

## Chapitre 3

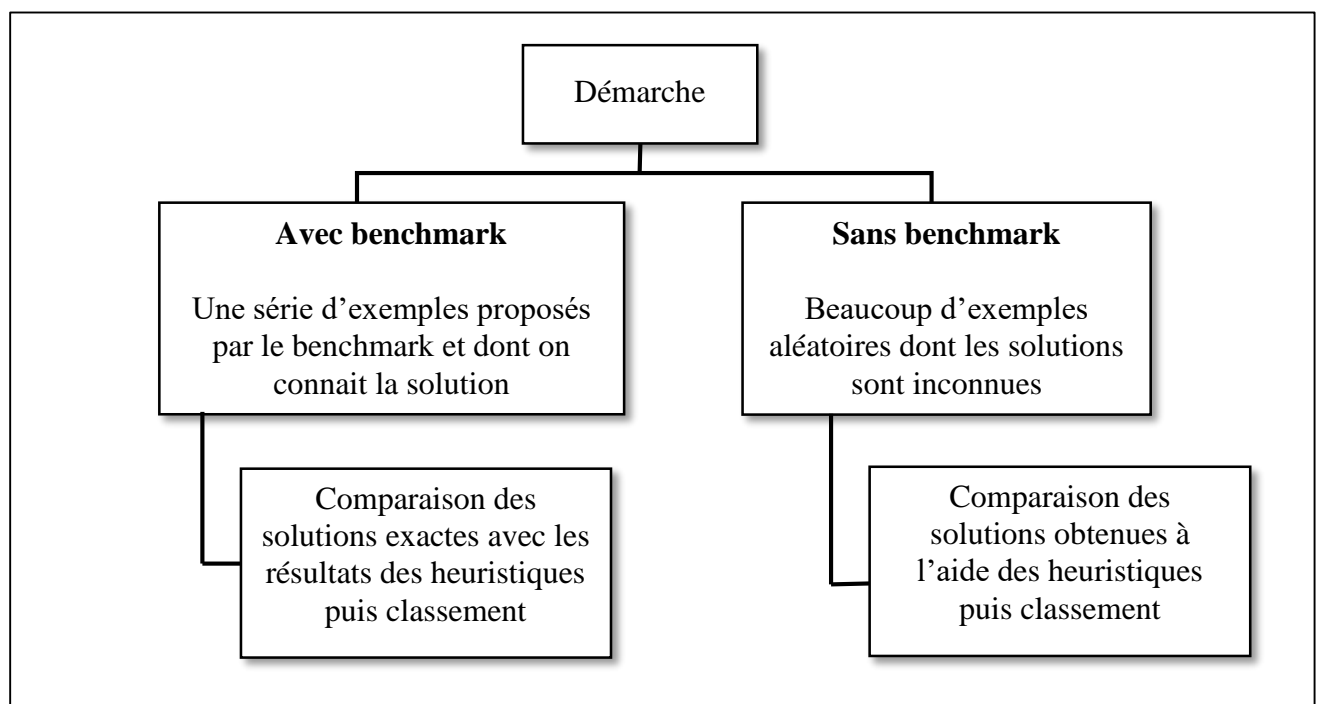
### Applications

#### 3.1 Introduction

De nombreuses entreprises de divers secteurs doivent à un moment donné, prendre des décisions stratégiques quant à l'endroit où construire des installations pour soutenir leurs activités. En outre, ces décisions ont un impact important tant en termes de satisfaction des clients que de gestion des coûts. L'un des facteurs critiques à prendre en compte dans ce processus est la localisation des clients que l'entreprise prévoit de servir.

Dans ce chapitre, nous présentons le langage Python et l'application Gurobi-optimizer, puis nous appliquons gurobipy à quelques variantes du FLP et VRP, et nous terminons par des exemples de Benchmarking.

On a deux différents problèmes : l'un où l'on connaît la solution exacte (benchmark), et l'autre où on ne la connaît pas (sans benchmark).



#### 3.2 Le langage Python:

**a. Définition :** Python est un langage de programmation créé par le programmeur Guido Van Rossum en 1981. Il tire son nom de l'émission Monty Python's Flying Circus. Ce langage de programmation présente de nombreuses caractéristiques intéressantes :

- Open-source.
- Portable.
- Orienté objet.
- Dynamique.
- Extensible.
- Support pour l'intégration d'autres langages.
- Langage interprété.

**b. Avantages :**

- ✓ Facile à apprendre et à utiliser.
- ✓ Fonctionne sur tous les principaux systèmes d'exploitation.
- ✓ Compilation transparente.

**c. Les bibliothèques de Python :**

On ne s'étend pas sur les détails de la syntaxe du langage mais uniquement sur les bibliothèques d'intérêt.

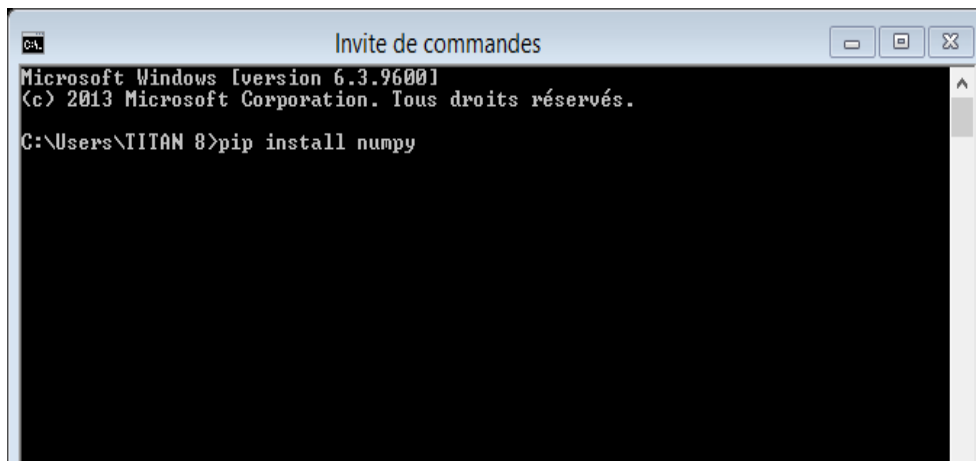
**1) La bibliothèque numpy :**

La bibliothèque numpy introduit le type ndarray ainsi que de nombreuses fonctions de calcul flottant notamment. Traditionnellement, on l'importe:

- soit directement dans l'environnement courant (from numpy import \*)
- soit sous un nom abrégé (import numpy as np).

La bibliothèque numpy est spécialisée dans la manipulation des tableaux (array). On l'utilisera essentiellement pour les vecteurs et les matrices. On peut l'installer sur Python, on utilise l'instruction suivante :

On utilisant la fenêtre DOS:



```
Microsoft Windows [version 6.3.9600]
(c) 2013 Microsoft Corporation. Tous droits réservés.

C:\Users\TITAN 8>pip install numpy
```

## 2) La bibliothèque Matplotlib.pyplot :

La bibliothèque matplotlib (et sa sous-bibliothèque pyplot) sert essentiellement à afficher des graphismes. Son utilisation ressemble beaucoup à celle de Matlab. Traditionnellement, on l'importe:

- soit directement dans l'environnement courant (from matplotlib.pyplot import \*).
- soit sous un nom abrégé (import matplotlib.pyplot as plt).

On peut l'installer sur Python, en utilisant l'instruction suivante dans la fenêtre DOS:

« pip install matplotlib »

## 3) Autres bibliothèques :

**Mini batch k-means** : est un algorithme de clustering, on peut l'installer sur Python, on utilise l'instruction suivante :

« pip install scikit-learn »

**Networkx** : est une bibliothèque Python pour l'étude des graphes et des réseaux, on peut l'installer sur Python, en utilisant l'instruction suivante :

« pip install networkx »

## f. Jupyter notebook :

Créé à partir de Python en 2014, Jupyter est un notebook de calcul (computational notebook) open source, gratuit et interactif. C'est une application web basée client permettant de créer et de partager du code, des équations, des visualisations ou du texte.

Son nom est en fait une référence à trois langages de programmation : Julia, Python et R. En effet, Jupyter prend en charge plus d'une quarantaine de langages de programmation.

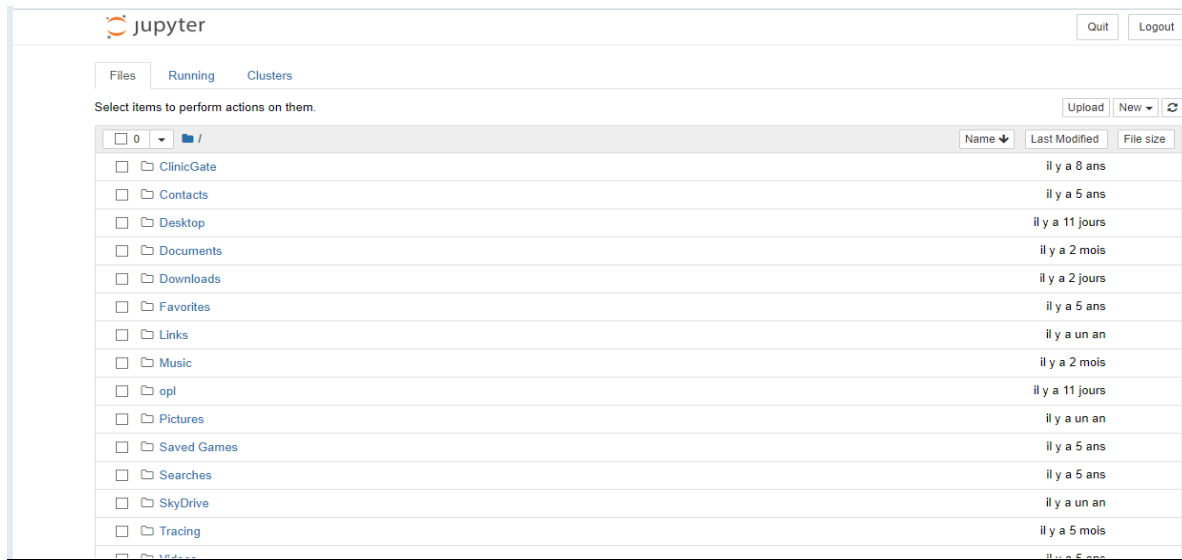
Les cas d'usage de Jupyter Notebook sont nombreux. Il est possible d'analyser une collection de textes, de créer de la musique ou de l'art ou même de développer des concepts d'ingénierie.

