Université Paris 6, Master ANDROIDE

Projet MOGPL

Olivier Bachollet, Arthur Ramolot

1er décembre 2014

Table des matières

1 Première modélisation du problème

1.1 Modélisation du problème

Fonction d'optimisation :

$\max \sum_{i=1}^{n} \sum_{j=1}^{m} u_{ij}.x_{ij}$

Contraintes :

$\prod_{i=1}^{n} \sum_{j=1}^{m} x_{ij} = 1$
$\prod_{j=1}^{n} \sum_{i=1}^{m} x_{ij} = 1$
$x_{ij} \in \{0,1\}, \forall i,j \in [0,n]$
$u_{ij} \in [0,M]$

Listing 1: Programme linéaire P0 avec pygurobi

```
# -*- coding: utf-8 -*-
"""
Created on Wed Nov 19 15:19:03 2014

@author: arthur
"""

from gurobipy import *
import numpy as np

def genUtils(M,N):
    u = np.ones((M,N))

    for i in range(N):
        for j in range(M):
            u[i][j]=np.round(np.random.triangular(0,M/2,M))

    return u

#N = np.random.random_integers(5,15)
N = 100
M = N

u = genUtils(M,N)
x = []
print u


m = Model("prjMogpl")

for i in range(N):
    tmp = []
```

```
    for j in range(M):
        name = "x"+str(i)+","+str(j)
        tmp.append(m.addVar(vtype=GRB.BINARY, name=name))
    x.append(tmp)

m.update()

obj = LinExpr()
obj = 0

for i in range(N):
    for j in range(M):
        obj += u[i][j]*x[i][j]

m.setObjective(obj,GRB.MAXIMIZE)

for i in range(N):
    m.addConstr(quicksum(x[i][j] for j in range(M))==1,
                          "contrainte%d" % i)

for j in range(M):
    m.addConstr(quicksum(x[i][j] for i in range(N))==1,
                          "contrainte%d" % (N+j))

m.optimize()
"""
print ""
print "Liste des objets :"
print u
print 'Solution optimale : '
for i in range(N):
    for j in range(M):
        print 'x'+str(i)+str(j), '=', x[i][j].x
print ""
print 'Valeur de la fonction objectif :', m.objVal
"""
```

1.2 Essais numériques

n	t	Moyenne	Minimum	Maximum
10	0.01	7.68	5.3	9.1
50	0.18	8.90	7.4	10
100	0.56	9.26	8.0	10
500	12.73	9.80	9.0	10
1000	69.79	9.98	9.0	10

Comme attendu, plus n est grand plus le calcul est long. Augmenter M n'influe pas sur les temps de calcul. Les autres données évoluent proportionellement avec la valeur maximale des objets.

2 Approche égalitariste

2.1 L'approche flot max

Pour écrire le problème de l'existence d'une affectation dans laquelle les satisfactions des agents seraient toute supérieures ou égales à lambda comme un problème de flot maximum dans un graphe il suffit de créé un graphe avec des sommets pour chaque objet et pour chaque agent, une source connecté à tous les agents, un puit connecté à tous les objet, et des arcs qui connecte les agents vers les objets qui leur donne une satisfaction supérieure ou égale à lambda. Chaque arcs a un flot maximum de 1.

Listing 2: Résolution problème flot max

```
# -*- coding: utf-8 -*-
"""
Created on Wed Nov 26 20:08:37 2014

@author: arthur
"""

from pygraph.classes.graph import graph
from pygraph.classes.digraph import digraph

from pygraph.algorithms.minmax import maximum_flow

import numpy as np

def genUtils(M,N):
    u = np.ones((M,N))

    for i in range(N):
        for j in range(M):
            u[i][j]=np.round(np.random.triangular(0,M/2,M))

    return u

#N = np.random.random_integers(5,15)
N = 100
M = N
u = genUtils(M,N)
x = []
#u = [[2,0,0],[0,3,1],[0,1,0]]
lmb = 0
sortie = 1
#print u

# Graph creation
while(sortie):
    gr = digraph()
```

```
    gr.add_nodes([0])
    gr.add_nodes([N+M+1])

    for i in range(N):
        gr.add_nodes([i+1])
        gr.add_edge((0,i+1), wt=1)

    for i in range(M):
        gr.add_nodes([N+i+1])
        gr.add_edge((N+i+1,N+M+1), wt=1)

    for i in range(N):
        for j in range(M):
            if(u[i][j]>=lmb):
                gr.add_edge((i+1,N+j+1), wt=1)
            else:
                gr.add_edge((i+1,N+j+1), wt=0)

    flows, cuts = maximum_flow(gr, 0, N+M+1)



    for i in range(N):
        k=0
        for j in range(M):
            if(flows[(i+1,N+j+1)]==0):
                k+=1
        if(k==N):
            sortie = 0

    if(sortie == 1):
        oldflow = flows
    lmb += 1

print u
print oldflow
```

