

Assignment Goals

- i. Understand how to work with epigenome browsers and interpret ATAC-seq data (similar idea as DNase I Hypersensitivity sites) on signal peaks.
- ii. Resolve read mapping uncertainty in RNA-Seq quantification.
- iii. Compare and contrast algorithms for finding paths in interaction networks.
- iv. Get familiar with the workflow of machine-learning modeling.

Submission Instructions

- To turn in your assignment, please log in to the server **pluto.biostat.wisc.edu** using your **pluto** username and password.
- Copy all relevant files to the directory

`/home/medinfo /bmi776-2022/hw3/<USERNAME>`

where **<USERNAME>** is your pluto (biostat) username. Please submit all of your Python source code and test that it runs on the pluto server.

- For the rest of the assignment, compile all your answers in a single file and submit as **solution.pdf**.
- Write the number of late days you used at the top of **solution.pdf**.
- For the written portions of the assignment, show your work for partial credit.

Part 1: Using an Epigenome Genome Browser (30 points)

In class, we discussed DNase I Hypersensitivity Sites (DHSs) and looked at peaks in different chromosome regions, and learned how those could indicate TF binding sites, and oftentimes overlap with ChIP-seq peaks on regulatory elements (e.g. enhancers and silencers). Similarly, there are different sequencing technologies available to help annotate the functions of DNA regions; in this problem we will look at ATAC-seq (Assay for Transposase-Accessible Chromatin using sequencing) epigenetics data, which is a popular and emerging method in molecular biology to detect open chromatin accessibility, especially at the cell-type level. The peaks of ATAC-seq data indicate open chromatin accessibility.

In particular, this part will utilize the “Corces_scATAC_BroadCellTypes” ATAC-seq data on the Washington University [Epigenome Browser](#) to identify which ATAC-seq signal peaks tend to be specific to which cell types in the brain. When you click the link above, you should see a few different options for tracks, but please select “Corces_scATAC_BroadCellTypes”(13 tracks). By right-clicking on the tracks, you can configure various options.

For instance, if you are given this region here, on Chromosome 19: 6799564 to 6804085 base pairs: chr19:6799564-6804085, based on the Washington Epigenome Browser, you would enter this chromosome information and region information in the blue box (near the + and – buttons in the middle of the top of the screen) above the chromosome sketch.



Then, please keep zooming out (using -1/3 or -5) till you discover an appropriate peak. You can configure the y-axes (on the left by the cell-type labels), by right clicking. For instance, all y-axes could be configured to have a height of 200 (instead of the default of 400). Please adjust as needed, and be consistent for all cell-types.

Hence, please note that example outputs for this region would be:

1. the link to the Epigenome Browser session:
<http://epigenomegateway.wustl.edu/legacy/?genome=hg38&session=drS3o1n4kj>
2. Screenshot of the Epigenome Browser with the session ID on the top right in red (that should be different for each user, which we will verify) (Figure 1A). Please ensure you adjust and zoom out the browser as needed to see the peaks. The yellow highlighted region corresponds to the chr19:6771658-6774320 region and zooming out shows how it fares relative to other regions along Chromosome 19.
3. Interpretation of the results: what cell-type has the strongest signal peak and which gene(s) are close to the peak.
4. Show a footprint of the peak (Figure 1B)

For example, we can see that chr19:6771658-6774320 corresponds to a signal peak in Microglia (the resident immune-system cell of the brain), and VAV1 is the closest gene, implying its potential microglial functions.



Figure 1 A chr19:6771658-6774320 shows a peak for Microglia cell-types, and this region is near VAV1 gene, which has immune-related functions.



Figure 1 B: Footprint Figure for chr19:6771658-6774320

Similarly, in this problem, you will be given various chromosome regions, and will be asked to do the same for each of 3 outputs (in (A)); in (B), you will do outputs 1 to 4 for 1 region.

