# TPs : Prog. Linéaire en Nombres Entiers avec SCIP

On rappelle qu'un programme linéaire est défini par :

- un ensemble de variables ;
- une fonction *objectif*, fonction linéaire de ces variables à maximiser ou minimiser ;
- et un ensemble de contraintes linéaires sur les valeurs que peuvent prendre ces variables.

De nombreux logiciels permettent de résoudre de tels problèmes, et il existe plusieurs formats de fichiers pour représenter des problèmes d'optimisation combinatoire afin de les résoudre avec ces logiciels.

Nous en verrons deux :

- le langage LP (brièvement), qui permet essentiellement d'écrire un objectif et des contraintes linéaire ;
- le langage ZIMPL, de plus haut niveau, qui offre des constructions permettant de décrire de manière plus concise certains problèmes courants d'optimisation combinatoire.

## Un exemple en langage LP

Le programme ci-contre est contenu dans le fichier exple_simple.lp sur Moodle. La syntaxe est très simple, quelques mots-clefs délimitent plusieurs sections :

\ commence une ligne de commentaires

`Maximize` ou `Minimize` annonce la fonction objectif à optimiser

`Subject To` annonce les contraintes, chaque contrainte doit avoir un label (ici c et c1)

`Bounds` annonce les restrictions sur les domaines des variables

`Binary` annonce la liste des variables binaires

`Generals` annonce la liste des autres variables entières

`End` marque la fin du programme

Les variables qui ne sont pas dans les listes `Binary` et `Generals` sont autorisées à avoir des valeurs non entières. Pour le programme ci-contre, l'optimum est atteint pour $x_1 = 1.5$, $x_2 = 0$, $Y = 7$.

```
\ LP format example
Maximize
  3 x1 + x2 + Y
Subject To
  c: x1 + x2 = 1.5
  c1: 2 x1 + 5 x2 + Y <= 10.5
Bounds
  0 <= x1 <= 5
  Y >= 2
Generals
  Y
Binary
  x2
End
```

Une description plus précise du format LP se trouve à l'adresse : lpsolve.sourceforge.net

## Prise en main du solveur

Nous utiliserons le logiciel SCIP. Sur les salles de TPs informatique de la FSI, la version 8 est installée sous linux / debian.

*Note importante : si vous souhaitez installer SCIP sur votre ordinateur personnel, faites-le* ***en dehors des heures de TPs****. Faites la première séance sur les ordinateurs de la salle de TPs, où SCIP est déjà installé. Voir en fin de document quelques conseils sur l'installation et l'utilisation à distance d'un PC de la FSI.*

Le programme fonctionne en mode «ligne de commande». Il est plus facile de se placer au préalable dans le répertoire où se trouvent les fichiers contenant les programmes linéaires. Par ailleurs, pour avoir un historique des commandes accessible avec les flèches durant l'interaction avec SCIP, on peut utiliser la commande `rlwrap` suivie du nom de l'exécutable. L'invite de commande du solveur est `SCIP>`

**Exercice 0.** Créer un nouveau répertoire dans votre *home* pour les TPs d'Algo. Avancée. Récupérer sur Moodle le fichier exple_simple.lp. Lancer un terminal, puis lancer la séquence de commandes suivantes :

`cd` *chemin vers répertoire de travail* . . . . . . . . . . . . . *# se placer dans le repertoire des TPs*
`rlwrap scip` . . . . . . . . . . . . . . . . . . . *# lance le solveur, avec historique de commandes*
`read exple_simple.lp` . . . . . . . . . . . . . . . . . . . . . *# lit la description d'un problème*
`optimize` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . *# lance la résolution*
*lorsque le solveur s'arrête, ou est arrêté, on peut voir si la résolution s'est bien passée sur la ligne commençant par `SCIP Status` : sur notre exemple, on doit avoir « problem is solved [optimal solution found]»*
`display solution` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . *# affiche la solution*
*on peut voir a valeur de la fonction à optimiser ('objective') et les valeurs affectées aux variables non nulles (donc ici `x2`=0), ainsi que leurs contributions à la fonction objective (obj :xx).*
`quit` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . *# termine SCIP*

**Exercice 1.** On s'intéresse au problème du SAC-À-DOS. Écrire, dans un fichier avec le suffixe `.lp`, un programme linéaire représentant l'instance ci-contre, et le résoudre avec SCIP. Quel est la valeur du sac optimal, et quels objets contient-il ?

