Dipayan Chakraborty
Florent Foucaud
Malika More
Adrien Wohrer

Université Clermont Auvergne
Méthodes d'optimisation
BUT Info 2A
2023-2024

TP1 : Utilisation du solveur de programmes linéaires dans Sagemath

VERSION AVEC SOLUTION

Rappel : structures de données en Python

En Python, une *liste* est un tableau dynamique qu'on peut manipuler de la façon suivante.

```
L = [1,4,"toto"] #création d'une liste avec 3 éléments
L.append(2)      #ajout de 2 en fin de L
L.insert(3,"x")  #insère "x" en 4e position de L
print(len(L))    #affichage de la longueur de L
print(L[1])      #affichage du deuxième élément de L
print(L[-11])    #affichage du dernier élément de L
if 3 in L:       #test si 3 est dans L
for a in L:      #boucle sur les éléments de L
L.remove("y")    #supprime toutes les occurences de "y" dans L
L.pop(2)         #supprime L[2]
L.clear()        #supprime tous les éléments de L
```

Un *dictionnaire* est une structure de données permettant d'associer des clés à des valeurs, on le manipule de la façon suivante.

```
dico = {2 : "toto", 5: "titi", "x": "tata"} #création d'un dictionnaire
dico = {}                                   #création d'un dictionnaire vide
dico[5] = "tutu"                            #mise à jour de l'élément de clé 5
dico[1] = "tete"                            #ajout d'un élément de clé 1
dico.keys()                                 #liste des clés
dico.values()                               #liste des valeurs
dico.pop(2)                                 #suppresion du couple clé/valeur de clé 2
```

Rappel de cours sur les flots

Un *flot* dans un graphe $G = (V, A)$ d'une sorce s à une destination t est une assignation $f : A \to \mathbb{R}^+$ telle que :

1. pour tout arc $\overrightarrow{xy} \in A$: $f(\overrightarrow{xy}) \leq c(\overrightarrow{xy})$, *(respect des capacités)*
2. pour tout sommet $v \in V - \{s, t\}$: $\sum_{x:x\to v} f(\overrightarrow{xv}) = \sum_{y:v\to y} f(\overrightarrow{vy})$. *(conservation locale du flot)*

Et on cherche à *maximiser* la valeur $\sum_{y:s\to y} f(\overrightarrow{sy})$ du flot.

Dans le programme linéaire correspondant, on définit une variable f_a pour chaque arc a.

$$\begin{array}{lccll} \text{maximiser :} & \sum_{y:s\to y} f_{\overrightarrow{sy}} & & & \\ \text{tel que :} & f_a & \leq & c(a) & \text{pour tout arc } a \\ & f_a & \geq & 0 & \text{pour tout arc } a \\ & \sum_{x:x\to v} f_{\overrightarrow{xv}} & = & \sum_{y:v\to y} f_{\overrightarrow{vy}} & \text{pour tout sommet } v \end{array}$$

Exercice 1 (Résolution de PL avec Sagemath)**.**

Pour lancer le logiciel sous Linux, lancer la commande `sage -n` dans un terminal[1]. Cela lance une interface (basée sur `jupyter notebook`) dans votre navigateur. Vous pouvez ensuite créer une nouvelle feuille de calcul en cliquant sur `Nouveau`→`Sagemath 9.x`. Ensuite on code en Python.

Un rappel sur les listes et dictionnaires en Python est disponible en fin du document.

