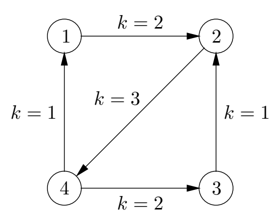
EE263 Homework 2 Solutions

Fall 2023

2.140. Communication over a wireless network with time-slots. We consider a network with n nodes, labeled $1, \ldots, n$. A directed graph shows which nodes can send messages (directly) to which other nodes; specifically, an edge from node j to node i means that node j can transmit a message directly to node i. Each edge is assigned to one of K *time-slots*, which are labeled $1, \ldots, K$. At time period $t = 1$, only the edges assigned to time-slot 1 can transmit a message; at time period $t = 2$, only the edges assigned to time-slot 2 can transmit a message, and so on. After time period $t = K$, the pattern repeats. At time period $t = K + 1$, the edges assigned to time-slot 1 are again active; at $t = K + 2$, the edges assigned to time-slot 2 are active, etc. This cycle repeats indefinitely: when $t = mK + k$, where m is an integer, and $k \in \{1, \ldots, K\}$, transmissions can occur only over edges assigned to time-slot k. Although it doesn't matter for the problem, we mention some reasons why the possible transmissions are assigned to time-slots. Two possible transmissions are assigned to different time-slots if they would interfere with each other, or if they would violate some limit (such as on the total power available at a node) if the transmissions occurred simultaneously. A message or packet can be sent from one node to another by a sequence of transmissions from node to node. At time period t, the message can be sent across any edge that is active at period t. It is also possible to store a message at a node during any time period, presumably for transmission during a later period. If a message is sent from node j to node i in period t, then in period $t + 1$ the message is at node i, and can be stored there, or transmitted across any edge emanating from node i and active at time period $t + 1$. To make sure the terminology is clear, we consider the very simple example shown below, with $n = 4$ nodes, and $K = 3$ time-slots.



In this example, we can send a message that starts in node 1 to node 3 as follows:

- During period $t = 1$ (time-slot $k = 1$), store it at node 1.
- During period $t = 2$ (time-slot $k = 2$), transmit it to node 2.
- During period $t = 3$ (time-slot $k = 3$), transmit it to node 4.
- During period $t = 4$ (time-slot $k = 1$), store it at node 4.
- During period $t = 5$ (time-slot $k = 2$), transmit it to node 3.

You can check that at each period, the transmission used is active, *i.e.*, assigned to the associated time-slot. The sequence of transmissions (and storing) described above gets the message from node 1 to node 3 in 5 periods. Finally, the problem. We consider a specific network with $n = 20$ nodes, and $K = 3$ time-slots, with edges and time-slot assignments given in `ts_data.json`. The labeled graph that specifies the possible transmissions and the associated time-slot assignments are given in a matrix $A \in \mathbb{R}^{n \times n}$, as follows:

$$A_{ij} = \begin{cases} k & \text{if transmission from node } j \text{ to node } i \text{ is allowed, and assigned to time-slot } k \\ 0 & \text{if transmission from node } j \text{ to node } i \text{ is never allowed} \\ 0 & i = j. \end{cases}$$

Note that we set $A_{ii} = 0$ for convenience. This choice has no significance; you can store a message at any node in any period. To illustrate this encoding of the graph, consider the simple example described above. For this example, we have

$$A_{\text{example}} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 2 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 \end{bmatrix}.$$

The problems below concern the network described in the data file and *not* the simple example given above.

a) *Minimum-time point-to-point routing.* Find the fastest way to get a message that starts at node 5, to node 18. Give your solution as a prescription ordered in time from $t = 1$ to $t = T$ (the last transmission), as in the example above. At each time period, give the transmission (as in 'transmit from node 7 to node 9') or state that the message is to be stored (as in 'store at node 13'). Be sure that transmissions only occur during the associated time-slots. You only need to give *one* prescription for getting the message from node 5 to node 18 in minimum time.

b) *Minimum time flooding.* In this part of the problem, we assume that once the message reaches a node, a copy is kept there, even when the message is transmitted to another node. Thus, the message is available at the node to be transmitted along any active edge emanating from that node, at any future period. Moreover, we allow multi-cast: if during a time period there are multiple active edges emanating from a node that has (a copy of) the message, then transmission can occur during that time period across *all* (or any subset) of the active edges. In this part of the problem, we are interested in getting a message that starts at a particular node, to all others, and we attach no cost to storage or transmission, so there is no harm is assuming that at each time period, every node that has the message forwards it to all nodes it is able to transmit to. What is the minimum time it takes before all nodes have a message that starts at node 7?

For both parts of the problem, you must give the specific solution, as well as a description of your approach and method.

