Theoretical Computer Sciences Project

Deadline 15 January

Sergio Peignier and Théotime Grohens

The homework is 2/3 of the **total grade**.
The project should be carried out in **groups of 2 (or 3) students**.
Upload the **code** and the **report** to the dedicated moodle web page **before the 17-th of January 2020**.

1 Graph Theory

1.1 Introduction

In this project we will apply graph algorithms to study the gene regulatory network (GRN) of *Saccharomyces cerevisiae*.

This species of yeast, it is a small single-cell eukaryote, with a short generation time, and two possible forms: an haploid one and a diploid one. Moreover, this organism can be easily cultured, and it has an important economic impact since it is extensively used for instance, in winemaking, baking, and brewing. Due to these characteristics, *Saccharomyces cerevisiae* is studied as an important model organism.

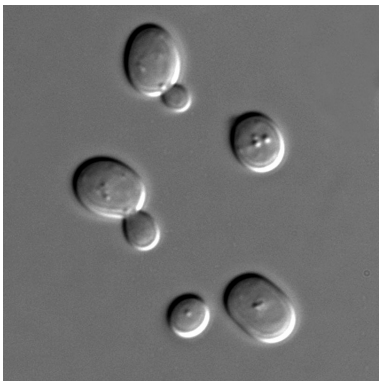

Figure 1: *Saccharomyces cerevisiae*

In this work we will study the gene regulatory network of *Saccharomyces cerevisiae*, using graph theory algorithms. The files that are provided for this project have been used in [MCK+12], as gold-standards to assess gene regulatory network inference algorithms, and they are the result of biological experiments based on ChIP binding data [MWG+06], and

systematic transcription factor deletions [HKI07]. Hereafter we describe each dataset in details:

- GRN_edges_S_cerevisiae.txt: contains the edges of the *S. cervisiae* regulatory network (from transcription factors to target genes). The intended meaning is that if there is an edge between transcription factor X and the target gene A, then X regulates the transcription of A.
- net4_transcription_factors.tsv: Is a file containing in a single column the identifiers of the transcription factors of *S. cervisiae* that were studied.
- net4_gene_ids.tsv: The two previous files, use specific identifiers to denote genes, and this file contains the gene name associated to each gene identifier.
- go_slim_mapping.tab.txt: Only columns 0 and 5 will be used in this work. Column 0 contains the gene name, and column 5 contains its Gene Ontology (GO) annotation (`http://www.geneontology.org/`). Notice that two different rows may give for the same gene different Gene Ontology annotations.

1.2 Exercises

Exercise 1. Exploration and characterization of the gene regulatory network

- Load the dataset and create a NetworkX graph instance.
- Plot the gene regulatory network, the plot should be readable, understandable, and informative. Which information did you decide to convey in your plot? Why?
- Describe the network by computing pertinent local and global metrics, explain your choices, represent the results graphically if necessary, and interpret the results.
- Implement and apply the k-shell decomposition algorithm.
- For at least 4 of the metrics that you have used: what is the time complexity of the algorithm that calculates it (explain)?

Exercise 2. Community detection

- You can choose between the Girvan Newman method and the Louvain algorithm to find communities in the graph.
- Describe both algorithms, and their time complexities (explain).
- Which algorithm did you choose, why?
- For the the Girvan Newman method, the user should select one of the output partitions, explain the criterion that could be used to make this choice, and its complexity.

- Study the GO composition of each community. To do this you can produce a counting matrix M, such that $M_{i,j}$ is the number of genes from community j that have GO annotation i[1].

- Is there a relationship between graph communities and particular cell functions?

2 Around the Traveling Salesman Problem

2.1 Introduction

The Traveling Salesman Problem is a famous NP-complete problem in computer science. It consists in finding the shortest path that visits every single node in a given graph. Such a path is called a Hamiltonian path.

In these exercises, we will implement an exact solution to the problem, as well as different heuristics that give approximate solutions in a reasonable time.

Exercise 1. Exact solution

We represent a weighted, non-oriented graph with an adjacency list. We also assume that a graph is always connected, i.e. that it is always possible to reach a node from any other node.

- In a complete graph, all nodes are connected to one another. What is the number of Hamiltonian paths in a complete graph of size n?

- Write a Python function that enumerates all Hamiltonian paths starting from a node i in a graph of size n, and returns the shortest one. What is its time complexity?

- Finally, write a Python function that finds an exact solution to the Traveling Salesman Problem for a graph of size n.

Exercise 2. The Nearest Neighbor heuristic

The Nearest Neighbor heuristic builds a Hamiltonian path incrementally. It starts from a given node u in the graph, and finds its nearest neighbor v that is not already in the path. Then, it adds the edge $u \to v$ to the path, and looks again for the nearest neighbor of v, until all nodes have been added to the path.

- Write a function that implements the Nearest Neighbor heuristic.

- What is its time complexity?

- Find a graph for which this heuristic does not compute the shortest Hamiltonian path.

[1]For the sake of clarity you may need to exclude the rows associated to GO having too many counts, since they denote too general features

Exercise 3. The Minimum Spanning Tree heuristic

A spanning tree of a graph $G = (V, E)$ is a subset F of its edges such that (V, F) is a tree. A minimum spanning tree (MST) of a graph is a spanning tree that minimizes the sum of the weights of its edges.

Here is an algorithm that computes a minimum spanning tree (Prim's algorithm).

Data: Graph $G = (V, E)$
Result: (V, F) a MST
$F \leftarrow \emptyset$
$W \leftarrow x_0$ an arbitrary element
while $W \neq V$ **do**
  Let $(x, y) \in E$ the shortest edge such that $x \in W$ and $y \notin W$
  $W \leftarrow W \cup y$
  $F \leftarrow F \cup (x, y)$
end
Result: (V, F)

Algorithm 1: Prim's algorithm

- Using an appropriate data structure, what is the time complexity of this algorithm?
- Prove that during each iteration of the `while` loop, the subgraph (W, F) is a tree. Deduce that Prim's algorithm returns a spanning tree.
- Show that the result of Prim's algorithm is a MST.
- Write a Python function that implements Prim's algorithm.
- The triangular inequality is the following inequality:
  $\forall x, y, z \in V, w(x, z) \leq w(x, y) + w(y, z)$,
  where $w(x, y)$ is the weight of edge $x \to y$ (more direct paths are shorter). Assuming that the graph verifies the triangle inequality, show that the length of the Hamiltonian cycle obtained by visiting the MST is less than twice the length of the shortest Hamiltonian cycle of the graph.

References

[HKI07] Zhanzhi Hu, Patrick J Killion, and Vishwanath R Iyer. Genetic reconstruction of a functional transcriptional regulatory network. *Nature genetics*, 39(5):683, 2007.

[MCK+12] Daniel Marbach, James C Costello, Robert Küffner, Nicole M Vega, Robert J Prill, Diogo M Camacho, Kyle R Allison, Andrej Aderhold, Richard Bonneau, Yukun Chen, et al. Wisdom of crowds for robust gene network inference. *Nature methods*, 9(8):796, 2012.

[MWG+06] Kenzie D MacIsaac, Ting Wang, D Benjamin Gordon, David K Gifford, Gary D Stormo, and Ernest Fraenkel. An improved map of conserved regulatory sites for saccharomyces cerevisiae. *BMC bioinformatics*, 7(1):113, 2006.