First & last name	Cyril Bousmar
NOMA UCLouvain	63401800

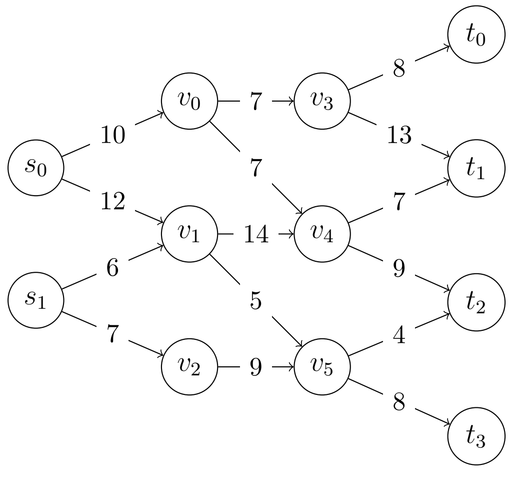
LINFO2266: Advanced Algorithms for Optimization

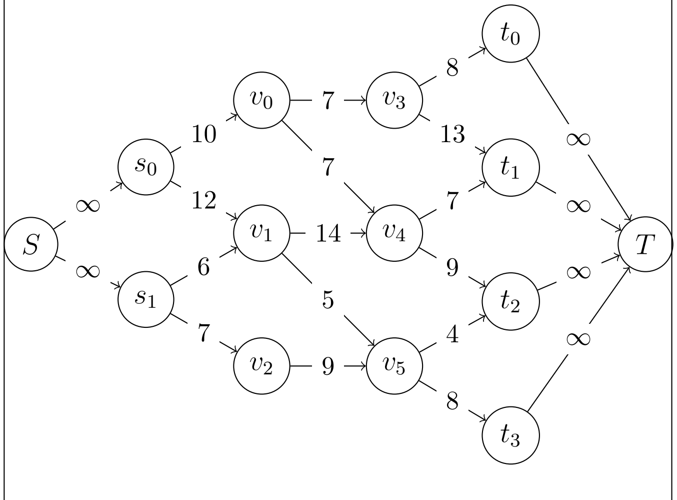
Project 3: Linear Programming and Maximum Flow

Exercise 1 Modeling flow problems

A variant of the maximum flow problem is the one with multiple sources and sinks. Fortunately, this problem can be reduced to the one with a single source and single sink and does thus not require a new algorithm.

1. How can you transform the following Maximum Flow Problem, with 2 sources (s_0, s_1) and 4 sinks (t_0, t_1, t_2, t_3), into a Maximum Flow Problem with 1 source and 1 sink? Answer with a clear illustration showing the reduction.





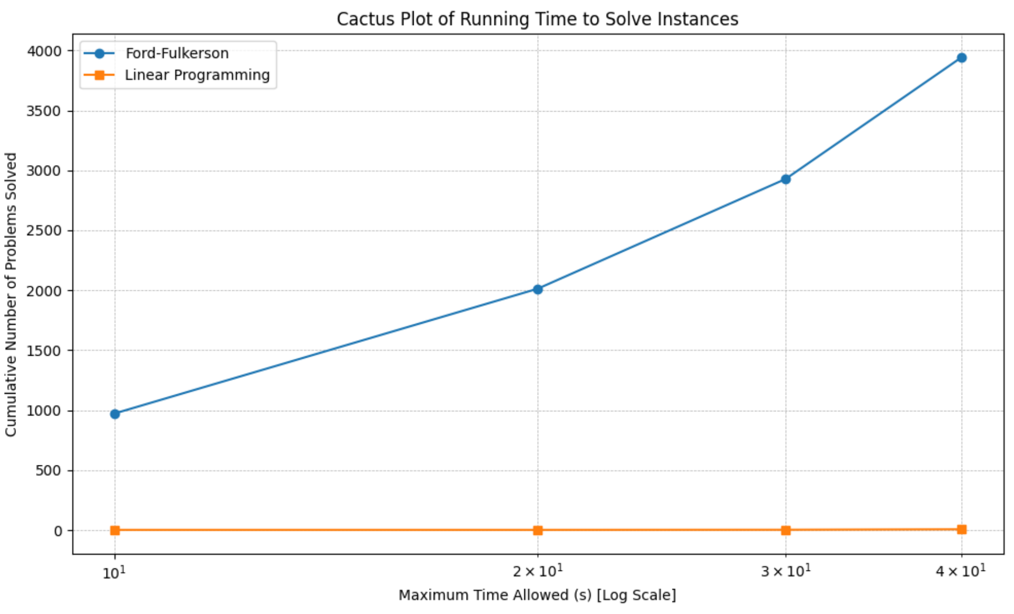
2. The `FlowNetwork` with V vertices and E edges, can be solved with a Linear Programming formulation of the form $\{\max\ c \cdot x \mid A \cdot x \leq b, x \geq 0\}$. Give the dimensions (number of variables and constraints) of the matrix A expressed using the cardinalities of V and E.

Matrix A is of dimensions $E + (V-2) \times E$, where $E + (V-2)$ is the number of constraints.

Exercise 2 Experimental comparison LP vs Dedicated Algorithm

We are interested to compare experimentally the time required by the Ford-Fukerson algorithm and the linear program for solving the maximum flow problem. We can perform time measurement on benchmark instances and report them graphically. Given the 50 instances `flowXXXX_YYYY.txt` within the `data/Flow` directory, analyze the running time of the 2 algorithms and give your observations.

1. Give the cactus plot of the running time to solve the instances between a `FlowMatrices` (+ the corresponding call to `LinearProgramming` to solve it) and a `FordFulkerson` solver. Your x axis should be the maximum time allowed (in log scale) and the y axis the cumulated number of problems solved within the maximum time. You can find one example of such a plot here.



2. Which solver performs best according to the cactus plot? Comment the differences you observe between the two methods, solely based on the plot.

Due to some issues, I could not get the computed values for the graphic on time, but we know what we were expecting.
The specialized algorithm, *Ford-Fulkerson* performs much better than *Linear Programming* which take much more time to compute instances solutions.

3. Give a comparison plot of the log-running time to solve the instances between a `FlowMatrices` (+ the corresponding call to `LinearProgramming` to solve it) and a `FordFulkerson` algorithm solver. Your x axis needs to be related to the log run time `FlowMatrices` formulation and your `y` axis to the log run time the `FordFulkerson`. Draw a diagonal line $x = y$ to ease the comparison. You can find one example of such a plot here

4. Which solver performs best according to the comparison plot? Comment the differences you observe between the two methods, solely based on the plot. What is the link with the diagonal line and what does it represent?

Any point on the diagonal represents an instance that is resolved within the same amount of time for both algorithm. Any point below this line shows that `FLowMatrices` with `LineaProgramming` performs better than `FordFulkerson`, and any point above means the opposite.

Exercise 3 Initialisation : Big-M VS Two-Phase

1. Different methods exist for initializing the simplex. One of them, the Two-Phase, was presented in the course and its code given in your project. Another, the Big-M, was left for you to implement. If you need to choose between one of those two methods for initializing the simplex in a random linear problem, which one would you choose? Why?

For each problem to solve using *Big-M*, a careful tuning of the value of M is required to achieve good performance and prevent instabilities due to its very high value.
Even if the *Two-Phase* requires the tuning of the ϵ, it seems that the same value could be used across all problems.
Moreover, splitting the problem in two phases: computation feasibility and problem optimization is simpler to read and to interpret.
Therefore, *Two-Phase* would be my pick.