TP Optimisation – session 1 – Programmation linéaire

1er janvier 2025

Objectif d'apprentissage : cette séance vise à prendre en main une interface de programmation pour résoudre programmatiquement des programmes linéaires. Nous utiliserons la bibliothèque `pulp` en Python.

> Les deux premières sections sont essentielles.

1 Exemple introductif

Lancer le script suivant et vérifier que la résolution est cohérente avec la solution proposée en cours.

```python
from pulp import *

prob = LpProblem("Exemple_simple", LpMaximize)

x1 = LpVariable("Première variable", 0)
x2 = LpVariable("Second variable", 0)

prob += x1 + x2, "Fonction objectif"

prob += 4 * x1 -     x2 <=  8, "Première contrainte"
prob += 2 * x1 +     x2 <= 10, "Seconde contrainte"
prob += 5 * x1 - 2 * x2 >= -2, "Troisième contrainte"

prob.solve()

for v in prob.variables():
    print(v.name, "=", v.varValue)

print("Objectif optimal = ", value(prob.objective))
```

2 Problème de l'élection

En partant de cette structure, répondre à la question posée pendant la séance de cours sur l'optimalité de la solution proposée pour le problème de l'élection.

```python
from pulp import *

prob = LpProblem("Election", LpMinimize)

x1 = LpVariable("Apocalypse zombie", 0)
```

```
x2 = LpVariable("Requins avec laser", 0)
x3 = LpVariable("Autoroutes pour voitures volantes", 0)
x4 = LpVariable("Vote des dauphins", 0)

prob += x1 + x2 + x3 + x4, "Cout total"

prob += -2 * x1 + 8 * x2           + 10 * x4 >=  50, "Ville"
prob +=  5 * x1 + 2 * x2                     >= 100, "Banlieues"
prob +=  3 * x1 - 5 * x2 + 10 * x3  - 2 * x4 >=  25, "Campagne"

prob.solve()

for v in prob.variables():
    print(v.name, "=", v.varValue)

print("Cout total = ", value(prob.objective))
```

De la même façon que cela est réalisé pour l'exemple https://coin-or.github.io/pulp/CaseStudies/a_blending_problem.html, résoudre le problème de l'élection en stockant toutes les données numériques dans des structures de données appropriés (listes et/ou dictionnaires), puis en construisant le programme linéaire dans un second temps. Cette façon générique de procéder qui sera à préférer pour les futurs programmes linéaires qui seront de tailles plus conséquentes.

```
from pulp import *

# Turn messages off
LpSolverDefault.msg = 0

strats = ["zombies", "requins", "autoroute", "dophins"]

ville = { "zombies": -2, "requins": 8, "autoroute": 0, "dophins": 10 }
banlieues = { "zombies": 5, "requins": 2, "autoroute": 0, "dophins": 0 }
campagne = { "zombies": 3, "requins": -5, "autoroute": 10, "dophins": -2 }

# Create the 'prob' variable to contain the problem data
prob = LpProblem("Election", LpMinimize)

# A dictionary called 'ingredient_vars' is created to contain the
# referenced Variables
vars = LpVariable.dicts("Strat", strats, 0)

# The objective function is added to 'prob' first
prob += lpSum(vars), "Cout total"

# The three constraints are added to 'prob'
prob += lpSum(ville[s] * vars[s] for s in strats) >= 50, "Ville"
prob += lpSum(banlieues[s] * vars[s] for s in strats) >= 100, "Banlieues"
prob += lpSum(campagne[s] * vars[s] for s in strats) >= 25, "Campagne"

# The problem is solved using PuLP's choice of Solver
prob.solve()
```

```
# Each of the variables is printed with it's resolved optimum value
for v in prob.variables():
    print(v.name, "=", v.varValue)

# The optimised objective function value is printed to the screen
print("Cout total = ", value(prob.objective))
```

3 Problème de flot maximum

Résoudre le problème de flot maximum entre les deux premiers sommets sur un graphe aléatoire de taille 100.

```
import networkx as nx
import matplotlib.pyplot as plt
import random as rand

n = 100
G = nx.fast_gnp_random_graph(n, 0.05, directed = True)
for e in G.edges():
    G[e[0]][e[1]]['capacity'] = rand.random()

nx.draw(G, with_labels = True)
plt.show()

# For reference
print(nx.maximum_flow_value(G, 0, 1))
```