



# Algorithme D'essaim Particulaire (PSO)

Présenté par:  
OUKCHIREN Ayoub  
EL ANBARI Imane  
SILMI Badr

Encadré par:  
Prof. LOMRI Amina





# Sommaire

- 01 Introduction**
- 02 Fondements théoriques**
- 03 Mécanisme de l'algorithme**
- 04 Application de l'algorithme**
- 05 Avantages et limitations**
- 06 Conclusion**

Particle Swarm Optimization Visualization

BUILD SCENE

START

NumDims: ☒ 2D ☐ 3D

# Populations:  1

Population 0 size: 100

► population 0

# Histoire

## Développement

James KENNEDY et Russel EBERHART ont été inspiré par le comportement des essaims d'oiseaux pour la création de l'algorithme

**1995**

## Première publication

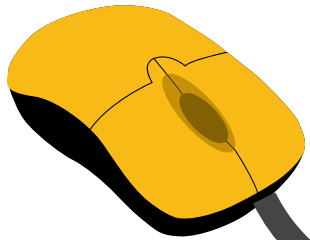
"Particule Swarm Optimisation" dans les actes de la conférences IEEE International Conference on Neural Networks

**1995**

## Améliorations et Adaptations

PSO adaptatif, PSO avec contraintes, PSO hybride avec d'autres méthodes d'optimisation ... etc.

**Au fil des années**



# Présentation générale

**Il est utilisé pour résoudre des problèmes d'optimisation en recherchant la meilleure solution dans un espace de recherche.**

**L'objectif est de guider les particules vers une solution optimale en utilisant un processus itératif de mise à jour des positions et des vitesses.**

**Utilise un ensemble de particules, représentant des solutions candidates, qui se déplacent dans l'espace de recherche en ajustant leurs positions et leurs vitesses.**

**Largement utilisé dans divers domaines, tels que l'optimisation de problèmes complexes, l'optimisation de fonctions mathématiques, la recherche de trajectoires, la planification, les réseaux de neurones, ...etc.**



# Inspiration de l'algorithme

## Essaims D'oiseaux ou Poisson

1. Coopération
2. Interaction sociale
3. Auto-organisation
4. Adaptabilité
5. Exploration et exploitation
6. Émergence de comportements collectifs





# Concept et Fonctionnement



## ● **Particules et position**

- Une particule représente une solution candidate dans l'espace de recherche. Elle est associée à une position et à une vitesse.
- La position d'une particule correspond à une configuration spécifique des paramètres de la solution.
- L'espace de recherche est défini par les limites des valeurs possibles des paramètres.

## ● **Fonctionnement de la mise à jour de la position et la vitesse**

Le PSO utilise un ensemble de particules pour explorer l'espace de recherche en ajustant leurs positions et vitesses. Il s'inspire du comportement des essaims naturels pour trouver des solutions optimales à des problèmes d'optimisation.

# Mécanisme de l'algorithme



## Initialisation

- 01** Les positions et les vitesses des particules sont définies de manière aléatoire.

## Mise à jour de la meilleure position actuelle

- 03** Chaque particule met à jour sa meilleure position personnelle si elle est améliorée.

## Mise à jour des positions et des vitesses

- 05** Les vitesses des particules sont mises à jour en fonction de règles spécifiques et les positions sont recalculées en utilisant les nouvelles vitesses.

## Evaluation de la fonction objectif

- 02** Chaque particule évalue la valeur de la fonction objectif pour sa position actuelle.

