

3. Draw a reasonable graph for this set of numbered pictures. (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. 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.
6. Hold this space for another instance, in case we need more.

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. Our sketched representation of weighted graphs remains great! Sketch all your instances if you haven't already.
4. 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.

-
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. The set of all valid solutions is the set of partitions of V into c subsets (where c is the requested number of categories). That turns out to be tricky to produce directly (challenge problem!).

Instead, write pseudocode to produce all the "labellings" of the nodes where each node is given a label from $\{1, 2, \dots, c\}$. (This will produce invalid solutions where some categories go unused, which we'll have to filter out. It will also produce duplicates where the same set of categories has simply been renamed. As always, **give a name** to your algorithm and its parameters, *especially* if your algorithm is recursive.)

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.