L'un des cas d'usage de Jupyter est la documentation live pour une bibliothèque ou un module. Alors que la documentation des modules Python est généralement statique, un notebook peut être utilisé comme un bac à sable interactif pour apprendre le fonctionnement d'un module. Pour installer le Jupyter Notebook, on peut utiliser la commande « pip » de Python.

« pip install notebook » « pip install jupyterlab »

Pour exécuter le notebook : dans la fenêtre de terminal écrire :

« jupyter notebook »



**Figure3.1 : L'environnement de Jupyter notebook sous Python**

### 3.3 Gurobi :

Le Gurobi Optimizer est un solveur commercial d'optimisation pour la programmation linéaire (LP), la programmation quadratique (QP), la programmation quadratique contrainte (QCP), la programmation linéaire mixte en nombres entiers (MILP), la programmation quadratique en nombres entiers mixtes (MIQP) et la programmation contrainte quadratique en nombres entiers mixtes (MIQCP). Gurobi porte le nom de ses fondateurs : Zonghao Gu, Robert Bixby et Edward Rothberg.

#### a. installation de Gurobi :

Pour utiliser Gurobi, une licence est requise. En tant qu'étudiant, la licence peut être obtenue gratuitement [www.gurobi.com](http://www.gurobi.com). Lors de la création d'un compte, sélectionnez "universitaire" et indiquez votre adresse e-mail. Assurez-vous que l'adresse e-mail est correctement orthographiée.

✓ Product updates and more.

Start your registration by designating your account type as either Commercial or Academic:

Are you an Academic or Commercial user?

First Name:

Last Name:

Company Email Address:

University:

Industry:

Academic Position:

Phone Number:

Country:

☐ Check this box if you also consult with commercial businesses.

\*Required. The information you provide to us will be used in accordance with the terms of our [Privacy Policy](#).

Après avoir téléchargé l'application, installez-la sur votre machine, puis allez à votre compte pour obtenir la licence.

Sélectionnez la licence, copiez (comme indiqué ci-dessous) la commande `grbgetkey` et exécutez-la dans la console. Sous Windows, ouvrez le menu Démarrer et recherchez `cmd.exe` dans le champ de recherche.

**GUROBI OPTIMIZATION**

Documentation Downloads & Licenses Support My Account

Products Customers Resources Academia Company Partners

Distributed Limit	0
Host Name	KING8
Host ID	86822e78

**Installation**

To setup this license file on a computer where Gurobi Optimizer is installed, copy and paste the following command to the Start/Run menu (Windows only) or a command/terminal prompt (any system): `grbgetkey b9228aa6-aea1-11ec-925d-0242ac120004` The `grbgetkey` command requires an active internet connection. If your computer has no internet access, or you get no response or an error message such as "Unable to contact key server", [Please click here for additional instructions](#). **Please note:** If you have license type - "WEBFLOATCS" - do not follow the installation instructions above. You can go directly to [licenses.gurobi.com](#) to the Web License Manager to access your Web License Service (WLS) license.

```
Gurobi 9.5.1
Python 3.7.6 <tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30> [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
Set parameter Username
Set parameter LogFile to value "gurobi.log"
Academic license - for non-commercial use only - expires 2022-05-28

Gurobi Interactive Shell (win64), Version 9.5.1
Copyright (c) 2022, Gurobi Optimization, LLC
Type "help(< >)" for help

gurobi>
```

Figure 3.2: L'environnement de Gurobi.

**b. Installation de Gurobi sous Python (gurobipy) :**

Vous pouvez utiliser pip pour télécharger et installer l'extension gurobipy simplement en ouvrant une fenêtre de terminal et en lançant la commande suivante :

« python -m pip install gurobipy »

On l'importe:

- soit directement dans l'environnement courant (from gurobipy import GRB)
- soit sous un nom abrégé (import gurobipy as gp).

**c. Fondamentaux de l'API Python :****1. Fonction globale importante :**

SetParam () : définir les paramètres du nouveau modèle dans cette session.

Read () : créer un objet modèle à partir d'un fichier modèle.

**2. Les objets primaires :**

Model : le modèle

Var : une variable.

Constr : une contrainte

**3. Importantes méthodes :**

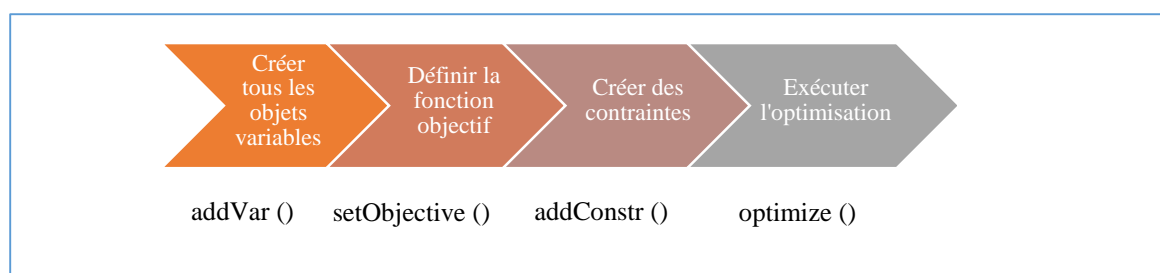
Model.setObjective () : définir la fonction objectif.

Model.optimize () : démarrer l'optimisation.

Model.reset () : réinitialiser l'état du modèle.

**Générateur de modèles :**

Pour la bonne création d'une instance de modèle avec gurobipy, la structure suivante est recommandée:



### 3.4 Facility location problem en gurobipy :

Les problèmes de localisation d'installations sont courants dans de nombreux secteurs, notamment la logistique et les télécommunications.

#### Exemple 1 :

Dans cet exemple, nous allons vous montrer comment résoudre un FLP qui consiste à déterminer le nombre et l'emplacement des entrepôts nécessaires pour approvisionner un groupe de supermarchés :

Une grande chaîne de supermarchés au Royaume-Uni doit construire des entrepôts pour un ensemble de supermarché qu'elle ouvre dans le nord de l'Angleterre. Les emplacements des supermarchés ont été identifiés, mais les emplacements des entrepôts doivent encore être déterminés. Plusieurs bons emplacements candidats pour les entrepôts ont été identifiés, mais des décisions doivent être prises quant au nombre d'entrepôts à ouvrir et à quel emplacements candidats les construire.

Notre objectif est de trouver le compromis optimal entre les coûts de livraison et les coûts de construction de nouvelles installations.

Nous importons le module Python Gurobi et d'autres bibliothèques, ensuite nous initialisons les structures de données avec les données:

```
[1]: import numpy as np
import random
from itertools import product
from math import sqrt
import gurobipy as gp
from gurobipy import GRB
import matplotlib.pyplot as plt
rnd = np.random
rnd.seed(42) # Change seed value to randomise
# parameter

customers = [(0,1.5),(2.5,1.2)]
facilities = [(0,0),(0,1),(1,0),(1,1),(1,2),(2,0),(2,1),(2,2)]
setup_cost = [3,2,3,1,3,3,4,3,2]
cost_per_mile = 1
```

Nous définissons une fonction qui détermine la distance euclidienne entre chaque installation et les sites des clients. Nous calculons les paramètres clés requis par le modèle MIP

```
[5]: # this function determines the euclidean distance between a facility and customer sites
def compute_distance(loc1, loc2):
    dx = loc1[0] - loc2[0]
    dy = loc1[1] - loc2[1]
    return sqrt(dx*dx + dy*dy)
# compute key parameters of MIP model formulation
num_facilities = len(facilities)
num_customers = len(customers)
cartesian_prod = list(product(range(num_customers), range(num_facilities)))
# compute shipping costs
shipping_cost = {(c,f): cost_per_mile*compute_distance(customers[c], facilities[f]) for c, f in cartesian_prod}
```

Nous déterminons maintenant le modèle MIP pour le FLP, puis nous lançons le processus d'optimisation, le Gurobi trouve le plan de construction des installations qui minimise les coûts totaux :

```
[6]: # MIP model formulation
m = gp.Model('facility_location')
select = m.addVars(num_facilities, vtype=GRB.BINARY, name='Select')
assing = m.addVars(cartesian_prod, ub=1, vtype=GRB.CONTINUOUS, name='Assing')
m.addConstrs((assing[(c,f)] <= select[f] for c,f in cartesian_prod), name='Setup2ship')
m.addConstrs((gp.quicksum(assing[(c,f)] for f in range(num_facilities))==1 for c in range(num_customers)), name='Demand')
m.setObjective(select.prod(setup_cost)+assing.prod(shipping_cost), GRB.MINIMIZE)
m.optimize()
```

➤ Les résultats de l'optimisation sont :

```
Optimal solution found (tolerance 1.00e-04)
Best objective 3.631308583792e+00, best bound 3.631308583792e+00, gap 0.0000%
```

Le résultat du modèle d'optimisation montre que la valeur minimale du coût totale est 3.63 millions de GBP.