## Mise à jour de la meilleure position globale

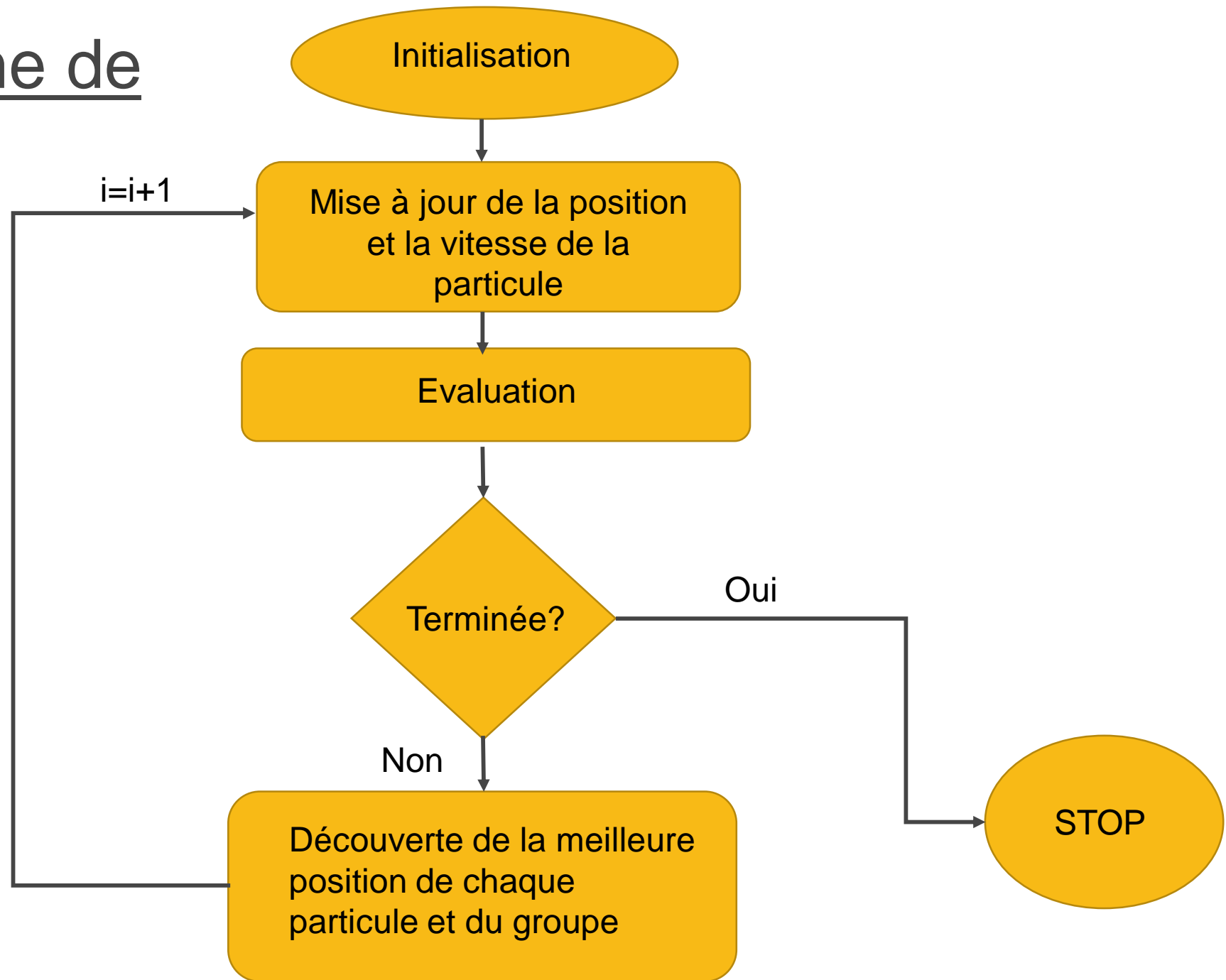
- 04** L'essaim met à jour la meilleure position globale si une particule trouve une meilleure solution.

## Répétition du processus

- 06** Les étapes 3 à 5 sont répétées jusqu'à ce qu'un critère d'arrêt soit atteint.