Solution. We first define the matrices $A^{(1)}$, $A^{(2)}$ and $A^{(3)}$, such that:

$$A^{(k)}_{ij} = \begin{cases} 1 & \text{if } A_{ij} = k \\ 1 & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

Thus, the matrix $A^{(k)}$ is the node adjacency matrix for time $t = k$. The significance of $A_{ii}^{(k)} = 1$ is that packets are allowed to be stored at any node, instead of being transmitted. Formally we would say that the graph has self loops at all nodes. Let $B^{(1)} = A^{(1)}$. The (i, j) element of $B^{(1)}$ is 1 if there is an active edge connecting node j to node i at time $t = 1$. We can also interpret $B_{ij}^{(1)}$ as the number of paths that go from node j to node i in 1 time epoch. Similarly let $B^{(2)} = A^{(2)}A^{(1)}$. We have that:

$$B_{ij}^{(2)} = \sum_k A_{ik}^{(2)} A_{kj}^{(1)}$$

An $A_{ik}^{(2)} A_{kj}^{(1)}$ element in the above summation will be nonzero if there is a valid length 2 path from node j to node i via node k. Thus $B_{ij}^{(2)}$ is the number of paths that go from node j to node i in 2 time epochs. In general let us define:

$$B^{(t)} = (A^{(r)} \cdots A^{(2)} A^{(1)})(A^{(K)} \cdots A^{(2)} A^{(1)})^m$$

where $t = mK + r$ and $r < K$. The element $B_{ij}^{(t)}$ tells us the number of allowed paths (including transmission and storage of a message) that go from node j to node i in t time epochs.

a) *Minimum-time point-to-point routing.* In order to calculate the minimum time required to get to node 18 from node 5, we just need to find the smallest value of k such that $B_{(18,5)}^{(k)}$ becomes nonnegative. This can be easily done by matrix multiplication. It was found that $T_{\min} = 8$. Also, since $B_{(18,5)}^{(8)} = 1$, there is a *unique* minimum-time path from node 5 to node 18. To reconstruct the minimum time path, we should start from the destination node and work our way up to the starting node. We should first note that $B^{(T_{\min})}$ can be rewritten as:

$$B^{(T_{\min})} = A^{(T_{\min})} B^{(T_{\min}-1)}$$

Now, if we want to find the penultimate point in the optimum path, we have to look at the 18th row of $A^{(T_{\min})}$ and the 5th column of $B^{(T_{\min}-1)}$. The nodes which correspond to elements which are jointly nonzero on both of these vectors make possible choices for the penultimate point. It doesn't matter which one of those we choose as the second to last point in the path. We can then repeat the above procedure, but this time for the second to last point. We keep doing this till we reach node 5. By following this method, we found that one (in this case unique) optimum path for the packet is:

- Start at node 5.
- During period $t = 1$, store it at node 5.
- During period $t = 2$, transmit it from node 5 to node 2.
- During period $t = 3$, store it at node 2.
- During period $t = 4$, transmit it from node 2 to node 9.
- During period $t = 5$, transmit it from node 9 to node 20.
- During period $t = 6$, store it at node 20.
- During period $t = 7$, store it at node 20.

- During period $t = 8$, transmit it from node 20 to node 18.

b) *Minimum time flooding.* To solve this problem, we just need to find the minimum value of k such that the 7th column of $B^{(k)}$ has no zero components. It was found that in this case $T_{\text{flood}} = 9$.

The following Julia code solves a general instance of this problem for any values of n and K.

```
using LinearAlgebra

include("readclassjson.jl")
ts_data = readclassjson("ts_data.json")
A = ts_data["A"]
K = ts_data["K"]
n = ts_data["n"]
At = zeros(n,n,K)

for k = 1:K
    local temp = Matrix(1.0I, n, n)
    for j in findall(i -> i==k,A)
        temp[j] = 1
    end
    At[:,:,k] = temp
end
start = 5
dest = 18
t = 1
exit_flag = 0
B = Matrix(1.0I, n, n)
C = zeros(n,n,0)
while(exit_flag == 0)
    for k = 1:K
        global B = At[:,:,k]*B
        global C = cat(C,B,dims=3)
        if(B[dest,start]>0)
            global exit_flag = 1
            break
        end
        global t = t+1
    end
end
t_min_path = t
display(t_min_path)
path = dest
new_dest = dest
for count = (t_min_path-1):-1:1
    last_slot = rem(count,K)+1
```

```
    row_dest = At[new_dest,:,last_slot]
    col_start = C[:,start,count]
    D = row_dest.*col_start
    index = findall(j -> j>0,D)
    global new_dest = index[1]
    global path = [new_dest path]
end
path = [start path]
display(path)
flood_node = 7
t = 1
exit_flag = 0
B = Matrix(1.0I, n, n)
while(exit_flag == 0)
    for k = 1:K
        global B = At[:,:,k]*B
        if(all(m -> m!=0,B[:,flood_node]))
            global exit_flag = 1
            break
        end
        global t = t+1
    end
end
t_min_flood = t
display(t_min_flood)
```

