

Le butinage: une optimisation naturelle d'un cycle de recherche

Lamnini Yassine

Cycles et Boucles: 2025-2026



Figure 1



Figure 2

Question centrale

Comment le trajet des abeilles permet-il d'approcher une solution optimale du problème du voyageur de commerce en maximisant le rapport nectar/distance traversée ?

Plan

- 1 Introduction d'un algorithme
- 2 Vers des applications plus réalistes
- 3 Comparaison avec d'autres algorithmes
- 4 Synthèse
- 5 Annexes

Algorithme

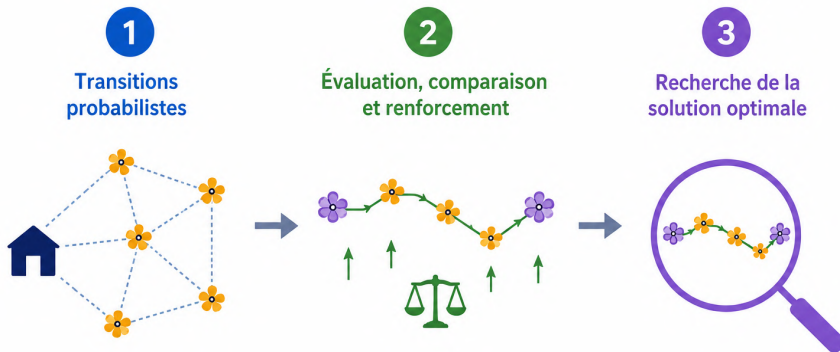
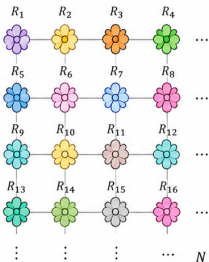


Figure 3

Etape 1

1 Réseau de fleurs et récompenses



Chaque fleur j possède une récompense R_j (gain)

2 Distances depuis i

	j (destination)					
	1	2	3	...	N	
1	$d_{1,1}$	$d_{1,2}$	$d_{1,3}$	...	$d_{1,N}$	
2	$d_{2,1}$	$d_{2,2}$	$d_{2,3}$	...	$d_{2,N}$	
3	$d_{3,1}$	$d_{3,2}$	$d_{3,3}$	...	$d_{3,N}$	
...	...	...	...	...	...	
N	$d_{N,1}$	$d_{N,2}$	$d_{N,3}$	...	$d_{N,N}$	

$d_{i,j}$: distance entre la fleur i et la fleur j
(toutes distinctes, $i \neq j$)

3 Matrice de probabilités P (avec récompenses)

	j (destination)					
	1	2	3	...	N	
1	$p_{1,1}$	$p_{1,2}$	$p_{1,3}$	...	$p_{1,N}$	
2	$p_{2,1}$	$p_{2,2}$	$p_{2,3}$	...	$p_{2,N}$	
3	$p_{3,1}$	$p_{3,2}$	$p_{3,3}$	...	$p_{3,N}$	
...	...	...	...	...	...	
N	$p_{N,1}$	$p_{N,2}$	$p_{N,3}$	...	$p_{N,N}$	

Probabilité de passer de i à j :

$$P[i, j] = \frac{\left(\frac{R_j}{d_{i,j}}\right)^2}{\sum_{k \neq i} \left(\frac{R_k}{d_{i,k}}\right)^2}$$

$$\sum_{j=1}^N P[i, j] = 1 \quad (\text{ligne } i)$$

R_j : récompense de la fleur j
(hors fleur i)

Plus j est proche (petit $d_{i,j}$)
et plus sa récompense R_j
est grande,
plus $P[i, j]$ est grande.

Figure 4

```

4 F = [] #liste des coords des fleurs
5 R = [] #liste des valeurs de nectar pour chaque fleur
6
7 # 1ère étape: Transitions probabilistes
8 def dist():
9     dist = np.zeros((N+1, N+1))
10    for i in range(1,N+1):
11        for j in range(i+1,N+1):
12            dist[i,j] = dist[j,i] = np.linalg.norm(F[i-1] - F[j-1])
13    for i in range(1,N+1):
14        dist[i,0] = dist[0,i] = np.linalg.norm(Nest-F[i-1])
15    return dist
16
17 dist = dist()
18
19 def init_P():
20     P = np.zeros((N+1, N+1))
21     for i in range(N+1):
22         for j in range(i+1,N+1):
23             P[i,j] = P[j,i] = 1 / (dist[i,j]**2)
24             P[i,j] *= R[j]**2
25             P[j,i] *= R[i]**2
26         P[i,:] /= P[i,:].sum()
27     return P
28
29 P = init_P()
30
31 def first(historic, best_seq):
32     if len(historic) <= 3 : return np.random.randint(N)
33     return best_seq[0]

```

Figure 5

```

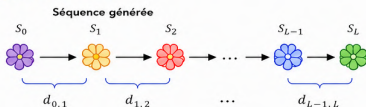
32 def next(seq, i_current):
33     candidates = [j for j in range(N) if j not in seq]
34     # trier par probabilité décroissante
35     candidates.sort(key=lambda t: P[i_current+1, t+1], reverse=True)
36     return candidates[0]
37
38 def run_seq(historic, best_seq):
39     seq = []
40     cur = first(historic, best_seq)
41     seq.append(cur)
42     gain = NectF
43     steps = 1
44     while gain < C and steps < N:
45         nxt = next(seq, cur)
46         seq.append(nxt)
47         cur = nxt
48         gain += NectF
49         steps += 1
50     return seq

```

Figure 6

Etape 2

1 ÉVALUATION



Distance totale (coût aller-retour)

$$L = \sum_{k=0}^{L-1} d_{S_k, S_{k+1}}$$

(un seul terme dans la somme)