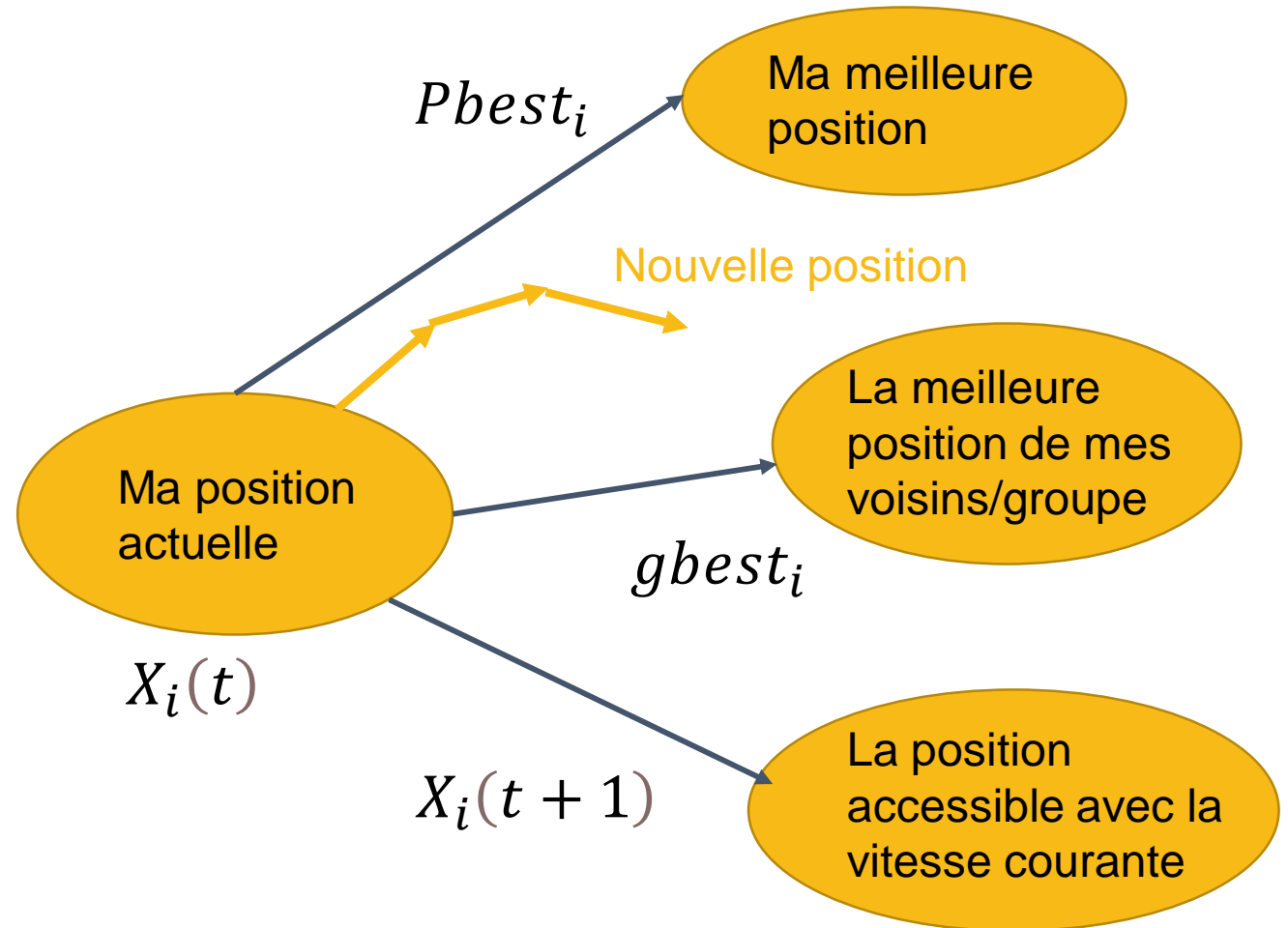


# Organigramme de l'algorithme



# Formalisation

- Un essaim de particule est caractérisé par :
  - a) **le nombre de particules** de l'essaim, noté nb ou S
  - b) **la vitesse maximale** d'une particule, notée  $\vec{V}_{\max}$
  - c) **la topologie** et **la taille** du voisinage d'une particule qui définissent son réseau social
  - d) **l'inertie** d'une particule, notée w
  - e) **les coefficients de confiance**, notés c1 et c2, qui pondèrent le comportement conservateur (càd la tendance à retourner vers la meilleure solution visitée).



# Formalisation

- Une particule est caractérisée, à l'instant  $t$ , par :
  - $\vec{x}_i(t)$  : sa position dans l'espace de recherche
  - $\vec{v}_i(t)$  : sa vitesse
  - $p_{best_i}$  : la position de la meilleure solution par laquelle elle est passée
  - $g_{best_i}$  : la position de la meilleure solution connue de son groupe
  - $\vec{x}_{p_{best_i}}$  : la valeur de fitness de sa meilleure solution
  - $\vec{x}_{g_{best_i}}$  : la valeur de fitness de la meilleure solution connu du groupe

$$V_{i,d}(t+1) = w.V_{i,d} + c_1 r_{1,d} (X_{best_{i,d}}(t) - X_{i,d}(t)) + c_2 r_{2,d} (X_{best_{i,d}}(t) - X_{i,d}(t))$$

$$\vec{X}_i(t+1) = \vec{X}_i(t) + \vec{V}_i(t+1)$$

# Algorithme



## Début

Initialisation des paramètres et la taille  $S$  de l'essaim

Initialisation des vitesses et des positions aléatoires des particules

Pour chaque particule  $P_{best} = X$

## Répéter

**Pour** (  $i$  allant de 1 à  $S$ ) **faire**

**Si** ( $F(X_i) > P_{best_i}$ ) **alors**

$P_{best_i} \leftarrow F(X_i)$

$X_{pbest_i} \leftarrow X_i$

**FinSi**

**Si** ( $F(X_i) > g_{best_i}$ ) **alors**

$g_{best_i} \leftarrow F(X_i)$

$X_{gbest_i} \leftarrow X_i$

**FinSi**

**Fin Pour**

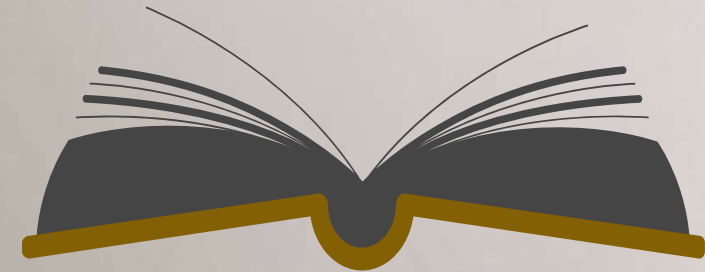
**Pour** (  $i$  allant de 1 à  $S$ ) **alors**

$V_i \leftarrow w * V_i + c1 * r1 * (X_{pbest_i} - X_i) + c2 * r2 * (X_{gbest_i} - X_i)$

$X_i \leftarrow X_i + V_i$

**Fin Pour**

**Jusqu'à** (condition d'arrêt)





# Extensions du PSO

## PSO adaptatif

Les paramètres de l'algorithme PSO sont ajustés dynamiquement pour s'adapter aux changements dans l'environnement du problème.

## PSO avec contraintes

Le PSO tient compte des contraintes du problème en plus de la fonction objectif pour générer des solutions respectant ces contraintes.

## PSO hybride

Le PSO est combiné avec d'autres techniques d'optimisation pour améliorer ses performances en exploitant les avantages complémentaires de ces techniques.

