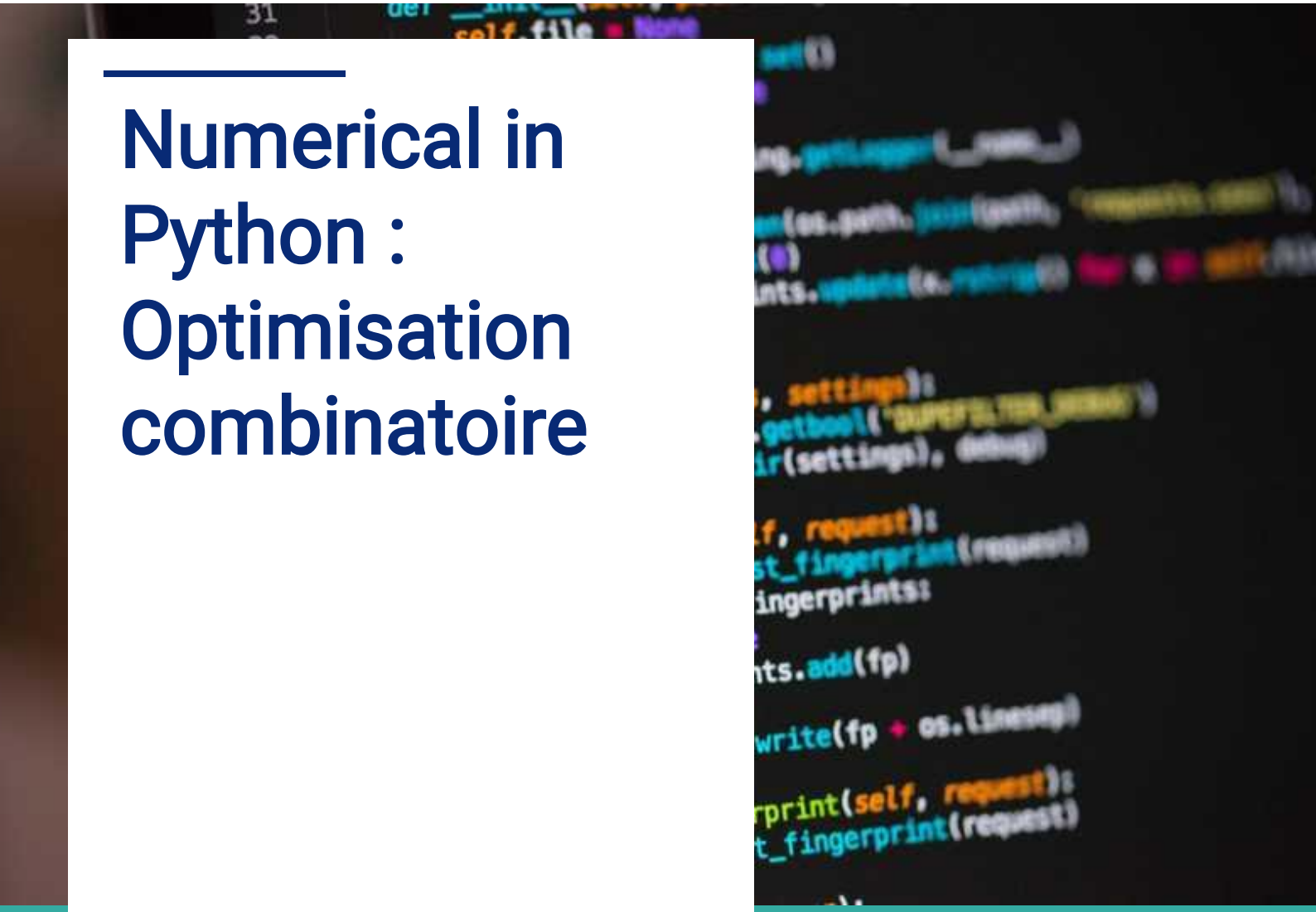


Numerical in Python : Optimisation combinatoire

17 OCTOBRE

ESIEA
Créé par : LAURENCE Camille, MAITROT Maxime, MARI Lucas

Table des matières

1. Modélisation linéaire

Nous avons :

-N nombre d'invités potentiels

-M Nombre de lien d'amitiés

Nous cherchons à inviter le plus d'invités à la condition que chacun se connaissent.

