

### Assignment Goals

- i. Implement the workflow of peptide-spectrum matching in the analysis of tandem mass spectrometry data.
- ii. Compare and contrast algorithms for finding paths in interaction networks.
- iii. Leverage epigenetic information to predict enhancer targets.
- iv. Use Gaussian processes to model biological time series.

### Submission Instructions

- To turn in your assignment, log in to the server **mi1.biostat.wisc.edu** or **mi2.biostat.wisc.edu** using your BMI (biostat) username and password.
- Copy all relevant files to the directory

**/u/medinfo/handin/bmi776/hw3/<USERNAME>**

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

- For the rest of the assignment, compile all of 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: Peptide-spectrum matching (50 points)

SEQUEST, now known as Comet, remains one of the most widely used tools for automatically matching peptide tandem mass spectra to database sequences. It has become an integral part of numerous computational tools for interpreting mass spectrometry based proteomic data. At the heart of the matching algorithm is the cross-correlation (xcorr) score, which measures the degree of similarity between an experimentally acquired spectrum and the theoretical spectrum constructed from a candidate peptide in the library. Here, we will implement the central component of the fast SEQUEST algorithm to identify peptide-spectrum matches (PSM).

Write a program, **psm.py**, that takes as input a batch of acquired spectra and a target peptide library, and outputs the list of PSMs that achieve the desired FDR. The full algorithm can be conceptually broken down into four steps:

#### **(A) Generate theoretical spectra for target and decoy peptides (Eng et al., 1994).**

Given a peptide sequence, your program should calculate the mass-to-charge ratio ( $m/z$ ) for all distinct  $b$ - and  $y$ -ions, and assign appropriate relative intensity (referred to as intensity hereafter) based on the following rules:

- a) Assume that all  $b$ - and  $y$ -ions carry a single positive charge (i.e.,  $z=1$ ). In practice, doubly charged ions are considered for input spectra with precursor charge 3 or higher.

- b) Discretize product ion  $m/z$ 's into bins of width 1 Da. For example, an ion of  $m/z$  688.6 falls into the 688<sup>th</sup> bin. In practice, the optimal bin width is 1.0005 Da for low-resolution tandem mass spectrometry.
- c) Assign intensity 50 to all  $b$ - and  $y$ -ions, and 25 to bins 1 Da away. For example, if a  $b$ -ion has  $m/z$  242.3, then the 242<sup>nd</sup> bin has intensity 50, and the 241<sup>st</sup> and 243<sup>rd</sup> bins have intensity 25.
- d) Assign intensity 10 to  $m/z$ 's representing the *neutral* loss of ammonia ( $\text{NH}_3$ , MW: 17) from all  $b$ - and  $y$ -ions, and those representing the neutral loss of water ( $\text{H}_2\text{O}$ , MW: 18) or carbon monoxide ( $\text{CO}$ , MW: 28) from a  $b$ -ion. These ions are called NL ions, and a  $b$ -ion missing a CO is an  $a$ -ion.
- e) If an  $m/z$  is assigned more than one intensity value, pick the largest one.
- f) The spectrum should cover the entire mass range of the peptide and its length (i.e., maximum  $m/z$ ) should be extended to multiples of 300. For example, a peptide of mass 1071.5 Da should give rise to a spectrum of length 1200 units.

Construct decoy peptides by reversing target peptide sequences. The order of amino acids in each target peptide is reversed, except that lysine (K) and arginine (R) at the C-terminus is kept in place to preserve the characteristic of trypsin digestion. For example, the target peptide DLSWTK will spawn a decoy peptide TWSLDK.

- (B) Preprocess input spectra (Eng et al., 2008).** Given an input spectrum, your program should first bucket the intensity peaks into bins of width 1 Da as in (A). Each bin only retains the highest peak that fell into it, whose intensity is then replaced by its square root. As in (A), the spectrum length is extended to multiples of 300. Next, divide the spectrum into 10 equal-sized sections and scale the intensity within each section proportionally such that the highest intensity has value 50. If a section contains no peaks, leave it as it is. Finally, compute the query spectrum  $y'$  as defined in the fast SEQUEST algorithm with the offset  $\tau$  ranging from -75 to 75 (excluding 0).
- (C) Score PSMs.** Given a query spectrum  $y'$ , calculate the xcorr score with respect to each peptide in the target-decoy library whose mass is within 1.25 Da of the *neutral* precursor mass associated with  $y'$ . Identify the *best* scoring PSM for each query spectrum.
- (D) Report PSMs that achieve the desired FDR.** Given the best scoring PSMs, calculate an xcorr score threshold such that the hits above it achieve the desired FDR. In practice, E-value is a better score for controlling FDR, but its calculation is more involved. Report all PSMs that score above the threshold.

Note that you do not have to implement the steps above sequentially. Instead, for example, you may obtain the query spectrum  $y'$  first, and construct the theoretical spectra on the fly while computing xcorr scores.

Your program should be callable from the command line as follows:

```
python psm.py --lib=<lib> --fdr=<fdr> --out=<out> <dataset>
```

where

- **<dataset>** is the name of an MS2 file containing the acquired spectra.
- **<lib>** is the name of a FASTA file containing the target peptides.
- **<fdr>** is the FDR we wish to achieve. We use 0.05 as the default value.
- **<out>** is the name of the text file into which the program will print the list of PSMs that achieve the desired FDR. See the example output for the formatting details.