2.2 L'approche programme linéaire

Fonction d'optimisation :

$\max y$

Contraintes :

$(\sum_{j=1}^{m} x_{ij}.u_{ij}) - y > 0, \forall i \in [0, n]$ $\prod_{i=1}^{n} \sum_{j=1}^{m} x_{ij} = 1$
$\prod_{j=1}^{n} \sum_{i=1}^{m} x_{ij} = 1$

$x_{ij} \in \{0,1\}, \forall i,j \in [0,n]$
$u_{ij} \in [0,M]$

Listing 3: Programme linéaire P1 avec pygurobi

```
# -*- coding: utf-8 -*-
"""
Created on Wed Nov 19 15:19:03 2014

@author: arthur
"""

from gurobipy import *
import numpy as np

def genUtils(M,N):
    u = np.ones((M,N))

    for i in range(N):
        for j in range(M):
            u[i][j]=np.round(np.random.triangular(0,M/2,M))

    return u

#N = np.random.random_integers(5,15)
N = 10
M = N
e=0.00001
y = 16
u = genUtils(M,N)
x = []
print u


m = Model("prjMogpl")

for i in range(N):
    tmp = []
    for j in range(M):
        name = "x"+str(i)+","+str(j)
        tmp.append(m.addVar(vtype=GRB.BINARY, name=name))
    x.append(tmp)

y = m.addVar(vtype=GRB.CONTINUOUS, name="y")

m.update()

obj = LinExpr()
obj = 0

```

```
obj += y

print "obj : ",y

m.setObjective(obj,GRB.MAXIMIZE)

for i in range(N):
    m.addConstr(quicksum(x[i][j]*u[i][j] for j in range(M))-y>=0
                                    ,"contrainte%d" % i)

for i in range(N):
    m.addConstr(quicksum(x[i][j] for j in range(M))==1
                                    ,"contrainte%d" % (N+i))

for j in range(M):
    m.addConstr(quicksum(x[i][j] for i in range(N))==1
                                    ,"contrainte%d" % (2*N+j))

m.optimize()
"""
print ""
print "Liste des objets :"
print u
print 'Solution optimale : '
for i in range(N):
    for j in range(M):
        print 'x'+str(i)+str(j), '=', x[i][j].x
print ""
print 'Valeur de la fonction objectif :', m.objVal
"""
```

2.3 Essais numériques

n	P1	Graphes
10	0.01	0.01
50	0.14	2
100	1.48	35

On constate que, d'après les résultats, Les Calculs avec l'algorithme de flot max sont beaucoup plus lent. La faute a l'étape de construction de graphe qui devient laborieuse lorsque n est grand.
Le programme linéaire P0 permet d'avoir un meilleur maximum mais en contrepartie peut avoir un minimum très bas. Tandis que le programme linéaire P1 permet d'avoir une moyenne supérieur et un meilleur minimum mais en contrepartie réduit le maximum.

