TP- Utilisation GLPK et de son langage de modélisation MathProg

Avant de commencer

- le lien de téléchargement est:
  - https://sourceforge.net/projects/winglpk/?source=typ_redirect

- Télécharger winglpk-4.65
- On a un zip. Copier le zip sur `C:\` puis déziper.
- Il est alors créé le répertoire : `C:\winglpk-4.65`
- L'exécutable se trouve dans `w32` ou `w64` selon votre ordinateur 32 ou 64 bits
- Modifier la variable d'environnement PATH
- Lui rajouter : …; `C:\winglpk-4.65\glpk-4.65\w64`
- Ouvrir une fenêtre « invite de commande »

Vérifiez que le solveur GLPK est accessible en tapant : `glpsol --help`

1. Utilisation de GLPK

Exercice 1 – Langage de modélisation MathProg

On veut modéliser et résoudre le problème :

$$\begin{cases} \max z = 3x_1 + 5x_2 \\ x_1 \le 4 \\ x_2 \le 6 \\ 3x_1 + 2x_2 \le 18 \\ x_1 \ge 0 \quad x_2 \ge 0 \end{cases}$$

a- Modélisation par le langage MathProg

A l'aide d'un éditeur de texte, créer le fichier texte suivant :

```
# modèle ex1.mod
var x1 >=0;
var x2 >=0;

maximize z :
    3*x1 + 5*x2;


subject to
C1: x1 <= 4;
C2: x2 <= 6;
C3: 3*x1 + 2*x2 <= 18;
```

.

b- Résolution par le « solveur » glpk

Lancer la commande : `glpsol --model ex1.mod`

Maintenant, dans ce format, on peut aussi gérer ses propres éditions. Rajouter `solve ;` pour résoudre le problème puis faites les éditions voulues.

```
# modèle ex1.mod au format MathProg
var x1 >=0;
var x2 >=0;

maximize z :
    3*x1 + 5*x2;


subject to
C1: x1 <= 4;
C2: x2 <= 6;
C3: 3*x1 + 2*x2 <= 18;

printf "lancement du solve \n";
solve;
# display fonction objectif et solution;
printf "x1=%f\n", x1;
printf "x2=%f\n", x2;
printf "z=%f\n", z;
```

Vous pouvez ensuite résoudre le contenu de ce fichier par la même commande :

```
glpsol --model ex1.mod
```

c- Vérifier le résultat obtenu en faisant la résolution graphique du problème

Exercice 2 : Généralisons – Utilisation de paramètres (données)

On veut modéliser et résoudre le problème du sac à dos :

$$\begin{cases} \max u = \sum_{i=1}^{n} C_i x_i \\ \sum_{i=1}^{n} A_i x_i \le B \\ x_i \in \{0,1\} \quad i = 1 \cdots n \end{cases}$$

1- Ecrire un fichier qui représente ce modèle (fichier texte de nom sac_a_dos.mod)

```
# Sac-à-dos en 0/1
param n; # Nombre d'objets
param C{i in 1..n}; # utilité de l'objet i
param A{i in 1..n}; # poids de l'objet i
param B;  # Capacité du sac

# Variable
```

```
var x{1..n} binary;

# Objectif
maximize z: sum{i in 1..n} C[i]*x[i];

# Contrainte
subject to
capa: sum{i in 1..n} A[i]*x[i] <= B;

printf "---- Avant résolution -----\n" ;

solve ;
display z ;
display x ;
```

2- Ecrire un fichier qui contient des données pour ce modèle (fichier texte de nom sac_a_dos.dat)

```
# Un fichier de donnees pour le probleme de sac a dos
data;
param n := 5;

param C :=  1 12
            2 15
            3 5
            4 16
            5 17;

param A :=  1 2
            2 6
            3 1
            4 7
            5 8;

param B := 20;

end;
```

3- Taper la commande :

```
glpsol --model sac_a_dos.mod --data sac_a_dos.dat
```

4- Déterminer la valeur optimale de la relaxation continue du précédent problème :
Les variables binaires 0-1 sont relâchées en variables continues entre 0 et 1 (bornes incluses).

```
glpsol --model sac_a_dos.mod --data sac_a_dos.dat --nomip
```

Comparer les valeurs obtenues.

