Linear Programming
Frédéric Giroire

TP

Linear Program Solvers

In this class, we learn how to solve a linear program on a computer using a solver (here GLPK). Then, we use the graph and linear program libraries of SAGEMATH to solve some combinatorial problems and networking problems. We finish by modelling a research problem: In a telecom backbone network, find a routing of the demands that minimizes the energy consumption of the network.

SAGEMATH is based on the PYTHON language. A crash course of PYTHON and a list on the functions to be used is given in this document.

Installation instructions and detailed documentation can be found:
- GLPK `http://www.glpk.fr`
- SAGEMATH `http://www.sagemath.org/index.html` The software can be used *online, without installation.* But you will need to create an account.

1 Using GLPK to solve Linear Programmes

We examine here the *input file format* of GLPK, its *commands* and *output format.* We then use it to solve a maximum matching problem.

1.1 Introduction to GLPK

1.1.1 File format.

Different formats exist. We use here CPLEX format which is widely used. Example:

```
Maximize
   obj: x1 + 2 x2 + 3 x3 + x4
Subject To
   c1: - x1 + x2 + x3 + 10 x4 <= 20
   c2: x1 - 3 x2 + x3 <= 30
   c3: x2 - 3.5 x4 = 0
Bounds
   0 <= x1 <= 40
   2 <= x4 <= 3
   x3 <= 4
General
   x4
Integer
  x3
End
```

1.1.2 GLPK commands.

Do not hesitate to use `glpsol --help` to have a list and explanation of the program commands.

To launch the solver and solve the LP defined in the file prog.lp in the CPLEX file format, use: `glpsol --cpxlp prog.lp -o output.txt` The output is written in the file `output.txt`.

1.1.3 Output format.

```
Problem:
Rows:       3
Columns:    4 (2 integer, 0 binary)
Non-zeros:  9
Status:     INTEGER OPTIMAL
Objective:  obj = 76 (MAXimum)

   No.   Row name        Activity     Lower bound   Upper bound
------ ------------    ------------- ------------- -------------
     1 c1                        4.5                          20
     2 c2                       12.5                          30
     3 c3                          0             0             =

   No. Column name       Activity     Lower bound   Upper bound
------ ------------    ------------- ------------- -------------
     1 x1                         40             0            40
     2 x2                       10.5             0
     3 x3           *              4             0             4
     4 x4           *              3             2             3

Integer feasibility conditions:

INT.PE: max.abs.err. = 0.00e+00 on row 0
        max.rel.err. = 0.00e+00 on row 0
        High quality

INT.PB: max.abs.err. = 0.00e+00 on row 0
        max.rel.err. = 0.00e+00 on row 0
        High quality

End of output
```

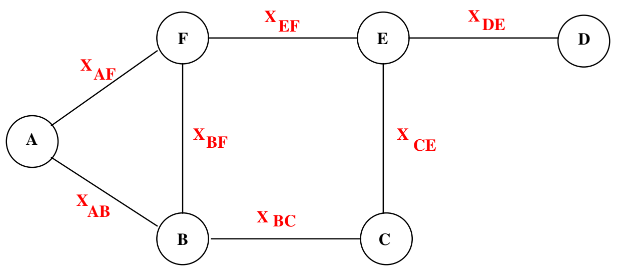
1.2 Solving a Graph Problem with GLPK

Again, we look at the Maximum Matching Problem. A small reminder.

Definition. Let $G = (V, E)$ be a graph.

- A *matching* $M \subseteq E$ is a collection of edges such that every vertex of V is incident to at most one edge of M.
- The *maximum cardinality matching* problem is to find a matching M of maximum size.

