

MS6a, Exercises Week 6, Model Solution

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 $\hat{\beta}_{ij} \neq 0$

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?

Before embarking on the **while** loop we need to initialise the $O(np)$ values of \mathbf{Y}^* and the $O(p^2)$ values of β and $\hat{\beta}$. In the **while** loop, finding the maximum $\hat{\beta}_{ij}$ value takes time $O(p^2)$. We further need to update the i 'th column of \mathbf{Y}^* , requiring time $O(n)$, and values of $\hat{\beta}_{ik}$ for all $k \neq i$, requiring time $O(p)$. In total, the time required is $O(np + np^2 + T(p^2 + n + p)) = O(np^2 + T(p^2 + n))$.

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 $O(p^2)$.

Assuming that finding the maximum $\hat{\beta}_{ij}$ value takes time $O(1)$, each update of a column in \mathbf{Y}^* takes time $O(n)$, and update of each of the $\hat{\beta}_{ik}$ values for $k \neq i$ takes time $O(\log p^2)$, the total time for running the BoostiGraph algorithm with T iterations become $O(np + np^2 + T(1 + n + p \log p)) = O(np^2 + T(n + p \log p))$ (actually, to initialise all the $\hat{\beta}$ values in time $O(np^2)$ and insert them in the data structure we need to

use either Fibonacci Heaps or Brodal Queues – with (Min-)Heaps, this part of the initialisation would take time $O((n + \log p)p^2)$.

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 $\mathbf{Y} = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}$?

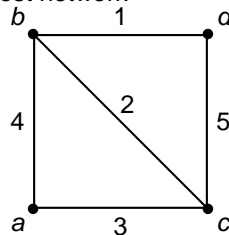
For $\mathbf{Y} = \begin{bmatrix} -1 & 1 \\ -1 & -1 \\ 1 & 1 \\ 1 & -1 \end{bmatrix}$, $\langle \mathbf{Y}_j^T, \mathbf{Y}_i \rangle = 0$ for $i \neq j$. Hence, no change will actually take place in the **while** loop,

so for any reasonable description of *it makes sense to continue* the algorithm should stop after the first iteration and report no edges.

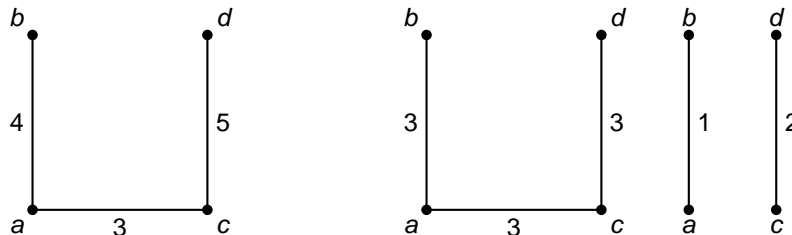
For $\mathbf{Y} = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}$, $\langle \mathbf{Y}_j^T, \mathbf{Y}_i \rangle = 2$ for $i \neq j$ so in the first iteration of the **while** loop one of the β_{ij} values, $i \neq j$, will be set to 2η . At the same time, the corresponding $\hat{\beta}_{ij}$ value will be reduced by 4η (and the i 'th column of \mathbf{Y}^* will be adjusted). Hence, in the next iteration the $\hat{\beta}_{ji}$ value (i.e. the entry with i and j reversed) will be the unique maximum, so the same updates will be applied to the corresponding β_{ji} value and the j 'th column of \mathbf{Y}^* . So β_{ij} and β_{ji} will be updated in tandem, unless one of the $\hat{\beta}_{ik}$ values is increased beyond the value of $\hat{\beta}_{ji}$ as a result of the update to $\hat{\beta}_{ij}$. For this reason it is not dramatically important whether we require both β_{ij} and β_{ji} to be non-zero, or just one of them, to report (i, j) as an edge.

