Graphs II: network inference

1. Introduction

In systems biology, a major goal is to derive models of biological processes, in particular of interactions between molecules: gene regulation (protein-DNA interactions), signaling (protein-protein interactions) or metabolic regulation (protein-metabolite interactions). Some of these processes are well understood and can be modeled in detail, for example using systems of ordinary differential equations (ODEs) or even more detailed models. Of other processes we know far less, but we can measure the responses of the system to perturbations and use these to derive approximate models of interactions. These two approaches are referred to as bottom-up and top-down, respectively.

One of the main challenges in top-down systems biology is network inference, i.e. deriving interaction networks from data. Often, the data are measured as time series and the goal is to derive models of regulation. For example, gene regulation networks can be inferred from time series gene expression data. However, such networks are often relatively simple representations, as there is usually insufficient data to derive models with many parameters. Network model types that have been applied include relevance networks (e.g. based on correlation or mutual information), Boolean networks, Bayesian networks, ODE models or linear models.

Boolean models constitute an interesting approach, with nodes modeling genes (or proteins) and incoming edges indicate that the expression of a gene at time t+1 depends on the expression of the genes connected to these edges at time t. The advantages are that the models are quite simple, modeling gene expression as being either “on” (1) or “off” (0), and that quite complicated dynamic interactions can be modeled. Their simplicity is also a disadvantage, as biology rarely is binary.

In this assignment, you will implement one of the first algorithms to derive Boolean interaction networks, the REVEAL algorithm. At the core of this algorithm is the concept of entropy and mutual information. The algorithm itself is quite simple; to find the regulators of a certain target gene, it looks for the smallest set of genes whose expression fully explains that of the target gene. A similar (but slightly more complex) procedure is at the heart of the well known ARACNE algorithm, which can also handle continuous variables and attempts to remove indirect interactions.

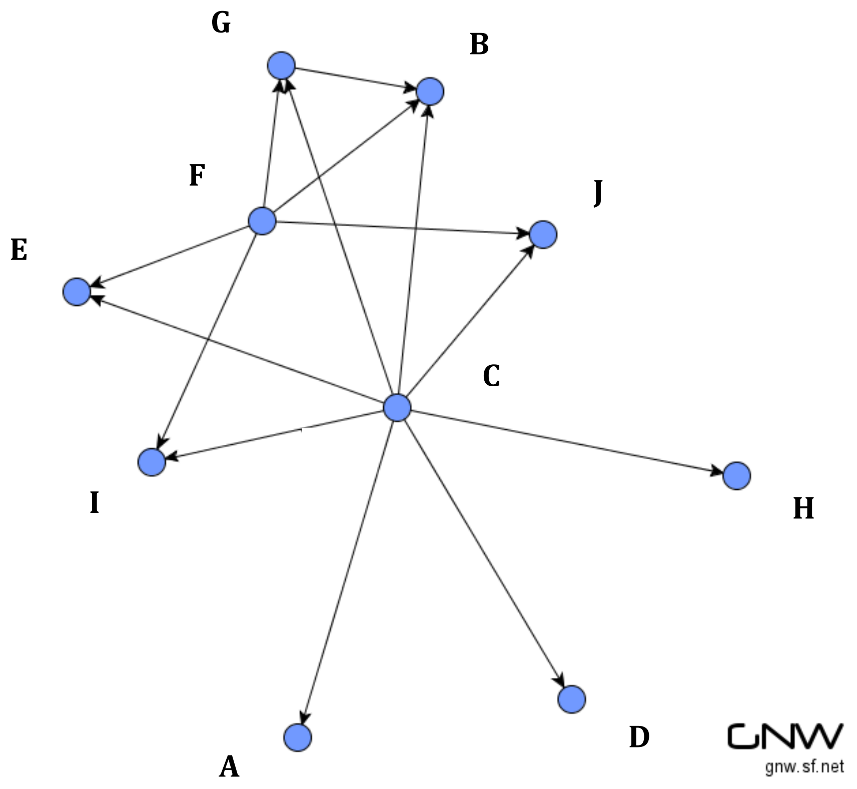
Reading material

A review paper by Kaderali and Radde gives a broad overview of the area of network inference and the various approaches taken [1]. The REVEAL algorithm is described in more detail in a 1998 publication at the Pacific Symposium on Biocomputing (PSB) by Liang *et al*.

2. Assignment

In this assignment you will implement the REVEAL algorithm to infer Boolean networks from time series data. The algorithm is briefly described on pages 40-42 of [1] and in full detail in [2].