(A) Please generate the peak signals (using Washington Epigenome Browser) for these given chromosome regions. Please specify which chromosome region has a peak specific to which cell-type. Please identify any genes associated with the region. And, provide

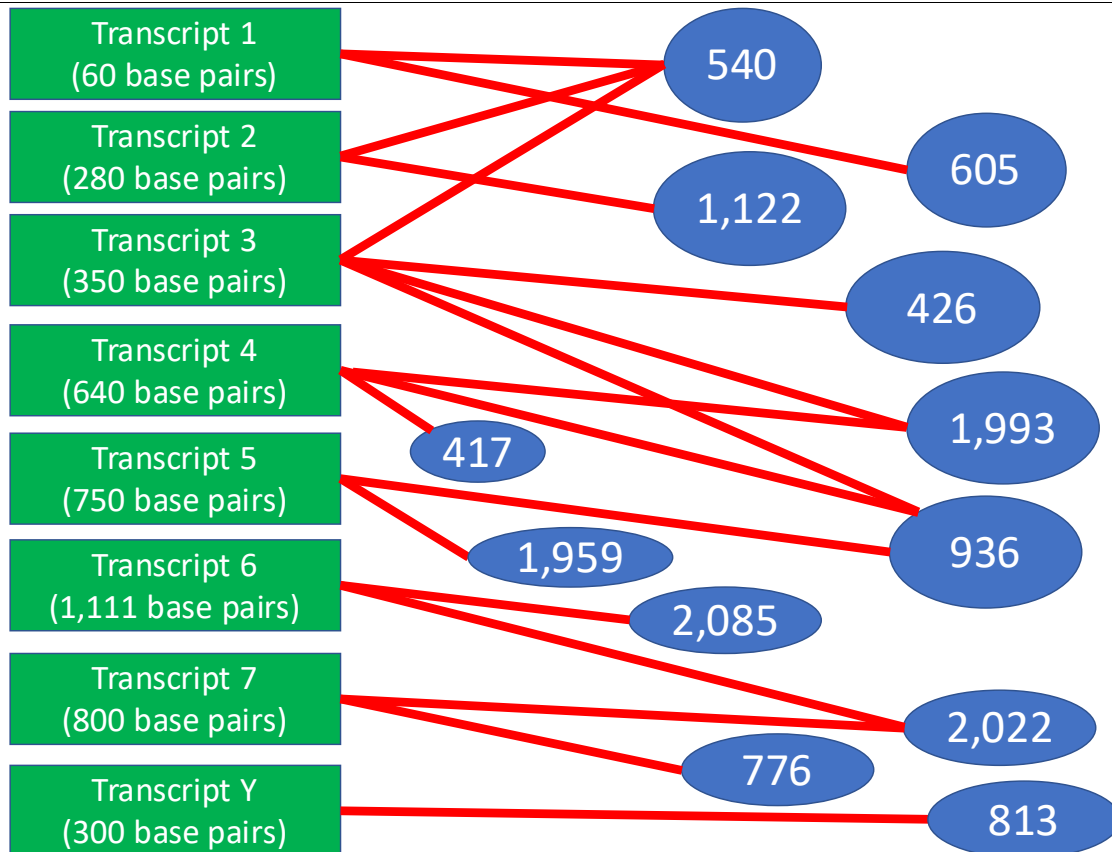
screenshots and URLs for each of the regions. That is, please output #1 to #3 from above. Please see the demo above for an example. **(20 points)**

chr#:start-end	Cell-type(s) with strongest peaks	Closeby gene(s)
chr1:109903217-109904779		
chr4:38331736-38338057		
chr7:152646-156292		
chr18:9848495-9851756		
chr5:11590287-11598674		

(B) Select a cell-type peak from chromosome 6 bases 15367922 to 15368482 and zoom in enough to visualize a footprint pattern. Please output #1 to #4 from above. Please refer to Figure 1B for an example (and please output # 1 to 4). **(10 points).**

Part 2: RNA-Seq Rescue Algorithm (35 points)

The full RSEM algorithm is too complicated to execute manually, but we can use the RNA-Seq rescue method to approximate one iteration of expectation-maximization. The bipartite graph below contains two types of nodes: transcripts and read groups. The transcript nodes contain a transcript ID and the transcript length in base pairs (bp). The read nodes contain the read counts for a group of reads that all align to the same transcripts. The edges designate the transcripts to which each read group aligns.