B Network Inference by Flow Requirements

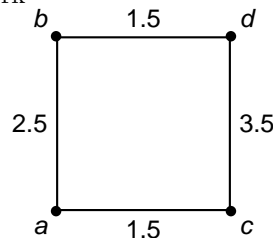
3. Construct a minimum network for the request network



Using the minimum network algorithm from [4, Ch. 12], we first create the maximum spanning tree and partition into uniform trees:



Creating a cycle of the uniform tree with more than two nodes by adding an edge between nodes b and d , we achieve the minimum 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?

The partition of the maximum spanning tree into uniform trees is just the star tree itself. The edge capacities in the cycle constructed will be $\frac{1}{2}$, so the sum of capacities for the minimum network constructed from a star tree with n nodes will be $\frac{n}{2}$. The sum of request values is $n - 1$, so the summed capacities is about half the summed requests.

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?

The sum of the weights on all edges in the uniform tree partition of the maximum spanning tree of a request network will be no larger than the sum of requests in the request network. As soon as we have three nodes in a connected request network, the partition of the maximum spanning tree into uniform trees will contain at least one tree with $n > 2$ nodes and edge weights $c > 0$. The sum of weights in this tree is $(n - 1)c$. It is converted into a cycle with edge weights $\frac{c}{2}$, so contributing only $\frac{nc}{2} < (n - 1)c$ to the total capacity of a minimum network. Hence, regardless of the structure of the request network, as long as it describes a connected set of requests reusing the requests as capacities will be a massive overkill.

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 $\max\{r(x, y) \mid y \neq x\} = u(x) = \max\{s(x, y) \mid y \neq x\}$).

Clearly $r(x, y) \leq u(x)$ and $r(x, y) \leq u(y)$ as $r(x, y)$ is part of the maximisation defining each. It follows that $r(x, y) \leq \min\{u(x), u(y)\} = s(x, y)$. Hence, $u(x) = \max\{r(x, y) \mid y \neq x\} \leq \max\{s(x, y) \mid y \neq x\}$.

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 .

Last things first. For a minimum network constructed from s , $\sum_{y \neq x} c(x, y) = \max\{s(x, y) \mid y \neq x\} = u(x)$ and thus it must also be a minimum network for r by the if and only if.

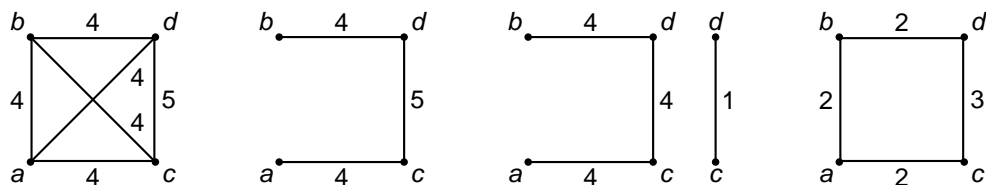
Assume that $w(x, y) > s(x, y)$ for some x, y . We then have $u(x) = \sum_{z \neq x} c(x, z) \geq w(x, y) > s(x, y)$ as $\{x\}$ vs. $V \setminus \{x\}$ is a x, y -cut in the network. Similarly, $u(y) = \sum_{z \neq y} c(y, z) \geq w(x, y) > s(x, y)$ for the same reason. But then $s(x, y) = \min\{u(x), u(y)\} > s(x, y)$, which is clearly a contradiction.

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 .

$w(x, y) = s(x, y)$ as we just saw. Assume that $u(x) \leq u(y) \Rightarrow s(x, y) = u(x)$. Now $w'(x, y) \leq \sum_{z \neq x} c'(x, z) = u(x) = w(x, y)$ where c' is the capacity function for the minimum network for r . If $u(y) < u(x)$ we can make the same argument using the sum of capacities of edges incident to 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.

The new request network, using request function s , a maximum spanning tree, its uniform tree decomposition and one possible resulting minimum network become



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

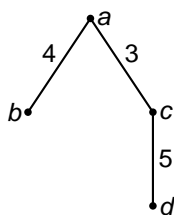
 Compute maximum flow $w(u, p_u)$ and corresponding minimum cut U, P with $u \in U, p_u \in P$

 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 $w(u, p_u)$

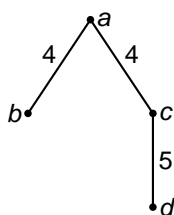
This algorithm can also be found in [4, Ch. 12] with a different formulation. Use this algorithm to compute flowtrees for the two minimum networks constructed in Problems 3 and 5.