➤ Ce plan détermine les emplacements sur lesquels construire un entrepôt

```
4]: # display optimal values of decision variables
for facility in select.keys():
    if (abs(select[facility].x) > 1e-6):
        print(f"\n n Build a warehouse at location {facility + 1}.")
```

```
n Build a warehouse at location 4.
```

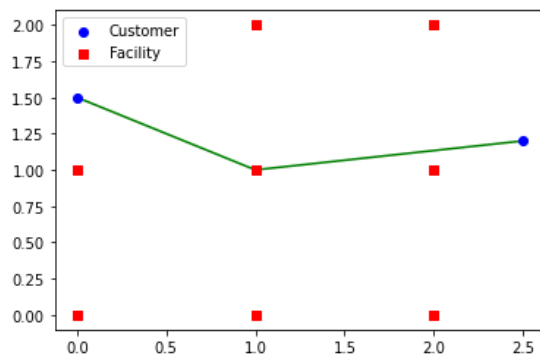
➤ Ce plan détermine le pourcentage des expéditions à envoyer de chaque installation construite à chaque client.

```
5]: # shipments from facilities to customers.
for customer, facility in assing.keys():
    if (abs(assing[customer, facility].x) > 1e-6):
        print(f"\n Supermarket {customer + 1} receives {round(100*assing[customer, facility].x, 2)} % of its demand from Warehouse 4")

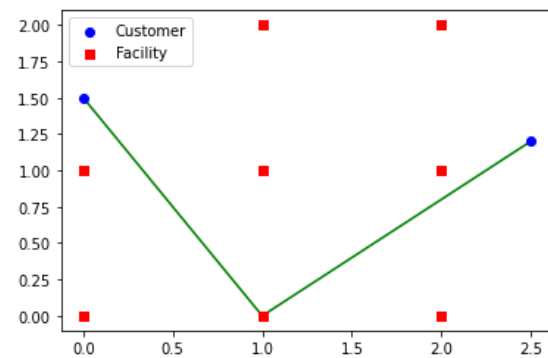
Supermarket 1 receives 100.0 % of its demand from Warehouse 4.
Supermarket 2 receives 100.0 % of its demand from Warehouse 4.
```



➤ On trace le graphe :



(a) Coûts = [3,2,3,1,3,3,4,3,2]



(b) Coûts = [5,8,1,4,3,10,15,4,11]

Figure 3.3 : Représentation du FLP pour 2 exemples de coûts.

➤ On inverse les clients (customers) et les dépôts (facilities)

➤ On obtient le graphe suivant :

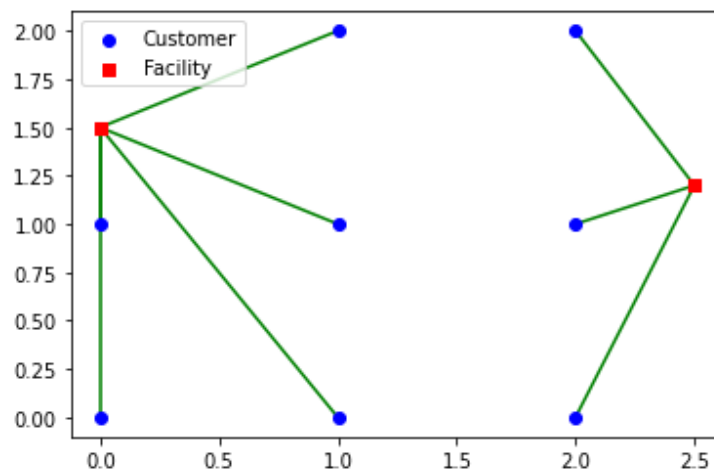


Figure 3.4 : Représentation du FLP après inversion.

### Exemple 2

Clients	10	20	30	40
10	C1	C2	C3	C4
20	C5	C6	C7	C8
30	C9	C10	C11	C12
40	C13	C14	C15	C16

Tableau 3.1 les coordonnées des clients.

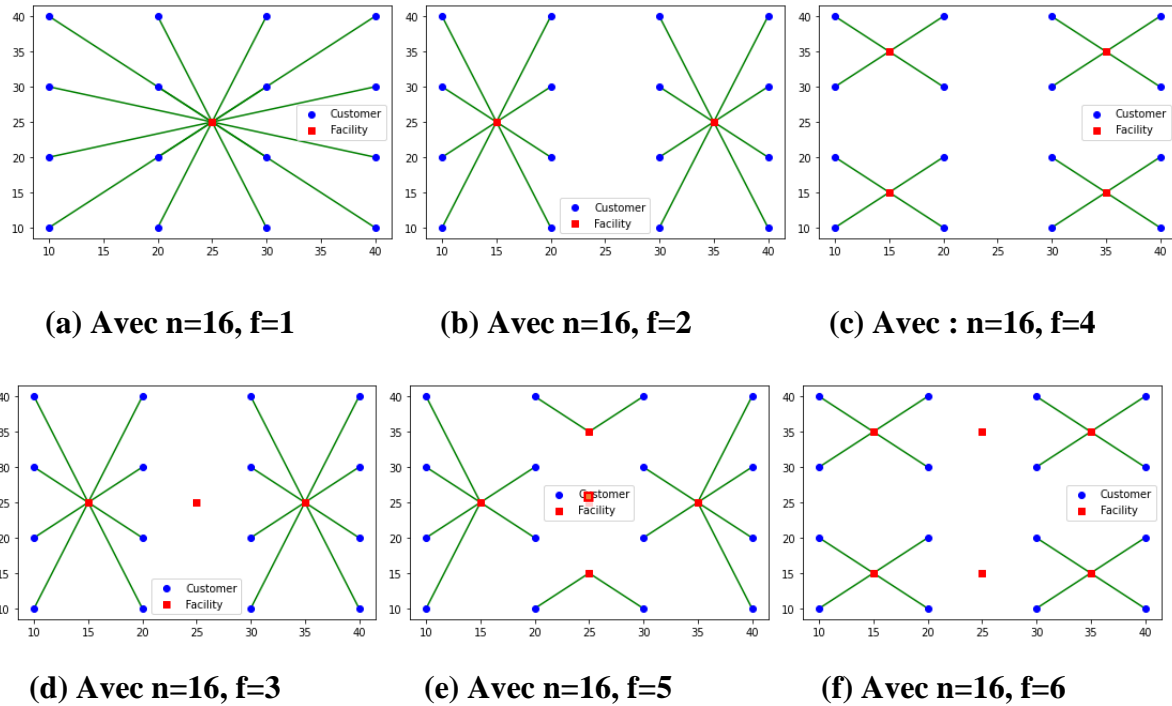


Figure 3.5 : Représentation du FLP avec différents dépôts.

**Exemple 3 :**

Dans cet exemple, nous choisissons des emplacements aléatoires pour les clients et les installations candidats. Les clients sont répartis selon des distributions gaussiennes autour de quelques centres de population choisis au hasard, tandis que les emplacements des installations sont distribués uniformément.

```
[1]: %matplotlib inline
import random
import gurobipy as gp
from gurobipy import GRB
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import MiniBatchKMeans
# Tested with Gurobi v9.0.0 and Python 3.7.0
seed = 10101
num_customers = 50000
num_candidates = 50
max_facilities = 8
num_clusters = 1000
num_gaussians = 10
threshold = 0.99

random.seed(seed)
customers_per_gaussian = np.random.multinomial(num_customers,
                                                [1/num_gaussians]*num_gaussians)

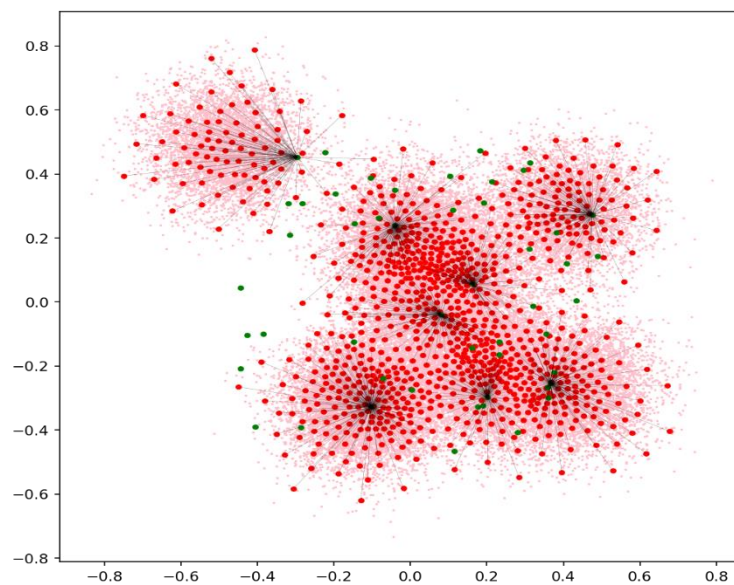
customer_locs = []
for i in range(num_gaussians):
    # each center coordinate in [-0.5, 0.5]
    center = (random.random()-0.5, random.random()-0.5)
    customer_locs += [(random.gauss(0,.1) + center[0], random.gauss(0,.1) + center[1])
                      for i in range(customers_per_gaussian[i])]
    # each candidate coordinate in [-0.5, 0.5]
facility_locs = [(random.random()-0.5, random.random()-0.5)
                 for i in range(num_candidates)]
print('First customer location:', customer_locs[0])

First customer location: (0.33164437091949245, -0.2809884943538464)
```

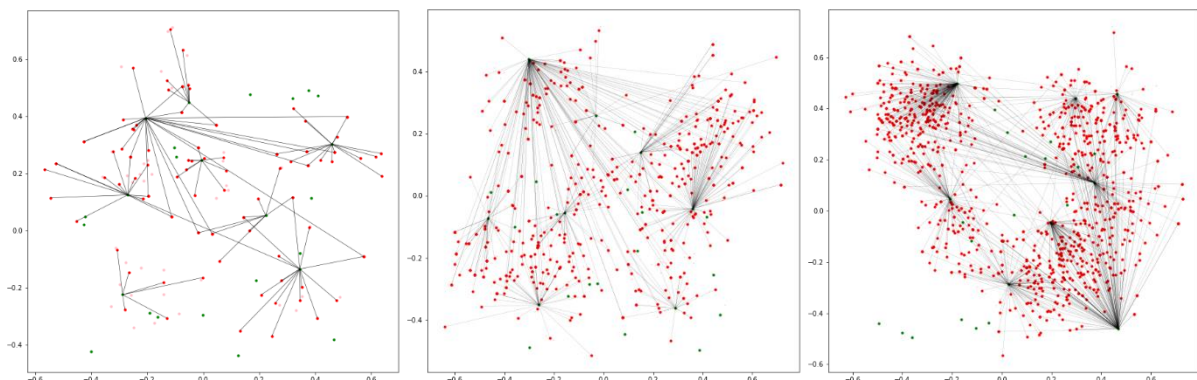
Pour limiter la taille du modèle d'optimisation, nous regroupons les clients individuels en clusters et optimisation sur ces clusters. Certains installations sont tout simplement trop éloignées du centre d'un cluster pour être pertinentes, alors filtrons heuristiquement toutes les distances qui dépassent un seuil donné. Construire des installations parmi les sites candidats afin de minimiser la distance totale vers les centres de clusters.