We could also solve the problem for $K = 3$ and $n = 20$ "by hand", by using the following Julia code:

```
# Import libraries & functions
using LinearAlgebra;
include("readclassjson.jl");

# Load in data
ts_data = readclassjson("ts_data.json");
A = ts_data["A"];
n = ts_data["n"];

# Create A1, A2, A3 matrices
A1 = diagm(ones(n));  # Self-loops
A1[A .== 1] .= 1;     # A1_ij = 1 if A_ij = 1
A2 = diagm(ones(n));  # Self-loops
A2[A .== 2] .= 1;     # A2_ij = 1 if A_ij = 2
A3 = diagm(ones(n));  # Self-loops
A3[A .== 3] .= 1;     # A3_ij = 1 if A_ij = 3

# Proceed through each step, one at a time, by hand
```

```
start = 5; dest = 18;
B1 = A1;
display(B1[dest, start])  # == 0
B2 = A2 * B1;
display(B2[dest, start])  # == 0
B3 = A3 * B2;
display(B3[dest, start])  # == 0
B4 = A1 * B3;
display(B4[dest, start])  # == 0
B5 = A2 * B4;
display(B5[dest, start])  # == 0
B6 = A3 * B5;
display(B6[dest, start])  # == 0
B7 = A1 * B6;
display(B7[dest, start])  # == 0
B8 = A2 * B7;
display(B8[dest, start])  # == 1, so Done!
```

Using the above code, the minimum time can be found. If we then wanted to reconstruct the actual path, we should start from $A^{(2)}$ and $B^{(7)}$ and work all the way up to $A^{(2)}$ and $A^{(1)}$. In order to find the minimum time flooding, we can use a similar code to the above. However, at each time-step t we should check `min(Bt(:,7))` and exit when that quantity is greater than 0. We mention one method that is *wrong*, but happens to give the correct answer for part (a). The method works like this:

- First, find the shortest path from node 5 to node 18, *ignoring* the time-slots, using any edge over which communication is allowed during some time-slot.
- Then, add a waiting period at each node (if needed), to make the path feasible.

In general, this method does not work; it does not give the shortest path from a source to destination. Some comments: Several people used CS style methods, such as breadth-first search or dynamic programming, to solve the problem. Of course, that's fine, provided the method was explained. But surely those of you who did this should have known the problem was *somehow* related to the topics of EE263 ... We were amazed that several people worked out the problem *by hand*, by enumerating (and in some cases drawing) all possible paths. We tried to make the problem instance big enough that this would be nearly impossible, but, obviously, we failed. *Next time*, the problem instance will have 50 nodes, 200 edges, and say 5 time slots.

3.260. Halfspace. Suppose $a, b \in \mathbb{R}^n$ are two given points. Show that the set of points in $\mathbb{R}^n$ that are closer to a than b is a halfspace, *i.e.*:

$$\{x \mid \|x-a\| \leq \|x-b\| \} = \{ x \mid c^\mathsf{T} x \leq d\}$$

for appropriate $c \in \mathbb{R}^n$ and $d \in \mathbb{R}$. Give c and d explicitly, and draw a picture showing a, b, c, and the halfspace.

Solution. It is easy to see geometrically what is going on: the hyperplane that goes right between a and b splits $\mathbb{R}^n$ into two parts; the points closer to a (than b) and the points closer to b (than a). More precisely, the hyperplane is normal to the line through a and b, and intersects that line at the midpoint between a and b. Now that we have the idea, let's try to derive it algebraically. Let x belong to the set of points in $\mathbb{R}^n$ that are closer to a than b. Therefore $\|x-a\| < \|x-b\|$ or $\|x-a\|^2 < \|x-b\|^2$ so

$$(x-a)^\mathsf{T}(x-a) < (x-b)^\mathsf{T}(x-b).$$

Expanding the inner products gives

$$x^\mathsf{T}x - x^\mathsf{T}a - a^\mathsf{T}x + a^\mathsf{T}a < x^\mathsf{T}x - x^\mathsf{T}b - b^\mathsf{T}x + b^\mathsf{T}b$$

