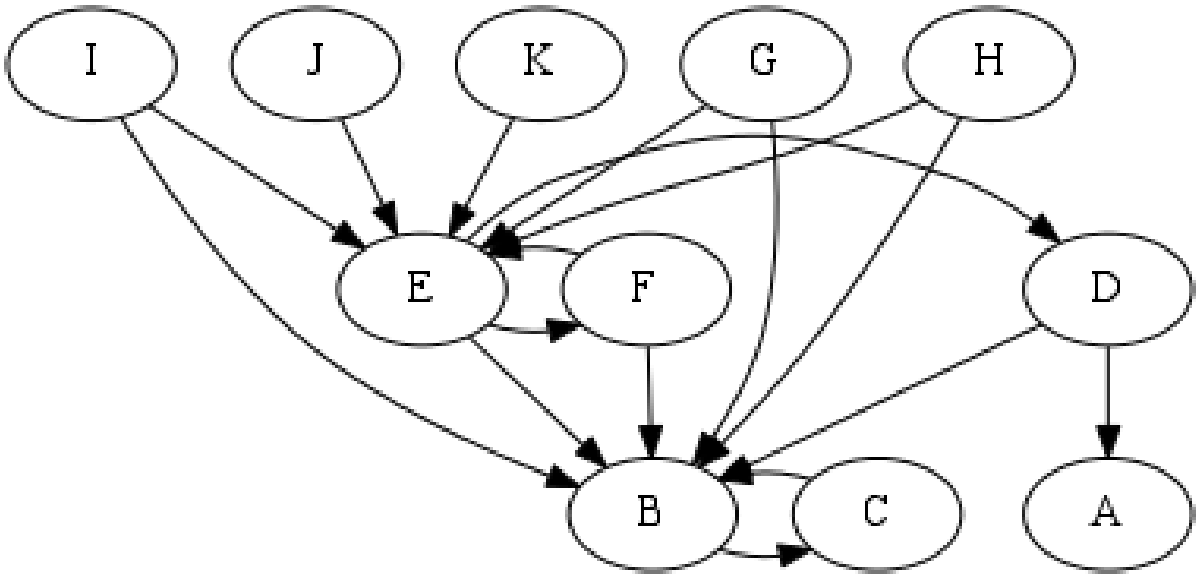
CPSC 320 Notes, PageRank and Clustering

September 28, 2016

1 PageRank

Imagine the following graph represents the "follows" structure of CS department faculty on Twitter:



Discuss these questions with your neighbour:

- Who's the biggest bigwig in the group?
- Who's the second biggest bigwig in the group?
- Which one is a bigger bigwig, A or C?
- How should an algorithm decide?
- Which one is Steve? (Just kidding.)

Now, cut out the following handy-dandy randomizers and follow the algorithm on the back of the page:

A	B	C	D	E	F	G	H	I	J	K	X

1. Repeat until Steve calls time:
   (a) Pick a random person p (among A–K) to start on.
   (b) Put a tick mark next to p.
   (c) Choose a random number 1 through 6. (See note below.)
   (d) While your random number is less than 6:
      i. Choose at random among the people p follows. (See second note below.)
      ii. Make the chosen person your new value of p.
      iii. Put a tick mark next to p (the new p).
      iv. Choose a random number 1 through 6.

Note 1: How should you choose a random number 1 through 6? Well, you can reuse your strips of paper and choose at random among A, B, C, D, E, and X. If you choose X, that's 6. Otherwise, you chose something less than 6.

Note 2: If p doesn't follow anyone, just quit (and return to the line "Repeat until Steve calls time").

1.1 Challenge Problem

What is the expected number of tick marks you write during a single run of this algorithm (i.e., all the steps beneath "Repeat until Steve calls time")?

2 Clustering

You're working on software to manage people's photos.

Your algorithm receives as input: a bunch of uncategorized photos, a number of categories to group them into, 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 list (of non-zero length) of the photos contained in that category. Every photo belongs to some category, and no photo belongs to more than one category. (I.e., 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.

2.1 Trivial and Small Instances

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

2. 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.)

3. The graph you've drawn is **not** an instance of this problem yet because something is missing. Read the specification above carefully, figure out what's missing, and add that missing element. Use the value 3.

4. Write down two more **small** instances of the categorization problem of different sizes. (Try to find the smallest non-trivial ones you can.)

5. Give possible 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 a good one.

6. Hold this space for another instance, in case we need more.

2.2 Represent the Problem

1. Explain in English how we can interpret the input to our algorithm as a graph. (The graph might be directed or undirected, weighted or unweighted; it's up to you!)

2. We've already seen an unweighted, undirected graph represented 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), where $u, v \in V$.

   Modify this to describe the input graph in this problem. Also describe the additional parameter (the desired number of categories).

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

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

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.

2.3 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 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 five-node instance into three categories: $\{1,3\},\{2,4\},\{5\}$ and $\{4,2\},\{5\},\{1,3\}$? 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, 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. (I.e., we are trying to minimize across all potential categorizations the maximum similarity between any two categories in the categorization.)

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

2.4 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.)

2.5 Brute Force?

1. Sketch an algorithm to produce everything with the "form" of a solution. (It will help to **give a name** to your algorithm and its parameters, *especially* if your algorithm is recursive.)

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

3. Exactly or asymptotically, how many such "solution forms" are there? (It will help to **give a name** to the number of solutions as a function of instance size.)

4. 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 possible solution is?

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

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

2.6 Lower-Bound

In terms of instance size, exactly or asymptotically, how "big" is an instance? (That is, how long will it take for an algorithm just to read the input to the problem?)

(We won't do more for now to lower-bound the problem.)

2.7 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.

2.8 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:

2.9 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).

2.10 Challenge Problem

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.