## EXERCISE DESCRIPTION

You will be asked to carry out two tasks to assess, respectively, your capacity:

1. to define a machine learning scenario and propose an approach to solve it (Task 1).
2. to implement, typically in `Python`, the code necessary to solve it (Task 2).

## PROCEDURE

1. First, you should propose me a solution for the Task 1.
2. Then, if your solution of Task 1 is convincing, I will give you as example my solution, along with a data set corresponding to (Task 2).
3. Finally, you should code the machine learning algorithm, typically in `Python`, to solve the problem (Task 2) using the data set provided.

Note that you are completely free to propose any solution to Task 1. ***Your solution could be even better than mine!!!***

**On the other hand, for Task 2, I'm only asking you to implement my solution and use my data set.** *But, again, you are completely free to implement your solution, if you think that I's better than mine, and adapt my data set to your proposed approach.*

**Depending on your success to complete Task 1 and Task 2, you will be invited to the final interview, where you will be asked to present your work.**

## CALENDAR

1. Submit by e-mail to me Task 1 solution (1-2 pages max. pdf file) no later than one week after receiving this document
2. Submit by e-mail to me Task 2 solution (source code and small report of 1-2 pages max. pdf file showing the results) no later than one week after Task 1 validation and Task 2 presentation.
3. **I will process your submission on the fly, then the faster applicants to successfully accomplish both tasks will be invited to the final interview first. *The sooner you finish, more chances you will have to get the PhD position***
4. If, unfortunately, you fail to accomplish both tasks before both deadlines, you will not be invited to the final interview
5. Final interview: ideally before end of June, date depending on your performance. It will consist of
    i. 10 min presentation of your work in Task1 and Task 2.
    ii. 10 min presentation about your PAST: your background (your training, your experience) highlight your experiences related to the PhD theses topic.
    iii. 10 min presentation about your FUTURE: your proposals / ideas about how to tackle the PhD theses topic.
    iv. Questions from the selection committee
6. ***Final decision: ideally before end of June.*** After at least. 4-5 final interviews, *and* ***based on this final interview performance, I will publish a ranking with the best 4-5 candidates.*** *The first one will have the right to take the position in priority.* Then, again, the sooner you finish tasks and interviews, more chances you will have.

# TASK 1: FORMULATE A NETWORKING OPTIMIZATION PROBLEM AS MACHINE LEARNINK TASK

## STATEMENT PROBLEM

Let be *G(N,E)* a graph representing a communication network where *N* is the node set and *E* is the link set. For each link $e \in E$, $a_e \in N$ is the starting node the link *e*, $b_e \in N$ is the ending node the link *e*, $c_e$ is the cost of using the link *e* and $u_e$ is the available capacity or bandwidth (in traffic units) of link *e*.

Let be *R* a set of connection requests, where a request $r \in R$ is characterized by the next quintuplet:

1. $s_r$: source node of the connection request *r*
2. $d_r$: destination node of the connection request *r*
3. $v_r$: traffic volume (in traffic units) node of the connection request *r (requested bandwidth)*
4. $a_r$: arrival time stamp (in time units) of connection request *r*
5. $l_r$: total life time (duration in time units) of the connection request *r*

Let be $P_{sd}$ the set of alternate paths *p* connecting the node *s* to the node *d*, where $E_p$ is simply the set of links belonging to a path $p \in P_{sd}$.

In this scenario, connection requests are arriving dynamically one after another. If the connection request *r* is admitted by the network control, $v_r$ traffic units along all the links belonging to a path connecting the node $s_r$ to the node $d_r$ will be blocked (allocated to the request *r*) during $l_r$ time units. We call that connection establishment.

Once the connection life time $l_r$ expired, the blocked $v_r$ traffic units along the path links are simply released: any following request could use this $v_r$ traffic units to set up its own connection. We call that connection release.

On the other hand, it could happen that the control plane simply rejects the connection request, typically, because there are no enough available network resources (here, available link bandwidths $u_r$) to set up the connection between $s_r$ and $d_r$.

Note that in this scenario the state of available network resources (or network state) dynamically changes after a connection establishment or a connection release. The dynamics of the connection establishments and releases depends on: (i) the stochastic process generating the connection requests, and (ii), the control algorithm deciding if a connection is admitted or not. I don't make any assumption about any of them because we will focus on solving the control admission problem in a greedy way, request after request, ignoring any information about the past established requests. The only thing that matters is the present moment: the current request *r* finding the network in a certain state of free resources (link bandwidths).

Then, the optimization problem to solve can be stated as:

For a given connection request *r* and a given network state (the values of available capacities $u_e$ at each link *e*), to find the *minimal cost path p** connecting the nodes *s* (request source) to the node *d* (request destination) meeting the capacity constraint ($v \leq u_e$) at each link $e \in E_{p*}$, that is, the links in path *p** need to have enough free capacity to accommodate the traffic volume of the request.

For the remainder of this exercise, we consider that cost $c_e = 1$, for each $e \in E$, free capacity $u_e \in \{0,1\}$, for each $e \in E$, request volume $v = 1$, and the set $P_{sd}$ consists of the three shortest paths from $s$ to $d$, regardless the values of $u_e$. Then, the problem can be reformulated as a capacitated shortest path problem:

For a given connection request $r$ and a given network state (values of available capacities (0 or 1) $u_e$ at each link $e$), to find the *shortest feasible path* $p^*$ connecting the nodes $s$ (request source) to the node $d$ (request destination), that is, with enough capacity.

Note: if path $p \in P_{sd}$ has at least one link $e$ with $u_e = 0$, the path cannot be used since the volume request is 1, that is, the path is not feasible.