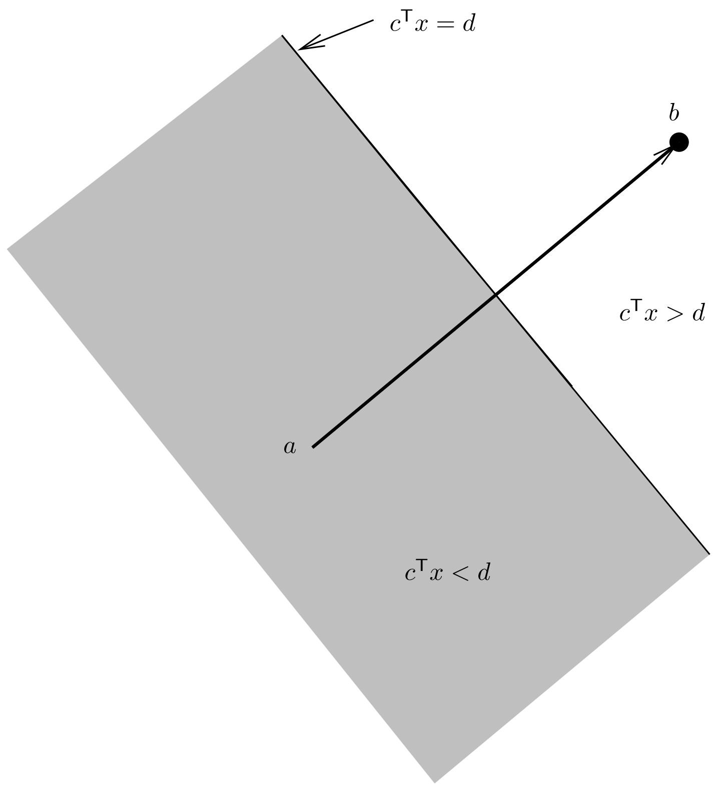
or

$$-2a^\mathsf{T}x + a^\mathsf{T}a < -2b^\mathsf{T}x + b^\mathsf{T}b$$

and finally

$$(b-a)^\mathsf{T}x < \frac{1}{2}(b^\mathsf{T}b - a^\mathsf{T}a). \tag{1}$$

Thus (1) is in the form $c^\mathsf{T}x < d$ with $c = b - a$ and $d = \frac{1}{2}(b^\mathsf{T}b - a^\mathsf{T}a)$ and therefore we have shown that the set of points in $\mathbb{R}^n$ that are closer to a than b is a halfspace. Note that the hyperplane $c^\mathsf{T}x = d$ is perpendicular to $c = b - a$.



3.350. Right inverses. This problem concerns the specific matrix

$$A = \begin{bmatrix} -1 & 0 & 0 & -1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

This matrix is full rank (*i.e.*, its rank is 3), so there exists at least one right inverse. In fact, there are many right inverses of A, which opens the possibility that we can seek right inverses that in addition have other properties. For each of the cases below, either find a specific matrix $B \in \mathbb{R}^{5\times 3}$ that satisfies $AB = I$ and the given property, or explain why there is no such B. In cases where there is a right inverse B with the required property, you must briefly explain how you found your B. You must also attach a printout of your Julia script or Jupyter notebook showing the verification that $AB = I$. (We'll be very angry if we have to type in your 5×3 matrix into Julia to check it.) When there is no right inverse with the given property, briefly explain why there is no such B.

a) The second row of B is zero.

b) The nullspace of B has dimension one.

c) The third column of B is zero.

d) The second and third rows of B are the same.

e) B is upper triangular, *i.e.*, $B_{ij} = 0$ for $i > j$.

f) B is lower triangular, *i.e.*, $B_{ij} = 0$ for $i < j$.

Solution.

a) The second row of B is zero. This means that the second column of A isn't used in forming AB. Let $\tilde{A}$ be the matrix A with its second column removed, and let $\tilde{B}$ denote the matrix B with its second row (which is supposed to be zero) removed. We have $\tilde{A}\tilde{B} = AB = I$, so $\tilde{B}$ is a right inverse of $\tilde{A}$. There is such a matrix if and only if $\tilde{A}$ is full rank, which it is. We can take $\tilde{B} = \tilde{A}^\mathsf{T}(\tilde{A}\tilde{A}^\mathsf{T})^{-1}$. Finally to construct B we simply insert a zero second row, moving rows 2, 3, 4 down by one. This gives the matrix

$$B = \begin{bmatrix} 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \frac{1}{2} \\ 1 & 0 & 1 \end{bmatrix}.$$

There are other possible choices as well.

b) The nullspace of B has dimension one. This means that B has rank 2, so the rank of AB is at most 2, which rules out the possibility that $AB = I$. So this is impossible.

c) The third column of B is zero. This implies B has a nullspace with dimension at least one, so by part (b) above, this is impossible too.