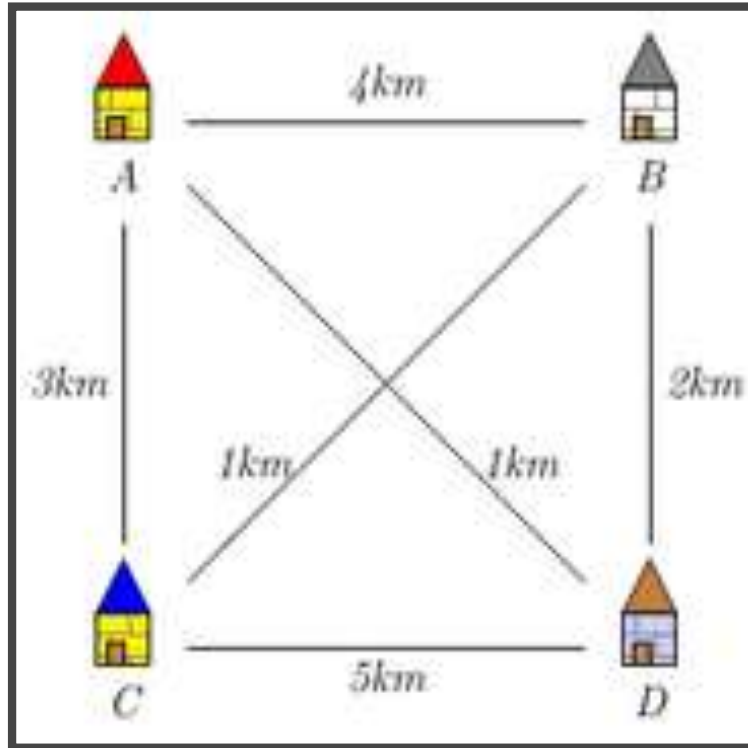
# Application de l'algorithme:

## Problème du voyageur de commerce (TSP)

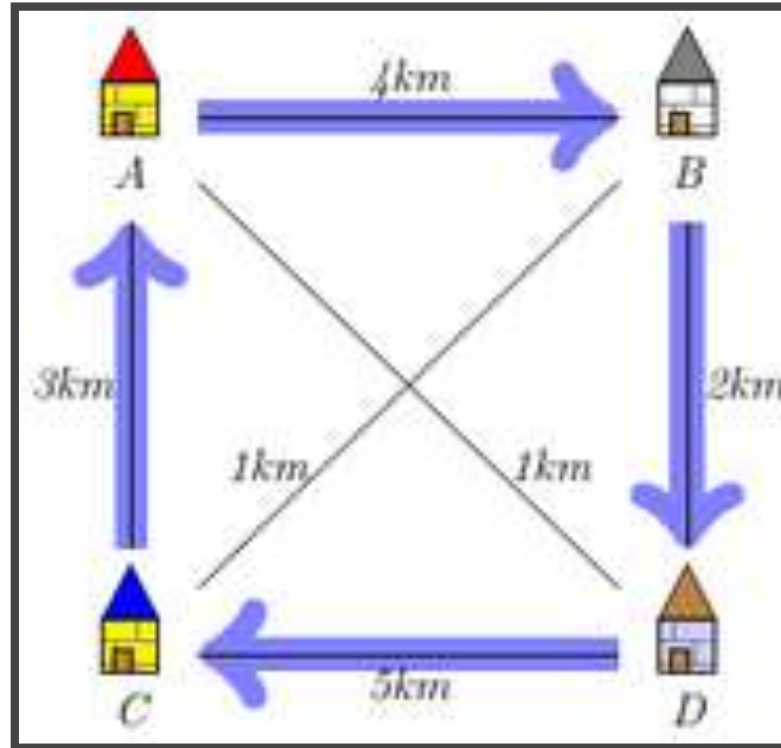
Le problème du voyageur de commerce consiste à trouver le chemin le plus court pour un voyageur qui doit visiter un ensemble de villes données une fois et revenir à la ville de départ. Chaque paire de villes est associée à une distance. L'objectif est de minimiser la distance totale parcourue par le voyageur.

Chaque particule représente un chemin possible pour le voyageur. Chaque particule contient une liste ordonnée des villes à visiter. L'algorithme PSO cherche à ajuster la position de chaque particule (c'est-à-dire l'ordre des villes) pour minimiser la distance totale parcourue.