If we iterate through the nodes in alphabetical order, choosing a as the original patriarch, for the minimum network constructed in Problem 3 we get the flowtree



by choosing the cut separating b from the rest of the nodes as the corresponding minimum cut for the flow of 4 between a and b .

For the minimum network constructed in Problem 5 one possible flowtree is



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

The only difference between the two flowtrees is that the edge between nodes a and c has weight 3 in the first and weight 4 in the second. It follows that maximum flows between nodes a or b on one side and nodes c or d on the other side is increased from 3 to 4 in the minimum network constructed in Problem 5.

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

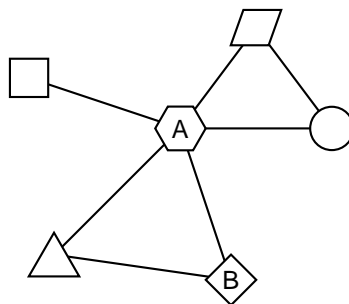
$$C_B = \frac{1}{n-1} \sum_{u \in V} C'_B(u^*) - C'_B(u)$$

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



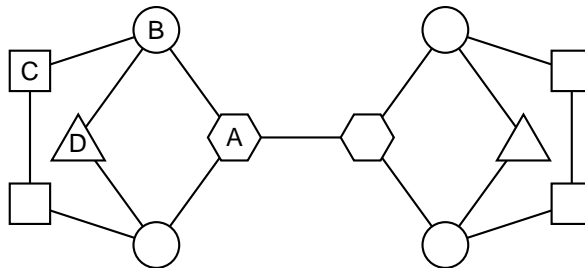
The node labelled A is on the shortest path between all pairs of other nodes, except for the two pairs of neighbours. Hence $C_b(A) = 8$. The node labelled B is not on the shortest path between any pair of other nodes so $C_b(B) = 0$.

8. Compute the FCI, based on $C_b(v)$ instead of $C_B(v)$, for this network.

Not only **B** but all nodes except **A** have C_b 0. With $n = 6$ we get $C'_b(A) = \frac{2 \cdot 8}{20} = \frac{4}{5}$ which is also the C_b based FCI of the network. It thus has a high degree of vulnerability to a targeted attack. This is pretty obvious, given the importance of **A** in the connectivity of the 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.

For the example to work, we basically need to have two nodes whose removal breaks the same large set of shortest paths. Consider the following network:

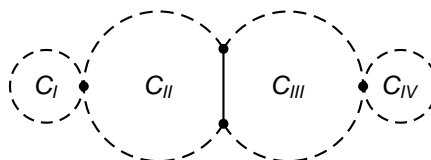


This network has four types of nodes, as illustrated. It essentially has two separate components, with the only connection between the components being the edge between the two nodes of type **A**. So removing a node of type **A** breaks all paths connecting nodes in different components. However, removing a node of type **A** will not affect shortest paths within its component, or indeed within the other component. So $C_b(A) = 30$.

Removing a node of type **B** breaks the shortest path connecting its type **C** neighbour with any node in the other component and the type **A** and the type **D** node in its own component. Consequently, $C_b(B) = 8$. Removing a type **C** node will only break the shortest path connecting its two neighbours, so $C_b(C) = 1$. Finally, removing a type **D** node does not increase the shortest distance between any other pair of nodes, as we can always use the type **A** node in the same component as an alternative. Hence, $C_b(D) = 0$. In conclusion, the two nodes with highest C_b score are the two type **A** nodes.