Capacité 20kg,

| num. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| poids | 11 | 7 | 5 | 5 | 4 | 3 | 3 | 2 | 2 | 2 | 2 | 1 |
| valeur | 20 | 10 | 25 | 11 | 5 | 50 | 15 | 12 | 6 | 5 | 4 | 30 |

# Le langage ZIMPL

On va maintenant utiliser un langage de plus haut niveau pour décrire les contraintes : ZIMPL est un langage propre à SCIP depuis la version 3; il permet des constructions beaucoup plus concises et proches du langage de modélisation mathématique. Le manuel du langage est disponible sur la page web de ZIMPL, il est aussi sur Moodle. À la lecture d'un programme ayant la terminaison `.zpl`, SCIP appelle un traducteur qui génère automatiquement[1] l'instance en langage LP.

Le programme ci-dessous est contenu dans le fichier sac-a-dos-12.zpl.

```
1 # Une instance de sac a dos simple a 12 objets numerotes de 1 a 12
2
3 set I := { 1..12 } ; # un ensemble d'indices
4
5 param C := 20 ; # la capacite du sac
6 param p[I] := <1> 11, <2> 7, <3> 5, <4> 5, <5> 4, <6> 3, <7> 3
7   , <8> 2, <9> 2, <10> 2, <11> 2, <12> 1;
8 param v[I] := <1> 20, <2> 10, <3> 25, <4> 11, <5> 5, <6> 50, <7> 15
9   , <8> 12, <9> 6 , <10> 5, <11> 4, <12> 30;
10
11 var x[I] binary; # un "tableau" de 12 variables booleennes
12
13 maximize valeur : sum <i> in I: v[i] * x[i];
14 subto poids : sum <i> in I: p[i] * x[i] <= C;
```

*Explications :* on déclare un ensemble d'indices en ligne 3; les *paramètres* de l'instance qu'on veut résoudre sont déclarées en lignes 5 à 9 : la capacité $C$ du sac, et des tableaux $p$ pour les poids des objets et $v$ pour leurs valeurs; la ligne 11 permet de déclarer un "tableau" de 12 variables de décision $x[1]$ à $x[12]$. La ligne 13] déclare l'objectif, qui est ici de maximiser $\sum_{i \in I} v_i x_i$ ; il faut donner un nom à la fonction objectif, ici `valeur`. Enfin la ligne 14 déclare la contrainte de poids; chaque contrainte doit aussi avoir un nom, ici `poids`.

**Exercice 2.** Récupérer le fichier sac-a-dos-12.zpl sur Moodle et le sauver dans votre répertoire de travail. Dans scip, taper les commandes suivantes :

[1]À condition que le bon *reader* ait bien été installé; c'est le cas avec SCIP3 sur les PC des salles de TPs informatique de la FSI.

```
rlwrap scip
read sac-a-dos-12.zpl
optimize
display solution
```

Question 2.1. Quelle est la valeur du sac optimal, et quels objets contient-il ?

L'instruction `read sac-a-dos-12.zpl` traduit en fait le programme ZIMPL en langage LP : c'est sur ce programme en langage LP que travaille le solveur SCIP.

Question 2.2. Toujours dans SCIP, exécuter la commande `write problem sac-a-dos-12-temp.lp` ; puis ouvrir avec un éditeur de texte le fichier `sac-a-dos-12-temp.lp` qui vient d'être généré : on reconnaît un programme en language LP qui correspond à la même instance.

**Exercice 3** (Pandémie)**.** On reprend l'exercice sur la pandémie vu en TDs. Une épidémie de maladie infectieuse a été observée dans un certain nombre $n$ de sites. Un ensemble de $m$ équipes de médecins doivent aller enquêter pour identifier la maladie, ce qui leur prend un certain temps $t_{ij}$ qui dépend du site $j$ et de l'équipe $i$. Chaque équipe peut enquêter au maximum sur 2 sites, et doit alors se déplacer du site $j_1$ au site $j_2$, ce qui prend un temps $d_{j_1 j_2}$.

| Eq / Lieu | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 10 | 12 | 14 | 5 |
| 2 | 6 | 10 | 10 | 4 |
| 3 | 12 | 12 | 16 | 6 |

| Lieu 1 / 2 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | | 6 | 6 | 8 |
| 2 | | | 7 | 8 |
| 3 | | | | 5 |

