CS 5046 Final Project Specification (Spring 2007)

T. M. Murali

April 10, 2007

This document specifies the final project for CS5046. I expect that the amount of effort required to finish the project corresponds to two–three homework assignments. Your project should use the object-oriented principles we have discussed in class. Make sure that you design classes appropriate to the specification. I anticipate that you will need to implement anywhere from three to five classes for this project. You can use `Class::Struct` to implement your classes. You can also use the more sophisticated and powerful `Class::MethodMaker`, if you want to. If you implement some non-object-oriented functionality, add a detailed comment that justifies your choice. The project is divided into two steps. You can write a single script to perform both steps, one after another. You can modify classes we have implemented during the lectures.

1. In this step, you have to compare a control data set and a treatment data set to find the list of genes up-regulated in the treatment and the list of genes down-regulated in the treatment.

   (a) Each gene expression data set is spread over multiple files. Each file contains the information for a single sample (e.g., a time point or a tissue sample from a patient). The first line of the file specifies the name of the sample. Every other line of the file contains two columns: the first column contains a gene identifier and the second column contains the expression value of the gene in that condition. You should read all these files in and assimilate them into a single gene expression data set of the type and format we have processed throughout the semester. You should handle the situation when a gene's expression value is present only in the files for some samples. You should also consider the possibility that different files may contain genes in different orders.

   Both the control data and the treatment data are stored in the same number of files. In addition, there is a one-one correspondence between the ith control file and the ith treatment file.

   (b) Now you have parsed one set of files containing the control data and another set of files containing the treatment data. Your next task is to compare the two datasets to identify genes that are up-regulated in response to the treatment and genes that are down-regulated in response to the treatment. A gene is *up-regulated* if its expression value in each sample in the treatment dataset is greater by a certain multiplicative factor than its expression value in the corresponding sample in the control dataset. For example, if the factor is 2, the expression value in the treatment must be at least two times as large as the value in the control. This factor must be a floating-point parameter that the user can define on the command-line using the `-regulation-bound` argument. Similarly, a gene is *down-regulated* if its expression value in each sample in the treatment dataset is smaller by the same multiplicative factor than its expression value in the corresponding sample in the control dataset.

2. You are given a file containing a network G of protein-protein interactions (PPIs). Each line of the file corresponds to a single PPI; the line contains the two identifiers separated by a tab. Your task is to implement a class to store the PPI network. At a minimum, your class should contain a constructor, a method to read in the network G from a file, methods to determine the number of proteins and the number of interactions in G, a method to count the number of interactors for a given protein, and a method to return the interactors for a protein. Most, if not all, of this functionality is present in the `Network` class we have already implemented.

3. For this part, we will tie together the previous two steps. You can assume that each gene has exactly one protein product and that the both the gene and its protein product have the same identifier. You have to identify the *response network* for the treatment, which set of interactions are activated in response to the treatment. The definition of the response network is also governed by an integer parameter `-distance`, specified on the command line, which must be at least 1. Suppose the distance parameter is d. Then the response network includes every protein previously identified as up- or down-regulated. In addition, for every pair of such proteins a and b such that the length of the shortest path between a and b in G is at most d, the response network includes the proteins and interactions in this shortest path.

   For example, if $d = 1$, we include a PPI from G in the response network if and only if each of the two interacting proteins is either up-regulated or down-regulated in response to the treatment. In other words, you can obtain the response network by deleting all proteins that are neither up- or down-regulated from the PPI network.

   Similarly, if $d = 2$, the response network includes all the proteins and interactions in the response network for $d = 1$. In addition, for every pair of proteins a and b in the response network that are not directly connected, if there is a protein c in G such that a interacts with c and b interacts with c. Note that c may or may not itself be up- or down-regulated.

   In general, the response network has four types of interactions:

   up connecting two up-regulated proteins,

   down connecting two down-regulated proteins, and

   mixed connecting an up-regulated protein to a down-regulated protein.

   bridge connecting an up-regulated protein or a down-regulated protein to a protein that is not up- or down-regulated.

   Print the response network to a file. Each line contains an interaction in the response network. In addition, the line also contains the type of the interaction, as just defined.

4. Consider the fact that you have to specify the names of all the files containing the control data set, the names of the all the files containing the treatment data set, the name of the file containing the PPI network, and the `-regulation-bound` and `-distance` parameters. Use the `Getopt::Long` or `Getopt::Euclid` modules from CPAN to define command line options for your programme.

The project may appear to be daunting at first glance. Try to solve it one step at a time. Make sure that you are satisfied that one step is correct before moving on to the next step. For each step, first decide on the class or classes you may need. Next, fix the interface for this class, i.e., the set

of methods that the user can invoke on objects belonging to the class. Write the documentation for these methods *before* implementing the methods. You may also find it helpful to write your script in a top-down manner: first, write comments in the script for each of the main steps involved (e.g., read the control data from the control files, read the treatment data from the treatment files, find the up-regulated genes, find the down-regulated genes, read the PPI network, compute the response network, and print the response network). Then write down the method invocations that will achieve these tasks. Obviously, you should have defined the corresponding variables first. Finally, implement each of the methods one by one. Some methods may be more complex than others. Break down the complex methods in a top down manner. If you use any modules from CPAN, please document their use clearly.

Note that we have not discussed how to compute shortest paths in undirected graphs. You can extend the breadth-first search methods we developed for DAG to this context.