On peut définir et résoudre un PL dans Sagemath de la façon suivante.

```
p = MixedIntegerLinearProgram(maximization=True)#on initialise un PL de maximisation.
V = p.new_variable(real=True, nonnegative=True) #V est un dictionnaire qui contiendra les
                                                 variables (ici positives ou nulles) du PL
x, y, z = V["x"], V["y"], V["z"]                #on initialise des variables dans le
                                                 dictionnaire. C'est optionnel, on peut aussi
                                                 manipuler directement V[0], V["toto"] etc.
p.set_objective( x + y + 3*z )                  #fonction objectif
#p.set_objective( V["x"] + V["y"] + 3*V["z"] )  #version équivalente
p.add_constraint( x + 2*y <= 4 )                #contrainte 1
p.add_constraint( 5*z - y == 8 )                #contrainte 2
p.show()                                        #affiche toutes les variables et
                                                                    contraintes du PL
opt = p.solve()                                 #résoudre le PL
print(opt, p.get_values(V))                     #afficher la valeur optimale du PL et les
                                                                  valeurs des variables
```

Cela correspond au programme linéaire suivant :

$$\begin{array}{lrcrcrcl} \text{maximiser} & x & + & y & + & 3z & & \\ \text{tel que} & x & + & 2y & & & \leq & 4 \\ & & & -y & + & 5z & = & 8 \\ & x & & & & & \geq & 0 \\ & & & y & & & \geq & 0 \\ & & & & & z & \geq & 0 \end{array}$$

On peut aussi faire une somme sur une liste L de variables du PL `p` (dans une contrainte ou dans la fonction objectif) de la façon suivante :

```
p.set_objective ( p.sum(x for x in L) )
```

À vous de jouer :

(a) Résoudre le problème du régime alimentaire à 5 variables présenté dans le cours magistral (voir ci-dessous) à l'aide d'un PL.

(b) Jouer avec les contraintes et les prix des denrées, pour obtenir un régime qui vous parait intéressant et/ou cohérent. Observer l'influence des différentes contraintes sur la solution.

- Types de nutriments et apport journalier recommandé :
  protéines (56g), vitamine C (110mg), fer (2mg)
- Types d'aliments : Ananas, Banane, Carotte, Datte, Endive

aliment	prix (€/kg)	protéines (g/kg)	vitamine C (mg/kg)	fer (mg/kg)
Ananas	3.1	5	478	3
Banane	2.1	10	70	12
Carotte	1.6	7.8	20	2.4
Datte	8.7	25	4	10
Endive	3.8	13	65	8

Tableau de données par aliment

1. Dans certaines versions on peut lancer `sage` puis, dans la console de sagemath, écrire `notebook()` mais cela ne semble pas fonctionner sur les machines de l'IUT.

Soient a, b, c, d, e les quantités d'ananas, bananes, carottes, dattes, endives (en kg). On obtient le programme linéaire suivant :

$$
\begin{array}{lrcrcrcrcrcl}
\text{minimiser :} & 3.1a & + & 2.1b & + & 1.6c & + & 8.7d & + & 3.8e & & \\
\text{tel que :} & 5a & + & 10b & + & 7.8c & + & 25d & + & 13e & \geq & 56 \\
& 478a & + & 70b & + & 20c & + & 4d & + & 65e & \geq & 110 \\
& 3a & + & 12b & + & 2.4c & + & 10d & + & 8e & \geq & 2 \\
& a & & & & & & & & & \geq & 0 \\
& & & b & & & & & & & \geq & 0 \\
& & & & & c & & & & & \geq & 0 \\
& & & & & & & d & & & \geq & 0 \\
& & & & & & & & & e & \geq & 0
\end{array}
$$

Solution.

(a)

```
p = MixedIntegerLinearProgram(maximization=False)
V = p.new_variable(real=True, nonnegative=True)
a,b,c,d,e=V[0],V[1],V[2],V[3],V[4]
p.add_constraint(5*a+10*b+7.8*c+25*d+13*e >= 56)  #contrainte protéines
p.add_constraint(478*a+70*b+20*c+4*d+65*e >= 110) #contrainte vitamine C
p.add_constraint(3*a+12*b+2.4*c+10*d+8*e >= 2)    #contrainte fer
p.set_objective(3.1*a+2.1*b+1.6*c+8.7*d+3.8*e)    #fonction objectif
p.show()
opt=p.solve()
print(opt, p.get_values(V))
```

(b) La solution est de manger 7,18kg de carottes pour 11,49euros. Ce n'est pas très varié ! C'est dû au fait que les carottes contiennent assez des trois nutriments. Si on augmente le prix des carottes à 1,7 euros, on obtient une autre solution uniquement à base de... bananes. On peut aussi baisser les taux de nutriments des carottes/bananes pour favoriser les autres aliments. On remarque que la contrainte sur le fer est très facile à satisfaire puisque tous les aliments en contiennent assez. On peut aussi limiter les quantités d'aliments à 0.5kg, par exemple, pour obtenir des solutions plus intéressantes.

