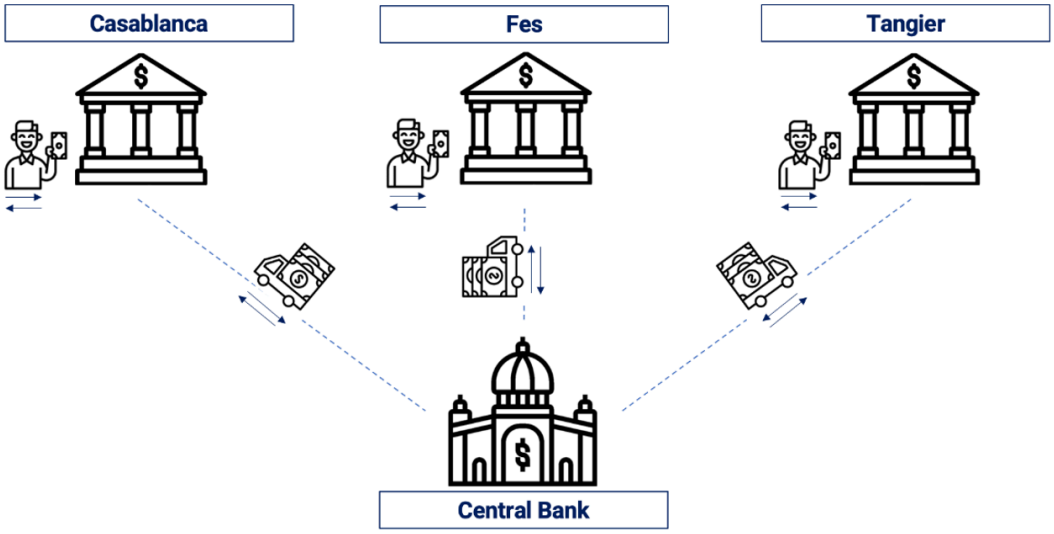
Cash Transfer Optimization with Linear Programming

September 6, 2024



1 Objective function

Minimize:

$\sum (x(supply)_{d,a} + x(collection)_{d,a})$ * $shipping\ cost_a$ + $\sum (y(supply)_{d,a} + y(collection)_{d,a})$ * $fixed\ cost_a$ + $\sum$ $closing\ balance_{d,a}$ * $opportunity\ cost\}$

2 Constraints

Define closing balance

• $closing\ balance_{d,a}$ == $initial\ balance_a$ + $\sum_{n=0}^{d}$ $x(supply)_{d,a}$ - $x(collection)_{d,a}$ + $\sum_{n=0}^{d}$ $transfer\ balance$

Maximum and minimum balance constraints

• $closing\ balance_{d,a} \leq max\ balance_a$

• $closing\ balance_{d,a} \geq min\ balance_a$

Big M method

• $x(supply)_{d,a} \geq$ -M*(1-$y(supply)_{d,a}$)

- $x(supply)_{d,a} \leq$ M*$y(supply)_{d,a}$
- $x(collection)_{d,a} \geq$ -M*(1-$y(collection)_{d,a}$)
- $x(collection)_{d,a} \leq$ M*$y(collection)_{d,a}$

Non-negativity constraints

- $x(supply)_{d,a} \geq 0$
- $x(collection)_{d,a} \geq 0$

3 Decision variables

Main variables

- $x(supply)_{d,a}$
- $x(collection)_{d,a}$

Auxiliary variables

- $y(supply)_{d,a}$
- $y(collection)_{d,a}$
- $closing\ balance_{d,a}$

Potential variables

- $min\ balance_{a}$
- $max\ balance_{a}$

4 Libraries

[ ]:
```
import gurobipy as gp
import pandas as pd
from gurobipy import GRB,quicksum,abs_
import os
from datetime import datetime
from datetime import timedelta
import numpy as np
```

5 Import and inspect dataset

[2]:
```
os.getcwd()
```

[2]:
```
'/Users/mydodethailung/Desktop/Artefact/CIH'
```

[3]:
```
#Import dataset
df_opticash = pd.read_excel("final_dataframe.xlsx")
```

6 Create model

[5]:
```
#Create a model object
m=gp.Model('Opticash')
```

```
Academic license - for non-commercial use only - expires 2022-10-29
Using license file /Users/mydodethailung/gurobi.lic
```