(A) Use the rescue method to calculate the *relative* abundance for the 8 transcripts. (20 points)

(B) Transcript Y is an RNA spike-in. 900 copies of transcript Y were mixed into the experimental sample when preparing the sample for RNA-Seq, meaning its absolute abundance is 900. Use the relative abundance from (A) to calculate the *absolute* abundances for the other 7 transcripts, rounded to the nearest integer. (15 points)

Part 3: Source-target paths in networks (35 points)

We will use the networkx Python package to compare and contrast two algorithms for finding source-target paths in a network. One algorithm optimizes the min-cost flow similar to ResponseNet. The other finds the k shortest weighted paths. In both cases, you are given an undirected network where each line in the input file lists a pair of nodes followed by its weight, which is interpreted as the cost of transmitting flow by the min-cost flow algorithm. The networkx flow algorithms require directed graphs, so we represent an undirected edge as a pair of directed edges with the same weight. In addition, you are provided with a list of source and target nodes. These nodes will be connected to an artificial source and an artificial target, as in ResponseNet.

Our objective is to find connections from the artificial source to the artificial target. Your task is to finish and test a mostly complete program, **find_paths.py**, which implements the min-cost flow and shortest path-based algorithms.

The networkx package is installed on the pluto server. The program is callable from the command line as follows:

```
python find_paths.py --edges=<edges> --flow=<flow> \  
--sources=<sources> --targets=<targets> --out=<out>
```

or

```
python find_paths.py --edges=<edges> --k=<k> \  
--sources=<sources> --targets=<targets> --out=<out>
```

where

- **<edges>** is a text file listing weighted undirected edges one per line.
- **<sources>** is a text file listing source nodes one per line.
- **<targets>** is a text file listing target nodes one per line.
- **<out>** is the name of the text file into which the program will print the identified source-target paths.
- **<flow>** is a positive number specifying the amount of flow to send from the artificial source to the artificial target.
- **<k>** is a positive integer specifying the number of shortest paths to find.

(A) (Path-finding implementation) Please complete the five code segments marked as TODO in **find_paths.py**. The networkx [documentation](#) will be useful for learning how it represents the [graph](#) data structure and implements the [min-cost flow](#) and [shortest path](#) based path-finding algorithms. Please use the example input files

example_graph.txt, **example_sources.txt** and **example_targets.txt** to test your code. When **find_paths.py** is run with **--flow=3**, you should obtain **example_paths_flow.txt** or the equally good **example_paths_alt_flow.txt**. When it is run with **--k=7** you should obtain **example_paths_shortest.txt**. **(15 points)**

- (B)** Please try out 2 different examples of flow values and 2 different k values and please analyze the resulting networks. You may use the provided **print_graph** and the networkx **draw** functions to visualize the graphs. Please draw the graph (using the Python functions or by hand) and provide the graph as part of your answer to this question. How can the graph explain the intuition behind the results? Please show the minimum cost flow path in the graph (from (A)). **(10 points)**.
- (C)** (*Min-cost flow vs. k shortest paths*) Based on the networkx documentation and your own experiments, discuss the strength and weakness of each method. To begin with, you may examine how certain edges 2-5 and 5-11 are used in the flow- and shortest path-based solutions in your test output. **(5 points)**
- (D)** (*Special case*) So far we have used infinite capacity on the edges incident to the artificial source and target, and a capacity of one on all real edges in the network. Describe a way to change the capacities such that the min-cost flow algorithm will return essentially the same solution as k shortest paths for some value of k . What value of k is relevant for this special case? **(5 points)**