Gain total (récompenses)

$$R = \sum_{k=0}^L r_{S_k}$$

(s_k : nœud visité, r_{s_k} : gain du nœud)

Critère d'évaluation : on cherche à **MAXIMISER** le rapport $\frac{R}{L}$
(gain par unité de distance)

2 COMPARAISON

Meilleure solution connue

(L^{best}, R^{best})

$$\text{Rapport}^{best} = \frac{R^{best}}{L^{best}}$$

Solution courante

(L^{cur}, R^{cur})

$$\text{Rapport}^{cur} = \frac{R^{cur}}{L^{cur}}$$

Comparaison des rapports
(gain / longueur)

$$\text{Si } \frac{R^{cur}}{L^{cur}} > \frac{R^{best}}{L^{best}}$$

⇒ Meilleure solution

(on remplace la meilleure connue par la solution courante)

3 RENFORCEMENT (si meilleure solution)

Transitions utilisées
dans la séquence



Mise à jour de la probabilité
de la transition $i \rightarrow j$

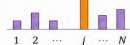
$$P_{i,j} \leftarrow P_{i,j} \times \gamma \quad (\gamma > 1)$$

Puis normalisation de la ligne i
pour garder $\sum_j P_{i,j} = 1$

Avant (ligne i)



Après (ligne i)



La transition utilisée ($i \rightarrow j$)
devient **plus probable**

Figure 7

```

52 # 2ème étape: Evaluation + Comparaison + Renforcement
53 def length(sequence):
54     L = 0
55     for k in range(len(sequence)-1):
56         L += dist[sequence[k]+1, sequence[k+1]+1]
57     # aller et retour au nid
58     L += dist[0, sequence[0]+1]
59     L += dist[sequence[-1]+1, 0]
60     return L
61
62 def reward(seq):
63     r = 0
64     for i in seq:
65         r += R[i]
66     return r
67
68 def Gain(seq):
69     return reward(seq) - c * length(seq)
70
71 def reinforce_seq(seq):
72     for k in range(len(seq)-1):
73         i, j = seq[k], seq[k+1]
74         P[i,j] *= gamma
75         P[i,:] /= P[i,:].sum()
76     # nid -> première fleur
77     P[0,seq[0]] *= gamma
78     P[0,:] /= P[0,:].sum()
79     # dernière fleur -> nid
80     P[seq[-1],0] *= gamma
81     P[seq[-1],:] /= P[seq[-1],:].sum()

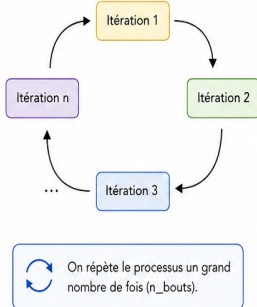
```

Figure 8

Etape 3

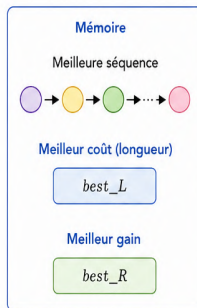
1 RÉPÉTITION DES ITÉRATIONS

On génère et évalue plusieurs séquences possibles.



2 MÉMORISATION DU MEILLEUR

À chaque itération, on mémorise la meilleure séquence trouvée jusqu'ici.



3 CHOIX DU MEILLEUR À LA FIN

Après toutes les itérations, on retourne la meilleure séquence trouvée.

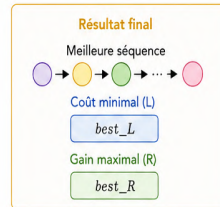


Figure 9

```
77 # 3ème étape: Recherche de solution optimale
78 def run(n_bouts):
79     historic = []
80     best_seq = None
81     r_max = 0
82     for _ in range(n_bouts):
83         seq = run_seq(historic, best_seq)
84         L = length(seq)
85         g = Gain(seq)
86         historic.append((seq, L))
87         if g/L >= r_max:
88             r_max = g/L
89             best_seq = seq
90             reinforce_seq(seq)
91     return historic, best_seq
```

Figure 10

Définition

La qualité d'un trajet est :

$$QL = \frac{g}{L} \text{ et } QL_{\%} = \frac{QL}{QL_{\text{opt}}}$$

- g : Gain accumulé par l'abeille (μL)
- L : distance totale parcourue lors du trajet (m)
- QL_{opt} : qualité du trajet optimal