Nous avons donc affaire à une maximisation :

Soit X une variable binaire

$X = \begin{cases} 0 \; si \; Xi \; est \; absent \\ 1 \; si \; Xi \; est \; présent \end{cases}$

On maximise Z = $\sum_{M}^{N} c_i \cdot Xi$

Sous condition que :

Si $X = 1, \forall n \notin V_{i,j} + X_n \leqslant 1$

2. GLPK

Tout d'abord, il fallait réalisé un algorithme permettant de convertir les fichiers d'instances donné en fichier .lp, traitable par GLPK.

Pour cela, il fallait tout d'abord comprendre la structure des instances données, décrite le sujet.

Rappel :

-La première ligne est composée de N nombre total d'invités et de M nombre total de liens d'amitiés

-Les N lignes suivantes sont composées de i et ci

-Les M lignes suivants, sont composées de i et j, indiquant que l'invité i connais l'invité j, et vice versa.

Une fois la structure connue, il nous a fallut isoler chaque élément et extraire les conditions pour respecter le format d'un fichier .lp à savoir :

-L'opération à effectuer (minimize ou maximize)

-L'équation

-La « section » Subject to, énumérant les conditions, et enfin les variables Xi qui seront utilisées.

Nous avons donc procédé au développement de l'algorithme.

En première étape, il fallait réussir à récupérer les informations selon le format des instances données.

```
11  # Ouverture de l'instance
12  File = open(sys.argv[1], 'r')
13  line = File.readlines()
```

Ensuite, nous avons configuré les variables N et M

```
15  # Configuration N et M
16  N = int(line[0].split()[0])
17  M = int(line[0].split()[1])
```

Puis, pour chaque n, nous créons un tableau contenant : l'indice n, une liste contenant toutes les connaissances

```
23  # Initialisation des Vi
24  while n < N:
25      test[n] = [n, []]
26      n = n + 1
27
28  # Configuration des Vi
29  for l in range(0, M):
30      test[int(line[N + 1 + l].split()[0])][1].append(int(line[N + 1 + l].split()[1]))
```

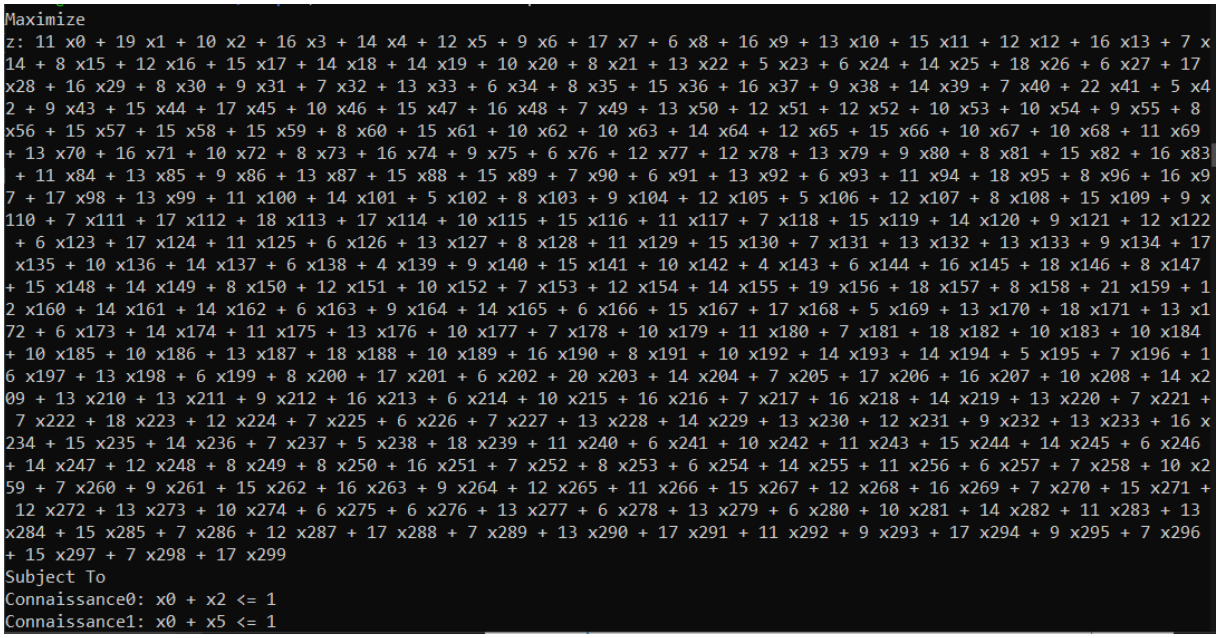
Nous procédons ensuite à l'écriture de z (grâce au N+1 lignes de l'instance)

```
36  # Ecriture de l'objectif à Maximiser
37  File2.write("z: ")
38  for k in range(0, N):
39      if k == N - 1:
40          File2.write(str(Ci[k]) + " x" + str(k))
41      else:
42          File2.write(str(Ci[k]) + " x" + str(k) + " + ")
```