7 Generate lists and dictionaries

7.1 Period of study

[ ]:
```
#We create a list that represents business days in our period of study
#On crée une liste qui représente les jours ouvrés de notre période d'étude
days = list(set(df_opticash["dateoperation"]))
days.sort()
days
```

[7]:
```
#Number of days in the period of study
#Nombre de jours dans la période choisie
nb_days = (days[-1] - days[0]).days
nb_days
```

[7]:
```
1459
```

[8]:
```
# Add a week column
df_opticash["week"] = df_opticash['dateoperation'].dt.isocalendar().week
```

[9]:
```
weeks = list(set(df_opticash['week']))
```

7.2 Minimum and maximum closing balance for each agency

[10]:
```
# We define the minimum and maximum closing balance for each agency
#On définit les seuils min et max
min_balance = {"Casa Goulmima": 100000, "Fes Dhar el Mehraz": 100000, "Tanger␣
 ↪Malabata":100000}
max_balance = {"Casa Goulmima": 400000, "Fes Dhar el Mehraz": 400000, "Tanger␣
 ↪Malabata": 500000}
```

7.3 Opportunity cost

[11]:
```
#Coût d'opportunité
#print("Enter the daily opportunity cost in a decimal form  (ex : 0.02 for 2%)")
#opportunity_cost_daily = float(input())
#Coût d'opportunité
opportunity_cost_daily = 0.00004
```

7.4 Agencies

[12]:
```
#We create a list that represents our 3 agencies
#On crée une liste qui représente nos agences (ici, 3)
agencies = list(set(df_opticash['nom_agence']))
agencies
```

[12]:
```
['Fes Dhar el Mehraz', 'Tanger Malabata', 'Casa Goulmima']
```

7.5 Initial closing balance

[13]:
```
#We store the initial closing balance at t=0. This step is necessary because␣
 ↪we'll use this variable when we define what the closing balance is␣
 ↪(constraint)
#On store le solde de fermeture initial (t=0). Cette étape est nécessaire car␣
 ↪nous utiliserons cette variables dans les contraintes

balance_init = {"Casa Goulmima": list(min_balance.values())[0], "Fes Dhar el␣
 ↪Mehraz": list(min_balance.values())[1], "Tanger Malabata": list(min_balance.
 ↪values())[2]}
balance_init
```

[13]:
```
{'Casa Goulmima': 100000,
 'Fes Dhar el Mehraz': 100000,
 'Tanger Malabata': 100000}
```

7.6 Logistic costs

[14]:
```
list(min_balance.values())
```

[14]:
```
[100000, 100000, 100000]
```

[15]:
```
#Coûts logistiques
fixed_costs ={'Casa Goulmima':119,'Tanger Malabata':120,'Fes Dhar el Mehraz':
 ↪129}
shipping_costs={'Casa Goulmima': 0.042 ,'Tanger Malabata': 0.033,'Fes Dhar el␣
 ↪Mehraz': 0.033}
```

8 Decision variables

8.1 Primary variables

8.1.1 Supply & collection

Define the supply variable $x(supply)_{d,a}$ and the collection variable $x(collection)_{d,a}$

[16]:
```
# X positive
#Variable de décision x positif
x_alim = m.addVars(days,agencies,lb =0,name = "x_alim")
```

[17]:
```
# X negative
# Variable de décision x négatif
x_collecte = m.addVars(days,agencies,lb = 0, name = "x_collecte")
```

8.2 Auxiliary variables

8.2.1 Binary variable for fixed costs

Define the binary variables $y(supply)_{d,a}$ and $y(collection)_{d,a}$

[18]:
```
# y = 1 if a supply/collection exists, 0 otherwise
#y = 1 si une alimentation existe, 0 otherwise
y_alim = m.addVars(days,agencies,vtype=GRB.BINARY,name = "y_alim")
y_collecte = m.addVars(days,agencies,vtype=GRB.BINARY, name = "y_collecte")

week_variable = m.addVars(weeks,agencies,vtype=GRB.INTEGER,name = "week")
```

8.2.2 Closing balance

Define the supply variable $closing\ balance_{d,a}$

[19]:
```
# Closing balance at day d for agency a
#Solde de fermeture en un point
closing_balance = m.addVars(days,agencies,name = "closing_balance")
```