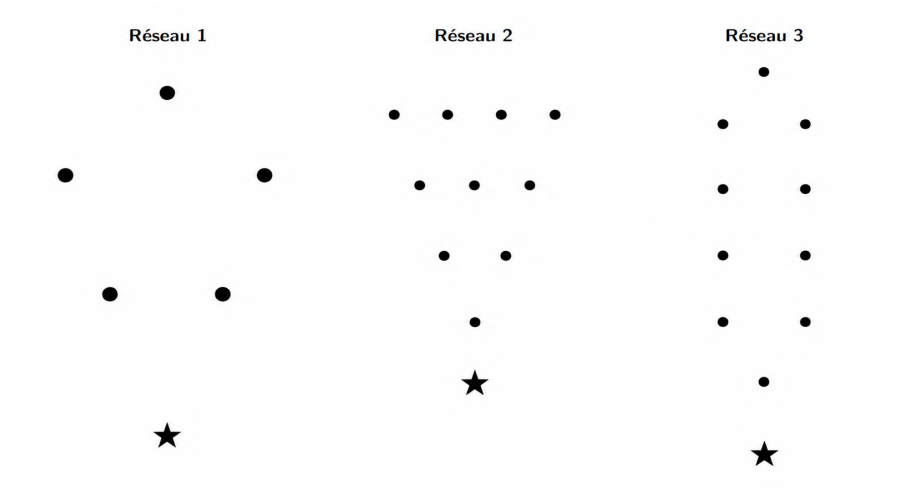


Figure 11

Trajets

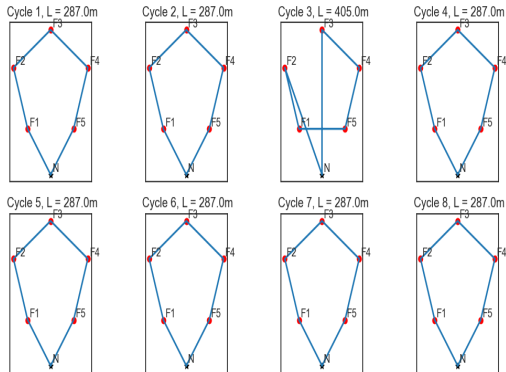


Figure 12

Evolution

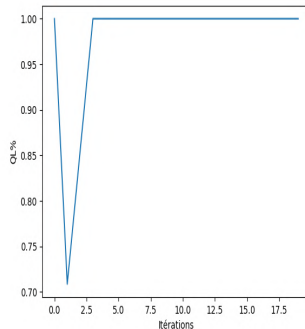


Figure 13

Trajets

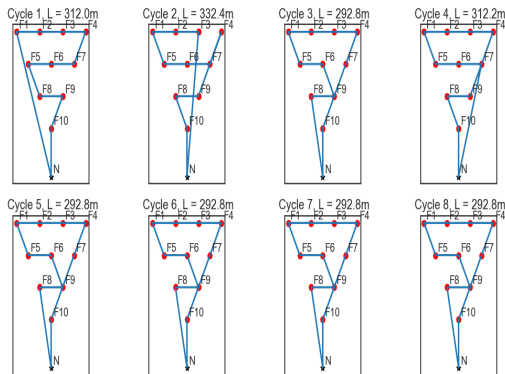


Figure 14

Evolution

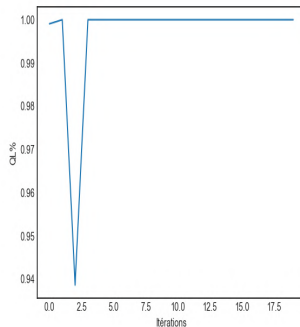


Figure 15

Trajets

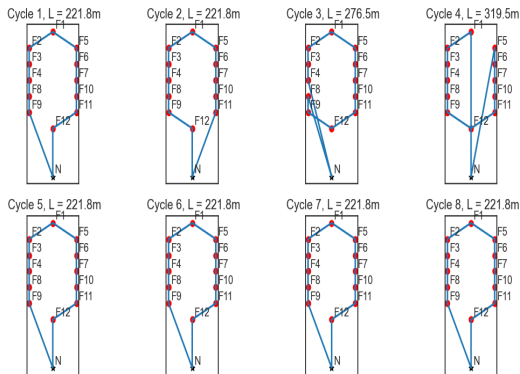


Figure 16

Evolution

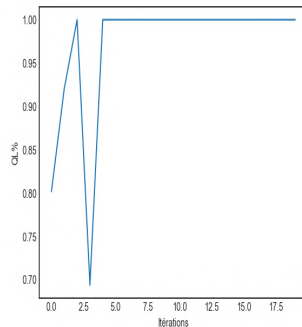


Figure 17

Plan

- 1 Introduction d'un algorithme
- 2 Vers des applications plus réalistes**
- 3 Comparaison avec d'autres algorithmes
- 4 Synthèse
- 5 Annexes

Distribution de fleurs

```
11 import numpy as np
12 import random
13
14 random.seed(42)
15
16 all_points = [
17     (i, j)
18     for i in range(n)
19     for j in range(n)
20 ]
21 F = random.sample(all_points, N)
22 F = np.array(F)
23
24 Nest = np.array((10,10))
```

Figure 18

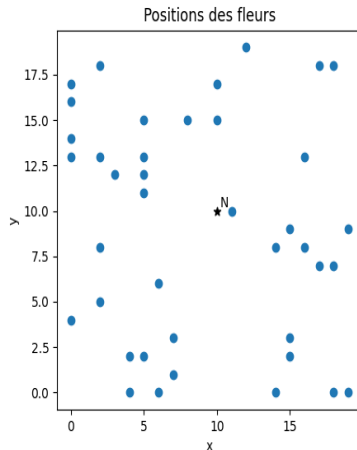


Figure 19

Hypothèse

La loi de variation est :

$$n(x, y) = \sum_{k=1}^K n_k \exp \left(-\frac{(x - x_k)^2 + (y - y_k)^2}{2\sigma_k^2} \right)$$

- K : nombre de zones riches en nectar
- n_k : Amplitude dans la zone k
- (x_k, y_k) : position du centre de la zone k
- σ_k : étalement spatial du nectar

