Homework 6

CS 4364/5364
Spring 2022

Due: 6 April 2022

Because of the reliance of the particular assignments in this class on mathematical notation, and the fact that all assignments will be submitted electronically, students are encouraged to use LaTeX to formalize their responses. **For those enrolled in the graduate section the use of latex is *required*.** This assignment (like all others) will be posted on the course `github`[1] as source code as well as in PDF form on the course website. Graduate students will need to include the `.tex` files as well as a PDF, this is optional but encouraged for undergraduates.

Question (35 points): **Give an algorithm, a proof of correctness, and proof of running time to solve the following problem in $O(kn^2c)$-time.**

Given integers $k \geq 1$ and $c \geq 1$, as well as two strings S and T both of length $n \gg k$, determine a similarity between them that is calculated as the sum of:

- **$c+1$ times the number of shared distinct exact k-mers**
- **c times the maximum number of shared distinct k-mers with one change that have not been accounted for already**
- **$c-1$ times the maximumnumber of shared distinct k-mers with two changes that have not been accounted for already**
- **...**
- **1 times the maximum number of shared distinct k-mers with c changes that have not been accounted for already**

normalized by the total number of distinct k-mers in both sequences. Here changes are replacements only and not indels.

As an example for the values of $k = 3, c = 2$ and the strings `AACTGT` and `TGTAAA` the similarity is $\frac{3}{4}$. The distinct 3-mers from the two strings are `AAC,ACT,CTG,TGT` and `TGT,GTA,TAA,AAA`. There is one 3-mer that matches exactly (`TGT`). There is one pair of 3-mers that match with one change (`AAC` and `AAA`). There is one pair of 3-mers that match with two changes (`CTG` and `GTA`) The remaining pair would need 3 changes (i.e. more than c changes). Thus the similarity is

$$\frac{(1 \times 3) + (1 \times 2) + (1 \times 1)}{8} = \frac{3}{4}.$$

[1] `github.com/deblasiolab/CS4364-documents`

Algorithm

1. Let $\mathcal{S}$ be a set, that will eventually contain all of the distinct k-mers from S.
2. Define the current k-mer as $C_S = s_1 s_2 ... s_k$ and a counter $c_S = k$
3. If $C_S \notin \mathcal{A}$, add it ($\mathcal{A} = \mathcal{A} \cup \{C\}$).
4. Update C_S and c to the next k-mer ($c_S + +; C_S = C_S[2...k]s_{c_S}$).
5. If $c_S \leq n$, go to Step 3.
6. Follow steps above to similarly construct $\mathcal{T}$ from T.
7. Let the total number of k-mers in both sets be $tk = |\mathcal{S}| + |\mathcal{T}|$.
8. Let the current allowable k-mer difference $d = 0$.
9. Let the final return value $r = 0$.
10. Create an empty graph G.
11. Create 2 1-dimensional boolean arrays U_S and U_T of length $|\mathcal{S}|$ and $|\mathcal{T}|$ respectively.
12. $\forall s \in \mathcal{S}, t \in \mathcal{T}$, calculate the hamming distance between s and t, if it is equal to d:
    - Add s and t as vertices to G if $U_S[s]$ and $U_T[t]$ are false respectively.
    - Add the edge (s, t) to G.
    - Set $U_S[s]$ and $U_T[t]$ to true.
13. Solve the maximal-cardinality matching problem on G.
14. For each pair of matched vertices s and t:
    - Remove the k-mers from the sets ($\mathcal{S} = \mathcal{S} \setminus \{s\}, \mathcal{T} = \mathcal{T} \setminus \{t\}$).
    - Update the return $r+ = (c - d + 1)$.
15. Update $d = d + 1$.
16. If $d \leq c$ go to Step 11.
17. Return r/tk.

Correctness

Steps 1 to 6 construct the list of unique k-mers from the strings. Since $\mathcal{S}$ and $\mathcal{T}$ are unordered sets, they will be unique in the sets of k-mers they contain. By using a known maximal-cardinality matching to find the subset of k-mers that provides the highest score we can ensure we will find the most matches at each value of d. The algorithm will only look for matched k-mers up to hamming distance c because of the check in Step 16. k-mers can only be matched once because in order for them to be matched they have to first be a node in a graph at some iteration, then be in a matching. If this is true, then by Step 14 they will be removed from $\mathcal{S}$ ($\mathcal{T}$) and thus cannot be in a graph is later iterations. Also, because each vertex in the graph cannot be in more than one matching a k-mer cannot be matched with more than one other in a single iteration.

Running Time

Step 2 takes $O(k)$-time. Step 3 takes $O(|\mathcal{S}|)$ time each iteration, and because after Step 4 this

increases by 1, and it can be repeated n times from Step 5, the total running time of these 3 steps is $O(n^2)$. Similarly Step 7 is also $O(n^2)$. Step 11 can take at most $O(n)$ time, tough the two sets are constantly decreasing. Calculating the hamming distance in Step 12 will take $O(k)$ time for each pair of k-mers, of which there are at most n^2, therefore this step is $O(kn^2)$. Because we keep a boolean array for each k-mer we don't need to search to determine if its already in the graph, thus the rest of Step 12 will take constant time for each possible k-mer pair. We know that in a graph with at most $2 \times n$ vertices, and n^2 edges the running time of a maximum cardinality matching algorithm such as Hopcroft-Karp will be $O(n\sqrt{n^2}) = O(n^2)$. Step 14 will deal with at most n matches, and processing each one will take constant time to deal with. Finally, because of Step 16 we will run Steps 11 to **??** st most c times; since Step 12 dominates the running time of all of these, we see the total running time of the loop is $O(ckn^2)$ time which dominates all of the steps before the loop.