Prob. 1	Prob. 2	Prob. 3	Prob. 4

Problem 1.

Let $n \times m$ be the size of the matrix. We call c_i the integer sum of the column i of the matrix, and r_i the integer sum of the row i of the matrix. We are going to show a way to compute a right rounding using max-flow algorithm.

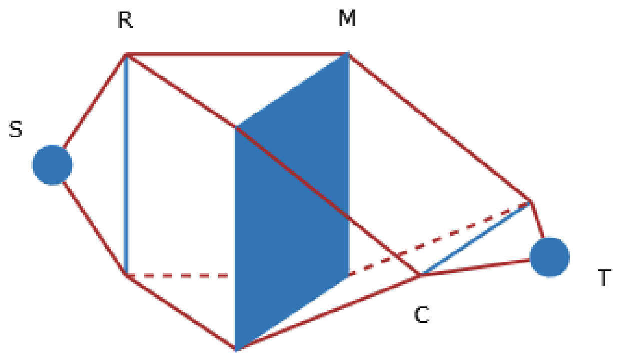

Figure 1: Blue = Vertices; Red = Edges

We construct the following graph (see Fig. 1) :

Vertices

- a source S
- a sink T
- n vertices called $R_1, ..., R_n$
- m vertices called $C_1, ..., C_m$
- n x m vertices called $M_{1,1}, ...M_{n,m}$

Edges

- n edges from S to R_i with capacity r_i
- m edges from C_i to T with capacity c_i
- n x m edges from R_i to $M_{i,j}$ with capacity 1
- n x m edges from $M_{i,j}$ to C_j with capacity 1

Our graph is made so that the not-rounded matrix gives us a possible flow of capacity $r_1 + ... + r_n$ which is maximal because it saturates both the source and the sink. However, since every capacity of an edge of the graph is an integer, the solution given by the max-flow algorithm will have integer values. That means we get our rounding with the solution of max-flow algorithm.

Furthermore our algorithm is polynomial, because the size of the graph is a linear function of the size of the input matrix and Max-Flow is polynomial.

Problem 2.

We are going to use min-cut algorithm. For this we construct the following graph (see Fig. 2) :

Vertices

- a source S
- a sink T
- n vertices called $F_1, ..., F_n$

Edges

- n edges from S to F_i with capacity b_i
- n edges from F_i to T with capacity a_i
- n x (n-1) / 2 edges from F_i to F_j with capacity c_{ij}

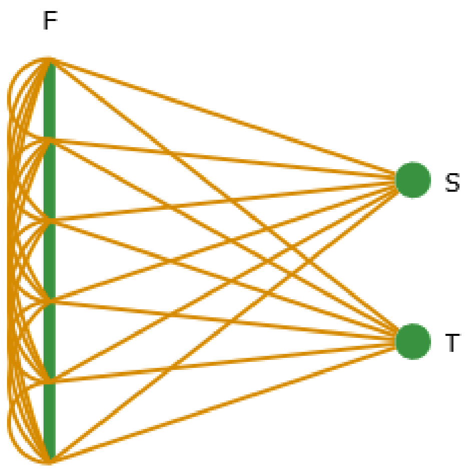

Figure 2: Green = Vertices; Yellow = Edges

We then realize that the capacity of a cut in this graph corresponds to the yearly cost of building firms linked to the source in town A and firms linked to the sink in town B. We conclude that min-cut algorithm gives an optimal repartition of the firms.

Furthermore our algorithm is polynomial, because the size of the graph is a linear function of the size of the input and Min-Cut is polynomial.

Problem 3.

1. The main idea is to convert our problem into a network flow one, which we know we can solve in polynomial time. To do so we give each directed edge a weight of one. Then we apply our network flow algorithm with source s and sink t. This gives us an integer value f of the maximum flow. We can show that there exist k edge-disjoint directed path from s to t **iff** $f \geqslant k$.

It is easy to see that if there exist k edge-disjoint directed path from s to t, then by using this k paths we will have a flow of k. Therefore the maximum flow is more than k.

Reciprocally, if $f \geqslant k$ then, as all edges have a weight of one, we can find f edge-disjoint directed path from s to t, only by using the edges that have a flow of one. Thus there exist k edge-directed paths from s to t.

2. We still want to apply a network flow algorithm but here there is some work to do with the graph to be able to do so. We are looking for node-disjoint paths, which means we need to ensure that each node can only see a flow of one. This can be done by duplicating all the nodes in our graph in order to separate the edges that are coming in a given node from the edges that are coming out. We then link the two duplicates by an edge of weight one so that the total flow in a node can be either one or zero (see figure 3).