However, removing both type **A** nodes only increases the 25 inter-component distances left when both type **A** nodes have been removed. Within each component we can use the type **D** node to connect the two type **B** nodes with just two steps. So $C_b(\{A, A'\}) = 25$. Removing just one type **A** node will increase the 25 inter-component distances left if we also remove a node from the other component. Removing a type **B** node in the other component will further increase the distance between its neighbouring type **C** node and the type **A** and the type **D** node in this component. So $C_b(\{A, B'\}) = 27$.

An alternative approach would be to aim at constructing a network where the shortest paths between many pairs have to use one one of two nodes. The removal of one still leaves a shortest path of the same length through the other, but removal of both breaks all shortest paths. Consider for example a network consisting of four cliques (a set of nodes where all pairs are connected by an edge) as sketched in

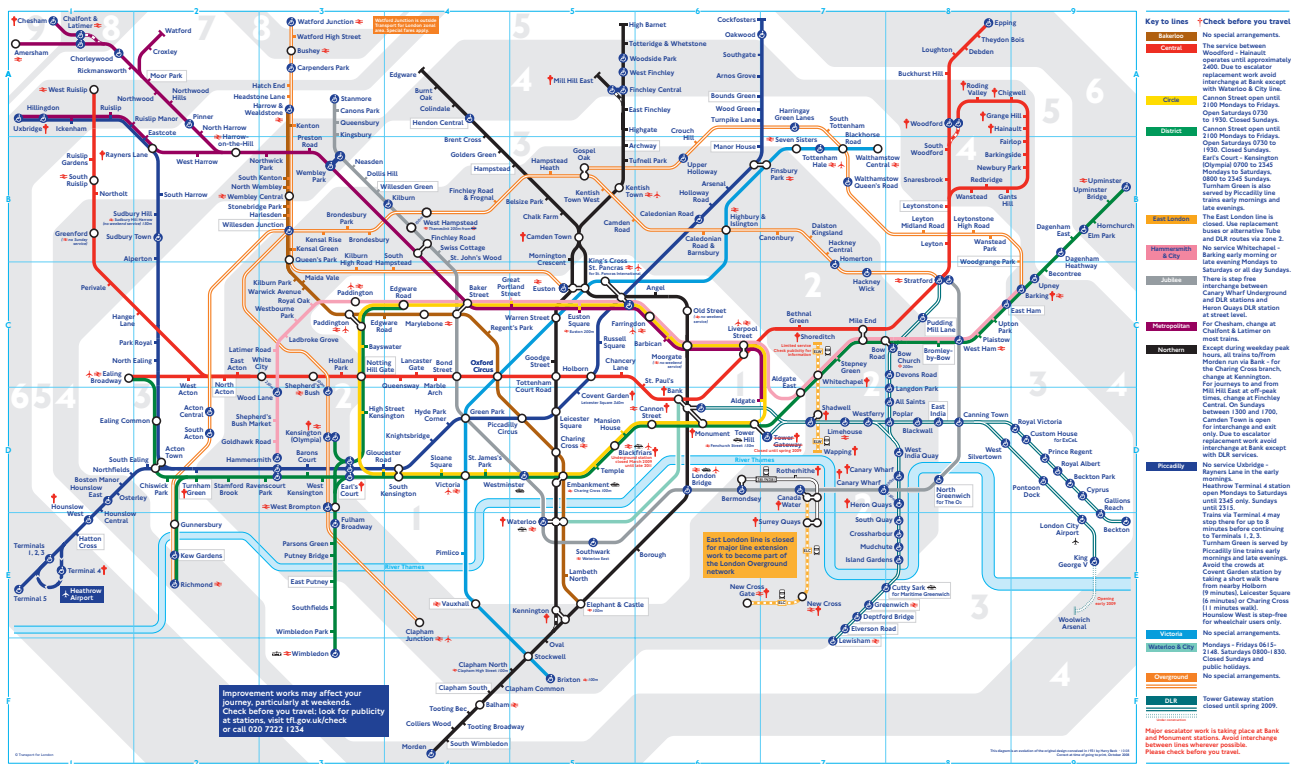


Clique C_I shares one node with clique C_{II} , clique C_{II} shares two nodes with clique C_{III} , and clique C_{III} shares one node with clique C_{IV} . Let cliques C_I and C_{IV} consist of n nodes each, and cliques C_{II}

and C_{III} consist of $3n$ nodes each. The only nodes with non-zero C_b values are the two nodes shared between cliques C_I and C_{IV} and their neighbouring clique. Removing both will disrupt approximately $13n^2$ shortest paths, while disrupting the two nodes shared between cliques C_{II} and C_{III} will disrupt approximately $16n^2$ shortest paths.