$k=1$

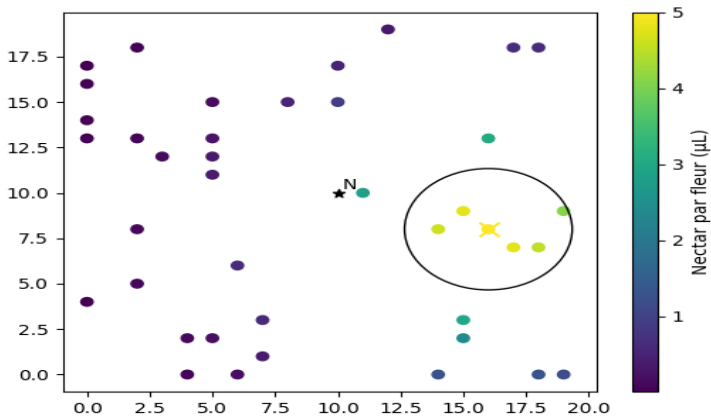


Figure 20

Trajets

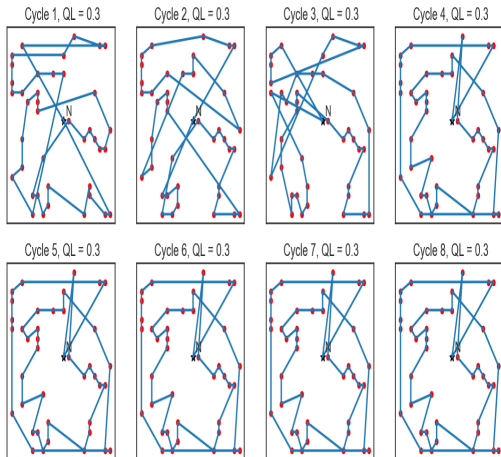


Figure 21

Evolution

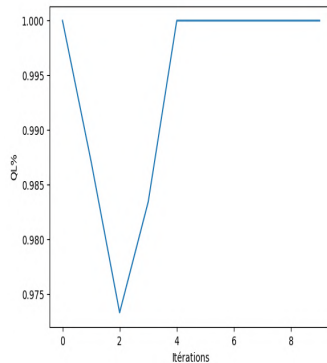


Figure 22

$k = 2$

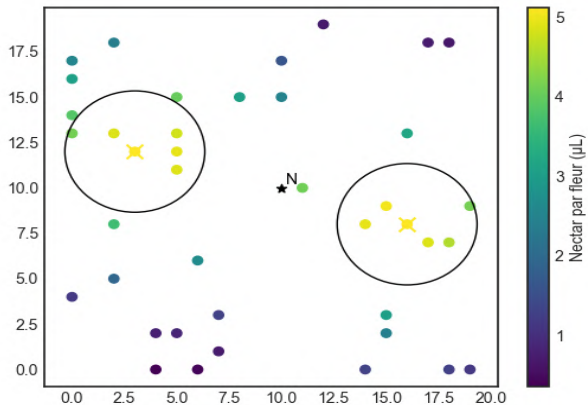


Figure 23

Trajets

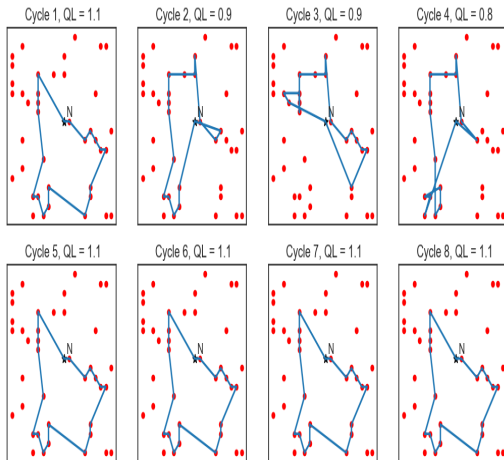


Figure 24

Evolution

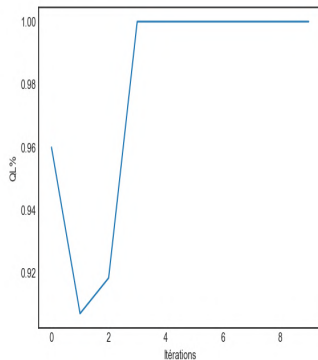


Figure 25

Plan

- 1 Introduction d'un algorithme
- 2 Vers des applications plus réalistes
- 3 Comparaison avec d'autres algorithmes**
- 4 Synthèse
- 5 Annexes

