

MS6a, Exercises Week 6

Rune Lyngsø

November 19, 2009

A Network Inference by Conditional Independence

1. We can write up the BoostiGraph algorithm [1] as Assume that \mathbf{Y} is an $n \times p$ matrix, i.e. we have n obser-

Algorithm 1 BoostiGraph Algorithm: estimate autoregression coefficient from data \mathbf{Y}

$\mathbf{Y}^* = \mathbf{Y}, \forall i, j : \beta_{ij} = 0$ // Initialise residuals and autoregression coefficients

$\hat{\beta}_{ij} = \langle \mathbf{Y}_j^T, \mathbf{Y}_i \rangle$ // Compute initial univariate regression coefficients

while it makes sense to continue **do**

$i, j = \arg \max \{ |\hat{\beta}_{ij}| \}$

$\beta_{ij} = \beta_{ij} + \eta \hat{\beta}_{ij}, \mathbf{Y}_i^* = \mathbf{Y}_i^* - \eta \hat{\beta}_{ij} \mathbf{Y}_j$ // Update autoregression coefficients and residuals

for $k \neq i$ **do**

$\hat{\beta}_{ik} = \langle \mathbf{Y}_k^T, \mathbf{Y}_i^* \rangle$ // Update univariate regression coefficients

Report (i, j) as edge iff $\beta_{ij} \neq 0$ and $\beta_{ji} \neq 0$

variations from each of p genes. In [1] it is assumed that $i, j = \arg \max \{ |\hat{\beta}_{ij}| \}$ can be computed in time $O(1)$ (it is also assumed that the inner products for updating the $\hat{\beta}$ values can be computed in time $O(1)$, but this is actually possible by using the fact that $\langle \mathbf{Y}_k^T, \mathbf{Y}_i^* - \eta \hat{\beta}_{ij} \mathbf{Y}_j \rangle = \langle \mathbf{Y}_k^T, \mathbf{Y}_i^* \rangle - \eta \hat{\beta}_{ij} \langle \mathbf{Y}_k^T, \mathbf{Y}_j \rangle$). This makes some sense, as the algorithm is implemented in Matlab and built-in iterations are immensely faster than programmed iterations.

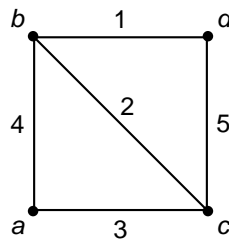
If there are no built-in operations allowing us to assume that we can find the maximum $\hat{\beta}$ value in $O(1)$ time, but that this operation is an implicit **for** loop, what is the time required for the Boostigraph algorithm assuming that it is run for T iterations?

The BoostiGraph algorithm looks a bit like Dijkstra's algorithm, in that in each iteration we choose an optimal element from our data structure and then proceed to update some elements in the data structure. It would thus seem to make sense to use an efficient priority queue to maintain the $\hat{\beta}$ values. However, the $\hat{\beta}$ values may be decreased as well as increased in an update. This means that we cannot always use the standard decreasekey priority queue operation for updates, but must anticipate having to delete and re-insert $\hat{\beta}$ values to update them. It thus doesn't make any difference which of the last three data structures in the table at http://en.wikipedia.org/wiki/Fibonacci_heap we use. Using one of them to maintain the $\hat{\beta}$ values, what would the time requirements for the BoostiGraph algorithm be? Remember that the number of $\hat{\beta}$ values is $O(p^2)$.

2. What is the outcome of the BoostiGraph algorithm on data $\mathbf{Y} = \begin{bmatrix} -1 & 1 \\ -1 & -1 \\ 1 & 1 \\ 1 & -1 \end{bmatrix}$? What about $\mathbf{Y} = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}$?

B Network Inference by Flow Requirements

3. Construct a minimum network for the request network



4. What is the sum of edge capacities for a minimum network for a request network corresponding to a star tree (one internal node connected to each of all the other (leaf) nodes) where all edges have request 1? How does this compare to the sum of request values?

Can you construct a connected request network such that the sum of request values equals the sum of capacities in a minimum network for any number of nodes?

5. For request function r define $u(x) = \max\{r(x, y) \mid y \neq x\}$ and $s(x, y) = \min\{u(x), u(y)\}$. As $\forall y \neq x : u(x) \geq s(x, y)$ it follows that $u(x) \geq \max\{s(x, y) \mid y \neq x\}$. Prove that $u(x) \leq \max\{s(x, y) \mid y \neq x\}$ (and thus that $\max\{r(x, y) \mid y \neq x\} = u(x) = \max\{s(x, y) \mid y \neq x\}$).