d) The second and third rows of B are the same. Let $\tilde{B}$ denote B with one of the (identical) rows 2 and 3 deleted. Then we have $AB = \tilde{A}\tilde{B}$, where $\tilde{A}$ is obtained from the matrix A by replacing its second column with the sum of its second and third columns, and deleting its third column. Thus, we need to find a right inverse for $\tilde{A}$, provided it is full rank. It is, so we can take $\tilde{B} = \tilde{A}^\mathsf{T}(\tilde{A}\tilde{A}^\mathsf{T})^{-1}$. Finally to construct B we simply insert a second copy of the second row of $\tilde{B}$ as a new third row. This gives

$$B = \begin{bmatrix} 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} \\ 1 & 0 & 1 \end{bmatrix}.$$

This matrix also happens to be the pseudo-inverse of A, $B = A^\mathsf{T}(AA^\mathsf{T})^{-1}$, and some of you noticed this immediately and used the pseudo-inverse to answer this question. That's a fine answer; it was our mistake to choose A so that the pseudo-inverse satisfied this condition. In general, of course, it would not.

e) B is upper triangular, *i.e.*, $B_{ij} = 0$ for $i > j$. If B is upper triangular, then it has the form

$$\begin{bmatrix} \tilde{B} \\ 0 \end{bmatrix},$$

where $\tilde{B}$ is square and upper triangular. If $AB = I$, then $\tilde{A}\tilde{B} = I$, where $\tilde{A}$ is the matrix formed from the first 3 columns of A. Thus we have $\tilde{A} = \tilde{B}^{-1}$. But the inverse of an upper triangular matrix is also upper triangular, so unless $\tilde{A}$ is upper triangular (and it isn't, in this case), we can't possibly have $\tilde{A}\tilde{B} = I$. So there is no such B in this case.

f) B is lower triangular, *i.e.*, $B_{ij} = 0$ for $i < j$. Let's label the columns of B as

$$b_1, \qquad b_2 = \begin{bmatrix} 0 \\ \tilde{b}_2 \end{bmatrix}, \qquad b_3 = \begin{bmatrix} 0 \\ 0 \\ \tilde{b}_3 \end{bmatrix},$$

where $\tilde{b}_2 \in \mathbb{R}^4$ and $\tilde{b}_3 \in \mathbb{R}^3$. To say that $AB = I$ is the same as saying that $Ab_1 = e_1$, $Ab_2 = e_2$, and $Ab_3 = e_3$, where e_1, e_2, e_3 are the unit vectors. We can solve these equations separately. The first equation is easy; the second we reduce to $\tilde{A}\tilde{b}_2 = e_2$, where here $\tilde{A}$ is A with its first column removed. The third is handled similarly. These equations do have a solution; we get

$$B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}.$$

Another way: we set it up as a set of 9 linear equations (one for each entry of $AB = I$) in $5 + 4 + 3 = 12$ variables. The variables are the first column of B (with 5 entries), the nonzero part of the second column of B (with 4 entries), and the nonzero part of the third second column of B (with 3 entries). We then attempt to solve these 9 equations in 12 variables. Some equations immediately give us the B matrix coefficients, while the others can be solved by inspection to obtain a rather simple matrix

$$B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}.$$

3.390. Some true/false questions. Determine if the following statements are true or false. No justification or discussion is needed for your answers. What we mean by "true" is that the statement is true for all values of the matrices and vectors given. You can't assume anything about the dimensions of the matrices (unless it's explicitly stated), but you can assume that the dimensions are such that all expressions make sense. For example, the statement "$A+B = B+A$" is true, because no matter what the dimensions of A and B (which must, however, be the same), and no matter what values A and B have, the statement holds. As another example, the statement $A^2 = A$ is false, because there are (square) matrices for which this doesn't hold. (There are also matrices for which it does hold, *e.g.*, an identity matrix. But that doesn't make the statement true.)

a) If all coefficients (*i.e.*, entries) of the matrix A are positive, then A is full rank.

b) If A and B are onto, then $A + B$ must be onto.

c) If A and B are onto, then so is the matrix $\begin{bmatrix} A & C \\ 0 & B \end{bmatrix}$.

d) If A and B are onto, then so is the matrix $\begin{bmatrix} A \\ B \end{bmatrix}$.

e) If the matrix $\begin{bmatrix} A \\ B \end{bmatrix}$ is onto, then so are the matrices A and B.

f) If A is full rank and skinny, then so is the matrix $\begin{bmatrix} A \\ B \end{bmatrix}$.

Solution.

a) If all coefficients (*i.e.*, entries) of the matrix A are positive, then A is full rank.

False. The matrix $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ has all entries positive and is singular, hence not full rank.

b) If A and B are onto, then $A + B$ must be onto.