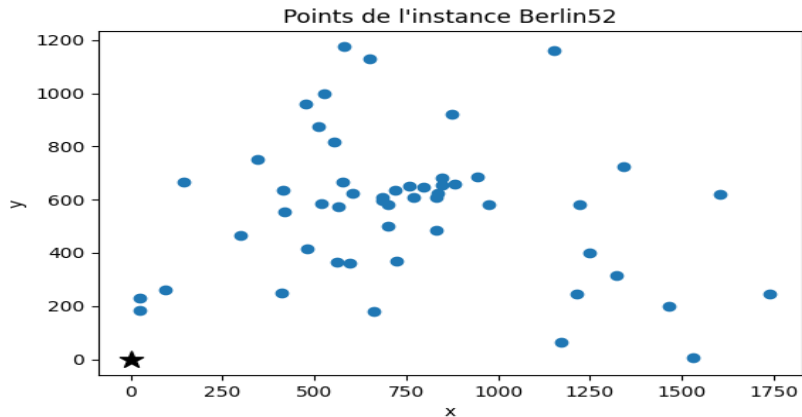


Figure 26

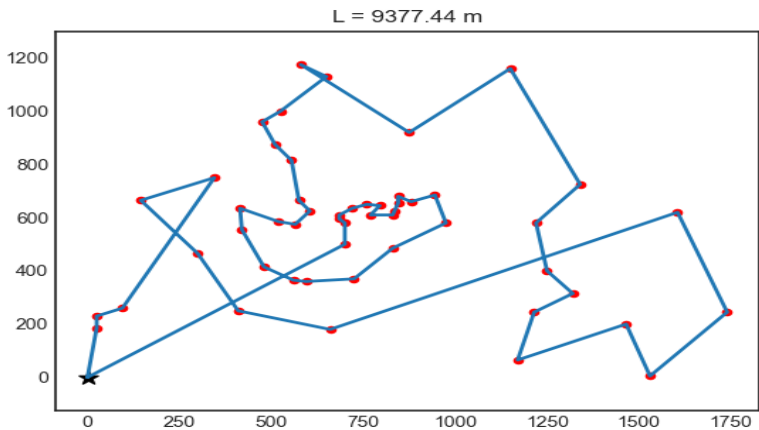


Figure 27

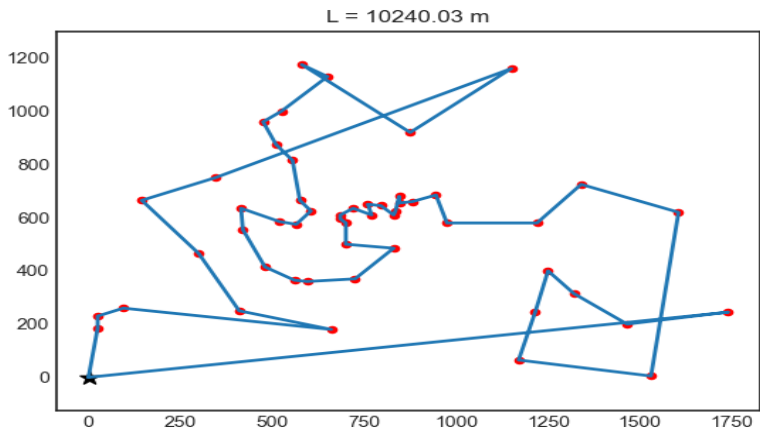


Figure 28

2-opt

```
94 tour = list(range(N))  
95 tour = two_opt(tour)
```

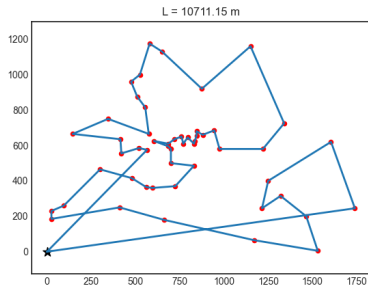


Figure 29

```
94 tour = run_glouton(F)  
95 tour = two_opt(tour)
```

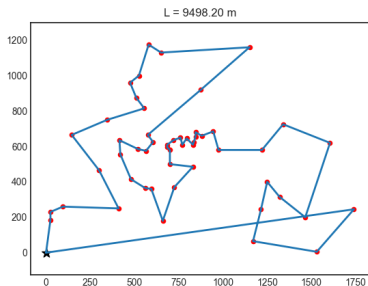


Figure 30

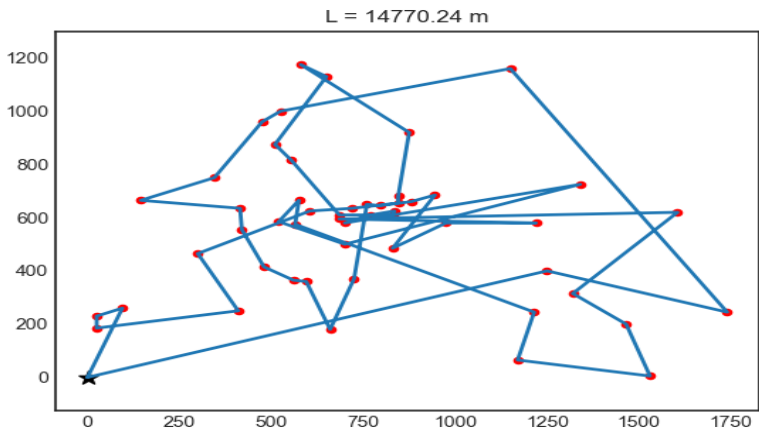


Figure 31

Algorithme	Longueur(m)	Temps(s)
A. A.	9377.44	0.022721
Glouton	10240.03	0.025576
2-Opt	10711.15	0.192914
2-Opt + Glouton	9498.20	0.147456
Algorithme génétique	14770.24	0.737773