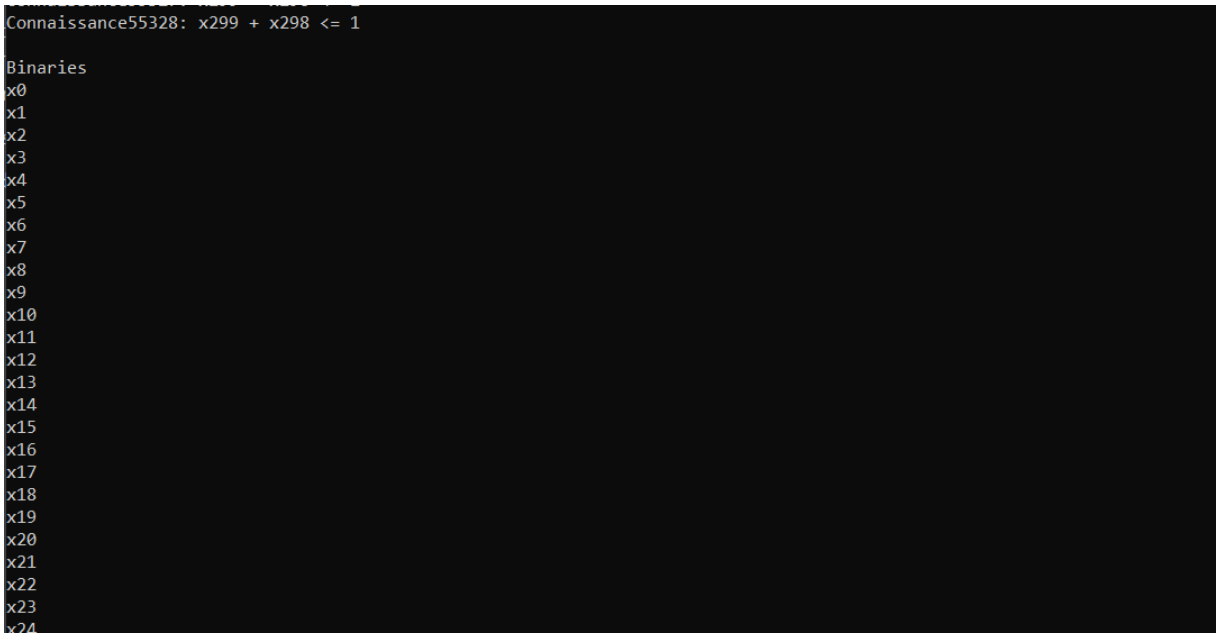
Evidemment, nous avons au préalablement créé le fichier « File2 », étant le fichier .lp
Une fois écrit, sous énumérons la condition de notre modèle linéaire pour chaque n Durant cette condition, nous prenons soit ne ne pas répéter des informations inutiles (exemple : si Vi connait Vj, on ne va pas réécrire que Vj connait Vi – ligne 49)

```
44  #Ecriture des contraintes
45  File2.write("\nSubject To\n")
46  for k in range(0, N): # On parcourt les Vi pour Chaque valeur de N
47      for n in range(0, N):
48          if test[k][1]:# Test si le Vi est vide
49              if n not in test[k][1] and n != k and k not in test[n][1]:# Test si n est dans Vk et si n n'est pas k et si k n'est pas dans Vn
50                  File2.write("Connaissance" + str(cpt) + ": x" + str(test[k][0]) + " + x" + str(n) + " <= 1\n")
51                  cpt += 1
```