Un modèle pour minimiser le coût total en PLNE, en supposant que le coût des missions est proportionnel à la durée totale de travail (= temps de réalisation des missions + temps de déplacement entre deux sites s'il y a lieu), est le suivant, lorsqu'il y a $n$ sites ($j$) et $m$ équipes ($i$) :

- Variables binaires : $x_{ij} = 1$ si l'équipe $i$ va sur le site $j$, $y_{ijk}$ si l'équipe $i$ fait le déplacement entre $j$ et $k$, dans n'importe quel sens, donc pour $j < k$.
- Objectif : Minimiser les durées totales de travail : Minimiser $\sum_{ijk} y_{ijk} \times d_{jk} + \sum_{ij} x_{ij} \times t_{ij}$ (expression linéaire)
- Sous les contraintes :
  - chaque site $j$ est visité au moins une fois : $\forall 1 \leq j \leq n$, $\sum_i x_{ij} \geq 1$ ;
  - chaque équipe $i$ visite au plus 2 sites : $\forall 1 \leq i \leq m, \sum_j x_{ij} \leq 2$ ;
  - une équipe $i$ fait le déplacement entre les sites $j$ et $k$ ssi elle visite le site $j$ et le site $k$, on modélise cela avec 3 contraintes linéaire : $x_{ij} + x_{ik} - 1 \leq y_{ijk}$, $y_{ijk} \leq x_{ij}$, $y_{ijk} \leq x_{ik}$, $\forall 1 \leq i \leq m, \forall 1 \leq j < k \leq n$.

Question 3.1. Récupérer le programme incomplet `pandemie.zpl`, le compléter et le résoudre avec SCIP. Quelles équipes vont sur quels site ?

Question 3.2. Justifier l'expression `with j < k` dans l'expression du coût à minimiser dans `pandemie.zpl`.

La contrainte sur les $y_{ijk}$ peut être exprimée comme une contrainte conditionnelle :

$$\forall 1 \leq i \leq m \forall 1 \leq j < k \leq n : \text{ SI } x_{ij} = 1 \text{ ET } x_{ik} = 1 \text{ ALORS } y_{ijk} = 1$$

Le langage ZIMPL permet d'écrire des contraintes conditionnelles :

```
subto depc : forall <i,j,k> in I*J*J with j<k :
  vif x[i,j] == 1 and x[i,k] == 1 then y[i,j,k] == 1 end ;
```

Question 3.3. Remplacer dans votre programme les contraintes linéaires sur les $y_{ijk}$ par cette contrainte condionnelle, et résoudre le problème à nouveau. Qu'est-ce qui a changé ? Pour voir comment la contrainte conditionnelle a été traduite en contraintes linéaires par SCIP, taper l'instruction `write problem pandemie.lp` et ouvrir ce dernier fichier : combien y a-t-il de nouvelles variables ?

Question 3.4. *(Travail Personnel)* Écrire en ZIMPL le programme correspondant à la minimisation de la date de fin de la mission (on ne cherche donc plus à minimiser le coût).

**Exercice 4** (DÉMÉNAGEURS)**.** Dans le problème des DÉMÉNAGEURS, on a un ensemble de $n$ objets numérotés de 1 à $n$, qu'on veut ranger dans des boites qui ont toutes la même capacité entière $C > 0$. Chaque objet a une taille entière $t_i > 0$, on suppose que pour chaque $i$, $t_i \leq C$. On veut minimiser le nombre de boites utilisées. On sait que c'est un problème NP-complet,

Une instance qu'on pourra prendre en exemple est :

- capacité $C = 9$ ;
- 24 objets, de tailles respectives 6, 6, 5, 5, 5, 4, 4, 4, 4, 2, 2, 2, 2, 3, 3, 7, 7, 5, 5, 8, 8, 4, 4, 5.

On veut écrire un programme ZIMPL pour résoudre ce problème, en utilisant deux familles de variables binaires :

- $x_{ij} = 1$ si et seulement si l'objet $i$ est mis dans la boite $j$ ;
- $y_j = 1$ si et seulement si la boite $j$ est utilisée.

Question 4.1. Quel majorant simple peut-on donner pour le nombre de boites nécessaires ?

