COSC 320: Clustering (part 1) *

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

- A bunch of uncategorized photos.
- A *similarity measure* for each pair of photos, where 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.
- The number of categories to group them into.

Your algorithm should create a *categorization*: the requested number of categories, where a category is a non-empty set of photos. Every photo belongs to some category, and no photo belongs to more than one category. So, a categorization is a *partition*. We'd like similar photos to be in the same category.

Step 1: Build intuition through examples.

1. Write down small and trivial instances of the problem. What data structure is useful to represent a problem instance? Write down also potential solutions for your instances. Are some solutions better than others? How so?

Step 2: Develop a formal problem specification

1. Develop notation for describing a problem instance.

2. Use your notation to flesh out the following group of photos into an instance.




3. Develop notation for describing a potential solution. Describe what you think makes a solution *good*. Can you come up with a reasonable criterion for deciding if one solution is better than another?

4. From here on, we'll all use the same definition of "good solution".

   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 *cost* of a categorization is the maximum similarity between any two of its categories. The lower the cost, the better the categorization, since we we don't **want** categories to be similar. So, we want to find a solution with minimum cost. We'll use the term "optimal solution" (rather than "good solution") to refer to solutions that have minimum cost.

   Write down optimal solutions and their costs for your previous examples.

Step 3: Identify similar problems. What are the similarities?

Step 4: Evaluate brute force.

1. A potential solution is the set of partitions of n photos into c subsets (where c is the requested number of categories). Suppose that $c = 2$. Roughly, how does the number of potential solutions grow asymptotically with n? Polynomially? Exponentially?

2. Given a potential solution, how can you determine how good it is, i.e., what is its cost? Asymptotically, how long will this take?

Step 5: Design a better algorithm.

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 a more general result.

2. Based on this insight, come up with algorithmic ideas for creating a categorization.