Traçons une carte avec :

- Les emplacements des clients sont représentés par de points roses.
- Les centroïdes des groupes de clients des représentés par de points rouges.
- Les candidats de FL sont représentés par des points verts.



**Figure 3.6 : FLP avec  $n_c = 50000$ .**



**(a) Avec  $n_c = 100$**

**(b) Avec  $n_c = 500$**

**(c) Avec  $n_c = 1000$**

**Figure 3.7 : Représentation du FLP avec différents  $n_c$ .**

## UFLP

Dans cet exemple, nous choisissons des emplacements aléatoires pour les clients et les dépôts.

```
[1]: import random
import numpy as np
import matplotlib.pyplot as plt
rnd = np.random
rnd.seed(42) # Change seed value to randomise
```

```
[2]: n = 20 # Number of Customer
xc = rnd.rand(n) * 200 # x-coordinate
yc = rnd.rand(n) * 100 # y-coordinate
m = 5 # Numebr of Facility
xf = rnd.rand(m) * 200 # x-coordiante
yf = rnd.rand(m) * 100 # y-coordiante
```

➤ Maintenant on trace le graphe :

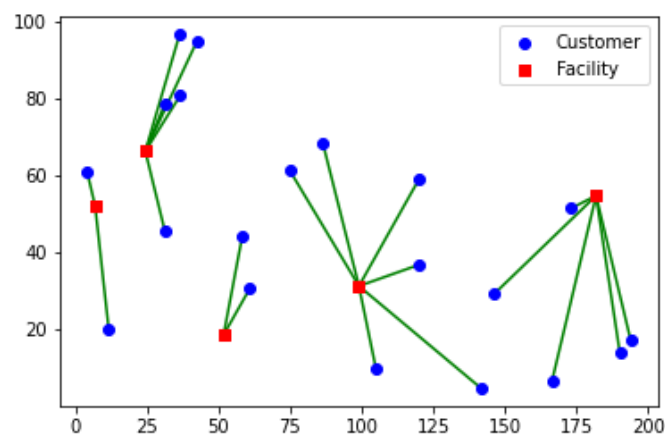
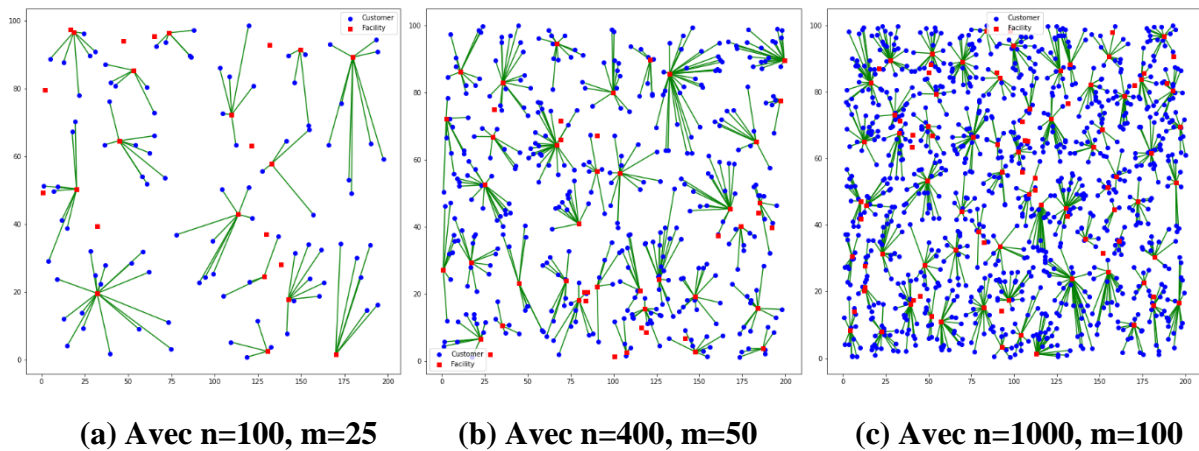
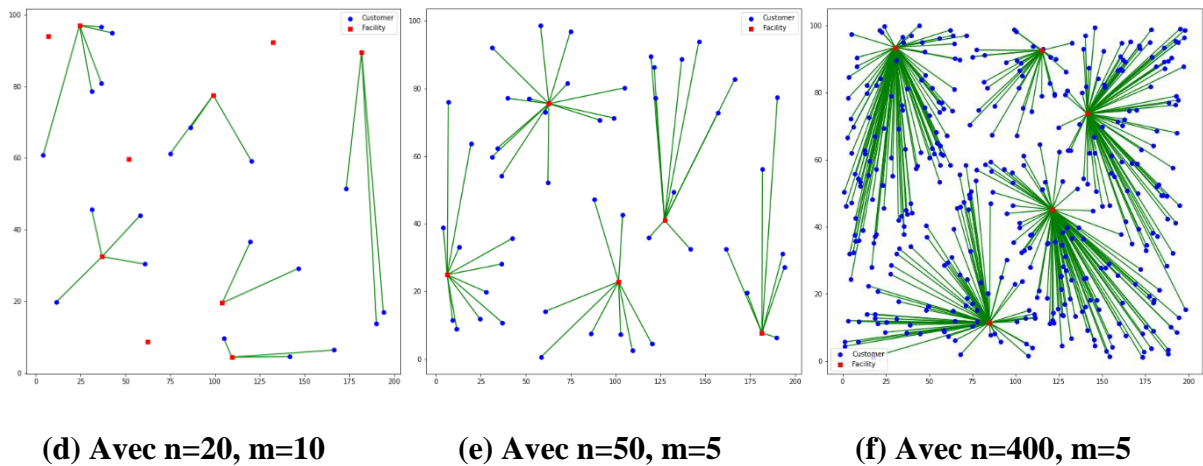


Figure 3.8 : Représentation du UFLP.



Figure 3.9 : Représentation du UFLP avec  $n, m$  différents.

### 3.5 Location Routing Problem en gurobipy :

#### VRP :

Nous choisissons des emplacements aléatoires pour les clients et le dépôt.

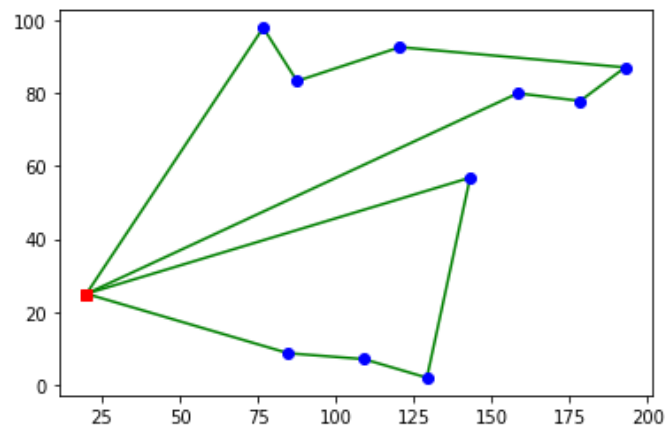
```
[1]: import numpy as np
import matplotlib.pyplot as plt
from gurobipy import Model, GRB, quicksum
rnd = np.random
rnd.seed(0)

[2]: n = 10 # number of clients
xc = rnd.rand(n + 1) * 200 # x-coordinate of clients and the depot
yc = rnd.rand(n + 1) * 100 # y-coordinate of clients and the depot

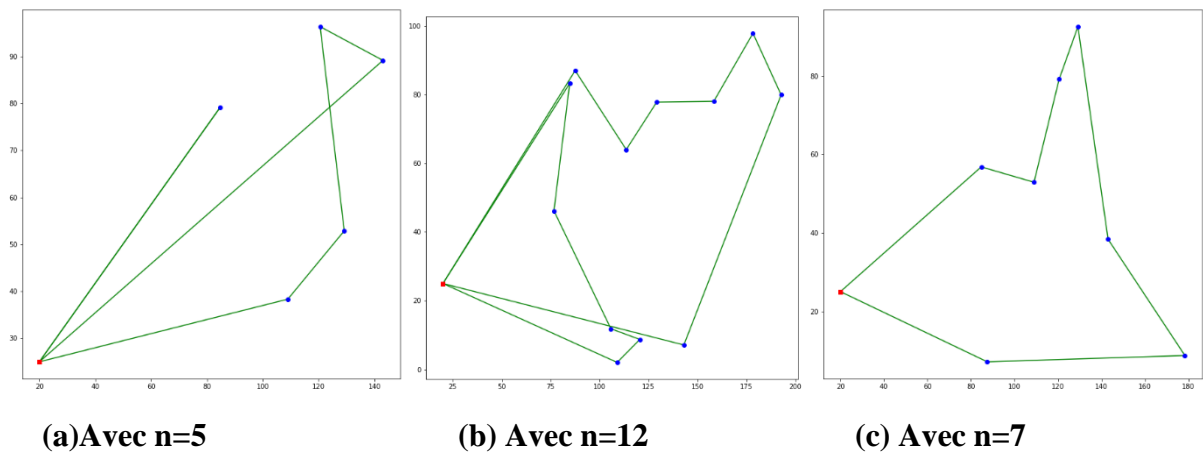
xc[0] = 20 # Set the depot to a very far location
yc[0] = 25 # Set the depot to a very far location
```