## YOUR TASK

Reformulate this capacitated shortest path problem as a *supervised machine learning classification problem*. That basically means (i) to define the data set to use and (ii) tp propose a machine learning approach to solve it.

For the question of the data set, you need to answer to these questions:

1. what a sample in this data set is.
2. what the features **X** (input variables) of a sample are.
3. what the label $y$ (output variable) of a sample is.

For the question of the approach, you are free to propose any machine learning technique.

## REFORMULATION AS MACHINE LEARNING PROBLEM (SOLUTION)

***My solution***: The previous capacitated shortest path problem can be reformulated as a supervised machine learning classification problem where the samples of the data set are *R* snapshots of the networks state. Each snapshot (sample) is taken when a new connection request arrives to the network and this request needs to be accommodated (a feasible path between *s* and *d* needs to be found) onto the current network state. Then, each sample at the data set is composed by:

- an input vector **x** = { $u_e$ for each link e $\in E$, *s, d, v, t*}, that is, a vector composed by the free capacity at each link concatenated to the connection source *s*, the connection destination *d*, the connection volume (equal to 1) and the *connection total duration (we ignore this feature for this problem).*
- an output variable (the label) *y*, indicating the path $p \in P_{sd}$ selected as solution following this convention:
  - y = 0: if there is no feasible path $p \in P_{sd}$
  - y = 1: if the optimal path is the first shortest path $p \in P_{sd}$
  - y = 2: if the optimal path is the second shortest path $p \in P_{sd}$
  - y = 3: if the optimal path is the third shortest path $p \in P_{sd}$

## TASK 2: IMPLEMENT A SIMPLE MACHINE LEARNING CODE SOLVING THE PROBLEM

### THE DATA SET FILES

You will have fifteen datasets available here.

https://unice-my.sharepoint.com/:f:/g/personal/ramon_aparicio-pardo_unice_fr/EIl5U6RoQcxBkw5RZ8UqUZABBF_ygZHUYmvzqtOsG6s3zQ?e=7OaOSz

A dataset file is a *json* file where a Python dictionary `dataset` is stored. The `dataset` dictionary is composed by:

- `dataset['X']`: python list of input vectors **x** as described above (each element in the list is an input vector **x** of the sample)
- `dataset['y']`: python list of output variables *y* as described above (each element in the list is label *y* of the sample)

Actually, each data set contains 5000 snapshots of the network state. Each snapshot was taken when a new connection request arrived at the network. The whole 5000 connection requests were generated using a certain statistical model (a traffic generator). The same traffic generator was used for the generation of all the data sets. The label *y* for each sample was found using a capacitated shortest path algorithm, that is the shortest path with enough free capacity to accept the connection request.

Finally, you are free to organize these datasets as you wish to create your training set and validation set.

## THE TOPOLOGY FILE

It's available here.

https://unice-my.sharepoint.com/:f:/g/personal/ramon_aparicio-pardo_unice_fr/Ell5U6RoQcxBkw5RZ8UqUZABBF_ygZHUYmvzqtOsG6s3zQ?e=7OaOSz

The topology file is a *json* file where a Python dictionary `topology` is stored. The `topology` dictionary describes the network graph and is composed by:

- `topology ['links']`: list of ending node pairs of a link. For example, a element [a b] in this list means: link (edge) from node *a* to node *b*.
- `topology ['adjMat']`: |N|-by-|N| binary matrix. If an element in the position *(i,j)* is equal to 1, it means that there is a link (edge) from node *i* to node *j*. If an element in the position *(i,j)* is equal to 0, it means that there is no link (edge) from node *i* to node *j*. *|N| is the number of nodes.*
- `topology ['incMat']`: |E|-by-|N| matrix. If an element in the position *(e,n)* is equal to 1, it means that node *n* is the destination (final) node of link *e.* If an element in the position *(e,n)* is equal to -1, it means that node *n* is the source (starting) node of link *e.* If an element in the position *(e,n)* is equal to 0, it means that node *n* is not present in link *e. |N| is the number of nodes. |E| is the number of edges (links).*
- `topology ['list_paths']`: List of all the possible paths considered in the problem. Since we have selected the three shortest paths for each node pair, that means that we have 3|N|(|N|-1) paths on this list. Warning: this list is organized as follows:
    - 1st level: as many elements as node pairs
    - 2nd level: for each node pair: 3 elements (as many as paths per node pair)
    - 3rd level: for each path: as many elements as nodes in the path: a path is represented a sequence of nodes
- `topology ['sd_vs_path_bin_table']`: |N|(|N|-1) -by-3|N|(|N|-1) binary matrix. If an element in the position *(sd,p)* is equal to 1, it means that there is a path *p* linking node pair *sd.* 0, otherwise. Path index *p* corresponds to the order in `topology ['list_paths']`and node pair index *sd* is calculated like this *sd = i*|N|+ j*, where *i* is the source node of path *p* and *j* is the destination node of path *p. |N| is the number of nodes.*
- `topology ['link_vs_path_bin_table']`: |E| -by-3|N|(|N|-1) binary matrix. If an element in the position *(e,p)* is equal to 1, it means that link *e* belongs to path *p. .* 0, otherwise. *|N| is the number of nodes. |E| is the number of edges (links).*
- `topology ['node_vs_path_bin_table']`: |N| -by-3|N|(|N|-1) binary matrix. If an element in the position *(n,p)* is equal to 1, it means that node *n* belongs to path *p. .* 0, otherwise. *|N| is the number of nodes.*

## YOUR TASK

From the provided labeled data set and graph, code a machine learning algorithm, typically a neural network, solving the classification problem. You can use your preferred language, but I suggest you `python`.