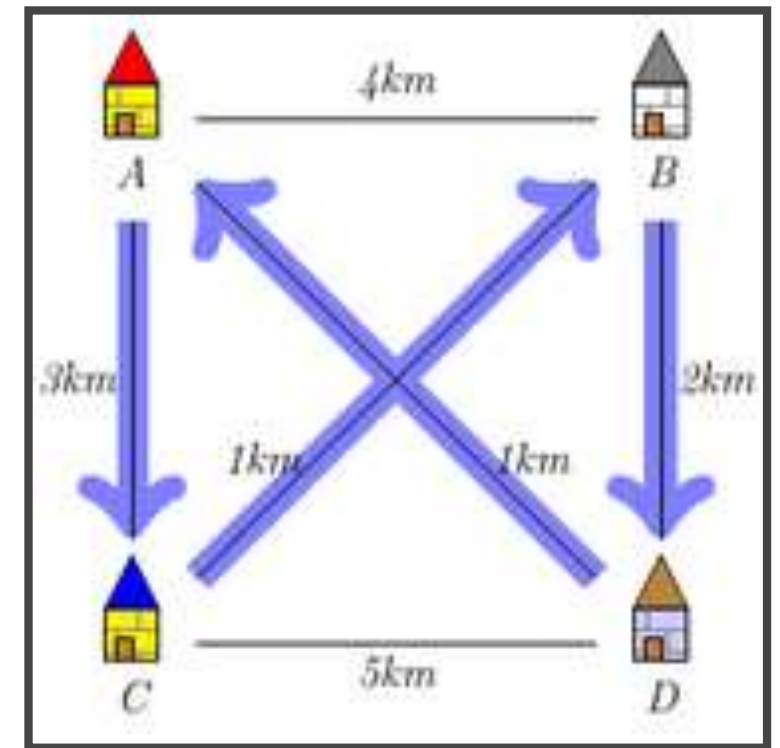
# Aperçu de l'objectif du problème



Ensemble des coordonnées des villes

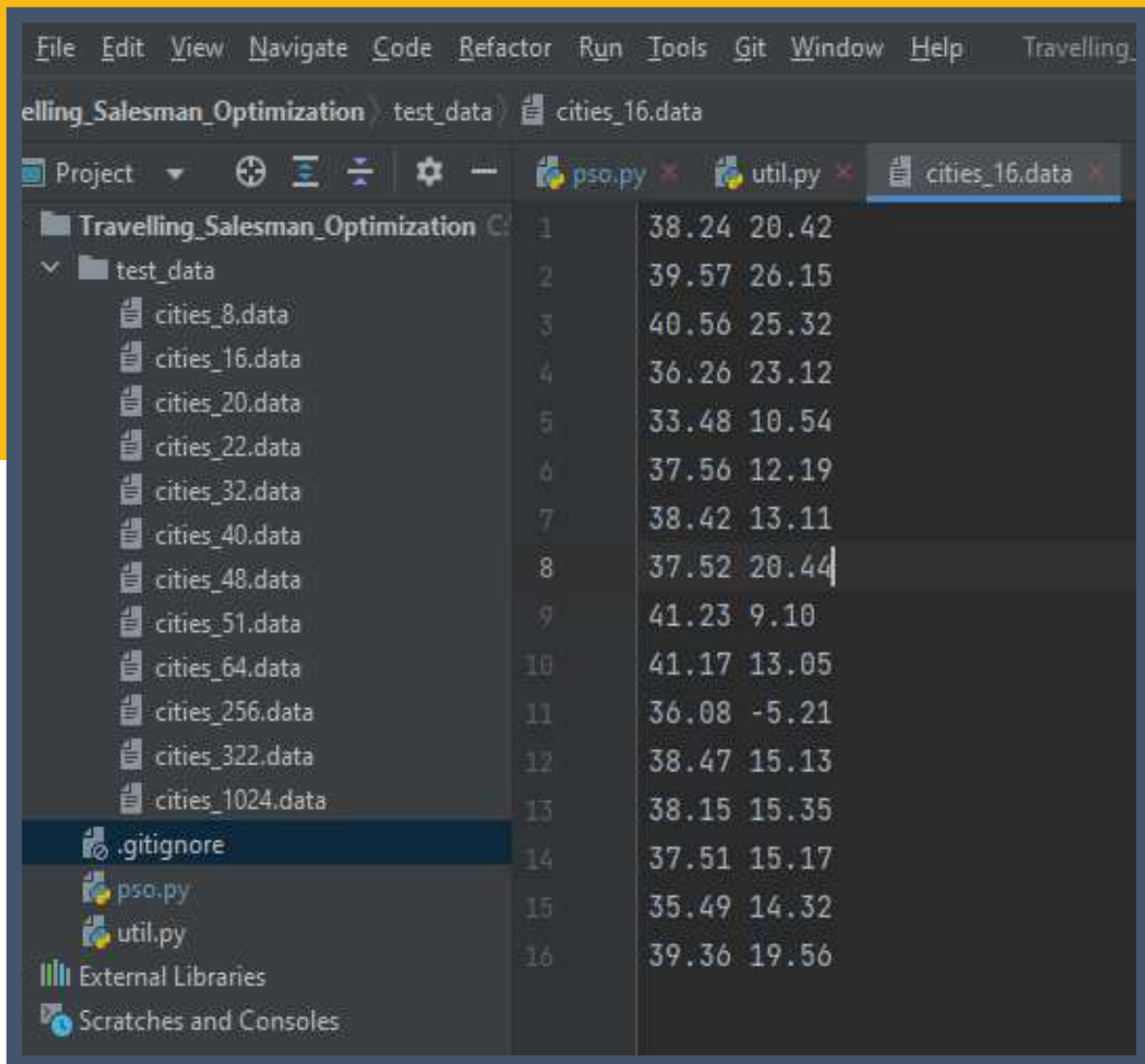


Distance initiale non optimisée



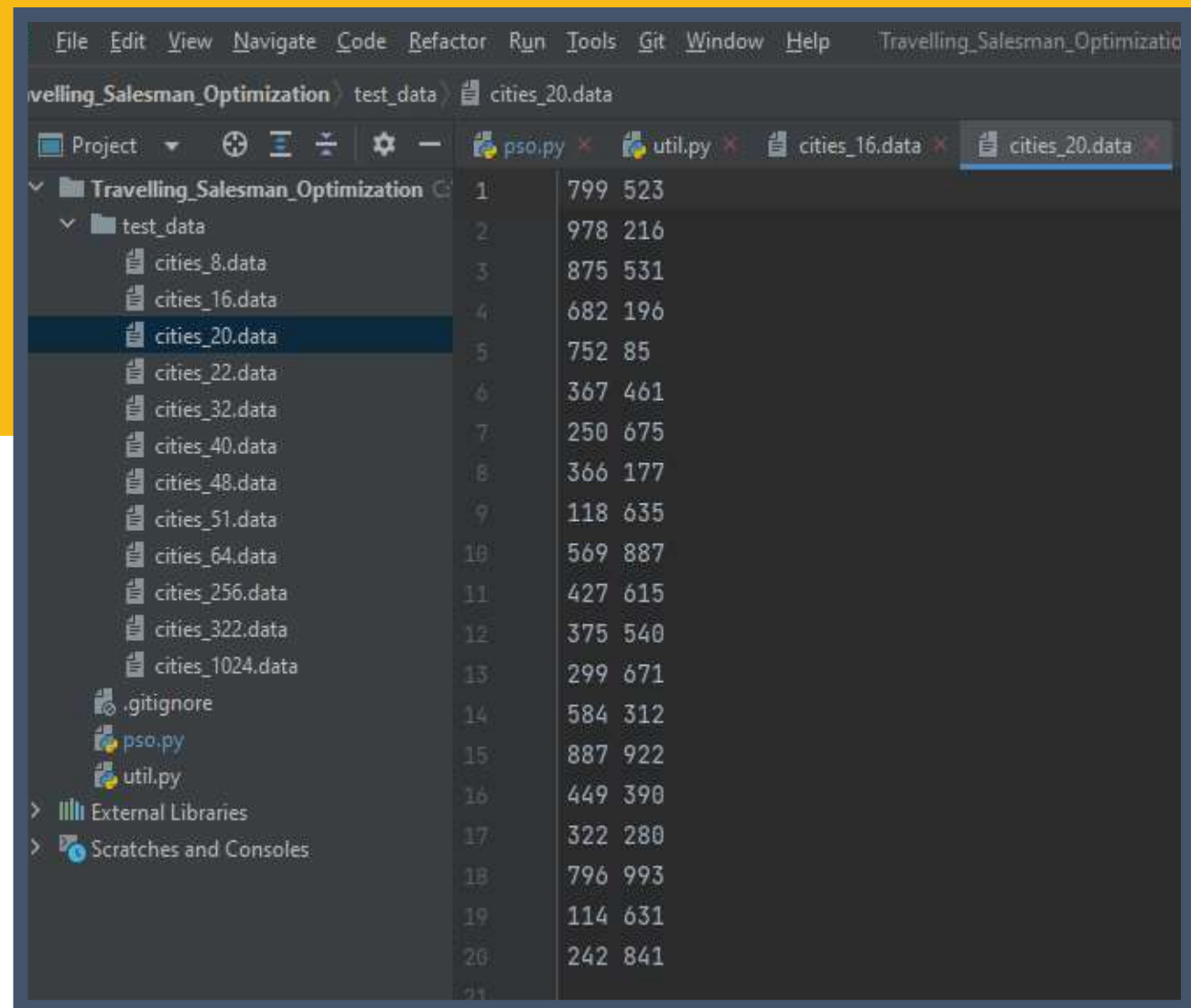
Distance optimale

# Coordonnées des villes:



The screenshot shows an IDE window titled "Travelling\_Salesman\_Optimization". The left sidebar displays the project structure with a tree view. The main editor area shows the content of the "cities\_16.data" file. The file contains a list of 16 cities, each with a number and two floating-point coordinates.

City	Coordinate 1	Coordinate 2
1	38.24	20.42
2	39.57	26.15
3	40.56	25.32
4	36.26	23.12
5	33.48	10.54
6	37.56	12.19
7	38.42	13.11
8	37.52	20.44
9	41.23	9.10
10	41.17	13.05
11	36.08	-5.21
12	38.47	15.13
13	38.15	15.35
14	37.51	15.17
15	35.49	14.32
16	39.36	19.56



The screenshot shows an IDE window titled "Travelling\_Salesman\_Optimization". The left sidebar displays the project structure with a tree view. The main editor area shows the content of the "cities\_20.data" file. The file contains a list of 20 cities, each with a number and two floating-point coordinates.

City	Coordinate 1	Coordinate 2
1	799	523
2	978	216
3	875	531
4	682	196
5	752	85
6	367	461