Table 1

Problème plus réduit

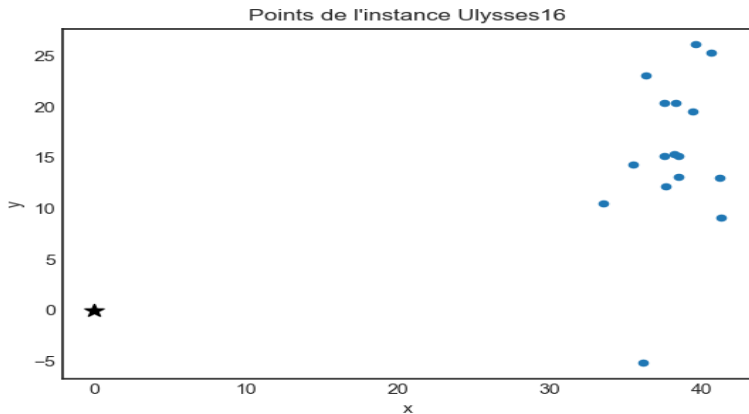


Figure 32

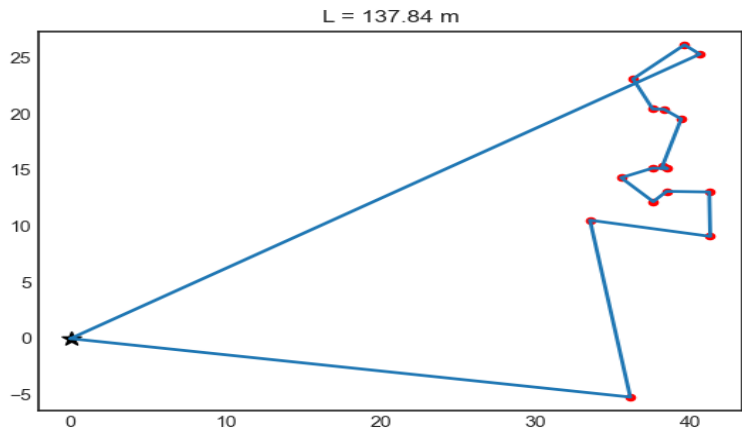


Figure 33

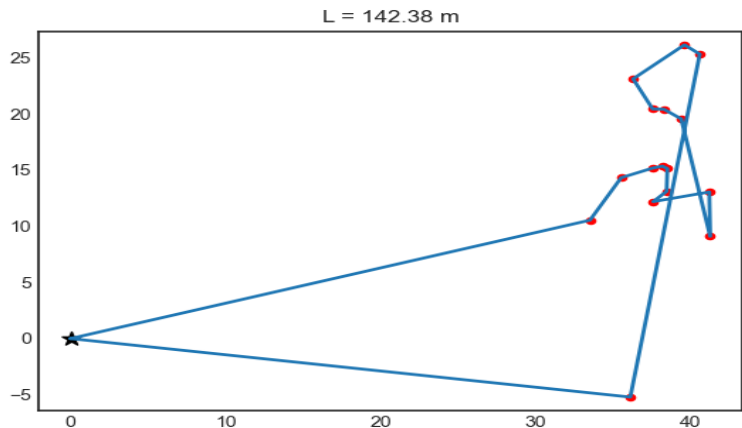


Figure 34

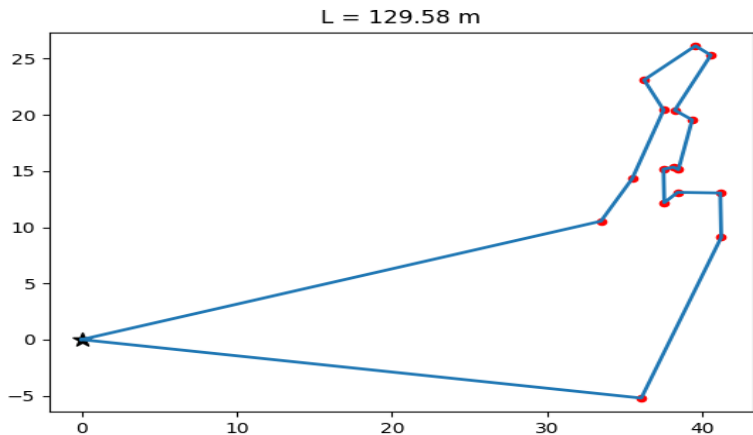


Figure 35

Algorithme	Longueur(m)	Temps(s)
A. A.	137.84	0.006793
Glouton	142.38	0.002389
Prog. dynamique	129.58	7.472813

Table 2

Positions des clients sur la carte de Casablanca



Figure 36

Situation problématique

Client	Prix(dh)	Espace(m ³)
1	100	0.03
2	200	0.05
3	300	0.06
4	150	0.04
5	1000	0.15
6	2000	0.30
7	5000	0.65
8	4400	0.60
9	2500	0.38
10	500	0.09

Client	Prix(dh)	Espace(m ³)
11	150	0.04
12	230	0.05
13	140	0.04
14	10000	1.1
15	9000	0.95
16	210	0.05

Table 3

Situation problématique

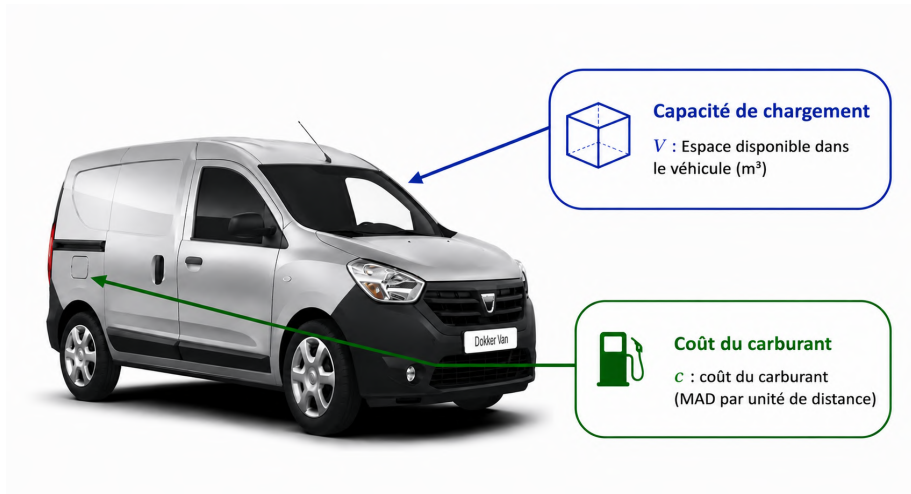


Figure 37

$L = 60.16 \text{ km} \mid g = 32754.88 \text{ MAD}$



Figure 38

L = 68.79 km | g = 35358.41 MAD



Figure 39

$L = 60.76 \text{ km} \mid g = 31754.43 \text{ MAD}$



Figure 40

Algorithme	Longueur(Km)	Gain(dh)	Temps(s)
A. A.	60.16	32754.88	26.708585
Glouton	68.79	35358.41	29.488834
Prog. dynamique	60.76	31754.43	377.72155

Table 4

Plan

- 1 Introduction d'un algorithme
- 2 Vers des applications plus réalistes
- 3 Comparaison avec d'autres algorithmes
- 4 Synthèse**
- 5 Annexes

