CPSC 320 2022W1: Tutorial Quiz 5, Individual Portion Solutions

November 2022

Dynamic Programming Vs. Zombies

The kingdom of Svenyarnia is about to wage battle against the Zombie King. The Zombie King leads an army of the undead, and he can raise all slain soldiers from the dead to join his ranks. This ability obviously gives him a huge military advantage, but the Svenyarnian strategists want to know just how big this advantage is. In particular, they want to know how big an army they would need to muster in order to have a reasonable chance of defeating the zombies.

The battle proceeds as follows. The soldiers from each army line up in single file. One soldier comes forward from each line and the pair will duel. If the living soldier wins, the zombie is (permanently) killed and the soldier moves to the back of his own line. If the zombie wins, the soldier dies and is instantly resurrected as a zombie and goes to the back of the zombie army's line, along with the victorious zombie. The battle continues until one army is eliminated. At each duel, there is a (constant) probability α between 0 and 1 that the living soldier will defeat the zombie soldier.

On this page, we will consider the probability $P(i, j)$ that a living army of size i will defeat a zombie army of size j.

1. What is the value of $P(1, 1)$ (i.e., the probability that one living soldier will defeat one zombie)?

   If there is one living soldier and one zombie, only one duel will take place, and the probability that the living soldier will win is α. Therefore, $P(1, 1) = \alpha$.

2. What is the value of $P(1, 2)$ (i.e., the probability that one living soldier will defeat an army of two zombies)?

   For one living soldier to defeat two zombies, the soldier has to defeat the first zombie and then the second zombie. (If he loses either duel, then the zombies will have won.) Since we assume that each duel is independent (i.e., the probability of victory for the living soldier is always α), the probability of the living soldier winning the first duel and then winning the second duel is equal to (probability of winning first duel) times (probability of winning second duel).

   Therefore, $P(1, 2) = \alpha^2$.

3. What is the value of $P(2, 1)$ (i.e., the probability that two living soldiers will defeat one zombie)?

   There are two ways that two living soldiers could defeat one zombie. The first way is if the first soldier defeats the zombie in the duel, which has probability α. If the first soldier loses, then it's still possible to defeat the zombie, but now the second soldier will have to defeat two zombies (since the first soldier will become part of the zombie army). The probability that this will happen is equal to (probability that first soldier loses) times (probability that second soldier defeats two zombies), which is $(1 - \alpha)\alpha^2$ (by question 2).

   The two events (first soldier wins) and (first soldier loses and then second soldier defeats the two zombies) are mutually exclusive, so the probability that one or the other will occur is equal to the sum of their respective probabilities.

   Therefore, $P(2, 1) = \alpha + (1 - \alpha)\alpha^2$.

4. Give a recurrence to define $P(m, n)$, which describes the probability that a living army of size m will defeat a zombie army of size n.

   The solution to this question will be given in the solutions to Assignment 5.

5. Give asymptotic bounds on the runtime and memory use of a memoized algorithm to compute $P(m, n)$, in terms of m and n. Justify your answer.

   The solution to this question will be given in the solutions to Assignment 5.

   Note that this question does not require you to actually **write** a memoized algorithm: the space complexity is the size of the table to store the recurrence values, and the runtime is the total amount of non-recursive work to compute all table entries.

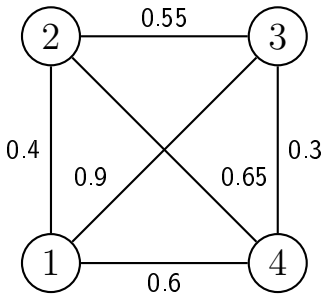
Clustering 2.0

Recall the photo categorization problem described in worksheets 5 and 6. An instance of the problem is given by

- n, the number of photos (numbered from 1 to n);
- E, a set of weighted edges, one for each pair of photos, where the weight is a similarity in the range between 0 and 1 (the higher the weight, the more similar the photos); and
- c, the desired the number of categories, where $1 \leq c \leq n$.

A solution to this problem is a *categorization*, which we define as a partition of photos into c (non-empty) sets, or categories. In the version of this problem we dealt with in class, our goal was to find the categorization that minimized the maximum inter-category edge weight. (Recall that an edge being "inter-category" means it is adjacent to photos in different categories. An edge that is adjacent to photos in the same category is said to be "intra-category.") Suppose now that we change the goal of the photo categorization to **maximize the minimum intra-category edge similarity**. We call this the *Max-Min Clustering Problem* (MMCP).

1. For the 4-node example below and $c = 2$, list all of the optimal MMCP solutions. No justification needed.



   The optimal solution is $\{(1, 3), (2, 4)\}$.

2. Let (n, E, c) be an input where all edges have distinct similarities, and $c \geq 2$. Suppose that (p, p', s) is the lowest-weight edge in E. In an *optimal* MMCP solution, must p and p' be in distinct categories? Justify your answer.