7	250	675
8	366	177
9	118	635
10	569	887
11	427	615
12	375	540
13	299	671
14	584	312
15	887	922
16	449	390
17	322	280
18	796	993
19	114	631
20	242	841



```
Travelling_Salesman_Optimization \ util.py
Project
  Travelling_Salesman_Optimization
    test_data
    .gitignore
    pso.py
    util.py
  External Libraries
  Scratches and Consoles
Commit
Pull Requests
Structure

1 import math
2 import random
3 import matplotlib.pyplot as plt
4
5
6 class City:
7     def __init__(self, x, y):
8         self.x = x
9         self.y = y
10
11     def distance(self, city):
12         return math.hypot(self.x - city.x, self.y - city.y)
13
14     def __repr__(self):
15         return f"({self.x}, {self.y})"
16
17
18 def read_cities(size):
19     cities = []
20     with open(f'test_data/cities_{size}.data', 'r') as handle:
21         lines = handle.readlines()
22         for line in lines:
23             x, y = map(float, line.split())
24             cities.append(City(x, y))
25     return cities
26
27
28 def write_cities_and_return_them(size):
29     cities = generate_cities(size)
30     with open(f'test_data/cities_{size}.data', 'w+') as handle:
31         for city in cities:
32             handle.write(f'{city.x} {city.y}\n')
33     return cities
34
```