Avantages

- Rapide
- Approche la solution optimale

Inconvénients

- Ne converge pas vers une solution fixe
- Sensible au départ

Plan

- 1 Introduction d'un algorithme
- 2 Vers des applications plus réalistes
- 3 Comparaison avec d'autres algorithmes
- 4 Synthèse
- 5 Annexes

```
36 def first():
37     L_min = float('inf')
38     j = 0
39     for k in range(1,N+1):
40         L = dist[0,k]
41         if L <= L_min:
42             L_min = L
43             j = k
44     return k-1
45
46 def next(i_c,seq):
47     L_min = float('inf')
48     j = 0
49     for k in range(1,N+1):
50         L = dist[i_c+1,k]
51         if k-1 not in seq and L <= L_min:
52             L_min = L
53             j = k-1
54     return j
55
```

```
65 def run_glouton(F):
66     seq = []
67     cur = first()
68     seq.append(cur)
69     gain = client
70     steps = 1
71     while gain < C and steps < N:
72         nxt = next(cur, seq)
73         seq.append(nxt)
74         cur = nxt
75         gain += client
76         steps += 1
77     return seq
78
79 seq = run_glouton(F)
```

2-opt

```
22 def dist(a, b):
23     return np.linalg.norm(a - b)
24
25 def two_opt(tour):
26     improved = True
27     n = len(tour)
28     while improved:
29         improved = False
30         for i in range(1, n-2):
31             for j in range(i+1, n-1):
32                 # coût avant / après swap
33                 a, b = tour[i - 1], tour[i]
34                 c, d = tour[j], tour[j + 1]
35                 if dist(F[a], F[b]) + dist(F[c], F[d]) > dist(F[a], F[c]) + dist(F[b], F[d]):
36                     tour[i:j+1] = reversed(tour[i:j+1])
37                     improved = True
38     return tour
```

Algorithme genetique

```
42 def init_population(n_pop):# génération initiale
43     pop = []
44     base = list(range(N))
45     for _ in range(n_pop):
46         t = base[:]
47         random.shuffle(t)
48         pop.append(t)
49     return pop
50
51 def crossover(p1, p2):# crossover
52     n = len(p1) - 1
53     a, b = sorted(random.sample(range(n), 2))
54     child = [-1] * n
55     child[a:b] = p1[a:b]
56     fill = [x for x in p2 if x not in child]
57     idx = 0
58     for i in range(n):
59         if child[i] == -1:
60             child[i] = fill[idx]
61             idx += 1
62     child.append(child[0])
63     return child
64
65 def mutate(tour, p=0.1):# mutation
66     if random.random() < p:
67         i, j = random.sample(range(len(tour)-1), 2)
68         tour[i], tour[j] = tour[j], tour[i]
69     return tour
70
71 def genetic_algorithm(F, n_pop, n_gen):
72     pop = init_population(n_pop)
73     for _ in range(n_gen):
74         pop = sorted(pop, key=lambda t: length(t))
75         new_pop = pop[:5] # élitisme
76         while len(new_pop) < n_pop:
77             p1, p2 = random.sample(pop[:20], 2)
78             child = crossover(p1, p2)
79             child = mutate(child)
80             new_pop.append(child)
81         pop = new_pop
82     pop.sort(key=lambda t: length(t), reverse= True)
83     return pop[0]
```

Prog. dynamique

```
30 n = len(points)
31
32 D = np.zeros((n, n))
33
34 for i in range(n):
35     for j in range(n):
36         D[i, j] = np.linalg.norm(points[i] - points[j])
37
38 dp = {}
39
40 # Initialisation
41 for j in range(1, n):
42     dp[(frozenset([j]), j)] = (D[0, j], 0)
43
44 # Remplissage DP
45 for size in range(2, n):
46     for subset in combinations(range(1, n), size):
47         S = frozenset(subset)
48         for j in subset:
49             prev_S = S - {j}
50             best_cost = float('inf')
51             best_parent = None
52             for i in prev_S:
53                 cost = dp[(prev_S, i)][0] + D[i, j]
54                 if cost < best_cost:
55                     best_cost = cost
56                     best_parent = i
57             dp[(S, j)] = (best_cost, best_parent)
58
59 S = frozenset(range(1, n))
60
61 best_cost = float('inf')
62 best_last = None
```

```
64 for j in range(1, n):
65     cost = dp[(S, j)][0] + D[j, 0]
66     if cost < best_cost:
67         best_cost = cost
68         best_last = j
69
70 tour = [0]
71
72 current = best_last
73 current_set = S
74
75 for _ in range(n - 1):
76     tour.append(current)
77     _, parent = dp[(current_set, current)]
78     current_set = current_set - {current}
79     current = parent
80
81 tour.append(0)
82
83 tour = tour[::-1]
84
85 path = points[tour]
```

Données numériques

```
1 # 1er axe
2 M = 8 # nbr. d'itérations
3 gamma = 1.2 # coeff. de renforcement
4 C = 60 # capacité de stockage de l'abeille (µL)
5 nect0 = 5 # Quantité de nectar fournie par la fleur (µL)
6
7 # 2ème axe
8 M >= 8
9 C = 60
10 N = 40 # nbr. de fleurs
11 n = 20 # taille des aretes du domaine des fleurs (m)
12 nect0 = 5 # Amplitude de la loi du nectar (µL)
13 sigma = 5 # Etalement spatial (m)
14
15 # 3ème axe
16 c = 0.00075 # cout de déplacement de la voiture (dh/m)
17 V = 3.3 # capacité de stockage de la voiture (m**3)
```

Coords des fleurs

```
10 # reseau 1
11 F = np.array([(-25.00, -34.41),
12               (-40.45, 13.14), (0.00, 42.53),
13               (40.45, 13.14), (25.00, -34.41)])
14 Nest = np.array((0, -70.0))
15 # reseau 2
16 F = np.array([(0, 80), (20, 80),
17               (40, 80), (60, 80), (10, 60),
18               (30, 60), (50, 60), (20, 40),
19               (40, 40), (30, 20)])
20 Nest = np.array((30, -10))
21 #reseau 3
22 F = np.array([(20, 70), (0, 60),
23               (0, 50), (0, 40), (40, 60),
24               (40, 50), (40, 40), (0, 30),
25               (0, 20), (40, 30), (40, 20), (20, 10)])
26 Nest = np.array((20, -20))
```