2.4 Approche un peu moins égalitariste

Fonction d'optimisation :

$\max(y + \epsilon \sum_{i=1}^{n} \sum_{j=1}^{m} x_{ij}.u_{ij})$

Contraintes :

$(\sum_{j=1}^{m} x_{ij}.u_{ij}) - y > 0, \forall i \in [0, n]$ $\prod_{i=1}^{n} \sum_{j=1}^{m} x_{ij} = 1$
$\prod_{j=1}^{n} \sum_{i=1}^{m} x_{ij} = 1$
$x_{ij} \in \{0, 1\}, \forall i, j \in [0, n]$
$u_{ij} \in [0, M]$

Listing 4: Programme linéaire P2 avec pygurobi

```
# -*- coding: utf-8 -*-
"""
Created on Wed Nov 19 15:19:03 2014

@author: arthur
"""

from gurobipy import *
import numpy as np

def genUtils(M,N):
    u = np.ones((M,N))

    for i in range(N):
        for j in range(M):
            u[i][j]=np.round(np.random.triangular(0,M/2,M))

    return u

#N = np.random.random_integers(5,15)
N = 10
M = N
e=0.00001
y = 16
u = genUtils(M,N)
x = []
print u


n = Model("prjMogpl2")

for i in range(N):
    tmp = []
    for j in range(M):
```

```
            name = "x"+str(i)+","+str(j)
            tmp.append(n.addVar(vtype=GRB.BINARY, name=name))
        x.append(tmp)

y = n.addVar(vtype=GRB.CONTINUOUS, name="y")

n.update()

obj = LinExpr()
obj = 0

obj += y
for i in range(N):
    for j in range(M):
        obj +=e*u[i][j]*x[i][j]

print "obj : ",y

n.setObjective(obj,GRB.MAXIMIZE)

for i in range(N):
    n.addConstr(quicksum(x[i][j]*u[i][j] for j in range(M))-y>=0
                                    ,"contrainte%d" % i)

for i in range(N):
    n.addConstr(quicksum(x[i][j] for j in range(M))==1
                                    ,"contrainte%d" % (N+i))

for j in range(M):
    n.addConstr(quicksum(x[i][j] for i in range(N))==1
                                    ,"contrainte%d" % (2*N+j))

n.optimize()
"""
print ""
print "Liste des objets :"
print u
print 'Solution optimale : '
for i in range(N):
    for j in range(M):
        print 'x'+str(i)+str(j), '=', x[i][j].x
print ""
print 'Valeur de la fonction objectif :', m.objVal
"""
```

3 Approche égalitariste en regrets

3.1 Regrets en programme linéaire

Fonction d'optimisation :

$\max y$

Contraintes :

$(\sum_{j=1}^{m} x_{ij}.u_{ij}) - y > 0, \forall i \in [0, n]$
$\prod_{i=1}^{n} \sum_{j=1}^{m} x_{ij} = 1$
$\prod_{j=1}^{n} \sum_{i=1}^{m} x_{ij} = 1$
$x_{ij} \in \{0, 1\}, \forall i, j \in [0, n]$
$u_{ij} \in [0, M]$

Listing 5: Programme linéaire P3 avec pygurobi

```
# -*- coding: utf-8 -*-
"""
Created on Wed Nov 19 15:19:03 2014

@author: arthur
"""

from gurobipy import *
import numpy as np

def genUtils(M,N):
    u = np.ones((M,N))

    for i in range(N):
        for j in range(M):
            u[i][j]=np.round(np.random.triangular(0,M/2,M))

    return u

def maxzi(u):
    ret=np.zeros(1,len(u))

    for i in range(len(u)):

        for j in range(len(u[i])):
            if(ret[i]<u[i][j]):
                ret[i]=u[i][j]
    return ret
#N = np.random.random_integers(5,15)
N = 100
M = N
```

```
u = genUtils(M,N)
zi=maxzi(u)
x = []
print u


m = Model("prjMogpl")

for i in range(N):
    tmp = []
    for j in range(M):
        name = "x"+str(i)+","+str(j)
        tmp.append(m.addVar(vtype=GRB.BINARY, name=name))
    x.append(tmp)
y = m.addVar(vtype=GRB.CONTINUOUS, name="y")
m.update()

obj = LinExpr()
obj = 0

obj += y

m.setObjective(obj,GRB.MINIMIZE)

for i in range(N):
    n.addConstr(y-(zi[i]-quicksum(x[i][j]*u[i][j] for j
                    in range(M)))>=0,"contrainte%d" % i)

for i in range(N):
    m.addConstr(quicksum(x[i][j] for j in range(M))==1
                         ,"contrainte%d" % i)

for j in range(M):
    m.addConstr(quicksum(x[i][j] for i in range(N))==1
                         ,"contrainte%d" % (N+j))

m.optimize()
"""
print ""
print "Liste des objets :"
print u
print 'Solution optimale : '
for i in range(N):
    for j in range(M):
        print 'x'+str(i)+str(j), '=', x[i][j].x
print ""
print 'Valeur de la fonction objectif :', m.objVal
"""
```