➤ Les résultats de l'optimisation sont :

```
Optimal solution found (tolerance 1.00e-04)
Best objective 6.987034212583e+02, best bound 6.987034212583e+02, gap 0.0000%
```



**Figure 3.10: Représentation de VRP avec  $n=10$ .**

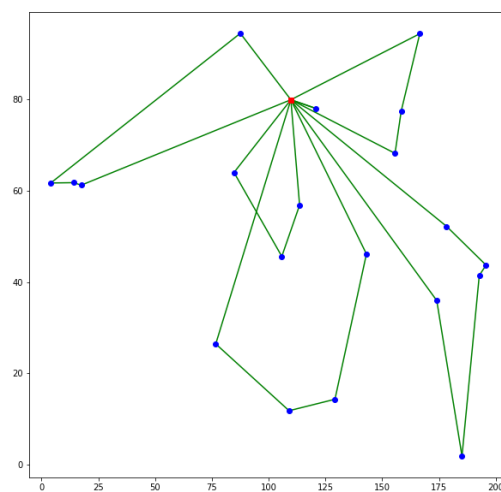


**Figure 3.11: Représentation de VRP avec différent  $n$ .**

### CVRP :

Le même exemple celui qui de VRP

➤ Après l'optimisation on obtient le graphe suivant :



**Figure 3.12: Représentation de CVRP avec  $n=20$ .**

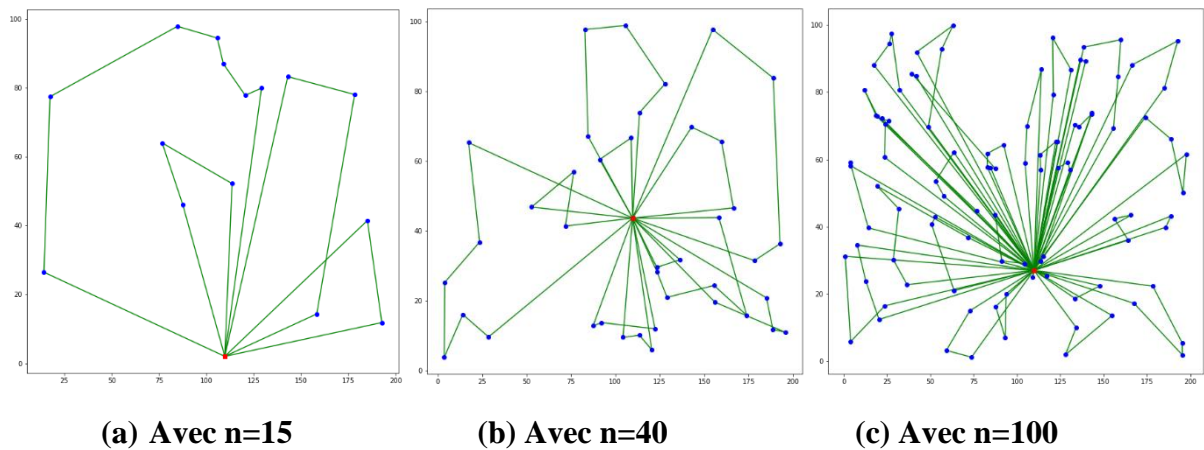


Figure 3.13: Représentation de CVRP avec différent n.

### 3.6 Benchmarking :

Le benchmarking est une technique de marketing ou de gestion de la qualité qui consiste à étudier et analyser les techniques de gestion, les modes d'organisation des autres entreprises afin de s'en inspirer et d'en tirer le meilleur. C'est un processus continu de recherche, d'analyse comparative, d'adaptation et d'implantation des meilleures pratiques pour améliorer la performance des processus dans une organisation. [Wikipédia]

#### 3.6.1 Quelque benchmark du FLP :

Une instance est un graphe dont on connaît la solution exacte.

Nous donnons quelques instances pour quelques variantes du FLP

La variante	Les instances
<b>CFLP</b>	<ul style="list-style-type: none"> <li>• Holmberg et al.</li> <li>• Beasley.</li> <li>• Real data instance (Cookie Factory data).</li> </ul>
<b>UFLP</b>	<ul style="list-style-type: none"> <li>• Beasley</li> </ul>
<b>P-median uncapacited</b>	<ul style="list-style-type: none"> <li>• Beasley</li> </ul>
<b>P-median capacited</b>	<ul style="list-style-type: none"> <li>• Beasley.</li> <li>• Lorena</li> <li>• Galvao and ReVelle.</li> </ul>

Tableau 3.2 Les instances de chaque variante de FLP.



### Un benchmark pour CFLP :

Cet ensemble de problèmes a été généré aléatoirement par Holmberg et al et comprend quatre sous-ensembles de problèmes. Les problèmes de test dans chaque sous-ensemble diffèrent pour les distributions des entrées et pour la taille, qui varie de 50 à 200 clients, et de 10 à 30 emplacements d'installations candidats.

Pr. No.	Opt.	Objective Value				% Deviation			
		<i>LH</i>	<i>bfs</i>	<i>dfs</i>	<i>bs</i>	<i>LH</i>	<i>bfs</i>	<i>dfs</i>	<i>bs</i>
p1	8848	8848	8848	8848	8848	0.000	0.000	0.000	0.000
p2	7913	7913	7913	7913	7913	0.000	0.000	0.000	0.000
p3	9314	9314	9314	9314	9314	0.000	0.000	0.000	0.000
p4	10714	10714	10714	10714	10714	0.000	0.000	0.000	0.000
p5	8838	8838	8838	8838	8838	0.000	0.000	0.000	0.000
p6	7777	7777	7777	7777	7777	0.000	0.000	0.000	0.000
p7	9488	9488	9488	9488	9488	0.000	0.000	0.000	0.000
p8	11088	11088	11088	11088	11088	0.000	0.000	0.000	0.000
p9	8462	8462	8462	8462	8462	0.000	0.000	0.000	0.000
p10	7617	7617	7617	7617	7617	0.000	0.000	0.000	0.000
p11	8932	8932	8932	8932	8932	0.000	0.000	0.000	0.000
p12	10132	10132	10132	10132	10132	0.000	0.000	0.000	0.000
p13	8252	8252	8252	8252	8252	0.000	0.000	0.000	0.000
p14	7137	7137	7137	7137	7137	0.000	0.000	0.000	0.000
p15	8808	8808	8808	8808	8808	0.000	0.000	0.000	0.000
p16	10408	10408	10408	10408	10408	0.000	0.000	0.000	0.000
p17	8227	8227	8227	8227	8227	0.000	0.000	0.000	0.000
p18	7125	7125	7125	7125	7125	0.000	0.000	0.000	0.000
p19	8886	8887	8886	8886	8886	0.011	0.000	0.000	0.000
p20	10486	10487	10486	10486	10486	0.010	0.000	0.000	0.000
p21	8068	8068	8068	8068	8068	0.000	0.000	0.000	0.000
p22	7092	7092	7092	7092	7092	0.000	0.000	0.000	0.000
p23	8746	8746	8746	8746	8746	0.000	0.000	0.000	0.000
p24	10273	10273	10273	10273	10273	0.000	0.000	0.000	0.000
Avg.						0.001	0.000	0.000	0.000

Figure 3.14 : les instances allant de p1 à p24.

Pr. No.	Opt. Sol.	Objective Value				% Deviation			
		<i>LH</i>	<i>bfs</i>	<i>dfs</i>	<i>bs</i>	<i>LH</i>	<i>bfs</i>	<i>dfs</i>	<i>bs</i>
p25	11630	11630	11630	11630	11630	0.000	0.000	0.000	0.000
p26	10771	10771	10771	10771	10777	0.000	0.000	0.000	0.056
p27	12322	12322	12327	12327	12327	0.000	0.041	0.041	0.041
p28	13722	13722	13727	13727	13727	0.000	0.036	0.036	0.036
p29	12371	12375	12379	12379	12379	0.032	0.065	0.065	0.065
p30	11331	11346	11399	11393	11394	0.132	0.600	0.547	0.556
p31	13331	13341	13436	13436	13436	0.075	0.788	0.788	0.788
p32	15331	15361	15436	15436	15436	0.196	0.685	0.685	0.685
p33	11629	11629	11629	11629	11629	0.000	0.000	0.000	0.000
p34	10632	10632	10632	10632	10632	0.000	0.000	0.000	0.000
p35	12232	12232	12232	12232	12232	0.000	0.000	0.000	0.000
p36	13832	13832	13832	13832	13832	0.000	0.000	0.000	0.000
p37	11258	11258	11258	11258	11258	0.000	0.000	0.000	0.000
p38	10551	10551	10551	10551	10551	0.000	0.000	0.000	0.000
p39	11824	11824	11824	11824	11824	0.000	0.000	0.000	0.000
p40	13024	13024	13024	13024	13024	0.000	0.000	0.000	0.000
Avg.						0.027	0.138	0.135	0.139

Figure 3.15 : les instances allant de p25 à p40.



