TP 3: Carpooling

Summary

This TP has three parts: In the first part we describe an optimization problem motivated by a carpooling scenario and explain how this problem can be modeled as a Max-Flow instance on a weighted directed graph. In the second part, we generalize this solution, analyze it theoretically and prove that it works correctly. In the third part we implement the Ford-Fulkerson method, which we saw in class, and use it to solve some example instances of the carpooling problem.

1 The basic set-up

Four colleagues, Alice, Bob, Charlie, and Derek, live in the same neighborhood and work in the same company. They therefore decide to carpool, that is, to share a single car to commute to and from work. Somewhat complicating things, though, all four of them are partially working from home and only going to the office some days. In order to properly estimate their needs they all declare their schedule, which is summarized in the table below. Here, a 0 entry means that a person will not participate in the carpool that day (working from home), a 1 entry means that the person will participate in the carpool and is available to drive, while a 2 entry means the person would like to participate, but cannot drive that day.

	Mon	Tue	Wed	Thu	Fri
A	0	0	1	1	0
B	2	0	0	1	0
C	1	1	1	0	0
D	1	2	0	0	1

In the example above, Alice wants to commute to work on Wednesday and Thursday, and is available to drive on both days. Bob wants to commute on Monday and Thursday, but cannot drive on Monday[1], so someone else will need to drive.

What we now want to decide is **who will drive on each day**. Of course, given a matrix such as the one above it is easy to come up with some schedule: as long as every column contains at least one 1 (or only 0s), we can assign a driver. We therefore consider a **fairness** objective.

Assume that all participants prefer **not** to drive[2]. Then, in the example above it would be unreasonable to ask Alice to drive on both Wednesday and Thursday: Alice only goes to work twice, so she should be allowed to relax on at least one of these days. On the other hand, perhaps asking Charlie to drive twice is not so unreasonable?

To quantify this type of question, we define each participant's **moral debt** as follows: if person i participates on day j (that is, the entry in position $[i, j]$ of the table is non-zero), then we charge person i a debt of $\frac{1}{d_j}$, where d_j is the total number of participants on day j. The total debt of participant i is then defined as the sum of her debt over all days, with the sum **rounded-up** to the nearest integer.

For example: Alice participates on Wednesday and Thursday, each of these days has two participants, so her total debt is $\lceil \frac{1}{2} + \frac{1}{2} \rceil = 1$. Charlie participates on three days, with $3, 2$, and 2 participants respectively, so

[1] Perhaps his car is not available, or he is still hung-over from the weekend...

[2] Because this allows them to play games on their phones or do other things...

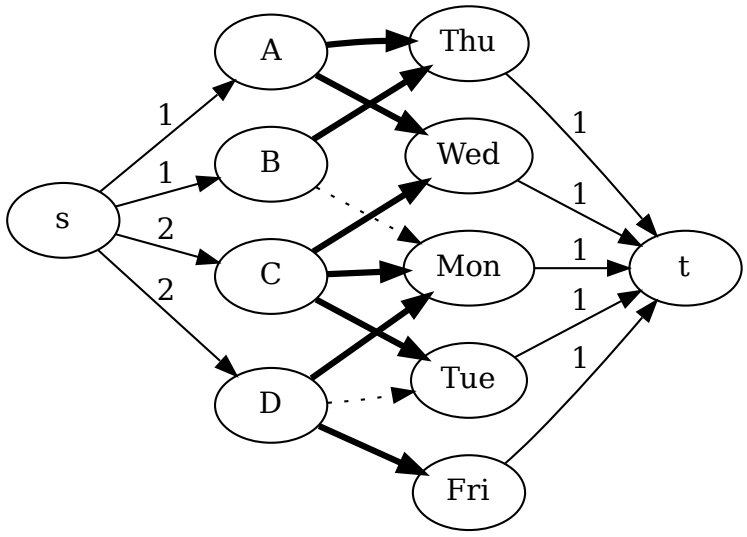

Figure 1: Max-Flow instance corresponding to our initial example. Bold arcs have capacity 1. Dotted arcs are removed from the graph and are only shown to indicate days when a participant wants to commute but cannot drive.

his debt is $\lceil \frac{1}{3} + \frac{1}{2} + \frac{1}{2} \rceil = 2$. With similar calculations, Bob has a debt of $\lceil \frac{1}{3} + \frac{1}{2} \rceil = 1$ and Derek a debt of $\lceil \frac{1}{3} + \frac{1}{2} + \frac{1}{1} \rceil = 2$.

Our goal now is to produce a schedule where for each participant i we satisfy the following conditions:

1. We only assign participant i as the driver on days j where i is available to drive (so the $[i, j]$ entry is 1 in the matrix).
2. The number of times that participant i is assigned to drive is at most equal to i's total debt.