The data is stored as a dictionary, where each keys is a node name and the value is a list of binary numbers corresponding to measurements over time. The goal is to produce a network, also represented as a dictionary where *target* nodes (to which edges point) are keys and *source* nodes are the values. To develop your code, the skeleton contains the toy example of Figures 1 and 5 of [2]; once ready, you will use your code to reveal the network in the figure below.



Use the code skeleton in **`reveal_skeleton.py`** to:

- Implement a function **`entropy(x,y,z,w)`** that calculates the entropy ([2], pp. 21-23) of a variable, $H(x)$, or the joint entropy of x with one, two or three or other lists of binary numbers, i.e. $H(x,y)$, $H(x,[y,z])$ or $H(x,[y,z,w])$. If you can, you can make it a function that can take any number of variables as input.
- Implement a function **`mutual_information(x,y,z,w)`** that calculates the mutual information ([2], pp. 21-23) $M(x,y)$, $M(x,[y,z])$ or $M(x,[y,z,w])$ (or, if you can, a function that can take any number of variables as input).
- Implement a function **`reveal(input,output)`** that finds, for each node in the dataset, the subset of nodes that best explains it. Follow the steps as outlined on pp. 22-24 of [2].

Questions:

1. Using the supplied function **`check_table(input,output)`**, list the entropy and mutual information values you calculated for the toy example and print the network you find.
2. Read in simulated input and output data from a file containing timeseries of 10,000 measurements for the 10-node yeast network in the figure above, using the supplied function **`read_tsv_file(file)`**. Explain how input and output are extracted from the single time series file.
3. Print the network you find in this data with *kmax* set to 1, 2 and 3. What do you notice?
4. Given the network structure, which nodes are the most likely regulators? Use DAVID (http://david.abcc.ncifcrf.gov/) and/or the SGD (http://yeastgenome.org/) to verify this by finding the functional annotation for each of the nodes.
5. Rerun **`reveal`** on subsets of the yeast dataset of 1000 and 100 timepoints. What network structures do you find?
6. Is the algorithm exact or approximate?
7. How is Occam's razor used in REVEAL?
8. What is the time complexity of the algorithm you implemented?
9. Is limiting *k* (to limit time complexity) a biologically plausible strategy?
10. How could you extend REVEAL to infer "imperfect" relations (i.e. for which the inputs do not fully explain the outputs, for example due to noise)?

Hints:

- Note that $M(x,y,z,w) = H(x) + H(y,z,w) - H(x,y,z,w)$.
- The log() function used in the entropy formula in [2] is actually the $\log_2()$. You can import that from numpy (as in the skeleton) or alternatively write your own function, using $\log_2(x) = \log(x)/\log(2)$.
- In the entropy formula, to avoid problems with zeroes when taking logs or dividing, add a very small value to the counts, for example **`1e-100`**.
- Round-off problems can be confusing, so don't check for **`a == b`** but for **`abs(a-b) < 0.001`** or something similar.
- For generating all subsets of a certain size *k* found in a set *s*, have a look at the function **`itertools.combinations(s,k)`**.

1. L. Kaderali and N. Radde. "Inferring gene regulatory networks from expression data", *Studies in Computational Intelligence* 94, 33–74, 2008.
2. S. Liang, S. Fuhrman and R. Somogyi. "Reveal, a general reverse engineering algorithm for inference of genetic network architectures." *Pacific Symposium on Biocomputing* 3:18—29, 1998.

Please, hand in your Python script, and a PDF file containing the answers to the questions above by e-mail, to algorithms@bioinformatics.nl no later than 14.00 on the day of the lecture.

3. Further reading

There is a wide body of literature on network inference. The Kaderali and Radde paper gives a good overview of the various model types used. On Blackboard you can find a recent paper in Nature (2012) that reports on the findings of the DREAM5 challenge. DREAM (Dialogue for Reverse Engineering Assessments and Methods) is an organization that sets up “challenges”, competitions in which researchers receive training data on which to develop their methods. They then apply their methods to a new, unlabeled set of test data and submit their findings, based on which the organization ranks the methods by their accuracy. The DREAM5 challenge dealt with network inference, and the paper shows the strengths and weaknesses of the various contenders. Interestingly, it shows that the best performance is obtained when the output of various methods is combined.

For discussion, we have selected two papers: one on dynamic Bayesian networks (DBNs), an extension of the basic Bayesian network paradigm to overcome the limitation that they cannot have cycles; and a paper on the Inferelator, a method that uses smart tricks to implement Boolean-like rules (AND, OR, NOT) in linear networks. When presenting the papers, please focus on the algorithmic challenges and implementation rather than the (biological) results. For these inference methods, in particular also the applicability to genome-scale data is of interest: do the algorithms scale well?