False. The 1×1 matrix $A = 1$ is full rank, and so is the matrix $B = -1$. But $A + B = 0$ (the 1×1 zero), which is not onto.

c) If A and B are onto, then so is the matrix $\begin{bmatrix} A & C \\ 0 & B \end{bmatrix}$.

True. To show this matrix is onto, we need to show that we can solve the equations

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} A & C \\ 0 & B \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

for any y_1 and y_2. (These are all vectors.) The bottom block row is $y_2 = Bx_2$. Using the fact that B is onto, we can find at least one x_2 such that $y_2 = Bx_2$. The top block row is

$$y_1 = Ax_1 + Cx_2,$$

which we can rewrite as

$$Ax_1 = y_1 - Cx_2.$$

Using the fact that A is onto, we can find at least one x_1 that satisfies this equation. Now we're done.

d) If A and B are onto, then so is the matrix $\begin{bmatrix} A \\ B \end{bmatrix}$.

False. Let A and B both be the 1×1 matrix 1. These are each onto, but $\begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ is not.

e) If the matrix $\begin{bmatrix} A \\ B \end{bmatrix}$ is onto, then so are the matrices A and B.

True. To say that $\begin{bmatrix} A \\ B \end{bmatrix}$ is onto means that for any vector y, we can find at least one x that satisfies

$$y = \begin{bmatrix} A \\ B \end{bmatrix} x.$$

Let's use this to show that A and B are both onto. First let's consider the equation $z = Au$. We can solve this by finding an x that satisfies

$$\begin{bmatrix} z \\ 0 \end{bmatrix} = \begin{bmatrix} A \\ B \end{bmatrix} x.$$

In a similar way can solve the equation $w = Bv$ for any vector w.

f) If A is full rank and skinny, then so is the matrix $\begin{bmatrix} A \\ B \end{bmatrix}$.

True. Since the matrix A is skinny and full rank, its has zero nullspace: whenever we have $Ax = 0$, we can conclude $x = 0$. The matrix $\begin{bmatrix} A \\ B \end{bmatrix}$ is also skinny, so to show it is full rank we must show that it, too, has zero nullspace. To do this suppose that

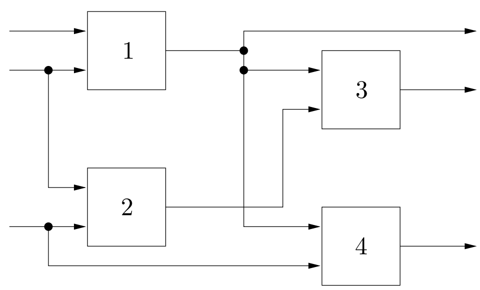
$$\begin{bmatrix} A \\ B \end{bmatrix} x = 0.$$

This means that $Ax = 0$ and $Bx = 0$. From the first, we conclude that $x = 0$. This shows that $\begin{bmatrix} A \\ B \end{bmatrix}$ is full rank.

3.500. Digital circuit gate sizing. A digital circuit consists of a set of n (logic) gates, interconnected by wires. Each gate has one or more inputs (typically between one and four), and one output, which is connected via the wires to other gate inputs and possibly to some external circuitry. When the output of gate i is connected to an input of gate j, we say that gate i *drives* gate j, or that gate j is in the *fan-out* of gate i. We describe the topology of the circuit by the *fan-out list* for each gate, which tells us which other gates the output of a gate connects to. We denote the fan-out list of gate i as $\mathrm{FO}(i) \subseteq \{1, \ldots, n\}$. We can have $\mathrm{FO}(i) = \emptyset$, which means that the output of gate i does not connect to the inputs of any of the gates $1, \ldots, n$ (presumably the output of gate i connects to some external circuitry). It's common to order the gates in such a way that each gate only drives gates with higher indices, *i.e.*, we have $\mathrm{FO}(i) \subseteq \{i+1, \ldots, n\}$. We'll assume that's the case here. (This means that the gate interconnections form a directed acyclic graph.)

To illustrate the notation, a simple digital circuit with $n = 4$ gates, each with 2 inputs, is shown below. For this circuit we have

$$\mathrm{FO}(1) = \{3, 4\}, \quad \mathrm{FO}(2) = \{3\}, \quad \mathrm{FO}(3) = \emptyset, \quad \mathrm{FO}(4) = \emptyset.$$



The 3 input signals arriving from the left are called *primary inputs*, and the 3 output signals emerging from the right are called *primary outputs* of the circuit. (You don't need to know this, however, to solve this problem.)

