity measure* for each pair of photos.

(A 0 similarity indicates two photos are nothing like each other; a 1 indicates two photos are exactly the same. All other similarities are in between.)

Your algorithm's job is to create a "categorization": the requested number of categories, where a category is just a (non-empty) set of the photos contained in that category. Every photo belongs to some category, and no photo belongs to more than one category. So, a categorization is a "partition".

Note: we assume that to your algorithm the photos are just nodes with no special content. The similarities encode everything it needs to know about them. So, for example, your algorithm cannot "find the prominence of a person in the photo" because that relies on the photos' contents.

Sketches (drawings) will be incredibly important for understanding this problem!

1 Trivial and Small Instances

1. Write down all the **trivial** instances of the categorization problem.

2. Write down another **small** instance of the categorization problem of different sizes. (Try to find the smallest non-trivial instance you can.)

3. Give solutions to your trivial/small instances by hand if you haven't already done so. In your small instances, briefly justify why your solution is the best one.

2 Represent the Problem

1. We represent an unweighted, undirected graph as $G = (V, E)$, where V is a set of nodes, E is a set of edges, and each edge is a tuple (u, v) that we consider to be unordered, where $u, v \in V$.

   That's not quite right for our problem. Modify the representation to describe the input graph in this problem. Also describe the additional parameter (the desired number of categories).

2. Go back up and rewrite one trivial and one small instance using these names.

3. Now draw a reasonable graph for this set of numbered pictures[1] (You need to choose similarities between each pair of photos; just quickly choose some that you like! Note that while you're doing this step, your algorithm does **not** do it! It receives the similarities as input.)

4. The graph you've drawn is **not** an instance of this problem yet because something is missing. **Read the specification at the start of this worksheet carefully**, figure out what's missing, and add that missing element. Use the value 2.

5. Our input graph has some constraints. We'll skip expressing "no self-loops" and "no duplicate edges", since we've done that before. Similarities must also be between 0 and 1, and **every** pair of photos has a similarity. Further, only certain numbers of categories make sense. Express these constraints.

[1] Picture of Homer Simpson retrieved from `https://simpsons.fandom.com/wiki/Donuts`, 10 Jul 2019; picture of Liz Lemon retrieved from `http://yestotally.com/food-drink/farewell-30-rock-cheesy-blasterz-and-homemade-pop-tarts/`, 10 Jul 2019; picture of Cartman retrieved from `https://www.instructables.com/id/How-to-make-your-own-South-Park-Cheesy-Poofs-4-/`, 10 Jul 2019.

3 Represent the Solution

1. What are the quantities that matter in the solution to the problem? Give them short, usable names.

2. Describe using these quantities what makes a solution **valid** and **good**.

3. Does your metric for the "goodness" of a categorization give the same result for these two categorizations of a four-node instance into two categories: $\{1,3\},\{2,4\}$ and $\{4,2\},\{1,3\}$? Should it? Why or why not?

   (Adjust your categorization as needed!)

4. Go back up to your trivial and small instances and write out one or more solutions to each using these names.

5. Go back up to your drawn representations of instances and draw at least one solution.

6. From here on, we'll all use the same "goodness" measure.

   First, we define the similarity between two categories C_1 and C_2 to be the maximum similarity between any pair of photos p_1, p_2 such that $p_1 \in C_1$ and $p_2 \in C_2$.

   Then, the "goodness" of a categorization is the maximum similarity between any two of its categories, and the best "goodness" is 0. (Note that we are indeed "minimizing a maximum". The "goodness"—or maybe it would be better to call this the "badness"—of a single solution is the maximum of its categories' similarities. We don't **want** categories to be similar. So, the best solution is the one among all valid solutions that has the lowest value for this measure.)

   Go back to the questions on the previous page and re-answer them with this "goodness" measure.

4 Similar Problems

You're starting to learn more problems and algorithms. Spend 3 minutes trying to brainstorm at least one similar problem. (**Any** problem is fine, not just ones from a textbook or Wikipedia page; your problem could come from our quizzes, lectures, or assignments.)

5 Brute Force?

1. Choose an appropriate variable (or variables!) to represent the "size" of an instance.

2. The set of all valid solutions is the set of partitions of V into c subsets (where c is the requested number of categories). Roughly, how does the number of valid solutions grow asymptotically with the instance size? Polynomially? Exponentially? To help with your thinking, focus on the case $c = 2$.

3. In this problem, you'll likely keep track of the best candidate solution you've found so far as you work through brute force. What will characterize how good a valid solution is?

4. Given a valid solution, how can you determine how good it is? Asymptotically, how long will this take?

5. Will a brute force approach be sufficient for this problem for the domain we're interested in?

6 Promising Approach

There is a **much** better approach.

1. Find the edge in each of your instances with the highest similarity. Should the two photos incident on that edge go in the same category? **Prove** your result.

2. Based on this insight, propose an efficient algorithm to create a categorization.

7 Challenge Your Approach

1. **Carefully** run your algorithm on your instances above. (Don't skip steps or make assumptions; you're debugging!) Analyse its correctness and performance on these instances:

2. Design an instance that specifically challenges the correctness (or performance) of your algorithm:

8 Repeat!

Hopefully, we've already bounced back and forth between these steps in today's worksheet! You usually *will* have to. Especially repeat the steps where you generate instances and challenge your approach(es).

9 Challenge Problems

1. Design an algorithm to directly produce exactly the set of valid solutions, i.e., partitions of the vertices into exactly c non-empty categories.

2. Using the algorithm we created, think of a **principled** way to decide how many categories there should be given no more input than the similarity graph.