5- L'intérêt d'avoir un modèle avec paramètres (et non des données en « dur ») est qu'on peut changer les données. Ecrire un autre fichier de données sac_a_dos2.dat avec par exemple n=10 et des données que vous choisissez et exécuter .

Exercice 3. Modéliser un programme linéaire quelconque avec m contraintes et n variables.
Soit le problème (P) suivant :

Max $z=\Sigma_{i=1,\ldots,n}\, c_i x_i$

Sous contraintes $\Sigma_{i=1,\ldots,n}\, a_{ki} x_i \leq b_k$ (k=1,…,m)
$x_i \geq 0$ (i=1,…,n)

1° Ecrivons le problème (P) en MathProg.

```
# programme linéaire quelconque
param n; # Nombre de variables
param m ; # Nombre de contraintes
param C{i in 1..n}; # utilité de l'objet i
param A{k in 1..m,i in 1..n}; # paramètres des contraintes
param B{k in 1..m};  # second membre

# Variable
var x{1..n} >=0;

# Objectif
maximize z: sum{i in 1..n} C[i]*x[i];

# Contrainte
subject to
contr{k in 1..m}: sum{i in 1..n} A[k,i]*x[i] <= B[k];

printf "---- Avant résolution -----\n" ;

solve ;
display z ;
display x ;
```

2° Modélisons les paramètres dans le cas de l'exercice 1

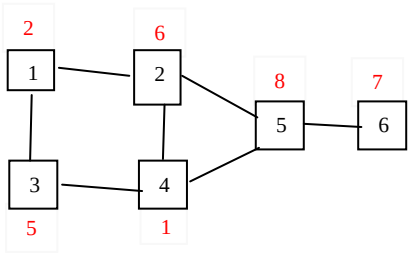
```
data;
param n := 2;
param m := 3;

param C :=
1    3
2    5
;

param A :   1 2 :=
1           1 0
2           0 1
3           3 2
;

param B :=

1           4
2           6
3           18
;

end;
```

3° Tester et vérifier que l'on retrouve bien les résultats de l'exercice 1

2. Exercices de modélisation

Exercice 4 – Problème du stable de poids maximum dans un graphe – Problème en variables binaires et contraintes d'exclusion

Soit un graphe G=(V,E). V les sommets, E les arêtes. Les sommets $u \in V$ sont munis de poids $c_u \geq 0$.
Un stable de G est un ensemble de sommets non adjacents 2 à 2 (non reliés par une arête). Le poids d'un stable est la somme des poids des sommets du stable.
Exemple : sommets V={1,2,… ,6} munis des poids {2,6,5,1,8,7}.



Voici quelques stables : S=∅, S={1},S={1,4},S={3,5},S={2,3,6}. Leurs poids sont respectivement : 0,2,3,13,18.

1° Modéliser par une variable binaire 0-1 le fait qu'un sommet $u \in V$ est (ou n'est pas) dans le stable S cherché.

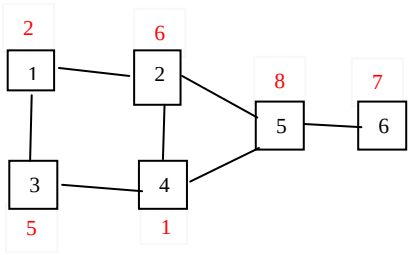
2° Modéliser par des contraintes linéaires le fait que deux sommets adjacents (reliés par une arête) ne peuvent pas être dans le stable S cherché en même temps.