Question 4.2. Écrivez le modèle mathématique modélisant ce problème avec les variables ci-dessus ; c'est-à-dire, donnez l'objectif et les contraintes :

- chaque objet doit être dans exactement une boite ;
- une boite est utilisée ($y_j = 1 > 0$) si et seulement si elle contient au moins un objet ;
- dans chaque boite, la somme des tailles des objets ne doit pas dépasser la capacité.

Question 4.3. Écrivez les commandes ZIMPL pour

- définir deux ensembles d'indices : un pour les objets, un pour les boites ;
- définir un tableau contenant les tailles des objets ;
- définir un paramètre qui est la capacité des boites ;
- définir un tableau de variables $y_j$ ;
- définir un tableau de variables $x_{ij}$.

Question 4.4. Traduisez en ZIMPL le modèle mathématique.

Question 4.5. Résolvez l'instance ci-dessus avec SCIP.

## Lecture de données dans un fichier

Il est possible avec ZIMPL de programmer la lecture des paramètres d'un problème à partir d'un fichier ayant un format spécifique à ce type de problème. Par exemple, le fichier `sac-a-dos-24.txt` ci-contre décrit une instance du sac-à-dos : les lignes commençant avec un # sont des commentaires ; la première ligne qui n'est pas un commentaire (ligne 4) indique la capacité ; les objets sont décrits à partir de la ligne 6, un objet par ligne : son nom dans la première colonne, puis son poids et sa valeur.

```
1 # instance du sac a dos - 24 objets
2 # solution : 1 1 0 1 1 1 0 0 0 1 1 ....
3 # capacite :
4    6404180
5 # objets = id poids valeur :
6 a 382745 825594
7 b 799601 1677009
8 c 909247 1676628
9 d 729069 1523970
```

Le programme sac-a-dos.zpl ci-dessous lit les paramètres du fichier `sac-a-dos-24.txt` :

```
param fichier := "sac-a-dos-24.txt" ;

set I := { read fichier as "<1s>" comment "#" skip 1 } ;

param C := read fichier as "1n" comment "#" use 1 ;
do print "capacite : " , capacite ;
```

```
do print "nb objets : " , card(I) ;
param p[I] := read fichier as "<1s> 2n" comment "#" skip 1 ;
param v[I] := read fichier as "<1s> 3n" comment "#" skip 1 ;

var x[I] binary;

maximize valeur : sum <i> in I: v[i] * x[i];
subto poids : sum <i> in I: p[i] * x[i] <= C;
```

*E*xplications : la nouveauté est la fonction read qui permet d'extraire des informations d'un fichier, à l'aide de «patrons». Le format général est :

read *nom de fichier* as *patron* [comment $s_1$] [match $s_2$] [skip $n_1$] [use $n_2$]

Le fichier est lu ligne par ligne, chaque ligne est découpée en *token* – les *tokens* sont délimités par des espaces, tabulations, virgules, points-virgules, ou « :», mais les chaînes de caractères entre «"» ne sont pas découpées. Dans le programme ci-dessus :

- read fichier as "<1s>" comment "#" skip 1 :
  "<1s>" indique qu'on récupère le 1er token de chaque ligne, interprété comme une chaîne de caractères (dans les fichiers de données pour le sac-à-dos, c'est l'identifiant des objets), et on le met entre <> car il va dans un *set* ; et skip1 indique qu'on ne prend pas la première ligne – lignes de commentaires non comptées (il y a la capacité sur la première ligne).
- read fichier as "1n" comment "#" use 1 :
  "1n" indique qu'on récupère le 1er token des lignes considérées, interprété comme un entier ; comment «#» indique que les lignes commençant par # sont des commentaires ; et use 1 indique qu'on ne lit qu'1 ligne ;
- read fichier as "<1s> 2n" comment "#" skip 1 :
  le patron "<1s> 2n" indique l'association entre l'identifiant qui est dans le premier token et la valeur qui est dans le second : le paramètre de la seconde colonne va être associé, dans le table $p$, à l'objet dont l'« indice » est le nom apparaissant dans la première colonne.

**Exercice 5** (Sac-à-dos – suite). Récupérer depuis Moodle le programme sac-a-dos.zpl et les fichiers sac-a-dos-xx.txt, et résoudre ces instances avec scip. Quelle instance a le temps de calcul le plus long ?

