Data Mining

Homework 2

Due: 11/12/2016, 23:59.

Instructions—Read carefully!

You must hand in the homeworks electronically and before the due date and time.
You should do the homeworks in the groups you have formed.

Handing in: You must hand in the homeworks by the due date and time by an email to Aris that will contain as attachment (not links!) a .zip or .tar.gz file with all your answers and subject
`[Data Mining class] Homework #`
where `#` is the homework number. In the text you must also mention the team members. After you submit, you will receive an acknowledgement email that your homework has been received and at what date and time. If you have not received an acknowledgement email within 2 days after you submit then contact the Aris.

The solutions for the theoretical exercises must contain your answers either typed up or hand written clearly and scanned.

The solutions for the programming assignments must contain the source code, instructions to run it, and the output generated (to the screen or to files).

For information about collaboration, and about being late check the web page.

Most of the questions are not very hard but require time and thought. **You are advised to start as early as possible, to work in groups, and to ask the instructor in case of questions.**

Problem 1. We mentioned in class that citation networks (networks representing citations between research articles) are, in theory, *directed and acyclic graphs* (DAGs).

1. Explain why we expect citation networks to be directed and acyclic.
2. However, in practice, we expect that citation networks have a small number of cycles. Why?
3. Assume that we want to test whether a given citation network has cycles. Propose an algorithm to detect if a directed graph has a cycle. What is the running time of your algorithm?

Problem 2. Assume that a graph is stored in the *edge incidence* model, that is, all edges incident to one vertex are stored sequentially.

Our sampling algorithm for estimating the triangle coefficient relies on sampling one connected triple of the graph with *uniform probability*. Namely, all connected triples should have *equal* probability to be sampled.

Describe a three-pass algorithm to sample a connected triple uniformly at random.

Note: This is already outlined in the slides, you are asked to describe the method in more detail.

Problem 3. Describe an algorithm to compute *exactly* the *diameter* of a given graph. What is the running time of your algorithm?

Describe an algorithm that uses the Flajolet-Martin sketching scheme to compute *approximately* the *diameter* of a given graph. What is the running time of your algorithm? How can you control the trade-off between accuracy and speed?

Problem 4. We are monitoring a graph, which arrives as a stream of edges $\mathcal{E} = e_1, e_2, \ldots$. We assume that exactly one edge arrives at a time, with edge e_i arriving at time i, and the stream is starting at time 1. Each edge e_i is a pair of vertices (u_i,v_i), and we use V to denote the set of all vertices that we have seen so far.

We assume that we are working in the *sliding window* model. According to that model, at each time t only the w most recent edges are considered *active*. Thus, the set of active edges $E(t,w)$ at time t and for window length w is

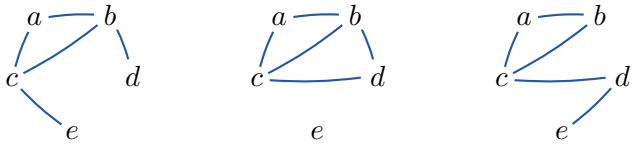
$$E(t,w) = \begin{cases} e_{t-w+1}, \ldots, e_t, & \text{if } t > w, \\ e_1, \ldots, e_t, & \text{if } t \leq w. \end{cases}$$

We then write $G(t,w) = (V,E(t,w))$ to denote the graph that consists of the active edges at time t, given a window length w.

As an example, given the stream of edges

$$e_1 = (c,e), e_2 = (b,d), e_3 = (a,c), e_4 = (c,b), e_5 = (a,b), e_6 = (c,d), e_7 = (d,e)$$

the graphs $G(5,5)$, $G(6,5)$ and $G(7,5)$ are shown below:



$G(5,5)$ $G(6,5)$ $G(7,5)$

Notice that for all $t \geq w$ the graph $G(t+1,w)$ results from $G(t,w)$ by adding one edge and deleting one edge.

We want to monitor the connectivity of the graph $G(t,w)$. In other words, we want to design an algorithm that quickly decides, at any time t, if the graph $G(t,w)$ is connected. In the previous example, the graphs $G(5,5)$ and $G(7,5)$ are connected, while the graph $G(6,5)$ is not connected.

1. Propose a streaming algorithm for deciding the connectivity of $G(t,w)$.

   Hint: An efficient streaming algorithm takes advantage of the fact that the graph $G(t+1,w)$ changes very little compared to $G(t,w)$. Therefore, our algorithm should be able to efficiently update the connectivity of $G(t+1,w)$ when a new edge e_{t+1} arrives at time $t+1$, given that the connectivity of $G(t,w)$ has already been computed.

2. How much space does your algorithm use? Try to design an algorithm that manages to use $O(n)$ space, where $n = |V|$.

3. What is the update time of your algorithm?

   Hint for 2 and 3: The space of your algorithm is the maximum amount of space used at any given moment. The update time is the time needed to compute the output at time $t+1$, given the state at time t and the new edge e_{t+1} that arrives at time $t+1$.

   You should provide your answer using the $O(\cdot)$ notation, written as a function of the window length w and the number of vertices n.

Problem 5. Here we are asking to implement nearest-neighbor search for text documents. You have to implement shingling, minwise hashing, and locality-sensitive hashing. We split it into several parts:

1. Implement a class that, given a document, creates its set of character shingles of some length k. Then represent the document as the set of the hashes of the shingles, for some hash function.

2. Implement a class, that given a collection of sets of objects (e.g., strings, or numbers), creates a minwise hashing based signature for each set.

3. Implement a class that implements the locally sensitive hashing (LSH) technique, so that, given a collection of minwise hash signatures of a set of documents, it finds the all the documents pairs that are near each other.

To test the LSH algorithm, also implement a class that given the shingles of each of the documents, finds the nearest neighbors by comparing all the shingle sets with each other.

We will apply the algorithm to detect whether people post double information. We will work with the recipes of the Homework 1.

We want to find recipes that are near duplicates. We will say that two recipes are near duplicates if the Jaccard coefficient of their shingle sets is at least 80%. We will use shingles of length 10 characters. Find values for r and b (see Section 3.4 in the book) that can give us the desired behavior. To plot the graph that gives the probability as a function of the similarity for different values of r and b you can use, for example, Wolfram Alpha.

To apply the algorithm you have the following tasks:

1. Find the near-duplicates among all the recipes of Homework 1 using LSH.

2. Find the near-duplicates among all the announcements of Homework 1 by comparing them with each other.

3. Report the number of duplicates found in both cases, and the size of the intersection.

4. Report the time required to compute the near duplicates in either case.

You will need a way to create a family of hash functions. One way is to use a hash function and a code similar to the following.

```
# Implement a family of hash functions. It hashes strings and takes an
# integer to define the member of the family.
# Return a hash function parametrized by i
import hashlib
def hashFamily(i):
  resultSize = 8        # how many bytes we want back
  maxLen = 20           # how long can our i be (in decimal)
  salt = str(i).zfill(maxLen)[-maxLen:]
  def hashMember(x):
    return hashlib.sha1(x + salt).digest()[-resultSize:]
  return hashMember
```

Note that this code is an overkill because we use a cryptographic hash function, which can be very slow, even though it is not needed to be as secure. However, for the necessities of the homework we will use it to avoid having to install some external hash library.

After you implement LSH apply it on the recipes data that you have downloaded. If you have not downloaded the full description ad (the optional part of the first homework) you will need to do it now. Report all the sets of ads that where duplicate, and the Jaccard similarity of their representation as sets.