3.2 Modélisation du problème sous forme de flot max

Listing 6: Résolution problème flot max avec regrets (algo hongrois)

```
# -*- coding: utf-8 -*-
"""
Created on Wed Nov 26 20:08:37 2014

@author: arthur
"""

from pygraph.classes.graph import graph
from pygraph.classes.digraph import digraph


from pygraph.algorithms.minmax import maximum_flow

import numpy as np

def genUtils(M,N,m):
    u = np.ones((M,N))

    for i in range(M):
        for j in range(N):
            u[i][j]=np.round(np.random.triangular(0,m/2,m))

    return u

def calcMin(t):
    print t,min(t)

def calcRegr(u):
    for i in range(len(u)):
        u[i,:] -= min(u[i,:])

    for i in range(len(u[0])):
        u[:,i] -= min(u[:,i])

    return u

def majRegr(u,cuts):
    mini = 99999

    for i in range(len(u)):
        for j in range(len(u[0])):
            if(cuts[i+1] == 0 and
               cuts[len(u)+j+1] == 1 and u[i][j]<mini):
                mini =  u[i][j]

    for i in range(len(u)):
```

```
        for j in range(len(u[0])):
            if(cuts[i+1] == 0 and cuts[len(u)+j+1] == 1):
                u[i][j]  -= mini
            elif(cuts[i+1] == 1 and cuts[len(u)+j+1] == 0):
                u[i][j] += mini
    return u
#    return u


#N = np.random.random_integers(5,15)
N = 10
M = N
u = genUtils(M,N,100)
x = []
#u = np.array(([2,2,2,5,6],[2,5,1,0,7],
#    [6,4,7,3,5],[5,3,3,7,0],[8,5,4,2,1]))
#u = [[2,0,0],[0,3,1],[0,1,0]]
sortie = 1

u = calcRegr(u)

while(sortie):
    # Graph creation
    cpt = 0
    gr = digraph()

    #Ajouts noeuds source et puit
    gr.add_nodes([0])
    gr.add_nodes([N+M+1])

    #ajouts noeuds agents et liens vers source
    for i in range(N):
        gr.add_nodes([i+1])
        gr.add_edge((0,i+1), wt=1)

    #ajouts noeuds objets et liens vers puit
    for i in range(M):
        gr.add_nodes([N+i+1])
        gr.add_edge((N+i+1,N+M+1), wt=1)


    for i in range(N):
        for j in range(M):
            if(u[i][j]==0):
                gr.add_edge((i+1,N+j+1), wt=1)
            else:
                gr.add_edge((i+1,N+j+1), wt=0)

    flows, cuts = maximum_flow(gr, 0, N+M+1)
```

```
    u = majRegr(u,cuts)


    for i in cuts:
            if(cuts[i] == 1):
                cpt += 1
    if(cpt == N+M+1):
        sortie = 0

    print sortie

print flows
```

3.3 Résultats

n	P1	Graphes
10	0.01	0.01
50	0.36	1
100	1.76	6

Même remarque que pour la résolution par flot max égalitariste

4 Extension à l'affectation multiple

P0 ne peut pas s'adapter au cas gÃľnÃľrale car ses seuls contraintes sont celle du cas spÃľcifique. P1, P2 et P3 peuvent s'adapter au cas gÃľnÃľrale en retirent les contraintes sur le nombre d'objet qu'un agent peux avoir et en ajoutant chaque objet en plusieurs exemplaires comme de nouveaux objet. De mÃłme pour l'algorithme de graphe Ãľgalitariste et celui des regrets dans le quelle on met le flot max de la source vers les agent Ãă m.