Flows and Matchings

The main idea of this exercise is to model this scenario as a Max-Flow problem. As we discussed in class, Max-Flow is one of the main tools we can use to solve **matching** problems, such as this one where we seek to match drivers with the days they will be assigned to drive. We construct a directed graph $G = (V, A)$ as follows:

1. The set of vertices V contains a vertex x_i for each participant, a vertex y_j for each day, and two special vertices s, t.
2. For each entry $[i, j]$ of the original matrix that is equal to 1 we construct an arc from x_i to y_j. This arc has capacity 1.
3. For each participant x_i we construct an arc sx_i with capacity equal to the moral debt of this participant.
4. For each day y_j we construct an arc $y_j t$ with capacity 1.

The graph corresponding to our example is shown in Figure 1.

We claim that there is an $s \to t$ flow with value equal to the number of days if and only if there exists a fair assignment. Why is this the case?

Question 1

Use the graph to find a feasible solution to the example. Furthermore, suppose we modify the example so that Alice declares she cannot drive on Wednesday (so we change entry $[1, 3]$ from 1 to 2) and Charlie declares he will work from home on Monday (so his debt is now 1). Prove that in this case no fair solution exists. Observe that this is the case, even though all columns contain at least one 1.

2 Theoretical Analysis

2.1 Generalization to large inputs

Consider now the following generalization of the previous scenario. We are given an $n \times m$ matrix M with entries from $\{0, 1, 2\}$. This is supposed to model a situation where n participants want to carpool and are fixing a schedule over m days.

One complication of this generalization is that, while previously all participants fit in a single car, we now may need several cars per day (depending on demand). For $j \in \{1, \ldots, m\}$ we define the **total car demand** for day j as $T_j := \lceil \frac{d_j}{5} \rceil$, where d_j is, as before, the number of participants for that day. So, T_j is the number of cars (and drivers) we need on day j, assuming each car can take 5 people.

Given the above, we define the **moral debt** of agent i for day j as $\frac{T_j}{d_j}$. As previously, the total moral debt for i is the sum of i's debts over all days, rounded up.

We now define a Max-Flow instance as before with the following modification:

- For each day $j \in \{1, \ldots, m\}$ we set the capacity of the arc $y_j t$ to T_j.

Question 2

Describe the Max-Flow instance that would encode the input given below, which has 7 participants and 10 days. Observe that the sum of the 7 non-rounded debt values is equal to the sum of the total car demands.

	1	2	3	4	5	6	7	8	9	10
1	1	0	1	1	0	0	1	1	1	0
2	1	1	0	0	2	0	1	0	2	1
3	0	0	0	0	1	1	1	1	0	1
4	1	1	2	1	0	0	2	0	1	0
5	0	2	0	1	1	1	0	1	1	1
6	1	1	1	0	0	0	2	0	1	0
7	1	0	0	0	1	0	1	0	1	0

Question 3

Prove that a carpooling instance has a feasible solution if and only if the constructed directed graph has a flow of value $\sum_{j \in \{1,\ldots,m\}} T_j$.

3 Implementing the Ford-Fulkerson Algorithm

For the last part of this exercise you are given a list of a few example instances following the simple format described below. You are asked to program the following:

1. A program which reads the carpooling problem instances and constructs the directed graph we described above. Here, you are allowed to use **adjacency matrices**, as we will not be dealing with very large instances and the algorithms we will implement are not very efficient.

2. An implementation of the Ford-Fulkerson algorithm, which you will apply to the instances of the program above.

3. Finally, for instances where a flow of value $\sum_j T_j$ exists, your program should output a fair assignment. For instances where this is not the case, your program should output a cut of the graph that proves that no such assignment exists. In particular, your program should find a list of participants L who want to commute on a set of days R such that (i) no other participant is available to drive on a day of R (ii) the total rounded-up debt of participants in L is strictly smaller than the total number of cars needed for days in R. Observe that if we find such sets L, R, this proves that the carpooling instance is infeasible and also that the resulting max-flow instance has a cut smaller than $\sum_j T_j$.

Input Format

To help you test your code, you are supplied with a few instances. Each instance is in the form of a text file. The first line contains a single integer which corresponds to the number of participants n. The second line contains a single integer which corresponds to the number of days m. Then, a list of $n \times m$ integers is given, in n lines with m integers per line, separated by spaces. All these integers are from $\{0, 1, 2\}$ and have the meanings defined above, that is, if the j-th integer of the i-th line has value 1, then participant i wants to commute on day j (and similar for values $0, 2$).