3° On a déjà défini les contraintes que doivent vérifier les sommets pour appartenir au même stable S (question 2). Il ne reste plus qu'à définir l'objectif.
Modéliser le problème du stable de poids maximum qui consiste à chercher parmi tous les stables de G un stable de poids maximum.

4° Créer des instances et tester.

Exercice 5 – Problème de recouvrement des arêtes d'un graphe par les sommets

Soit un graphe G=(V,E). V les sommets, E les arêtes. Les sommets $u \in V$ sont munis de poids $c_u \geq 0$.
Un ensemble R de sommets de G tel que toute arête de G a au moins une de ses extrémités dans R est appelé recouvrement. Le poids de R est la somme des poids des sommets de R. On cherche un recouvrement de poids minimum.
Exemple : sommets V={1,2,… ,6} munis des poids {2,6,5,1,8,7}.



Voici quelques recouvrements : R=V, R={2,3,6}, R={1,4,5}, R={1,4,6}. Leurs poids sont respectivement : 29, 18, 11, 10.

1° Modéliser par une variable binaire 0-1 le fait qu'un sommet $u \in V$ est (ou n'est pas) dans le recouvrement R cherché.

2° Modéliser par des contraintes linéaires le fait que chaque arête doit être recouverte.

3° On a déjà défini les contraintes que doivent vérifier les sommets d'un recouvrement (question 2). Il ne reste plus qu'à définir l'objectif.
Modéliser le problème de la recherche d'un recouvrement de poids minimum qui consiste à chercher parmi tous les recouvrements de G un recouvrement de poids minimum.

4° Créer des instances et tester.

Exercice 6 – problème d'affectation

On a n tâches et n ouvriers. On doit affecter chaque tâche à chaque ouvrier. Une tâche j exécutée par l'ouvrier i coûte D_{ij}. La matrice D donne tous les coûts possibles. On veut minimiser le coût total d'affectation. Modéliser le problème. Penser à des variables de décision à 2 indices.
Ecrire le programme et tester le cas suivant :

D (coûts)	Tâche 1	Tâche 2	Tâche 3	Tâche 4
Ouvrier 1	8	3	1	5
Ouvrier 2	11	7	1	6
Ouvrier 3	7	8	6	8
Ouvrier 4	11	6	4	9

Exercice 7 – problème d'affectation - Trier une liste par un programme linéaire

On dispose d'une liste D de n nombres entiers. On veut faire trier cette liste en ordre croissant par un programme linéaire. Ce programme aura (notamment) pour variables v_i i=1 à n où v_i est la valeur du nombre en position i dans la liste triée : $v_1 \leq v_2 \leq \ldots \leq v_n$.
Exemple :
`D=[6,3,4]` $\Rightarrow$ `v=[3,4,6]`

Une astuce recommandée est la suivante : on remarque que si on a une liste D et une liste C de coefficients décroissants pour minimiser $\Sigma_{i=1,\ldots,n} D_i C_i$ il faut que D soit rangé dans l'ordre croissant.
Exemple :
`C=[3,2,1] D=[6,3,4] 3×6+2×3+1×4=28`
`C=[3,2,1] D=[3,4,6] 3×3+2×4+1×6=23`

Ce problème de tri revient à un problème d'affectation . Il faut affecter les nombres D_i (i=1,…,n) à une position j (j=1,…,n) de telle façon que les nombres soient affectés dans l'ordre croissant : le plus petit en 1ère position, le deuxième plus petit en 2ème position etc….

Exercice 8 – Problème d'ouverture d'entrepôts

Le problème est le suivant.
Etant donnés :
- n : nombre de clients
- m : nombre de dépôts

- D_{ij} : coût de raccordement entre le client i et le dépôt j
- C_j : coût d'ouverture du dépôt j

On veut ouvrir des dépôts et affecter chaque client à un dépôt ouvert. L'objectif à minimiser est la somme des coûts d'ouverture des dépôts et des coûts de raccordement des clients aux dépôts auxquels ils sont affectés.