   In an optimal solution, p and p' must be in different categories. Because (p, p', s) is the lowest-weight edge in E', s defines the worst cost (according to this particular metric) of **any** categorization. If

all edge weights are distinct, then **every** solution in which (p, p', s) is inter-category will be better (i.e., have a larger minimum intra-category edge weight) than any solution in which (p, p', s) is intra-category.

3. In class, we saw that an optimal greedy algorithm for minimizing the maximum inter-category edge weight was to initialize all photos in their own category and merge the photos adjacent to the highest-weight intercategory edge until we achieved the desired number of categories. Below is a greedy algorithm for MMCP, which starts with all photos in a single category and separates the photos adjacent to the lowest-weight intracategory edge until we achieve the desired number of categories.

```
function MMCP-Greedy(n, E, c)
▷ n ≥ 1 is the number of photos
▷ E is a set of edges of the form (p, p', s), where s is the similarity of p and p'
▷ c is the number of categories, 1 ≤ c ≤ n
    create a list of the edges of E, in increasing order by similarity
    let C be the categorization with all photos in one category
    Num-C ← 1                ▷ Initial number of categories
    while Num-C < c do
        remove the lowest-similarity edge (p, p', s) from the list
        if p and p' are in the same category S of C then
            ▷ split S into two new categories T and T' as follows:
            put p in T
            put p' in T'
            for each remaining p'' in S do
                put p'' in T if p'' is more similar to p than p'
                put p'' in T' otherwise
            end for
            ▷ now S is replaced by T and T' in C
            Num-C ← Num-C + 1
        end if
    end while
    return C
end function
```

What categorization does MMCP-Greedy produce on the 4-node example given in question 1? No justification needed.

It produces the categorization $\{(1, 3), (2, 4)\}$.

4. Let E' be the set of all edges removed from the list ove all iterations of the **while** loop of MMCP-Greedy, and let C^T be the categorization returned by the algorithm. Which statement is true?

- ● All edges of E' must be inter-category in C^G
- ○ All edges of E' must be intra-category in C^G
- ○ Edges of E' could be either intra-category or inter-category in C^G

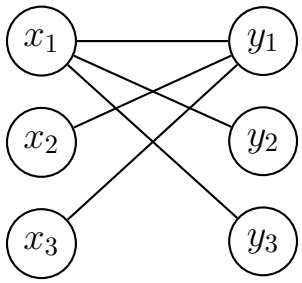
Greater Than or Equal to Three's a Crowd

Consider the following SMP-like problem. You're in charge of matching up n employers and n computer science co-op students for summer internships. Instead of having ranked preference lists, each employer e has a list of students s they're willing to work with and vice versa. You want to determine if it's possible to generate a perfect matching in which everyone is paired with someone they're willing to work with. We call this the *CO-OP2 Problem*.

For example, if $n = 2$ and e_1 is willing to work with s_1 or s_2, e_2 is willing to work with s_2, s_1 is willing to work with e_1, and s_2 is willing to work with e_1 or e_2, this is a YES instance (because we can pair e_1 with s_1 and e_2 with s_2). However, if we modify the instance so that s_2 is only willing to work with e_1, then this becomes a NO instance.

Recall that a *bipartite graph* $G = (V, E)$ is an undirected graph whose node set can be partitioned as $V = X \cup Y$, with the property that ever edge $e \in E$ has one end in X and the other end in Y. A *matching* M in G is a subset of the edges $M \subseteq E$ such that each node appears in at most one edge in M. The *Bipartite Matching Problem* (also discussed in Section 7.5 of your textbook) is that of finding a matching in G of largest possible size.

For example, in the bipartite graph below, a (non-unique) largest matching is $M = \{(x_1, y_2), (x_3, y_1)\}$.



1. Complete the following reduction from CO-OP2 to Bipartite Matching.

   REDUCTION: Given an instance of CO-OP2 with n employers $e_1, e_2, \ldots, e_n$ and n students $s_1, s_2, \ldots, s_n$, create a bipartite graph where:

   There are [n] vertices on the left side of the graph and [n] vertices on the right.

   Each vertex on the left side of the graph, x_i, represents [the employer e_i]

   Each vertex on the right side of the graph, y_j, represents [the student s_j]

   For each x_i and each y_j, we add an edge (x_i, y_j) if:

   [The employer e_i is willing to work with student s_j, and s_j is willing to work with e_i.]

   Now, solve the Bipartite Matching instance. The answer to the CO-OP2 problem is YES if and only if the returned matching has size [n].

2. Explain why your reduction from question 1 runs in polynomial time. (Note that when we refer to the time complexity of a reduction from problem A to problem B, we are referring to the time complexity all parts of the reduction **except** the time to solve the reduced instance(s) of B.)

   Defining the vertices of the graph for Bipartite Matching takes $O(n)$ time. Defining the edges will take $O(n^2)$ time in the worst case (since we need to go through n lists of employer/student preferences, each of which has length at most n). Converting the Bipartite Matching solution back to the answer to CO-OP2 takes constant time. Therefore, the reduction takes $O(n^2)$ (and therefore polynomial) time total in the worst case.