Ainsi, voici à quoi ressemble notre fichier .lp :



```
Maximize
z: 11 x0 + 19 x1 + 10 x2 + 16 x3 + 14 x4 + 12 x5 + 9 x6 + 17 x7 + 6 x8 + 16 x9 + 13 x10 + 15 x11 + 12 x12 + 16 x13 + 7 x
14 + 8 x15 + 12 x16 + 15 x17 + 14 x18 + 14 x19 + 10 x20 + 8 x21 + 13 x22 + 5 x23 + 6 x24 + 14 x25 + 18 x26 + 6 x27 + 17
x28 + 16 x29 + 8 x30 + 9 x31 + 7 x32 + 13 x33 + 6 x34 + 8 x35 + 15 x36 + 16 x37 + 9 x38 + 14 x39 + 7 x40 + 22 x41 + 5 x4
2 + 9 x43 + 15 x44 + 17 x45 + 10 x46 + 15 x47 + 16 x48 + 7 x49 + 13 x50 + 12 x51 + 12 x52 + 10 x53 + 10 x54 + 9 x55 + 8
x56 + 15 x57 + 15 x58 + 15 x59 + 8 x60 + 15 x61 + 10 x62 + 10 x63 + 14 x64 + 12 x65 + 15 x66 + 10 x67 + 10 x68 + 11 x69
+ 13 x70 + 16 x71 + 10 x72 + 8 x73 + 16 x74 + 9 x75 + 6 x76 + 12 x77 + 12 x78 + 13 x79 + 9 x80 + 8 x81 + 15 x82 + 16 x83
 + 11 x84 + 13 x85 + 9 x86 + 13 x87 + 15 x88 + 15 x89 + 7 x90 + 6 x91 + 13 x92 + 6 x93 + 11 x94 + 18 x95 + 8 x96 + 16 x9
7 + 17 x98 + 13 x99 + 11 x100 + 14 x101 + 5 x102 + 8 x103 + 9 x104 + 12 x105 + 5 x106 + 12 x107 + 8 x108 + 15 x109 + 9 x
110 + 7 x111 + 17 x112 + 18 x113 + 17 x114 + 10 x115 + 15 x116 + 11 x117 + 7 x118 + 15 x119 + 14 x120 + 9 x121 + 12 x122
 + 6 x123 + 17 x124 + 11 x125 + 6 x126 + 13 x127 + 8 x128 + 11 x129 + 15 x130 + 7 x131 + 13 x132 + 13 x133 + 9 x134 + 17
 x135 + 10 x136 + 14 x137 + 6 x138 + 4 x139 + 9 x140 + 15 x141 + 10 x142 + 4 x143 + 6 x144 + 16 x145 + 18 x146 + 8 x147
+ 15 x148 + 14 x149 + 8 x150 + 12 x151 + 10 x152 + 7 x153 + 12 x154 + 14 x155 + 19 x156 + 18 x157 + 8 x158 + 21 x159 + 1
2 x160 + 14 x161 + 14 x162 + 6 x163 + 9 x164 + 14 x165 + 6 x166 + 15 x167 + 17 x168 + 5 x169 + 13 x170 + 18 x171 + 13 x1
72 + 6 x173 + 14 x174 + 11 x175 + 13 x176 + 10 x177 + 7 x178 + 10 x179 + 11 x180 + 7 x181 + 18 x182 + 10 x183 + 10 x184
+ 10 x185 + 10 x186 + 13 x187 + 18 x188 + 10 x189 + 16 x190 + 8 x191 + 10 x192 + 14 x193 + 14 x194 + 5 x195 + 7 x196 + 1
6 x197 + 13 x198 + 6 x199 + 8 x200 + 17 x201 + 6 x202 + 20 x203 + 14 x204 + 7 x205 + 17 x206 + 16 x207 + 10 x208 + 14 x2
09 + 13 x210 + 13 x211 + 9 x212 + 16 x213 + 6 x214 + 10 x215 + 16 x216 + 7 x217 + 16 x218 + 14 x219 + 13 x220 + 7 x221 +
 7 x222 + 18 x223 + 12 x224 + 7 x225 + 6 x226 + 7 x227 + 13 x228 + 14 x229 + 13 x230 + 12 x231 + 9 x232 + 13 x233 + 16 x
234 + 15 x235 + 14 x236 + 7 x237 + 5 x238 + 18 x239 + 11 x240 + 6 x241 + 10 x242 + 11 x243 + 15 x244 + 14 x245 + 6 x246
+ 14 x247 + 12 x248 + 8 x249 + 8 x250 + 16 x251 + 7 x252 + 8 x253 + 6 x254 + 14 x255 + 11 x256 + 6 x257 + 7 x258 + 10 x2
59 + 7 x260 + 9 x261 + 15 x262 + 16 x263 + 9 x264 + 12 x265 + 11 x266 + 15 x267 + 12 x268 + 16 x269 + 7 x270 + 15 x271 +
 12 x272 + 13 x273 + 10 x274 + 6 x275 + 6 x276 + 13 x277 + 6 x278 + 13 x279 + 6 x280 + 10 x281 + 14 x282 + 11 x283 + 13
x284 + 15 x285 + 7 x286 + 12 x287 + 17 x288 + 7 x289 + 13 x290 + 17 x291 + 11 x292 + 9 x293 + 17 x294 + 9 x295 + 7 x296
+ 15 x297 + 7 x298 + 17 x299
Subject To
Connaissance0: x0 + x2 <= 1
Connaissance1: x0 + x5 <= 1
```



```
Connaissance55328: x299 + x298 <= 1

Binaries
x0
x1
x2
x3
x4
x5
x6
x7
x8
x9
x10
x11
x12
x13
x14
x15
x16
x17
x18
x19
x20
x21
x22
x23
x24
```