Proposez un modèle pour ce problème. Il faut deux familles de variables binaires (0-1) qui sont :
-variables pour décider de l'ouverture d'un dépôt
-variables pour modéliser l'affectation d'un client i à un dépôt j.

Tester le modèle sur l'instance suivante : n=5, m=3,

Coût construction	Entrepôt 1	Entrepôt 2	Entrepôt 3
	1	4	1

Coût raccordement	Client 1	Client 2	Client 3	Client 4	Client 5
Entrepôt 1	1	3	8	1	1
Entrepôt 2	7	4	2	2	5
Entrepôt 3	2	5	7	4	4

Exercice 9 – Fonction pseudo-booléenne quadratique – Linéarisation

Soit une fonction $f(x)=\Sigma_{i=1,\ldots,n}\, c_{ii}\, x_i \; + =\Sigma_{i\neq j=1,\ldots,n}\, c_{ij}\, x_i\, x_j$ quadratique de variables $x_i \in \{0,1\}$. On doit minimiser $f(x)$ c'est-à-dire trouver une instanciation des variables x telle que $f(x)$ soit minimum. On veut résoudre ce problème par un programme linéaire.
Pour cela, à chaque produit $x_i\, x_j$ on substitue une variable y_{ij} prenant les mêmes valeurs que le produit.
1° Vérifier que les 4 contraintes suivantes, liant x et y, suffisent à cela :
$y_{ij} \geq 0$; $y_{ij} \leq x_i$; $y_{ij} \leq x_j$; $y_{ij} \geq x_i + x_j - 1$

2° Soit le problème suivant :
min $f(x) = -x_1-x_2-x_3-x_4+3x_1x_2-3x_1x_3+x_1x_4+2x_2x_3-2x_2x_4+2x_3x_4$ avec $x_1, x_2, x_3, x_4 \in \{0, 1\}$
Ecrire le programme linéaire correspondant et résoudre avec GLPK

Exercice 10 – Ordonnancement de tâches sur une machine

n tâches doivent être exécutées sur une machine unique. Chaque tâche i, a une durée d(i), une date d'échéance e(i). Les tâches s'exécutent sur une unique machine. Les tâches ne sont pas sécables (une tâche commencée va jusqu'à son terme). Si une tâche se termine après sa date d'échéance, elle est en retard. Le retard d'une tâche i est R(i)=max{0, t(i)-e(i)}, où t(i) est la date de fin de la tâche i.
Il faut trouver un ordonnancement des tâches qui minimise la somme des retards des tâches.
Modéliser ce problème par un modèle mathématique en variables mixtes :
variables binaires x_{ij}=1 si tâche i avant j et x_{ij}=0 sinon,
variables continues t_i date de fin de la tâche i et R_i retard de la tâche i.

Résoudre l'exemple suivant.

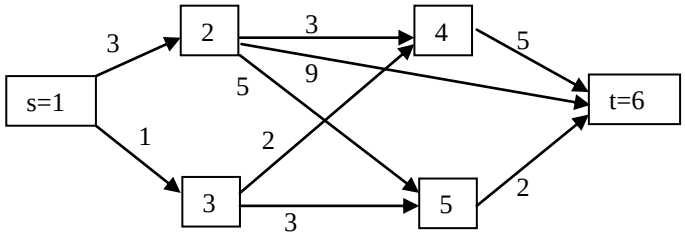
Exemple :
n=5.

Tâches	1	2	3	4	5
Durée d	4	3	7	2	2
Echéances e	5	6	8	8	17

L'ordonnancement optimal est 1, 2, 4, 3, 5 . Le retard total=11.