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 UNDERGROUND

The full list of C_b scores, when changing from one line to another is also considered an edge, is: Amersham: 0 Brixton: 0 Chesham: 0 Cockfosters: 0 Edgware: 0 Elephant & Castle: 0 Epping: 0 Harrow & Wealdstone: 0 Heathrow Terminal 5: 0 High Barnet: 0 Kensington (Olympia): 0 Mill Hill East: 0 Morden: 0 Mornington Crescent: 0 Richmond: 0 Stamford Brook: 0 Upminster: 0 Uxbridge: 0 Walthamstow Central: 0 Watford: 0 West Ruislip: 0 Wimbledon: 0 Northwood Hills: 5 Swiss Cottage: 5 Preston Road: 8 Willesden Green: 10 Northwood: 11 Dollis Hill: 14 Stamford Brook: 16 Arsenal: 22 Chiswick Park: 23 Hainault: 29 Grange Hill: 33 Elephant and Castle: 35 Ravenscourt Park: 35 Northwick Park: 47 West Kensington: 50 Goodge Street: 58 Borough: 118 Covent Garden: 126 Bromley-by-Bow: 146 Stepney Green: 150 Marylebone: 169 St John's Wood: 172 Holloway Road: 197 Russell Square: 199 Neasden: 229 Whitechapel: 235 Charing Cross: 252 Shepherd's Bush Market: 257 Pinner: 258 Leicester Square: 259 Kilburn: 261 Goldhawk Road: 264 Blackhorse Road: 266 Burnt Oak: 266 Canons Park: 266 Croxley: 266 Heathrow Terminals 1, 2, 3: 266 Hillingdon: 266 Kenton: 266 Kew Gardens: 266 Lambeth North: 266 Oakwood: 266 Ruislip Gardens: 266 South Wimbledon: 266 Theydon Bois: 266 Totteridge and Whetstone: 266 Upminster Bridge: 266 Wimbledon Park: 266 Fairlop: 274 Cannon Street: 279 Monument: 284 Chigwell: 286 Aldgate: 326 Wood Lane: 336 Bow Road: 353 Tower Hill: 354 Regent's Park: 367 Mansion House: 368 Ealing Broadway: 424 Piccadilly Circus: 430 Caledonian Road: 438 Aldgate East: 442 Old Street: 471 Latimer Road: 482 Blackfriars: 490 West Acton: 506 North Harrow: 512 West Hampstead:

516 Barkingside: 518 Colindale: 530 Colliers Wood: 530 Debden: 530 Gunnersbury: 530 Heathrow Terminal 4: 530 Hornchurch: 530 Ickenham: 530 Queensbury: 530 South Kenton: 530 South Ruislip: 530 Southfields: 530 Southgate: 530 Tottenham Hale: 530 Woodside Park: 530 Chalfont & Latimer: 531 Angel: 533 Roding Valley: 538 Ladbroke Grove: 668 Temple: 681 Newbury Park: 766 Arnos Grove: 792 Chorleywood: 792 East Putney: 792 Elm Park: 792 Hatton Cross: 792 Hendon Central: 792 Kingsbury: 792 Loughton: 792 North Wembley: 792 Northolt: 792 Ruislip: 792 Seven Sisters: 792 Tooting Broadway: 792 West Finchley: 792 Westbourne Park: 861 Canning Town: 876 Oval: 887 St Paul's: 926 Park Royal: 944 St James's Park: 956 Chancery Lane: 969 North Ealing: 978 Alperton: 1007 Gants Hill: 1014 North Greenwich: 1018 Tottenham Court Road: 1041 Bounds Green: 1052 Brent Cross: 1052 Buckhurst Hill: 1052 Dagenham East: 1052 Greenford: 1052 Hounslow West: 1052 Putney Bridge: 1052 Rickmansworth: 1052 Ruislip Manor: 1052 Tooting Bec: 1052 Wembley Central: 1052 Royal Oak: 1067 Embankment: 1126 Sudbury Town: 1131 Canary Wharf: 1196 Kennington: 1196 Redbridge: 1267 Sudbury Hill: 1278 Holborn: 1283 Balham: 1310 Dagenham Heathway: 1310 Eastcote: 1310 Golders Green: 1310 Hounslow Central: 1310 Parsons Green: 1310 Perivale: 1310 Stonebridge park: 1310 Wood Green: 1310 Finchley Central: 1314 Vauxhall: 1345 Canada Water: 1380 South Harrow: 1431 Ealing Common: 1434 Queensway: 1501 Wanstead: 1519 Pimlico: 1521 Becontree: 1566 Bermondsey: 1566 Clapham South: 1566 East Finchley: 1566 Fulham Broadway: 1566 Hampstead: 1566 Hanger Lane: 1566 Harlesden: 1566 Hounslow East: 1566 Turnpike Lane: 1566 Lancaster Gate: 1604 Southwark: 1656 Wembley Park: 1736 Marble Arch: 1737 High Street Kensington: 1741 Belsize Park: 1820 Clapham Common: 1820 Highgate: 1820 Manor House: 1820 Osterley: 1820 Upney: 1820 West Brompton: 1820 Willesden Junction: 1820 Woodford: 2070 Archway: 2072 Barking: 2072 Boston Manor: 2072 Chalk Farm: 2072 Clapham North: 2072 Kensal Green: 2072 Moor Park: 2086 London Bridge: 2088 Bayswater: 2101 East Acton: 2116 Sloane Square: 2194 North Acton: 2224 South Woodford: 2279 White City: 2300 East Ham: 2322 Northfields: 2322 Queen's Park: 2322 Tufnell Park: 2322 Shepherd's Bush: 2499 Snaresbrook: 2523 Kentish Town: 2570 Kilburn Park: 2570 South Ealing: 2570 Upton Park: 2570 Stockwell: 2642 Knightsbridge: 2678 Holland Park: 2715 West Harrow: 2789 Hyde Park Corner: 2803 Maida Vale: 2816 Plaistow: 2816 Rayners Lane: 2921 Barbican: 3052 Warwick Avenue: 3060 Warren Street: 3092 Finsbury Park: 3093 Farringdon: 3158 Highbury & Islington: 3207 West Ham: 3213 Victoria: 3364 Acton Town: 3695 Bank: 3724 Turnham Green: 4014 Bond Street: 4216 Oxford Circus: 4326 Notting Hill Gate: 4337 Barons Court: 4403 Leytonstone: 4510 Hammersmith: 4518 Westminster: 4618 Moorgate: 4659 Leyton: 4712 Edgware Road: 4989 Waterloo: 5006 Stratford: 5021 Camden Town: 5039 Euston Square: 5275 Great Portland Street: 5313 Gloucester Road: 5461 South Kensington: 5521 Harrow-on-the-Hill: 5642 Bethnal Green: 5660 Paddington: 5875 Mile End: 6158 Euston: 6565 Earl's Court: 6791 Green Park: 7209 Liverpool Street: 7305 Finchley Road: 7656 King's Cross St Pancras: 9978 Baker Street: 11641

The most crucial stations are thus Baker Street, King's Cross St Pancras, and Finchley Road. An analysis of the possibility of a rational choice of targets in the 7/7 bombings is presented in [3].

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] F. Jordán. Predicting target selection by terrorists: a network analysis of the 2005 london underground attacks. *International Journal of Critical Infrastructures*, 4(1/2):206–214, 2008.
- [4] D. Jungnickel. *Graphs, Networks, and Algorithms*. Springer, 2007.