On the following graph,



The problem can be modeled with the following LP:

$$
\begin{array}{ll}
\text{Var.:} & x_{AB} = 1 \text{ if } AB \in M, \\
 & x_{AB} = 0 \text{ otherwise} \\
 & \\
\max & x_{AB} + x_{BC} + x_{CE} \\
 & + x_{DE} + x_{EF} + x_{AF} + x_{BF} \\
\text{s.t.} & \\
 & x_{AB} + x_{AF} \leq 1 \\
 & x_{AB} + x_{BC} + x_{BF} \leq 1 \\
 & x_{BC} + x_{CE} \leq 1 \\
 & x_{DE} \leq 1 \\
 & x_{CE} + x_{EF} + x_{DE} \leq 1 \\
 & x_{BF} + x_{EF} + x_{AF} \leq 1 \\
 & x_{AB}, x_{BC}, x_{CE}, x_{DE}, x_{EF}, x_{AF}, x_{BF} \geq 0 \\
 & x_{AB}, x_{BC}, x_{CE}, x_{DE}, x_{EF}, x_{AF}, x_{BF} \in \mathbb{N}
\end{array}
$$

This LP is easily translated into a format readable by GLPK:

```
Maximize
  xAB+xBC+xCE+xDE+xEF+xAF+xBF
Subject to
  c1: xAB + xAF <= 1
  c2: xAB + xBC + xBF <= 1
  c3: xBC + xCE <= 1
  c4: xDE <= 1
  c5: xCE + xEF + xDE <= 1
  c6: xBF + xEF + xAF <= 1
Binary:
  xAB, xBC, xCE, xDE, xEF, xAF, xBF
```

The output given by GLPK is

```
Problem:
Rows:       6
Columns:    13 (7 integer, 7 binary)
Non-zeros:  14
Status:     INTEGER OPTIMAL
Objective:  obj = 3 (MAXimum)

   No.   Row name        Activity     Lower bound   Upper bound
------ ------------    ------------- ------------- -------------
     1 c1                          1                             1
     2 c2                          1                             1
     3 c3                          1                             1
     4 c4                          1                             1
     5 c5                          1                             1
     6 c6                          1                             1

   No. Column name       Activity     Lower bound   Upper bound
------ ------------    ------------- ------------- -------------
     1 xAB                         0             0
     2 xBC                         1             0
     3 xCE                         0             0
     4 xDE                          1             0
     5 xEF                          0             0
     6 xAF                          1             0
     7 xBF          *               0             0             1

Integer feasibility conditions:

INT.PE: max.abs.err. = 0.00e+00 on row 0
        max.rel.err. = 0.00e+00 on row 0
        High quality

INT.PB: max.abs.err. = 0.00e+00 on row 0
        max.rel.err. = 0.00e+00 on row 0
        High quality

End of output
```

The maximum matching is of size 3. The three edges of the matching are BC, DE, and AF.

2 Using Sagemath

Concise Formating

As we have seen in the former lessons, graph problems can be written in a more concise form and for a general graph. For example, here is a formulation of the MINIMUM VERTEX COVER Problem: as:

$$\begin{array}{lll} \text{Var.:} & x_i = 1 \text{ if } i \in C, & \\ & x_i = 0 \text{ otherwise} & \\ \min & \sum_{i \in V} x_i & \\ \text{s. t.} & & \\ & x_i + x_j \geq 1 & (\forall ij \in E) \\ & x_i \in \{0,1\} & (\forall i \in V) \end{array}$$

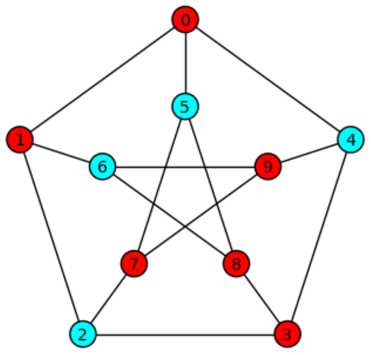
For the solver, the user has to write a small script in his language of preference to obtain the same generality. The script takes a graph as an input and outputs the LP files that will be the input of GLPK.

Here, to avoid have to script input-output we will use the graph library of the Sage software.