Each gate has a (real) *scale factor* or *size* x_i. These scale factors are the design variables in the gate sizing problem. They must satisfy $1 \leq x_i \leq x^{\max}$, where $x^{\max}$ is a given maximum allowed gate scale factor (typically on the order of 100). The total area of the circuit has the form

$$A = \sum_{i=1}^{n} a_i x_i,$$

where a_i are positive constants.

Each gate has an *input capacitance* C_i^{in}, which depends on the scale factor x_i as

$$C_i^{\text{in}} = \alpha_i x_i,$$

where α_i are positive constants.

Each gate has a *delay* d_i, which is given by

$$d_i = \beta_i + \gamma_i C_i^{\text{load}} / x_i,$$

where β_i and γ_i are positive constants, and C_i^{load} is the *load capacitance* of gate i. Note that the gate delay d_i is always larger than β_i, which can be intepreted as the minimum possible delay of gate i, achieved only in the limit as the gate scale factor becomes large.

The load capacitance of gate i is given by

$$C_i^{\text{load}} = C_i^{\text{ext}} + \sum_{j \in \text{FO}(i)} C_j^{\text{in}},$$

where C_i^{ext} is a positive constant that accounts for the capacitance of the interconnect wires and external circuitry.

We will follow a simple design method, which assigns an equal delay T to all gates in the circuit, *i.e.*, we have $d_i = T$, where $T > 0$ is given. For a given value of T, there may or may not exist a feasible design (*i.e.*, a choice of the x_i, with $1 \leq x_i \leq x^{\max}$) that yields $d_i = T$ for $i = 1, \dots, n$. We can assume, of course, that $T > \max_i \beta_i$, *i.e.*, T is larger than the largest minimum delay of the gates.

Finally, we get to the problem.

a) Explain how to find a design $x^\star \in \mathbb{R}^n$ that minimizes T, subject to a given area constraint $A \leq A^{\max}$. You can assume the fanout lists, and all constants in the problem description are known; your job is to find the scale factors x_i. Be sure to explain how you determine if the design problem is feasible, *i.e.*, whether or not there is an x that gives $d_i = T$, with $1 \leq x_i \leq x^{\max}$, and $A \leq A^{\max}$.

   Your method can involve any of the methods or concepts we have seen so far in the course. It can also involve a simple search procedure, *e.g.*, trying (many) different values of T over a range.

   Note: this problem concerns the general case, and not the simple example shown above.

b) Carry out your method on the particular circuit with data given in the file `gate_sizing_data.json`

   The fan-out lists are given as an $n \times n$ matrix `F`, with i, j entry one if $j \in \text{FO}(i)$, and zero otherwise. In other words, the ith row of `F` gives the fanout of gate i. The jth entry in the ith row is 1 if gate j is in the fan-out of gate i, and 0 otherwise.

Comment. You do not need to know anything about digital circuits; *everything* you need to know is stated above.

Solution.

a) We define the fanout matrix F as $F_{ij} = 1$, if $j \in \text{FO}(i)$, and $F_{ij} = 0$ otherwise. The matrix F is *strictly upper triangular*, since $\text{FO}(i) \subseteq \{i+1, \dots, n\}$.

   Using the formulas given above, and $d_i = T$, we have

$$\begin{aligned}
T &= d_i \\
&= \beta_i + \gamma_i \frac{C_i^{\text{load}}}{x_i} \\
&= \beta_i + \gamma_i \frac{C_i^{\text{ext}} + \sum_{j \in \text{FO}(i)} C_j^{\text{in}}}{x_i} \\
&= \beta_i + \gamma_i \frac{C_i^{\text{ext}} + \sum_{j \in \text{FO}(i)} \alpha_j x_j}{x_i}.
\end{aligned}$$

Multiplying by x_i we get the equivalent equations

$$Tx_i = \beta_i x_i + \gamma_i \left(C_i^{\text{ext}} + \sum_{j \in \text{FO}(i)} \alpha_j x_j \right),$$

which we can express in matrix form as

$$Tx = \mathbf{diag}(\beta)x + \mathbf{diag}(\gamma)C^{\text{ext}} + \mathbf{diag}(\gamma)F\,\mathbf{diag}(\alpha)x.$$

Defining

$$K = \mathbf{diag}(\beta) + \mathbf{diag}(\gamma)F\,\mathbf{diag}(\alpha),$$

we can write the equations as

$$(TI - K)x = \mathbf{diag}(\gamma)C^{\text{ext}},$$

a set of n linear equations in n unknowns. So this problem really does belong in EE263, after all.