	Solution optimale	Temps	Mémoire
Instance1.txt	75	5879.9s	556Mb
Instance2.txt	91	10007.0s	717Mb
Instance3.txt	84	10804.7s	882.6Mb

Pour des raisons de temps, nous n'avons pas pu faire tourner le programme GLPK sur toutes les instances. En effet, le temps de traitement étant très long, nous avons choisi ne faire que les 3 premières instances. Nous n'avons aucune piste de réflexion quant à ces temps, qui sont 5 fois plus élevé que dans le sujet.

3. Algorithme glouton

Pour pallier au problème du temps de traitement de GLPK, nous avons programmé un algorithme glouton. Le but d'un algorithme glouton est de gagner significativement en temps de traitement, au dépend de la précision de la réponse.
Les algorithmes gloutons dépendent eux-mêmes d'heuristiques, c'est-à-dire, de conditions qui incrémenteront ou non le score si elles sont respectées.

Ici notre heuristique se traduit de la manière suivante :

```
67      def Heuristique(self):
68
69          var = self.CalculeConnsConfirmees()
70          Index = 0
71          Tot = []
72
73          for i in range(self.S_nbr_contraintes):
74
75              Tot.append(0)
76
77              for j in range(len(S_TabVji[i])):
78
79
80
81
82                  Tot[i] = Tot[i] / self.S_Tab_Count[i]
83
84              if Tot[i] > Tot[Index]:
85
86                  Index = i
87
88          return Index
```

Tout d'abord, on stock les connexions confirmées et on initialise 2 variables : Index, et Tot tableau vide.
Ensuite, pour toutes les contraintes de l'instances on parcourt le tableau des connaissances de j et on divise tot[i] par le score des invités.
Si tot[i] est plus grand que le plus grand tot[i] stocké, on sauvegarde i dans Index. Ainsi, l'invité ayant le rendement (nombre de connaissances / son score) est sauvegardé et retourné.