A typical entry in a FASTA file containing the target peptides is

```
>YLR371W ROM2 SGDID:S000004363, ..., which also encodes a  
GEP" _6_Trypsin_93_18_1905.02038_3.47  
SGVEAAIDDSDIPNNEMK
```

where **1905.02038** is the *neutral* peptide mass, and **SGVEAAIDDSDIPNNEMK** is the peptide sequence. You can safely disregard other elements in the header.

A typical input spectrum in an MS2 file is

```
S    10    10    636.34  
Z     2    1271.67  
187.4 12.5  
193.1 19.5  
...
```

where **10** is the scan number, **636.34** is the precursor  $m/z$  ( $[M+2H]^{2+}/2$  in the example), **2** is the precursor charge, **1271.67** is the predicted  $[M+H]^+$ , and the lines below are **<m/z, intensity>** pairs in the spectrum (W. Hayes et al., 2004). Note that you can easily infer the *neutral* precursor mass  $M$  from the Z line. There could be more than one Z line present before the spectrum data that gives distinct precursor charge and predicted mass. In that case, you should match the spectrum *separately* to candidate peptides from different mass ranges, generating one best PSM per range. You can safely disregard the lines that begin with H and I (not shown in the example above).

Example input files **queries.ms2**, its corresponding output file, the target peptide library **library.fasta**, and the template **psm.py** with argument parsing code can be

found in the **hw3\_files** directory. Your program will be evaluated on the example inputs and additional datasets that will be kept private.

## **Part 2: Source-target paths in networks (25 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 biostat 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*) 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. You may use the provided `print_graph` and the networkx `draw` functions to visualize the graph, and 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`.

Test your code on the input files `test_graph.txt`, `test_sources.txt` and `test_targets.txt`. Run `find_paths.py` with `--flow=3` and store the results in `test_paths_flow.txt`. Then, run it with `--k=8` and store your results in `test_paths_shortest.txt`. (15 points)

- (B) (*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 the edges 2-5 and 5-11 are used in the flow- and shortest path-based solutions in your test output. (5 points)
- (C) (*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)

### Part 3: Enhancer targets (10 points)

DNA looping can bring proteins bound to enhancer sites into contact with proteins bound to promoter sites, enabling the enhancer-bound proteins to influence the transcription of genes that are up to one million base pairs away. Hi-C data provide a direct way to detect *enhancer targets*, that is, distant genes that may be influenced by an enhancer site. Unfortunately, one is often constrained to work with the data that are available rather than those that would be ideal for addressing a biological question. Here, we will explore how to use other types of data to predict enhancer targets.

For *one* of the two scenarios below, outline an algorithm that predicts enhancer targets. The outline should focus on the major steps and does not need to contain full algorithmic details.

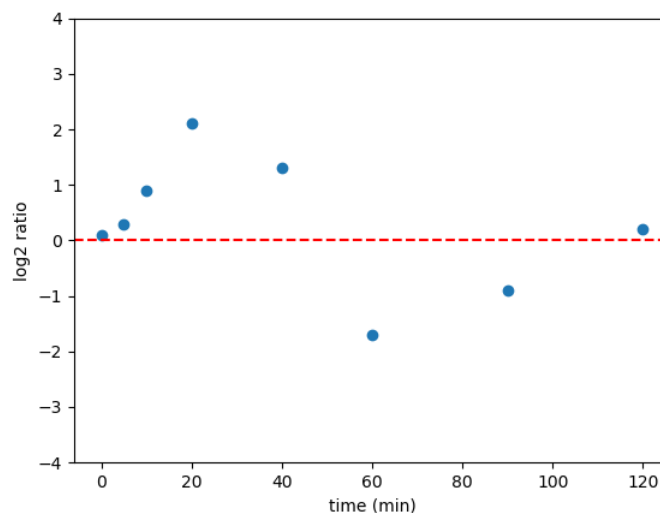
- (A) (*Genetic variation*) Suppose you have data from a resource like GTEx that provides genotypes and tissue-specific gene expression for a large number of individuals.

- (B) (*Histone modification*) The histone modification H3K27ac indicates active enhancer sites. Suppose you have H3K27ac and gene expression data from a resource like ENCODE that covers hundreds of cell lines.

**Part 4: Gaussian processes for biological time series (15 points)**

Suppose we are studying how neural stem cells respond to an environmental toxin. We perform RNA-Seq on cells in the control and treatment groups to obtain gene expression data at 0, 5, 10, 20, 40, 60, 90 and 120 min. Gaussian processes (GP) with a squared exponential kernel are well suited for modeling biological data collected over a time course. Following GP regression, the posterior mean is smooth over time, and the confidence intervals track uncertainty between the measured time points.

- (A) Shown below is a time series of  $\log_2$  fold change of gene expression (i.e.,  $\log_2$  ratio between treatment and control). Assume a GP prior with zero mean and a squared exponential kernel. Sketch the mean and 95% confidence interval of the GP posterior (note the irregular time intervals). You may further assume that the kernel parameters (i.e., length scale  $l$  and variance  $\sigma^2$ ) have been optimized to maximize the data likelihood. Explain your reasoning. (5 points)



- (B) Differential expression analysis and clustering are the natural first steps toward understanding gene functions. Describe a GP-based statistical test that can be applied separately to each gene to assess whether its temporal expression profile in the normal condition differs from that under drug exposure, or a GP-based probabilistic model that clusters genes by their temporal expression profiles to reveal shared biological functions. You are encouraged (but not required) to work with the  $\log_2$  ratios as in (A). Clearly state and justify your test/model formulation as well as your assumptions. (10 points)