La classe permet de représenter une ville avec ses coordonnées (x, y) et de calculer la distance entre deux villes.

La fonction lit les coordonnées des villes à partir d'un fichier, crée des objets City pour chaque paire de coordonnées, et retourne une liste contenant ces objets City.

Génère des villes aléatoires, les écrit dans un fichier en les séparant par des sauts de ligne, puis renvoie la liste des villes générées.

```

27
28 1 usage  Ramez
29 def write_cities_and_return_them(size):
30     cities = generate_cities(size)
31     with open(f'test_data/cities_{size}.data', 'w+') as handle:
32         for city in cities:
33             handle.write(f'{city.x} {city.y}\n')
34     return cities
35
36 2 usages  Ramez
37 def generate_cities(size):
38     return [City(x=int(random.random() * 1000), y=int(random.random() * 1000)) for _ in range(size)]
39
40 2 usages  Ramez
41 def path_cost(route):
42     return sum([city.distance(route[index - 1]) for index, city in enumerate(route)])
43
44 Ramez
45 def visualize_tsp(title, cities):
46     fig = plt.figure()
47     fig.suptitle(title)
48     x_list, y_list = [], []
49     for city in cities:
50         x_list.append(city.x)
51         y_list.append(city.y)
52     x_list.append(cities[0].x)
53     y_list.append(cities[0].y)
54
55     plt.plot(x_list, y_list, 'ro')
56     plt.plot(x_list, y_list, 'g')
57     plt.show(block=True)

```

Génère une liste de villes aléatoires en utilisant la classe City. Les coordonnées des villes sont générées de manière aléatoire dans la plage de 0 à 1000.

Calcule le coût total d'un itinéraire en sommant les distances entre chaque paire de villes consécutives dans l'itinéraire.

Crée un graphique affichant les villes du problème TSP, avec les points des villes représentés en rouge et une ligne verte reliant les villes dans l'ordre. Le graphique est affiché à l'aide de la bibliothèque Matplotlib.

```
1 import random
2 import math
3 import matplotlib.pyplot as plt
4 from util import City, read_cities, write_cities_and_return_them, generate_cities, path_cost
5
6 class Particle:
7 >     def __init__(self, route, cost=None): ...
13
14 >     def clear_velocity(self): ...
16
17 >     def update_costs_and_pbest(self): ...
22
23 >     def path_cost(self): ...
25
26
27 class PSO:
28
29 >     def __init__(self, iterations, population_size, gbest_probability=1.0, pbest_probability=1.0, cities=None): ...
41
42 >     def random_route(self): ...
44
45 >     def initial_population(self): ...
50
51 >     def greedy_route(self, start_index): ...
60
61 >     def run(self): ...
119
120
121 > if __name__ == "__main__": ...
```

```
Particle 291: cost 119.21143561564695, route [(33.48, 18.54), (37.51, 15.17), (41.23, 9.1), (38.47, 15.13), (36.88, -5.21), (37.56, 12.19), (38.42
Particle 292: cost 140.6992858892789, route [(37.51, 15.17), (36.26, 23.12), (38.42, 13.11), (40.56, 25.32), (38.24, 20.42), (33.48, 18.54), (36.8
Particle 293: cost 126.54464096547159, route [(40.56, 25.32), (36.26, 23.12), (38.47, 15.13), (33.48, 18.54), (38.15, 15.35), (37.56, 12.19), (38.
Particle 294: cost 135.05877982773094, route [(39.36, 19.56), (38.24, 20.42), (39.57, 26.15), (37.56, 12.19), (35.49, 14.32), (36.88, -5.21), (38.
Particle 295: cost 147.6707643805474, route [(39.36, 19.56), (40.56, 25.32), (37.51, 15.17), (35.49, 14.32), (38.47, 15.13), (38.15, 15.35), (37.5
Particle 296: cost 150.70201916087194, route [(36.26, 23.12), (38.47, 15.13), (37.56, 12.19), (35.49, 14.32), (38.42, 13.11), (40.56, 25.32), (41.
Particle 297: cost 145.6263871309294, route [(38.47, 15.13), (35.49, 14.32), (41.17, 13.05), (38.42, 13.11), (37.52, 20.44), (41.23, 9.1), (36.26,
Particle 298: cost 106.48368229965463, route [(36.26, 23.12), (38.15, 15.35), (37.56, 12.19), (37.51, 15.17), (38.42, 13.11), (38.47, 15.13), (35.
Particle 299: cost 138.20734805859072, route [(37.51, 15.17), (36.26, 23.12), (39.36, 19.56), (41.17, 13.05), (37.52, 20.44), (33.48, 18.54), (38.
Particle 300: cost 104.73494586816572, route [(38.24, 20.42), (37.52, 20.44), (39.36, 19.56), (38.15, 15.35), (38.47, 15.13), (37.51, 15.17), (35.
```