Exercice 2 (Flots dans les graphes)**.**

Sagemath est aussi très pratique pour manipuler des graphes. Beaucoup de fonctions de base sur les graphes sont implémentées dans Sagemath (voir `https://doc.sagemath.org/html/en/reference/graphs/index.html`). Il existe aussi beaucoup d'algorithmes avancés qui y sont implémentés. (D'ailleurs, souvent, la méthode utilisée est la PL.)

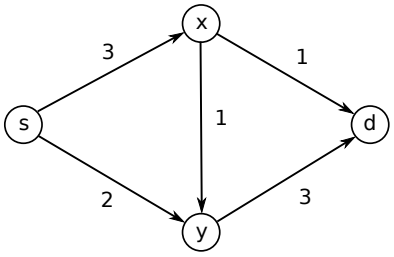
Pour créer et manipuler un graphe orienté et étiqueté dans Sagemath, on peut écrire :

```
G = DiGraph()                  #Crée un graphe orienté vide
G.add_vertex(name="toto")      #ajoute un sommet nommé "toto"
G.add_vertex(name="a")
G.add_vertex(name="b")
G.add_edge("a","b",6)          #Ajoute l'arc "a"->"b" avec l'étiquette (label) 6
for e in G.edges():            #boucle sur la liste des arcs de G
    print(e[0],e[1],e[2])      #affiche l'origine, la destination, et l'étiquette de l'arc e
for v in G.vertices():         #boucle sur la liste des sommets de G
```

```
    print(G.neighbors_out(v))  #affiche la liste des voisins sortants de v
    print(G.neighbors_in(v))   #affiche la liste des voisins entrants de v
    print(G.neighbors(v))      #affiche la liste de tous les voisins de v
    print(G.incoming_edges(v)) #affiche la liste des arcs entrant dans v
    print(G.outgoing_edges(v)) #affiche la liste des arcs sortant de v
G.show(edge_labels=True)       #dessine le graphe en affichant les étiquettes
```

À vous de jouer :

(a) Dans Sagemath, créer le graphe orienté étiqueté qui correspond au réseau de gauche de la figure 1.



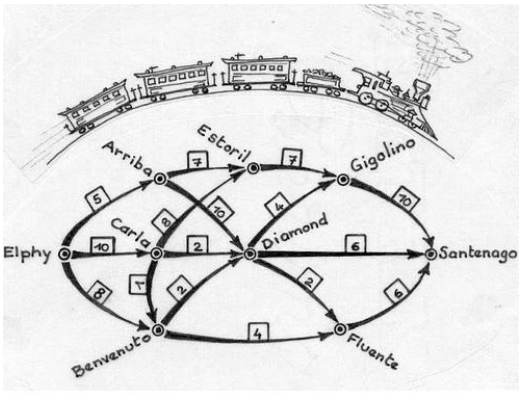


FIGURE 1 – Deux graphes

(b) Créer le programme linéaire qui résoud le problème de flot pour aller de s à d (méthode vue en cours, voir rappel en fin du document si nécessaire). Résoudre le PL et afficher la solution. Pour manipuler une variable correspondant à un arc a entre "sommet1" et "sommet2" dans le dictionnaire de variables V, vous pouvez par exemple écrire directement : `V[("sommet1","sommet2")]` ou `V["sommet1sommet2"]` ou encore `V[a]`. Vous pouvez en effet choisir librement les clés du dictionnaire de variables, à voir ce qui est plus pratique.