Remember that a network is a minimum network for r iff $u(x) = \sum_{y \neq x} c(x, y)$ where c is the capacity function of the network. We just saw that request functions r and s lead to identical $u(x)$ values. Use this to prove that for a minimum network constructed from s it cannot be the case that $w(x, y) > s(x, y)$, where the flow function $w(x, y)$ is the maximum flow between x and y (hint: observe that the sum at the beginning of this paragraph is the value of a cut in the network). Also use it to conclude that a minimum network for s is also a minimum network for r .

Let finally w' be the flow function for a minimum network for r and w the flow function for a minimum network for s . Prove that it cannot be the case that $w'(x, y) > w(x, y)$ for any pair of vertices x and y .

It follows that by using s instead of r as request function, we still get a minimum network for r , but a minimum network with maximum flow between any pair of nodes x, y that is at least as large as the maximum flow between x and y for any other minimum network for r . Apply this technique to the request network in Prob. 3.

C Network Robustness

6. We can compute a flowtree, i.e. a tree where the minimum weight on the path connecting any two nodes u and v equals the maximum flow between u and v using the following algorithm:

Algorithm 2 Compute flowtree from flow network (V, E, c)

Start with one set containing all nodes and choose an arbitrary patriarch p for this set

Flowtree $T = \emptyset$

for all $u \neq p$ **do**

Let S_u denote the set u is currently in and p_u the patriarch of this set

Compute maximum flow $w(u, p_u)$ and corresponding minimum cut U, P with $u \in U, p_u \in P$

Split S_u into $S_u \cap U$ with patriarch u and $S_u \cap P$ with patriarch p_u

Add edge (u, p_u) to T with weight $w(u, p_u)$

This algorithm can also be found in [3, Ch. 12] with a different formulation. Use this algorithm to compute flowtrees for the two minimum networks constructed in Problems 3 and 5.

For which pairs of nodes is the maximum flow larger in the second network?

One measure of the importance of a node in a network is called the *Betweenness Centrality* (or BC from here on). It measures the number of number of pairs of other nodes that would have the shortest path connecting

them disrupted if the node is eliminated from the network. More formally the BC of node v is

$$C_B(v) = \sum_{\substack{s \neq v \neq t \\ s < t}} \frac{\sigma_{s,t}(v)}{\sigma_{s,t}}$$

where $\sigma_{s,t}$ is the number of shortest paths connecting s to t and $\sigma_{s,t}(v)$ is the number of shortest paths connecting s to t that goes through v . There are several examples of biological networks where this can be viewed as a good measure of how crucial a node is for performance, e.g. in a regulation network longer paths could introduce more noise and in a metabolic network longer paths would usually result in increased overhead. Here we will use a slightly modified version of this measure,

$$C_b(v) = \sum_{\substack{s \neq v \neq t \\ s < t}} \mathbb{1}_{\sigma_{s,t}(v) = \sigma_{s,t}},$$

i.e. we count the number of pairs that would see their shortest distance increased by the elimination of v .

The BC provides a measure of the importance of an individual node in the network. To measure the robustness of a network to targeted attacks, we want to summarise these node importances into a number capturing whether importance is more or less evenly distributed over all nodes or a single node is crucial for the network. It can be observed [2, Ch. 7] that $C_B(v) \leq \frac{n^2 - 3n + 2}{2}$ (the same holds for our $C_b(v)$) so by dividing by this number we get a measure

$$C'_B(v) = \frac{2C_B(v)}{n^2 - 3n + 2}$$

that is normalised to the range from 0 to 1. The Freeman centrality index (FCI) of a graph is now defined as

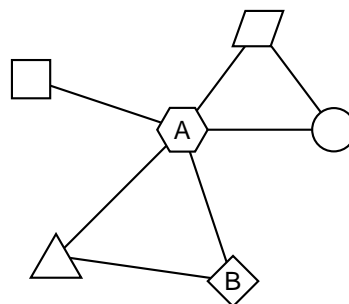
$$C_B = \frac{1}{n-1} \sum_{u \in V} C'_B(u^*) - C'_B(u)$$

where

$$u^* = \arg \max \{ C'_B(u) \mid u \in V \}$$

and provides a measure in the range from 0 to 1 of how crucial the most important node is in shortest paths. By modifying all-pairs shortest paths algorithms to remember the number of paths attaining shortest distance, the above measures can be computed efficiently.