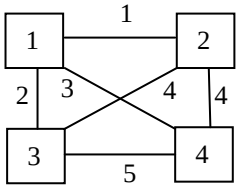
Exercice 11 – Plus court chemin dans un graphe

Soit un graphe orienté dont les arcs sont munis de poids positifs et 2 sommets *s* et *t* de ce graphe. On veut trouver le plus court chemin de *s* à *t*.
Le graphe a *n* sommets et est défini par sa matrice d'adjacence *E* de *n* lignes et *n* colonnes et la matrice *C* de *n* lignes et *n* colonnes des coûts des arcs.
Ecrire le modèle (linéaire) permettant de résoudre ce problème. Tester votre modèle sur l'instance suivante :



Exercice 12 – Bipartition de graphe et linéarisation

Un graphe complet d'ordre n avec des poids positifs ou nuls sur les arêtes. On veut faire une partition des sommets du graphe en 2 paquets. Le paquet 1 contient k sommets et le paquet 2 n-k sommets. Le coût d'une partition est la somme des poids des arêtes « coupées » c'est-à-dire avec une extrémité dans le paquet 1 et l'autre dans le paquet 2. On veut faire une partition de coût minimum.
On définit, pour chaque sommet *i*, la variable x_i binaire telle que $x_i = 1$ si le sommet *i* est dans le paquet 1 et 0 sinon.
1-Montrer que l'arête [i,j] est « coupée » si et seulement si $x_i(1- x_j)+ x_j(1- x_i) = 1$.
2-En déduire la fonction objectif du problème à minimiser.
3-Cette fonction objectif contient des produits $x_i\ x_j$ (i<j). Pour linéariser chaque produit, on introduit une variable y_{ij} qui vaudra $x_i\ x_j$. Proposer les contraintes linéaires liant la variable y_{ij} et les variables x_i et x_j et qui feront que l'égalité $y_{ij} = x_i\ x_j$ sera bien réalisée quand x_i et x_j sont binaires.
4-Ecrire le modèle linéarisé et tester l'exemple suivant avec k=2 :



Exercice 13 – Voyageur de commerce

Un voyageur de commerce doit visiter n villes et revenir à son point de départ. Les villes sont numérotées de 1 à n. Le point de départ est la ville 1. Les distances entre les villes sont données dans une matrice D . Le voyageur de commerce désire faire un tour de longueur minimale.

Proposer plusieurs modèles (voir ci-dessous) et résoudre l'instance suivante avec GLPK. Comparer les modèles au niveau de la relaxation continue et du temps de calcul.

```
data;
param n := 9;

param D :  1 2 3 4 5 6 7 8 9 :=
      1      0    796   482   640   583   449   801   244   876
      2      796  0     685   995   212   562   11    900   79
      3      482  685   0     309   463   761   701   538   763
      4      640  995   309   0     766   921   1011  405   1074
      5      583  212   463   766   0     338   229   676   292
      6      449  562   761   921   338   0     579   699   569
      7       801 11    701   1011  229   579   0     908   91
      8      244  900   538   405   676   699   908   0     969
      9       876 79    763   1074  292   569   91    969   0;

end;
```

- modèle 1 : dans le tour, chaque ville occupe une position : la 1ère ou la 2ème ou la 3ème etc … . Contraintes linéaires mais l'objectif quadratique sera à linéariser. Proposer une linéarisation basique et des améliorations de cette linéarisation.
- modèle 2 : basé sur les arcs utilisés (parcourus). Ce modèle entraine le problème des sous-tours qu'il faudra éliminer. Comment éliminer les sous-tours ?

Exercice 14 – Sudoku

Proposez un modèle et un fichier de données pour trouver la solution de la grille de Sudoku ci-dessous. Affichez le résultat.

7			**2**		**9**			**1**
4	**9**	**1**		**3**		**8**	**6**	**2**
	8						**9**	
5			**6**		**3**			**8**
3	**6**	**2**				**9**	**5**	**4**
8			**9**		**4**			**3**
	5						**1**	
6	**7**	**8**		**4**		**2**	**3**	**9**
1			**3**		**6**			**5**