Our new graph has $2|V|-2$ vertices and $|E|+|V|-2$ edges, so we can solve our problem in polynomial time using a network flow algorithm. It provides us with the value of the maximum flow f. We conclude using the fact that : there exist k node-disjoint directed path from s to t **iff** $f \geqslant k$.

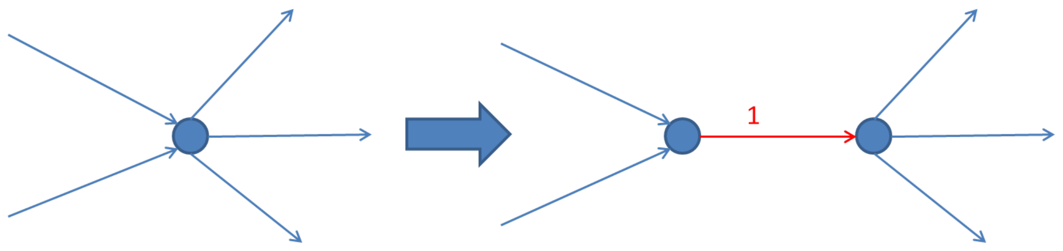

Figure 3: Graph transformation for a node

3. Here we need to reason in terms of min-cut and use the max-flow min-cut theorem to do the link with the property proved in question 1.

Let's suppose that we have a cut C between s and t such that its value c is strictly less than k : $c < k$. Let's prove that we can construct a cut of value c either between s and u or between u and t. Our cut C is able to separate either s and u or u and t. By the property of question 1., this violates the hypothesis :"there exist k edge-disjoint paths from s to u and re-exist k edge-disjoint paths from u to t".

This proves that the minimal cut between s and t has a value of at least k, which concludes the proof :

If there exist k edge-disjoint paths from s to u and re-exist k edge-disjoint paths from u to t, then there does exist k edge-disjoint directed paths from s to t.

4. Here is a counter-example.

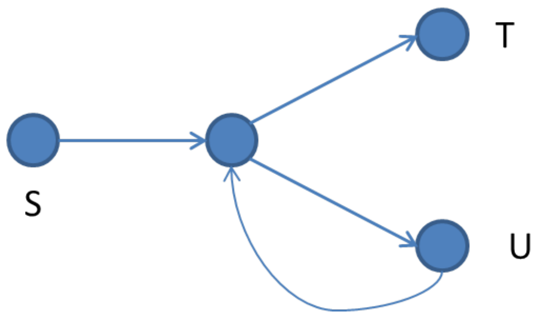

Figure 4: Counter-example with $k = 1$

Problem 4.

Let suppose that we want to classify data in a set of disjoint classes $C = C_1, ..., C_k$. To classify this data, we will present two algorithms.

The first algorithm that we present is a simple one: at a given step, if the remaining possible classes are $C' = C_1, ..., C_i$, we test if the sample is of class C_i or in one of the classes $C_1, ..., C_i - 1$ by a 2-class classification; and continue until we have found the correct class.
Classifying a datum with this algorithm takes $O(k)$ (wich is quite slow), but we need only few steps of training i.e $\Theta(k)$.
With this algorithm, if we know the probability distribution of the classes, we can sort the classes by their probabilty so that we make fewer classification steps in average.

To make the classification of a datum faster, the second algorithm that we present finds the class of a datum by dichotomic search among the classes: at a given step , if the remaining possible classes are $C' = C_i, ..., C_j$, then we determine wether the sample is in $C_i, ..., C_{i+\lfloor \frac{j-i}{2} \rfloor}$ or in $C_{i+1+\lfloor \frac{j-i}{2} \rfloor}, ..., C_j$ by a 2-class clasification.
This way, after the training, we can classify any sample really fast: we can do it in logarithmic time. However, the training takes much more time, i.e $\Theta(klog(k))$ training steps are needed.
With this algorithm, if we know the probabilty distribution of the classes, we can try to enforce that at every step the two posibilities (that the datum is in $C_i, ..., C_{i+\lfloor \frac{j-i}{2} \rfloor}$ or in $C_{i+1+\lfloor \frac{j-i}{2} \rfloor}, ..., C_j$) are as likely as possible.
To enforce that, we can use the Huffmann algorithm to build an Huffmann tree: starting from the separate classes $C_1, ..., C_k$ that will be the leaves of the classification tree, we iteratively group the two less likely nodes of the tree by setting them the same parent node and giving to this node the sum of his children nodes probabilities as his probability. The construction of a Huffmann tree can be done in $O(klog(k)$ if we need to sort the classes by probability.
With this way, we will have a really good classification system in average.