(c) Faites de même pour le graphe de droite de la figure 1, pour un flot qui va d'Elphy à Santenago (vous pouvez aussi directement le faire via la question (d) si vous voulez). Ce graphe peut être créé avec le code suivant :

```
G = DiGraph()
G.add_vertex(name="Elphy")
G.add_vertex(name="Arriba")
G.add_vertex(name="Carla")
G.add_vertex(name="Benvenuto")
G.add_vertex(name="Diamond")
G.add_vertex(name="Estoril")
G.add_vertex(name="Fluente")
G.add_vertex(name="Gigolino")
G.add_vertex(name="Santenago")
G.add_edge("Elphy","Arriba",5)
G.add_edge("Elphy","Carla",10)
G.add_edge("Elphy","Benvenuto",8)
G.add_edge("Carla","Estoril",8)
G.add_edge("Carla","Diamond",2)
G.add_edge("Carla","Benvenuto",1)
G.add_edge("Arriba","Estoril",7)
G.add_edge("Arriba","Diamond",10)
G.add_edge("Benvenuto","Diamond",2)
G.add_edge("Benvenuto","Fluente",4)
```

```
G.add_edge("Estoril","Gigolino",7)
G.add_edge("Diamond","Gigolino",4)
G.add_edge("Diamond","Santenago",6)
G.add_edge("Diamond","Fluente",2)
G.add_edge("Gigolino","Santenago",10)
G.add_edge("Fluente","Santenago",6)

G.show(edge_labels=True) #affichage
```

(d) Écrire une fonction générique `def flot(G,s,t)` qui prend en entrée un graphe étiqueté G (les étiquettes sont les capacités initiales), le sommet source `s` et le sommet destination `t`. Dans cette fonction, on définit et on résout le PL associé au problème du flot dans G. La fonction renvoie la solution optimale sous forme d'un couple (graphe étiqueté, valeur optimale du flot), où les étiquettes des arcs représentent les valeurs de flot pour ces arcs.

Solution.

(a)

```
G=DiGraph()
G.add_edge("s","x",label="3")
G.add_edge("s","y",label="2")
G.add_edge("x","y",label="1")
G.add_edge("x","d",label="1")
G.add_edge("y","d",label="3")
```

(b)

```
p = MixedIntegerLinearProgram(maximization=True)
V = p.new_variable(real=True, nonnegative=True)

#fonction objectif: par exemple la valeur des arcs qui sortent de s
p.set_objective( V["sx"] + V["sy"] )

#contraintes de capacités des arcs
p.add_constraint( V["sx"]<=3 )
p.add_constraint( V["sy"]<=2 )
p.add_constraint( V["xy"]<=1 )
p.add_constraint( V["xd"]<=1 )
p.add_constraint( V["yd"]<=3 )

#contraintes d'égalité de flot sur chaqie sommet
p.add_constraint( V["sx"] == V["xy"] + V["xd"]) #sommet x
p.add_constraint( V["sy"] + V["xy"] == V["yd"]) #sommet y

p.show()
opt = p.solve()
print("flot optimal: " + str(opt), " --- valeurs du flot: ", p.get_values(V))
```

Un flot optimal a valeur 4.

(c) Le PL est similaire à (b), mais avec 16 contraintes pour les capacités des arcs, et 7 contraintes pour l'égalité du flot sur les sommets. Le flot optimal a une valeur de 20.

(d)

```
def flot(G,s,t):
    p = MixedIntegerLinearProgram(maximization=True)
    V = p.new_variable(real=True, nonnegative=True)

    p.set_objective(p.sum(V[e] for e in G.outgoing_edges(s)))

    for e in G.edges():
        p.add_constraint(V[e] <= e[2])

    for x in G.vertices():
        if x != s and x != t:
            p.add_constraint(p.sum(V[e] for e in G.incoming_edges(x)) ==
                                                    p.sum(V[e] for e in G.outgoing_edges(x)))

    p.show()
    opt = p.solve()
    print("flot optimal: " + str(opt), "--- valeurs du flot: ", p.get_values(V))

    return opt
```