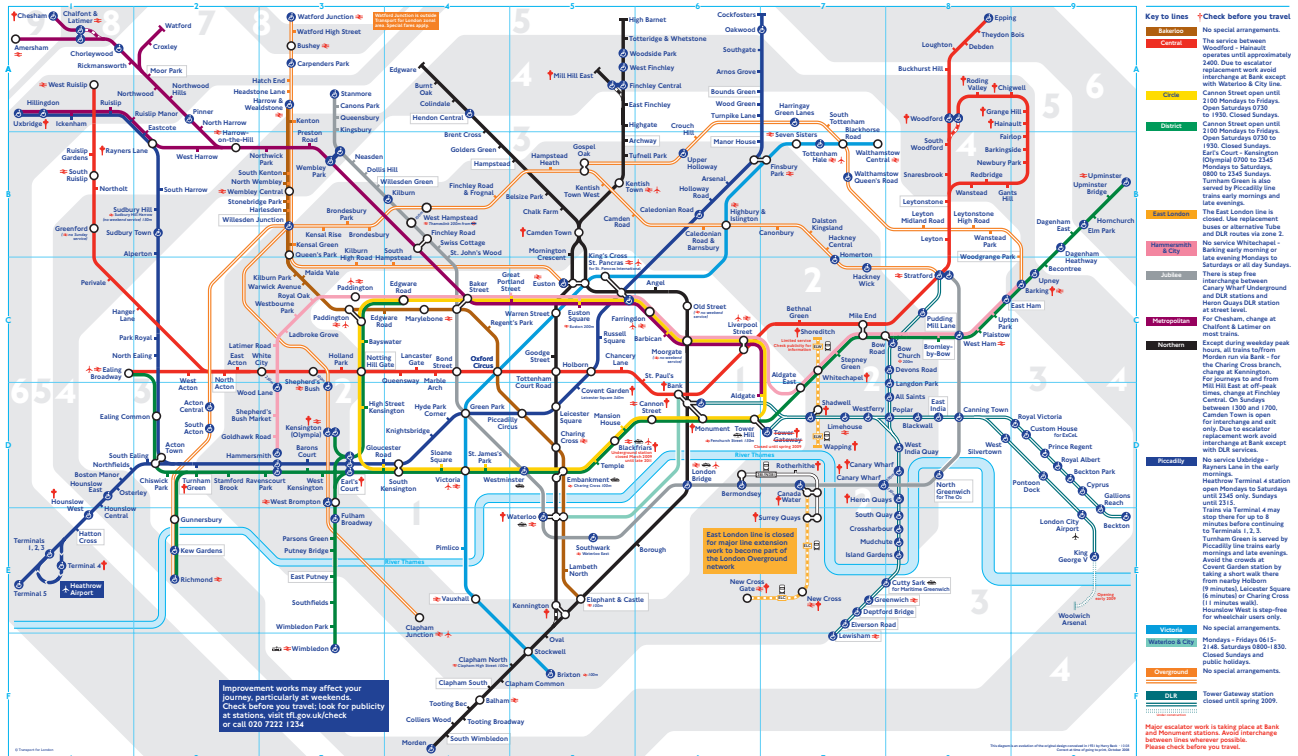
7. Compute $C_b(v)$ for the nodes marked A and B in the network



8. Compute the FCI, based on $C_b(v)$ instead of $C_B(v)$, for this network.
9. One can easily generalise the C_b score to sets, such that we count the number of pairs for which the shortest distance increases if all nodes in the set are eliminated. Design a network where the two element set with the highest C_b score is not the set of the two nodes with the highest C_b scores.

10. Transportation networks are another example where the BC measure captures how critical a node is. Which three stations on the map of the London Underground below would you guess have the highest C_b scores?

Tube Map



MAYOR OF LONDON

Website tfl.gov.uk 24 hour travel information 020 7222 1234

Transport for London

References

- [1] S. Anjum, A. Doucet, and C. C. Holmes. A boosting approach to structure learning of graphs with and without prior knowledge. *Bioinformatics*, 25(22):2929–2936, 2009.
- [2] A. Cento. *The Airline Industry: Challenges in the 21st Century*. Physica-Verlag Heidelberg, 2008.
- [3] D. Jungnickel. *Graphs, Networks, and Algorithms*. Springer, 2007.