9 Maximum balance

[20]:
```
#max_balance = m.addVars(agencies,name = "max_balance")
```

10 Constraints

[21]:
```
week = []
agency = []
value = []
for element in y_alim:
    week.append(element[0].week)
    agency.append(element[1])

week = pd.DataFrame(week)
agency = pd.DataFrame(agency)
value = pd.DataFrame(y_alim.values())
```

```
ag_week = pd.merge(week, agency, left_index=True, right_index=True)
final = pd.merge(ag_week, value, left_index=True, right_index=True)
final = final.rename(columns={"0_x": "week", "0_y": "agency",0:"value"})
final
```

[21]:

	week	agency	value
0	1	Fes Dhar el Mehraz	<gurobi.Var *Awaiting Model Update*>
1	1	Tanger Malabata	<gurobi.Var *Awaiting Model Update*>
2	1	Casa Goulmima	<gurobi.Var *Awaiting Model Update*>
3	1	Fes Dhar el Mehraz	<gurobi.Var *Awaiting Model Update*>
4	1	Tanger Malabata	<gurobi.Var *Awaiting Model Update*>
…	…	…	…
2974	53	Tanger Malabata	<gurobi.Var *Awaiting Model Update*>
2975	53	Casa Goulmima	<gurobi.Var *Awaiting Model Update*>
2976	53	Fes Dhar el Mehraz	<gurobi.Var *Awaiting Model Update*>
2977	53	Tanger Malabata	<gurobi.Var *Awaiting Model Update*>
2978	53	Casa Goulmima	<gurobi.Var *Awaiting Model Update*>

[2979 rows x 3 columns]

[22]:
```
#for w in weeks:
    #for a in agencies:
       # m.addConstr(week_variable[w,a] == (final.groupby(['agency','week']).
 ↪sum('value'))[w,a])
```

[23]:
```
final['value'][11].getAttr
```

[23]: <bound method Var.getAttr of <gurobi.Var *Awaiting Model Update*>>

10.0.1 Define closing balance

[ ]:
```
#Create a list of transfers
agency_date_dict = zip(df_opticash['dateoperation'],df_opticash['nom_agence'])
transfers = df_opticash['transfers']

dict_transfers = dict(zip(agency_date_dict,transfers))
dict_transfers
```

Define the constraint that sets the closing balance :

$closing\ balance_{d,a} == initial\ balance_{a} + \sum_{n=0}^{d} x(supply)_{d,a}$ - $x(collection)_{d,a} + \sum_{n=0}^{d} transfer\ balance$

[26]:
```
#Define closing balance
#Définir le solde de fermeture
for index,val in enumerate(days):
    for a in agencies:
```

```
        m.addConstr(closing_balance[val,a] == balance_init[a] +␣
 ↪quicksum((x_alim[d,a]-x_collecte[d,a])  for d in days[0:index]) +␣
 ↪dict_transfers[val,a],name="closing_balance_def")
```

Define the maximum closing balance constraint :

$closing\ balance_{d,a} \leq max\ balance_a$

10.0.2 Maximum closing balance

[27]:
```
#Seuil maximum
#Le solde en tout point doit être inférieur au solde max
for d in days:
    for a in agencies:
        m.addConstr(closing_balance[d,a] <=␣
 ↪max_balance[a],name="closing_balance_max")
```

Define the maximum closing balance constraint :

$closing\ balance_{d,a} \geq min\ balance_a$

10.0.3 Minimum closing balance

[28]:
```
#Seuil minimum
#Le solde en tout point doit être supérieur au solde min
for d in days:
    for a in agencies:
        m.addConstr(closing_balance[d,a] >=␣
 ↪min_balance[a],name="closing_balance_min")
```

Use the big M method to account for fixed costs :

$x(supply)_{d,a} \geq$ -M*(1-$y(supply)_{d,a}$)

$x(supply)_{d,a} \leq$ M*$y(supply)_{d,a}$

$x(collection)_{d,a} \geq$ -M*(1-$y(collection)_{d,a}$)

$x(collection)_{d,a} \leq$ M*$y(collection)_{d,a}$

10.0.4 Double Big-M method for fixed costs

[29]:
```
#Auxiliary variable y
#Big M method

M = 100000000

#Big-M for x positive
```

```
for d in days:
    for a in agencies:
            m.addConstr(x_alim[d,a] >= -M*(1-y_alim[d,a]))
            m.addConstr(x_alim[d,a] <=  M*y_alim[d,a])


#Big-M for x negative
for d in days:
    for a in agencies:
            m.addConstr(x_collecte[d,a] >=  -M*(1-y_collecte[d,a]))
            m.addConstr(x_collecte[d,a] <=  M*y_collecte[d,a])
```