Pr. No.	Opt. Sol.	Objective Value				% Deviation			
		<i>LH</i>	<i>bfs</i>	<i>dfs</i>	<i>bs</i>	<i>LH</i>	<i>bfs</i>	<i>dfs</i>	<i>bs</i>
p41	6589	6589	6590	6590	6590	0.000	0.015	0.015	0.015
p42	5663	5663	5663	5663	5663	0.000	0.000	0.000	0.000
p43	5214	5214	5214	5214	5214	0.000	0.000	0.000	0.000
p44	7028	7028	7028	7028	7028	0.000	0.000	0.000	0.000
p45	6251	6251	6251	6251	6251	0.000	0.000	0.000	0.000
p46	5651	5651	5651	5651	5651	0.000	0.000	0.000	0.000
p47	6228	6228	6238	6238	6238	0.000	0.161	0.161	0.161
p48	5596	5596	5600	5596	5600	0.000	0.071	0.000	0.071
p49	5302	5302	5302	5302	5302	0.000	0.000	0.000	0.000
p50	8741	8741	8757	8757	8757	0.000	0.183	0.183	0.183
p51	7414	7469	7425	7425	7425	0.742	0.148	0.148	0.148
p52	9178	9178	9178	9178	9178	0.000	0.000	0.000	0.000
p53	8531	8531	8603	8603	8603	0.000	0.844	0.844	0.844
p54	8777	8777	8777	8777	8777	0.000	0.000	0.000	0.000
p55	7654	7672	7654	7654	7654	0.235	0.000	0.000	0.000
Avg.						0.065	0.095	0.090	0.095

Figure 3.16 : les instances allant de p41 à p55.

Pr. No.	Opt. Sol.	Objective Value				% Deviation			
		<i>LH</i>	<i>bfs</i>	<i>dfs</i>	<i>bs</i>	<i>LH</i>	<i>bfs</i>	<i>dfs</i>	<i>bs</i>
p56	21103	21163	21120	21120	21120	0.284	0.081	0.081	0.081
p57	26039	26167	26061	26061	26061	0.492	0.084	0.084	0.084
p58	37239	37792	37261	37261	37261	1.485	0.059	0.059	0.059
p59	27282	27300	27282	27282	27282	0.066	0.000	0.000	0.000
p60	20534	20534	20534	20534	20534	0.000	0.000	0.000	0.000
p61	24454	24454	24454	24454	24454	0.000	0.000	0.000	0.000
p62	32643	32643	32651	32651	32651	0.000	0.025	0.025	0.025
p63	25105	25105	25105	25105	25105	0.000	0.000	0.000	0.000
p64	20530	20530	20530	20530	20530	0.000	0.000	0.000	0.000
p65	24445	24445	24445	24445	24445	0.000	0.000	0.000	0.000
p66	31415	31504	31503	31506	31495	0.283	0.280	0.290	0.255
p67	24848	24876	24849	24849	24849	0.113	0.004	0.004	0.004
p68	20538	20538	20538	20538	20538	0.000	0.000	0.000	0.000
p69	24532	24532	24532	24532	24532	0.000	0.000	0.000	0.000
p70	32321	32393	32333	32333	32333	0.223	0.037	0.037	0.037
p71	25540	25562	25540	25540	25540	0.086	0.000	0.000	0.000
Avg.						0.189	0.036	0.036	0.034

Figure 3.17 : les instances allant de p56 à p71.

### Les algorithmes de résolutions du CFLP sous python :

Supposons qu'il y ait 10 à 30 installations(n) et 50 à 200 clients(m). Nous souhaitons choisir :

- (1) lequel des n établissements ouvrir
- (2) l'affectation des clients aux installations
- (3) La demande totale affectée à une installation ne doit pas dépasser sa capacité.

L'objectif est de minimiser la somme du coût d'ouverture et du coût d'affectation.

	Opt bench	LS	LS-OPT	SA	SA-OPT	TS	TS-OPT	GR	GR-OPT
P1	8848	9463	615	8958	110	8963	115	9440	592
P4	10714	12438	1724	11068	354	10989	275	12126	1412
P10	7617	8224	607	7626	9	7700	83	7726	109
P30	11331	13678	2347	11769	438	11880	549	16239	4908
P35	12232	16458	4226	12577	345	13505	1273	21389	9157
P46	5651	8319	2668	6253	602	6156	505	12639	6988
P52	21103	23016	1913	22376	1273	23882	2779	23882	2779
P66	31415	50097	18682	41954	10539	47501	16086	53882	22467
P70	32321	52630	20309	43073	10752	52561	20240	53882	21561
P71	25540	38067	12527	33842	8302	38241	12701	39121	13581

**Tableau 3.3 : Les résultats de benchmark après la résolution.**

➤ Tableau classement :

	P1	P4	P10	P30	P35	P46	P52	P66	P70	P71	Nombre de fois classé
<b>LS</b>	4	4	4	3	3	3	2	3	3	2	0
<b>SA</b>	1	2	1	1	1	2	1	1	1	1	8
<b>TS</b>	2	1	2	2	2	1	3	2	2	3	2
<b>GR</b>	3	3	3	4	4	4	3	4	4	4	0

**Tableau 3.4: Le classement des algorithmes.**

### 3.6.2 Quelques benchmarks du VRP :

Nous donnons quelques instances pour le VRP :

La variante	Les instances ()
<b>VRP</b>	<ul style="list-style-type: none"> <li>• K.F.Wong and J.E.Beasley.</li> <li>• J.E Beasley.</li> <li>• N.Christofides and J.E.Beasley.</li> <li>• C.D.T.Watson-Gandy and N.Christofides.</li> <li>• N.Christofides, A.Mingozzi, P.Toth and C.Sandi.</li> </ul>
<b>CVRP</b>	<ul style="list-style-type: none"> <li>• Augerat et al.</li> <li>• Christofides and Eilon.</li> <li>• Fisher.</li> <li>• Christofides, Mingozzi and Toth.</li> <li>• Golden et al.</li> <li>• Li et al.</li> <li>• Uchoa et al.</li> <li>• Arnold, Gendreau and Sörensen.</li> </ul>
<b>VRPTW</b>	<ul style="list-style-type: none"> <li>• Solomon.</li> <li>• Homberger and Gehring.</li> </ul>

**Tableau 3.5 : Les instances de chaque variante de VRP.**

#### Un benchmark pour CVRP :

Les instances de **Augerat et al** sont les suivantes (Solution exacte) :

<b>Instance A-n32-k5</b>	1	21 31 19 17 13 7 26
	2	12 1 16 30
	3	27 24
	4	29 18 8 9 22 15 10 25 5 20
	5	14 28 11 4 23 3 2 6

**Tableau 3.6 : Les tournées obtenues de l'instance A-n32-k5 (Solution exacte).**

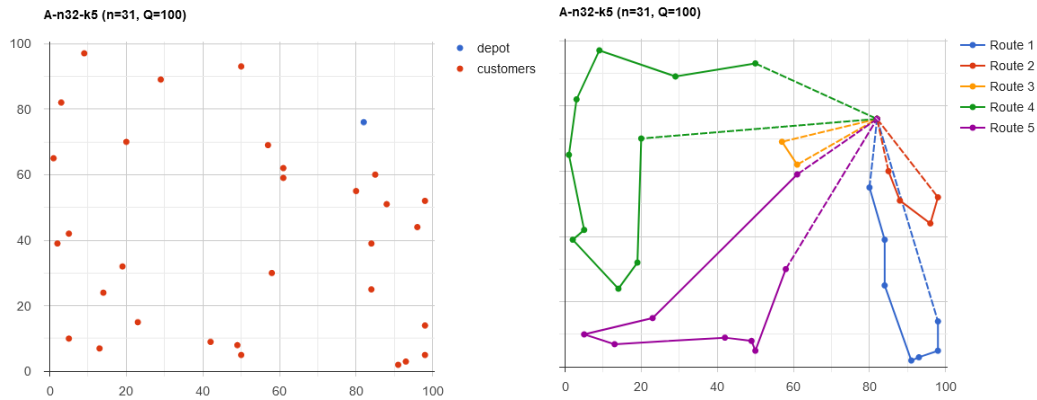


Figure 3.18 : Représentation de l'instance A-n32-k5 (Solution exacte).

Les algorithmes de résolutions du l'instance A-n32-k5 sous python :

1. On résout l'instance par l'algorithme de colonie de fourmis AC (Ant Colony):

Instance A-n32-k5	1	24 27
	2	30 16 1 12
	3	6 3 2 23 4 11 28 14
	4	18 8 9 22 15 10 25 29 5 20
	5	7 13 17 19 31 21 26

Tableau 3.7 : Les tournées obtenues de l'instance A-n32-k5 avec AC.