Solving a graph problem with Sage

Here is how to write the MINIMUM VERTEX COVER Problem using *Sage*. The program takes the Petersen graph as an input.

```
# Define a graph
  sage: g=graphs.PetersenGraph()

# Define the linear program as a minimization problem
  sage: p=MixedIntegerLinearProgram(maximization=False)

# Defining the variables
 sage: b=p.new_variable(binary=True)

# Setting the constraints
  sage: for (u,v) in g.edges(labels=None):
  ...     p.add_constraint(b[u]+b[v],min=1)

# Setting the objective function
  sage: p.set_objective(sum([b[u] for u in g]))

# Write the LP in a file under the LP format.
  sage: p.write_lp("max-matching-sage-lp.lp")

# Solving the linear program
  sage: p.solve()

# Printing the solution
  sage: b = p.get_values(b)
  sage: m = [u for u in g if b[u] == 1]
  sage: print m
  [0, 1, 3, 7, 8, 9]

# Drawing the solution
  sage: g.show(vertex_colors={"red":m})
```



3 Crash course of Python

Python can be seen as a very powerful scripting language allowing to write very quickly programs. In particular, it is a convenient language for class: all commands can be tested in a terminal as the language is interpreted (and the compilation is not necessary); the documentation is incorporated inside the terminal. You can get the doc of a function or object by writing its name followed by `?` and the source code with `??`. Additionaly, the list of methods of an object can be obtained by putting a dot after an object and pressing *tab*).

```
sage: g ?
Type:           Graph
Base Class:     <class 'sage.graphs.graph.Graph'>
String Form:    Graph on 0 vertices
Namespace:      Interactive
Length:         0
File:           /Applications/sage/local/lib/python2.6/site-packages/sage/graphs/graph.py
Docstring:
       Undirected graph.

       A graph is a set of vertices connected by edges. See also the
       Wikipedia article on graphs.

       One can very easily create a graph in Sage by typing:

          sage: g = Graph()
...

sage: g.
Display all 256 possibilities? (y or n)
g.add_cycle                             g.is_directed
g.add_edge                              g.is_drawn_free_of_edge_crossings
g.add_edges                             g.is_equitable
g.add_path                              g.is_eulerian
g.add_vertex                            g.is_even_hole_free
...
```

Below are presented quickly differences with other programming languages like C or Java, common commands, and nice tricks existing in this language.

Variables in Python are directly used without declaration.

```
x=10
name="frederic"
```

Contrary to language like C or Java, blocks are defined by indentation (and not by brackets {}). The indentation is done with a *tab* or a succession of 3 *spaces.* For example, a conditional statement would be defined in the following way in Python:

```
if x > 5 and x/2==0:
   print "OK"
```

Functions are defined by using the keyword `def`.

```
def sum(x,y):
   value = x+y
   return value
```

Lists are handled in a very natural way in Python. Defining a list:

```
l = [2,10,5]
```

Iterating on the elements of a list:

```
for x in l:
   print x
```

There is a very elegant functionality in Python called *list comprehension.* This is the possibility of creating a new list from a list.

```
l2 = [2*x for x in l if x > 4]
```

This command creates a list with the elements of l that are greater than 4.
A nice trick with tuples

```
x = (3,"2e element")
l = (x,(4,"et oui"))
for (a,b) in l:
   print "The value of ",b,"is ",a"."
```

Python commands and Sagemath functions for the exercises

The script below :
- Defines a graph g with 5 vertices, using constructor `Graph(n)`.
- Iterates on the vertices of the graphs and print their names. Note that, by default, nodes are designated by numbers.
- Adds an edge between vertices 1 and 2 with label 4, add and edge between vertices 3 and 4 with label 7, using method `g.add_edge(u,v,w)`.
- Iterates on the edges, using method `g.edges()`. Note that an edge is a triple, and that the values of its elements can be retrieved very elegantly.

```
sage: g = Graph(5)
sage: for v in g:
....:     print v
....:
0
1
2
3
4
sage: g.add_edge(1,2,4)
sage: g.add_edge(3,4,7)
sage: for (u,v,w) in g.edges():
....:     print "Label of edge ",u,v,": ",w
....:
Label of edge  1 2 :  4
Label of edge  3 4 :  7
```