For choices of T for which $TI - K$ is nonsingular, there is only one solution of this set of linear equations,

$$x = (TI - K)^{-1}\,\mathbf{diag}(\gamma)C^{\text{ext}}.$$

If this x happens to satisfy $1 \le x_i \le x^{\max}$, and $A = a^\mathsf{T} x \le A^{\max}$, then it is a feasible design. Our job, then, is to find the smallest T for which this occurs. If it occurs for no T, then the problem is infeasible.

Let's analyze the issue of singularity of $TI - K$. The matrix K is upper triangular, with diagonal elements β_i. So $TI - K$ is upper triangular, with diagonal elements $T - \beta_i$. But these are all positive, by our assumption. So the matrix $TI - K$ is nonsingular.

Thus, for each value of T (larger than $\max_i \beta_i$) there is exactly one possible choice of gate sizes. Among the ones that are feasible, we have to choose the one corresponding to the smallest value of T.

We can solve this problem by examing a reasonable range of values of T, and for each value, finding x. We check whether x is feasible, by looking at $\min_i x_i$, $\max_i x_i$, and A. We take our final design as the one which is feasible, and has smallest value of T. Alternatively, we can start with a value of T just a little bit larger than $\max_i \beta_i$, then increase T until we find our first feasible x, which we take as our solution.

b) The following code generates x for a range of value of T, and plots $\min_i x_i$, $\max_i x_i$, and A, versus T.

```
using LinearAlgebra

include("readclassjson.jl")

data = readclassjson("gate_sizing_data.json")
n = data["n"]
F = data["F"]
```

```
xmin = data["xmin"]
xmax = data["xmax"]
a = data["a"]
gamma = data["gamma"]
beta = data["beta"]
alpha = data["alpha"]
Cext = data["Cext"]
Amax = data["Amax"]

deltaT=0.001;
Trange= maximum(beta) + deltaT:deltaT:10;

for i = Trange
    K = diagm(beta) + diagm(gamma)*F*diagm(alpha);
    x = (i*I(n)- K) \ diagm(gamma)*Cext;

    if a'*x <= Amax && minimum(x) >= xmin && maximum(x) <= xmax
        global T = i
        break
    end
end

print("T = ", T)
```

The output of the code is

```
T= 2.519
```

Figure 1 shows how the minimum and maximum gate sizes, and the total area, vary with T, with the blue lines showing the limits. This shows that the feasible designs correspond to $2.5194 \leq T \leq 5.088$.

A few more comments about this problem:

- Since the matrix $TI - K$ is upper triangular, we can solve for x very, very quickly. In fact, if we use sparse matrix operations, we can easily compute x very quickly (seconds or less) for a problem with $n = 10^5$ gates or more. You didn't need to know this; we're just pointing it out for fun.
- The plots above show that as T increases, all of gate sizes decrease. This implies that $\min_i x_i$, $\max_i x_i$, and A all decrease as T increases. This means you can use a more efficient bisection search to find the optimal T. Again, you didn't need to know this; we're just pointing it out.

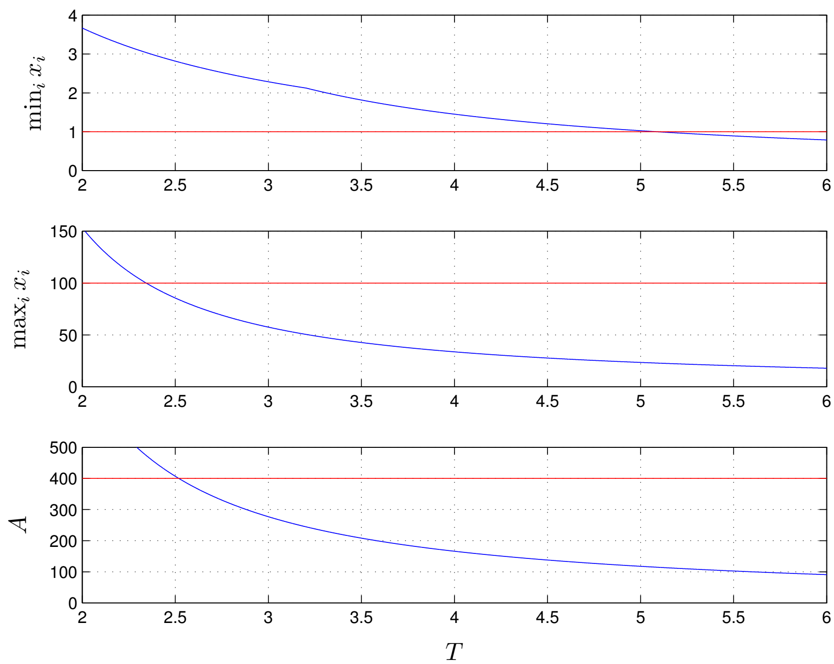


Figure 1: $\max_i x_i, \min_i x_i$, and A versus T.