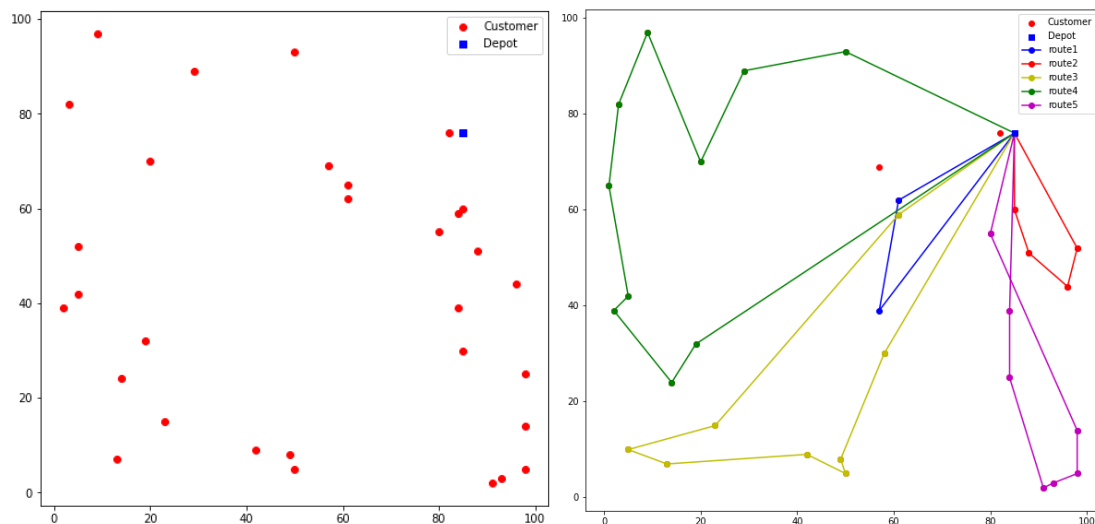
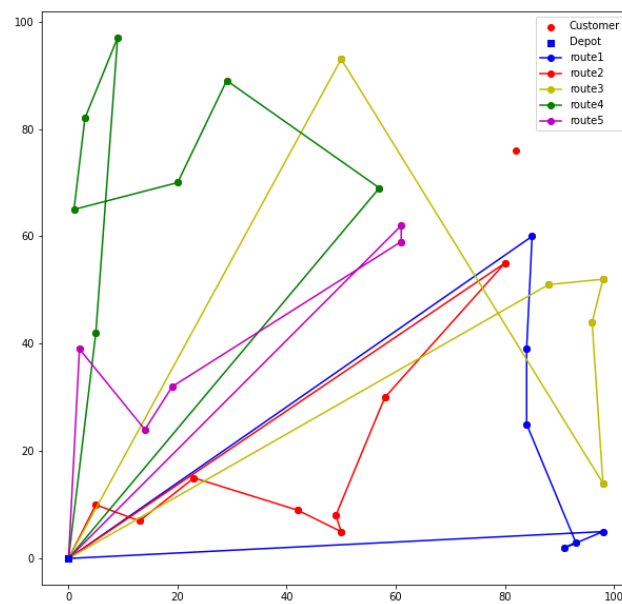


Figure 3.19 : Représentation de l'instance A-32-k5 avec AC.

2. On résout l'instance par l'algorithme **Greedy** :

<b>Instance A-n32-k5</b>	<b>1</b>	30 7 13 19 17 31
	<b>2</b>	26 6 3 2 23 28 4 11
	<b>3</b>	16 12 1 21 20
	<b>4</b>	27 5 29 15 10 25 22
	<b>5</b>	24 14 18 8 9

**Tableau 3.8 : Les tournées obtenues de l'instance A-n32-k5 avec Greedy.**



**Figure 3.20 : Représentation de l'instance A-32-k5 avec Greedy.**

## ➤ l'instance B-n78-k10 :

<b>Instance B-n78-k10</b>	<b>1</b>	1 52 24 21 43 67 69 72 17
	<b>2</b>	2 63 49 27 56 38 58
	<b>3</b>	5 30 70 29 37 65
	<b>4</b>	8 22 71 31 73 32 76 7 57 47
	<b>5</b>	9 19 33 55 35 59 4 36 14 41 10
	<b>6</b>	11 64 40 53 68 18 25 42
	<b>7</b>	15 12 34 46 45 51
	<b>8</b>	16 66 60 6 62 54 20 3 75
	<b>9</b>	23 26 44 77
	<b>10</b>	28 74 48 13 39 61 50

Tableau 3.9: Les tournées obtenues de l'instance B-n78-k10 (solution exacte).

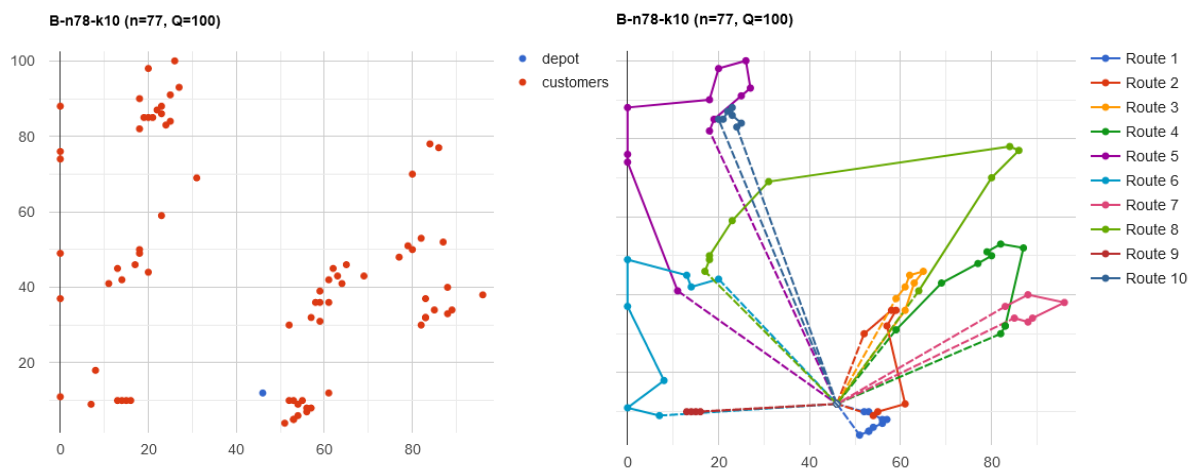


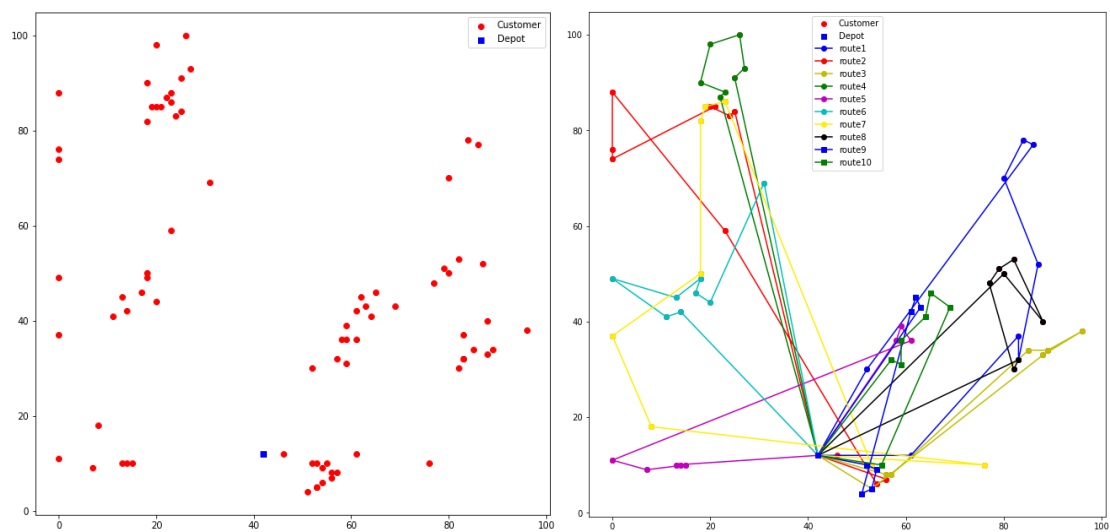
Figure 3.21: Représentation de l'instance B-n78-k10 (solution exacte).

### Les algorithmes de résolutions du l'instance B-n78-k10 sous python :

1. On résout l'instance par l'algorithme de colonie de fourmis AC (Ant Colony):

<b>Instance B-n78-k10</b>	<b>1</b>	2 20 54 3 31 71 51 56
	<b>2</b>	61 50 28 74 41 14 36 6 21 43
	<b>3</b>	24 12 46 34 15 67 69
	<b>4</b>	48 13 4 59 35 55 33
	<b>5</b>	63 5 65 64 11 23 44 26
	<b>6</b>	25 10 68 18 66 16 42 62
	<b>7</b>	72 39 19 9 60 53 40 77
	<b>8</b>	76 45 73 32 7 8 22
	<b>9</b>	37 70 30 1 52 58 17
	<b>10</b>	38 57 29 75 49 47 27

**Tableau 3.10: Les tournées obtenues de l'instance B-n78-k10 avec AC.**

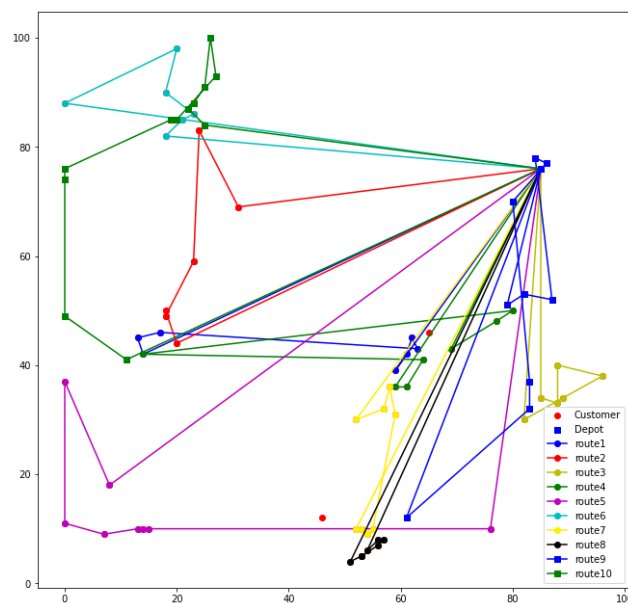


**Figure 3.22 : Représentation de l'instance B-n78-k10 avec AC.**

2. On résout l'instance par l'algorithme de **Greedy** :

<b>Instance B-n78-k10</b>	<b>1</b>	25 18 16 37 70 30 5
	<b>2</b>	62 5 6 66 60 42
	<b>3</b>	8 34 46 45 12 15
	<b>4</b>	49 65 75 25 76 7 57
	<b>5</b>	40 53 64 11 26 44 23 77
	<b>6</b>	9 74 39 4 59 36
	<b>7</b>	17 72 58 38 47 63 27 2
	<b>8</b>	1 24 52 43 67 69 21
	<b>9</b>	56 22 71 51 3 20 54 31 73 32
	<b>10</b>	61 48 13 33 35 55 28 19 14 41 68 10