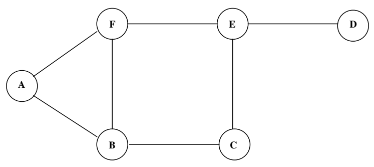
- Defining a digraph with 7 vertices, using constructor `DiGraph(n)`.
- To retrieve the in-neighbors and out-neighbors of a vertex, use the function `d.neighbors_in(v)` and `d.neighbors_out(v)`.

```
sage: d = DiGraph(7)
sage: d.add_edge(3,4,7)
sage: d.add_edge(3,1,8)
sage: d.neighbors_out(3)
[1, 4]
```

4 Exercises

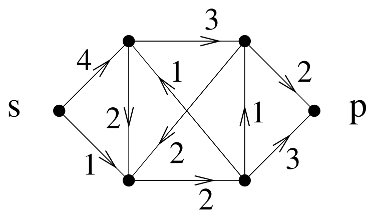
4.1 Solving linear programs using GLPK

Exercise 1 Consider the VERTEX COVER PROBLEM on the following graph:



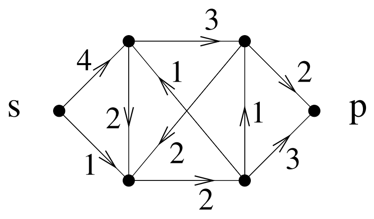
Solve the problem using GLPK.

Exercise 2 Consider the SHORTEST PATH PROBLEM on the following graph:



Solve the problem using GLPK.

Exercise 3 Consider the MAXIMUM FLOW PROBLEM on the following graph:



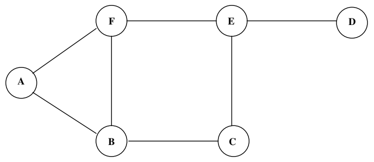
Solve the problem using GLPK.

4.2 Solving Graph and Network Problems using Sage

Exercise 4 [Maximum Independant Set with Sage] Write a Sage script to solve the MAXIMUM INDEPENDANT SET PROBLEM on the Petersen graph.

Exercise 5 [Maximum Matching with Sage]

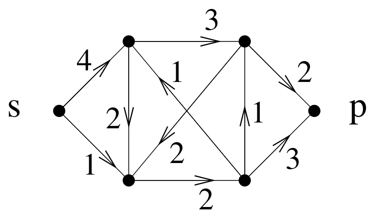
1. Write a Sage script that solves the MAXIMUM MATCHING on the following graph (use a function taking a graph as a parameter `maximumMatching(g)`, define the graph below and call the function)



2. Relax the integer property of the variables and relaunch the program on the above graph. What is the solution and its value? Comment.

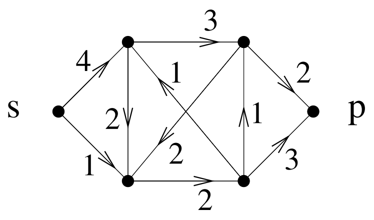
Exercise 6 [Shortest Path with Sage]

Write a Sage script that solves the SHORTEST PATH PROBLEM from the node s to the node t on the following weighted graph and plots the shortest path in red on the digraph. (use a function taking a directed graph as a parameter `shortestPath(d)`, define the digraph below with labels on the arcs, and call the function).



Exercise 7 [Flows with Sage*]

1. Write a Sage script that solves the MAXIMUM FLOW PROBLEM from the node s to the node t on the following network (use a function taking a directed graph as a parameter `maximumFlow(d)`, define the digraph below with labels on the arcs for the capacities, and call the function). Plots the flows and capacities on the arcs of the digraph.