11 Objective function

Define the objective function

Minimize:

$\sum$ $(x(supply)_{d,a}$ + $x(collection)_{d,a})$ * $shipping\ cost_a$ + $\sum$ $(y(supply)_{d,a}$ + $y(collection)_{d,a})$ * $fixed\ cost_a$ + $\sum$ $closing\ balance_{d,a}$ * $opportunity cost $

[30]:
```
#Objective function
m.setObjective(quicksum(shipping_costs[a]*(x_alim[d,a]+x_collecte[d,a]) for d␣
↪in days for a in agencies)
               + quicksum(fixed_costs[a]*(y_alim[d,a]+y_collecte[d,a]) for d in␣
↪days for a in agencies)
               + quicksum(closing_balance[d,a]*opportunity_cost_daily for d in␣
↪days for a in agencies),GRB.MINIMIZE)
```

12 Solution

12.1 Print solution

[ ]:
```
def printSolution():
    if m.status == GRB.OPTIMAL:
        print('\n Optimal cost is : %g' % m.objVal)
    else:
        print('No solution:', m.status)

m.optimize()
printSolution()
```

12.2 Store solutions

[32]:
```
#We store solutions
sol_pos = m.getAttr('x', x_alim)
sol_neg = m.getAttr('x', x_collecte)
y_pos = m.getAttr('x', y_alim)
y_neg = m.getAttr('x', y_collecte)
closing = m.getAttr('x', closing_balance)


date = []
agence = []
alim = sol_pos.values()
collecte = sol_neg.values()
fixed_cost_alim = y_pos.values()
fixed_cost_col = y_neg.values()
solde_fermeture = closing.values()

for d in range(len(sol_pos.keys())):
    date.append(sol_pos.keys()[d][0])
    agence.append(sol_pos.keys()[d][1])


dataframe_decisions_manual = pd.DataFrame({'agence':agence,'date':
 ↪date,'alimentations':alim, 'collectes' : collecte,'y_alim' :␣
 ↪fixed_cost_alim,'y_col' : fixed_cost_col,"solde_fermeture" :␣
 ↪solde_fermeture})
```

[33]:
```
dataframe_decisions_manual
```

[33]:

	agence	date	alimentations	collectes	y_alim	y_col	\
0	Fes Dhar el Mehraz	2017-01-02	0.0	0.0	0.0	0.0	
1	Tanger Malabata	2017-01-02	0.0	0.0	0.0	0.0	
2	Casa Goulmima	2017-01-02	0.0	0.0	0.0	0.0	
3	Fes Dhar el Mehraz	2017-01-03	0.0	0.0	0.0	0.0	
4	Tanger Malabata	2017-01-03	0.0	0.0	0.0	0.0	
…	…	…	…	…	…	…	
2974	Tanger Malabata	2020-12-30	477037.5	0.0	1.0	0.0	
2975	Casa Goulmima	2020-12-30	0.0	257261.0	0.0	1.0	
2976	Fes Dhar el Mehraz	2020-12-31	0.0	0.0	0.0	0.0	
2977	Tanger Malabata	2020-12-31	0.0	0.0	0.0	0.0	
2978	Casa Goulmima	2020-12-31	0.0	0.0	0.0	0.0	

	solde_fermeture
0	100000.0
1	100000.0
2	100000.0

```
3            118351.7
4            149436.8
…               …
2974         323814.5
2975         205658.0
2976         400000.0
2977         100000.0
2978         400000.0

[2979 rows x 7 columns]
```

[34]:
```
#Write an excel file
dataframe_decisions_manual.to_excel("dataframe_decision_manual_scenario_atf.
 ↪xlsx")
```

[ ]:
```
for v in m.getVars():
        print('%s %g' % (v.varName, v.x))
```

[36]:
```
dataframe_decisions_manual.groupby('agence').agg({'y_alim':'sum','y_col':
 ↪'sum','solde_fermeture':'mean'})
```

[36]:

agence	y_alim	y_col	solde_fermeture
Casa Goulmima	104.0	306.000000	204497.339053
Fes Dhar el Mehraz	33.0	231.000000	201745.142991
Tanger Malabata	46.0	255.999999	229909.763706