**Exercice 6** (déménageurs, suite). On veut résoudre des instances de déménageurs décrites dans des fichiers comme u120_00.bpa, formatés comme suit :

- la première ligne indique le nom de l'instance ;
- la deuxième ligne a trois nombres : le premier est la capacité, le second est le nombre d'objets, le troisième n'est pas utilisé ;
- les lignes suivantes indiquent les tailles des objets, une par ligne.

Question 6.1. Écrivez un programme ZIMPL qui lit une instance de déménageurs d'un tel fichier et la résout.

Remarque : pour lire les tailles dans un tableau, on pourra utiliser les deux instructions suivantes :

```
set tmp[<i> in I] := {read file as "<1n>" skip 1+i use 1} ;
param taille[<i> in I] := ord(tmp[i],1,1);
```

Vous pouvez tester votre programme sur les instances fournies sur Moodle, les résultats attendus pour certaines de ces instances y sont aussi donnés.

Question 6.2. *(Travail Personnel)* La résolution du programme linéaire n'est pas performante à cause des nombreuses symétries du problème, puisque les boites sont interchangeables ; par exemple, si on échange les objets des 2 premières boites, ça ne change pas fondamentalement la répartition, ni le nombre de boites utilisées. Proposez des contraintes à ajouter à votre programme pour éviter ces symmétries. (Une petite recherche sur Internet peut être utile...) Comparez les performances de votre nouveau programme à ce que vous aviez précédemment.

**Exercice 7** (Le voyageur de commerce)**.** Le problème du voyageur de commerce a été vu en TDs : un voyageur de commerce doit se déplacer dans $n$ villes, en minimisant la distance totale parcourue. On suppose ici que la distance $c_{ij}$ entre deux villes $i$ et $j$ est simplement la distance euclidienne entre ces deux villes, et qu'on connait les coordonnées de toutes les villes dans un repère à deux dimensions : si $(x_i, y_i)$ sont les coordonnées de la ville $i$, et $(x_j, y_j)$ celles de la ville $j$, alors $c_{ij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$.

Question 7.1. Programmez la résolution de ce problème, en supposant que les coordonnées sont données dans un fichier, avec le nombre de villes sur la première ligne, ensuite les coordonnées de chaque ville, une ville par ligne : d'abord $x$, puis $y$ (il n'y a donc que deux colonnes). Deux fichiers sont donnés sur Moodle : `tsp5.txt` et `tsp101.txt`.

Question 7.2. Résolvez avec scip l'instance `tsp5.txt`.

Question 7.3. L'instance `tsp101.txt` est sans doute trop difficile à résoudre. Testez des instances plus petites, en ne prenant que les $k$ premières villes (on pourra utilise pour cela l'expression «`use k` » dans l'instruction `read ...` de ZIMPL). Quel est le nombre maximum de villes pour lequel vous arrivez à la solution optimale en temps raisonnable ?

# À propos de l'installation de scip

**Les enseignant-e-s de TPs ne sont pas là pour vous aider à réussir cette installation.** En plus des séances de TPs prévues sur l'emploi du temps, vous avez accès à certaines salles de PCs en dehors des heures de TPs, pour finir les TPs sur les ordinateurs de ces salles.

Vous pouvez aussi accéder à distance à un environnement linux ayant les même logiciels que sur les PCs du U3. Des instructions détaillées se trouvent sur l'espace Moodle Ressources informatiques FSI, section *Utilisation des ressources*, page *Outils pour l'utilisation des logiciels à distance. Le nom du serveur a changé : c'est maintenant `fsi-ens-d11.univ-tlse3.fr`.*

Si vous voulez utiliser scip sur votre ordinateur personnel, il faut télécharger la dernière version de « SCIP Optimization Suite » (disponible gratuitement à l'adresse scipopt.org) qui contient, en plus du *solveur* scip lui-même, d'autres programmes permettant de lire des PL écrits dans des formats variés. L'installation est souvent très facile, mais parfois ça prend du temps. Le plus simple est de récupérer le paquet précompilé en fonction de votre OS. Ça ne marche pas toujours, il faut alors recompiler à partir de l'archive scipoptsuite-x.y.z.tgz.

Avec Ubuntu 22.04 par exemple, il faut recompiler, le paquet deb a des dépendances non satisfaites.