2. There exists a second classic linear formulation of the MAXIMUM FLOW PROBLEM, the *path-formulation.*

We note $\mathcal{P}$ the set of all paths going from s to t. In this formulation, there is a variable $f_p \in \mathbb{R}^+$ per path $p \in \mathcal{P}$ giving the value of the flow on path p. The MAXIMUM FLOW PROBLEM can be written as:

$$\begin{array}{lll} \max & \sum_{p\in\mathcal{P}} f_p & \\ \text{s. t.} & & \\ & \sum_{ij\in A} \leq c_{ij} & (\forall ij \in A) \\ & f_p \in \mathbb{R}^+ & (\forall p \in \mathcal{P}) \end{array}$$

Write a second Sage function solving the MAXIMUM FLOW PROBLEM using this formulation.

3. Launch both functions on complete digraphs of sizes 3,4,5, ... Compare the implementation time. Comment.

4. A Telecom company wants to send a maximum flow on the above network, but it wants to transport flow only on paths with less than 4 edges, as longer paths imply longer delays. Which formulation the company should use? Comment.

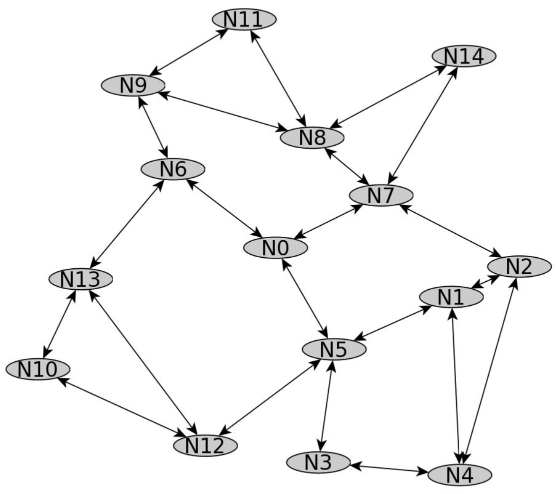
Exercise 8 [Multicommodity Flows with Sage]

1. Write a Sage script to solve the MULTICOMMODITY FLOW PROBLEM on the Petersen graph for an all-to-all demand. You should define a function taking as parameter a network and a demand matrix.

2. Consider a variant of the MULTICOMMODITY FLOW PROBLEM in which a flow should be an integer and should use only one path from its source to its destination. Write the corresponding LP. Launch it on the Petersen graph. Comment (nature of the solution, execution time).

4.3 Modelling and Solving a Research Problem

Exercise 9 [Reducing Network Energy Consumption *] A backbone network is a core network of today's Internet. It is operated by a large company such as France Telecom in France. Schematically, it is made of computers routing the traffic called routers and of optical fibers linking them. Its role is to transfer the Internet traffic from a big city to an other big city. It can be modelled as a network, that is a graph $G = (V, E)$ (note that we consider here a graph and not a digraph as in the course) with a weight function $c : E \rightarrow \mathbb{R}$ and a demand matrix D, where D_{ij} is the traffic demand from city i to city j. The capacity in our problem are all equal and of value 1. An example of such networks is given below:



We consider here the problem of reducing backbone network energy consumption. Recent studies have shown that the energy consumption of routers does not depend of the traffic load that they route, but mainly on the number of active links between them. We suppose here that links that are not used can be turned-off.

1. Model the problem of minimizing the backbone energy consumption while routing all the demands as a linear program.

2. We want to study 4-regular square grid networks that are often used by operators. Use the `Sage` software to find the routing using the minimum amount of energy for a 3 by 3 grid for an all-to-all traffic demand of intensity 1/10. What is the percentage of energy that can be saved?

3. The traffic of the operator is highly dynamic. For example, it varies during the day and is of low level at night. We want to implement an algorithm adapting to the level of trafic. To do so, we first want to assess the percentage of energy that can be saved for different levels of traffic.

   What is the maximum level traffic $\alpha_{\max}$ that can be routed by a $n \times n$-grid? Write a sage script that returns the energy saving for 5 different levels of traffic between 0 and $\alpha_{\max}$ in a 3×3 grid.

4. Try to do the same study for 4×4-grids, 5×5-grids,... what happens? Propose solutions.