Traçage des trajets

```
90 def Trace(seq):
91     XY = [Nest] # nid
92     for k in seq:
93         XY.append(F[k])
94     XY.append(Nest) # retour nid
95     return np.array(XY)
96
97 historic, best_seq = run(8)
98 Data = [(Trace(seq[0]),length(seq)) for seq in historic]
99
100 nb = len(Data)
101 nb_cols = 4
102 nb_ligs = 2 # exactement 2 lignes
103
104 import matplotlib.pyplot as plt
105
106 plt.style.use("seaborn-v0_8-white")
107 fig, axes = plt.subplots(nb_ligs, nb_cols, figsize=(20, 12))
108 axes = axes.flatten()
109
110 for idx, ax in enumerate(axes):
111     if idx < nb:
112         traj, L = Data[idx][0], Data[idx][1]
113         x, y = traj[:,0], traj[:,1]
114         # tracé du trajet
115         ax.plot(x, y, linewidth=2)
116         # fleurs
117         for i in range(N):
118             fx, fy = F[i]
119             ax.scatter(fx, fy, c="red")
120             ax.text(fx+2, fy+2, f"F{i+1}", fontsize=12)
121         # nid
122         ax.scatter(Nest[0], Nest[1], c="black")
123         ax.text(Nest[0]+2, Nest[1]+2, "N", fontsize=12)
124         ax.set_aspect('equal') # IMPORTANT : pas plt.gca()
125         ax.set_xticks([])
126         ax.set_yticks([])
127         ax.set_title(f" Cycle {idx+1}, L = {L:.1f}m", fontsize=14)
128 plt.show()
```

Contrôle de l'évolution des sols

```
140 def quality(seq):
141     return reward(seq) / length(seq)
142
143 historic, best_seq = run(M)
144
145 I = list(range(M))
146
147 Qual_evol= np.array([quality(historic[i]) for i in I])
148 Qual_evol /= max(Qual_evol)
149
150 plt.plot(I,Qual_evol)
151 plt.xlabel("Itérations")
152 plt.ylabel("QL%")
153 plt.show()
```

Coords des instances

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 F = []
5 with open("C:\\Users\\hp\\OneDrive\\Bureau\\Tipe\\berlin52.txt", "r") as f:
6     start = False
7     for line in f:
8         if "NODE_COORD_SECTION" in line:
9             start = True
10            continue
11            if "EOF" in line:
12                break
13            if start:
14                _, x, y = line.split()
15                F.append((float(x), float(y)))
16 f.close()
17
18 x = [p[0] for p in F]
19 y = [p[1] for p in F]
20
21 plt.scatter(x, y)
22 plt.scatter(0,0, c='black')
23 plt.gca().set_aspect('equal')
24
25 plt.title("Points de l'instance Berlin52")
26 plt.xlabel("x")
27 plt.ylabel("y")
28
29 plt.show()
```

Carte de Casa

```
1 import osmnx as ox
2 import networkx as nx
3 import itertools
4 from itertools import combinations
5
6 # Importation de la carte de Casablanca
7 G = ox.graph_from_place("Casablanca, Morocco", network_type="drive")
8 G = ox.project_graph(G)
9 nodes = list(G.nodes)
10
11 # Recherche des coords d'un dépôt sur la carte
12 lon = -7.6362819
13 lat = 33.5907202
14 xs = np.array([G.nodes[n]['x'] for n in nodes])
15 ys = np.array([G.nodes[n]['y'] for n in nodes])
16 # distance euclidienne (dans la projection métrique)
17 dist = (xs - lon)**2 + (ys - lat)**2
18 nest = nodes[np.argmin(dist)]
19
20 # Choix de clients
21 random.seed(42)
22 clients = random.sample(nodes, M)
```

Trajets de Casa

```
125 # Tracçage du trajet
126 def build_full_route(seq):
127     full_route = []
128     # nest -> premier client
129     u = nest
130     v = clients[seq[0]]
131     segment = nx.shortest_path(G,u,v,weight="length")
132     full_route += segment[:-1]
133     # clients -> client
134     for i in range(len(seq)-1):
135         u = clients[seq[i]]
136         v = clients[seq[i+1]]
137         segment = nx.shortest_path(G, u, v, weight="length")
138         full_route += segment[:-1]
139     # dernier client -> nest
140     u = clients[seq[-1]]
141     v = nest
142     segment = nx.shortest_path(G, u, v, weight="length")
143     full_route += segment
144     return full_route
145
146 full_route = build_full_route(seq0)
147
148 fig, ax = ox.plot_graph(G, show=False, close=False, node_size=0, edge_color="lightgray",
149 bgcolor="white")
150
151 ox.plot_graph_route(G, full_route, route_color="red", route_linewidth=3, node_size=0, ax=ax,
152 show=False, close=False)
153
154 # points clients
155 ax.scatter([G.nodes[n]['x'] for n in clients], [G.nodes[n]['y'] for n in clients], s=20,
156 color="blue")
157
158 # nid
159 ax.scatter(G.nodes[nest]['x'], G.nodes[nest]['y'], s=150, marker='*', color="orange")
160
```

```
160 # numérotation des clients
161 for i, n in enumerate(clients):
162     x = G.nodes[n]['x']
163     y = G.nodes[n]['y']
164     ax.text(
165         x + 0.0002, # petit décalage horizontal
166         y + 0.0002, # petit décalage vertical
167         str(i),
168         fontsize=8,
169         color="black"
170     )
171
172 ax.set_title(f"L = {L:.2f} km | g = {g:.2f} MAD")
173 plt.show()
```