**Tableau 3.11: Les tournées obtenues de l'instance B-n78-k10 avec Greedy.**



**Figure 3.23 : Représentation de l'instance B-n78-k10 avec Greedy.**



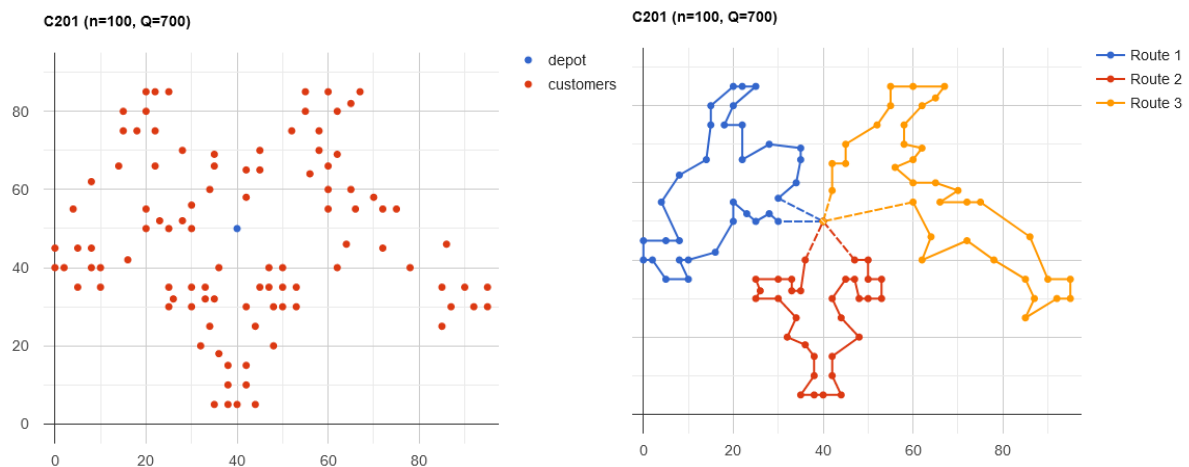
### Un benchmark pour VRPTW :

Soit les instances de **Solomon** suivantes :

#### 1. l'instance C201 :

<b>1</b>	20 22 24 27 30 29 6 31 32 33 35 37 38 39 36 34 28 26 23 25 9 13 17 18 19 15 16 14 12 11 10 8 21
<b>2</b>	67 63 69 66 62 74 72 61 64 68 65 49 55 54 53 56 58 60 59 57 40 44 46 41 42 45 51 50 52 47 43 48
<b>3</b>	93 5 75 2 1 99 100 97 92 94 95 98 7 3 4 89 91 88 84 83 82 85 71 70 73 80 79 76 81 78 77 96 87 86 90

**Tableau 3. 12: Les tournées obtenues de l'instance C201 (solution exacte).**



**Figure 3.24 : Représentation de l'instance C201 (solution exacte).**

Les algorithmes de résolutions du l'instance C201 sous python :

#### 1. Par l'algorithme de AC :

<b>1</b>	93 5 75 2 1 99 100 97 92 94 95 98 7 3 4 89 91 88 84 86 83 82 85 76 71 70 73 80 79 81 78 77 96 87 90
<b>2</b>	67 63 62 74 72 61 64 66 69 68 65 49 55 54 53 56 58 60 59 57 40 44 46 45 51 50 52 47 43 42 41 48
<b>3</b>	20 22 24 27 30 29 6 32 33 31 35 37 38 39 36 34 28 26 23 18 19 16 14 12 15 17 13 25 9 11 10 8 21

**Tableau 3.13 : Les tournées obtenues de l'instance C201 avec AC.**

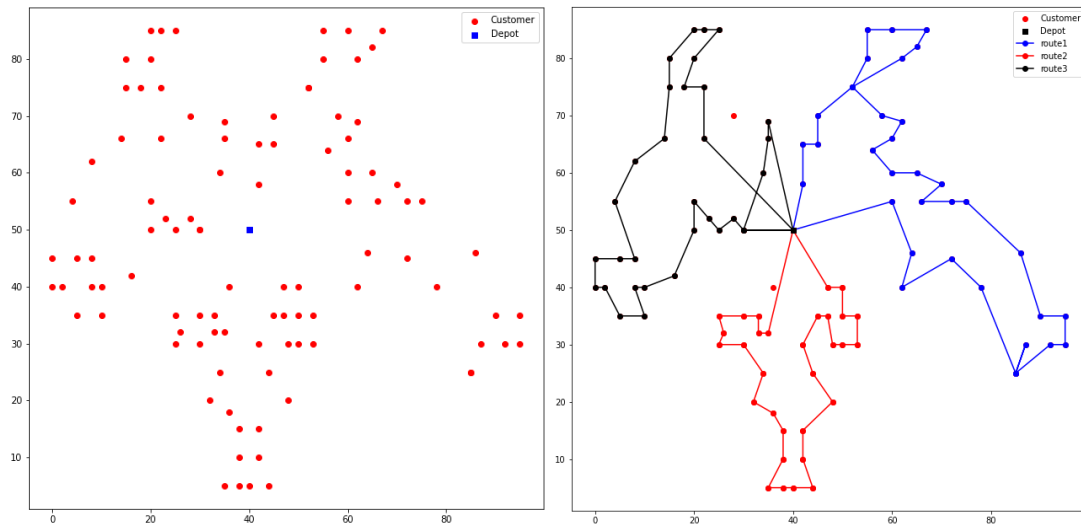


Figure 3.25 : Représentation de l'instance C201 avec AC.

## 2. Par l'algorithme de PSO :

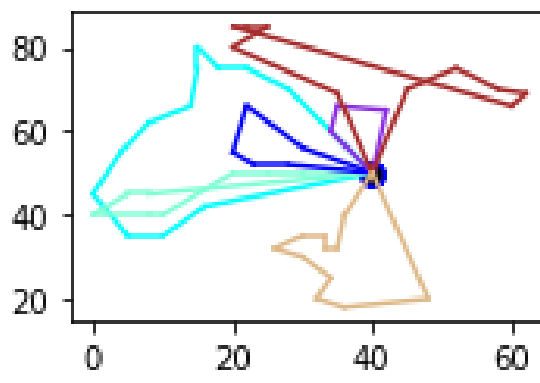


Figure 3.26 : Représentation de l'instance C201 avec PSO.

## 2. L'instance C101 :

1	5 3 7 8 10 11 9 6 4 2 1 75
2	13 17 18 19 15 16 14 12
3	20 24 25 27 29 30 28 26 23 22 21
4	32 33 31 35 37 38 39 36 34
5	43 42 41 40 44 46 45 48 51 50 52 49 47
6	57 55 54 53 56 58 60 59
7	67 65 63 62 74 72 61 64 68 66 69
8	81 78 76 71 70 73 77 79 80
9	90 87 86 83 82 84 85 88 89 91
10	98 96 95 94 92 93 97 100 99

Tableau 3. 14 : Les tournées obtenues de l'instance C101-n100 (solution exacte).

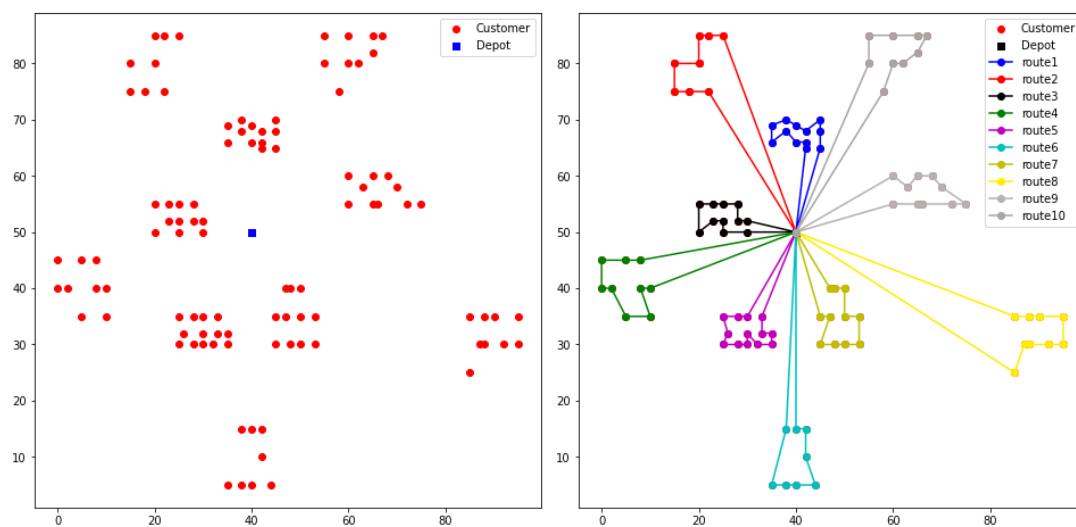


Figure 3.27: Représentation de l'instance C101-n100 (solution exacte).

Les algorithmes de résolutions du l'instance C101-n100 sous python :

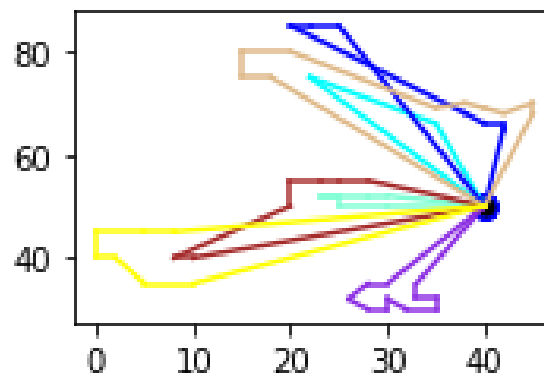


Figure 3.28: Représentation de l'instance C101-n100 avec PSO.