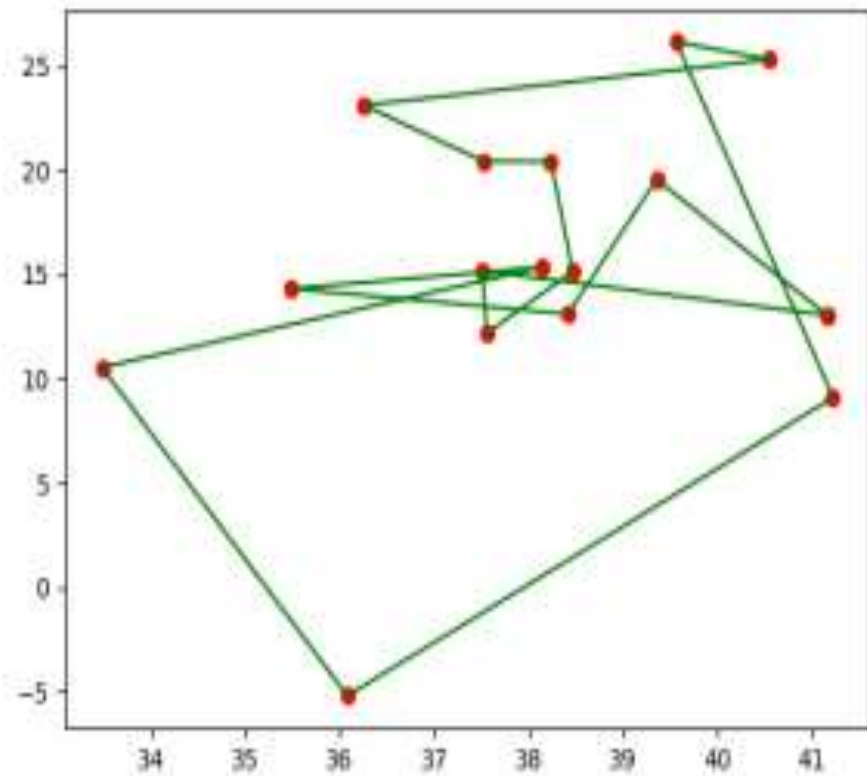
## Résultats

```
initial cost is 99.690326173317797
```

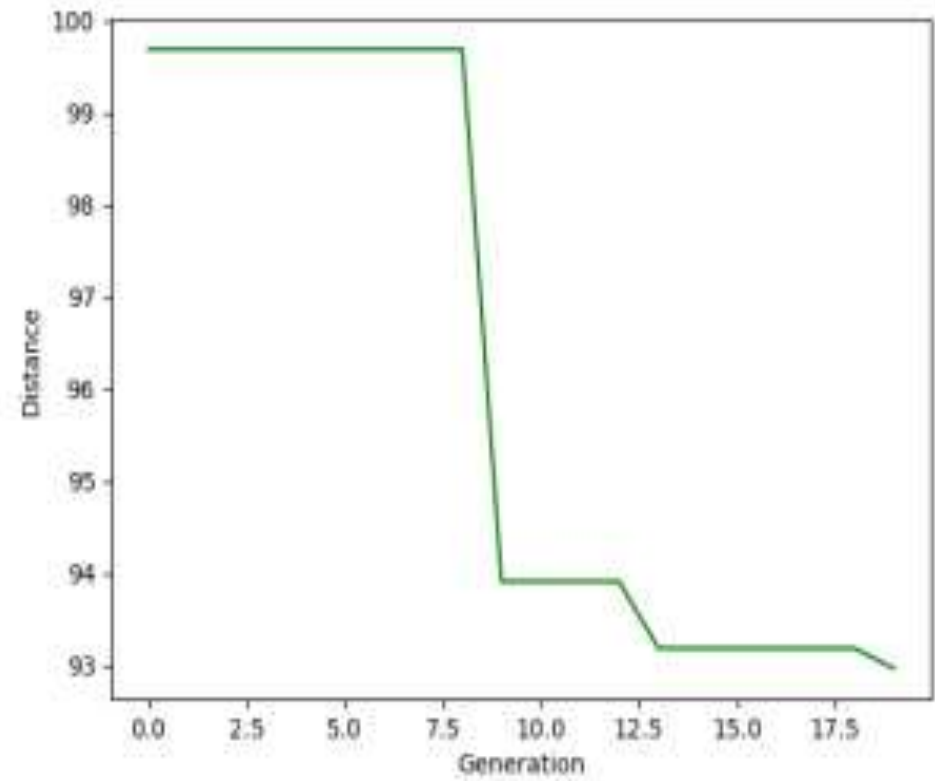
```
cost: 76.307852451975 | guest: [(36.26, 23.12), (37.52, 20.44), (37.51, 15.17), (38.15, 15.35), (38.47, 15.13), (38.42, 13.11), (37.56, 12.19), (35.49, 1
```



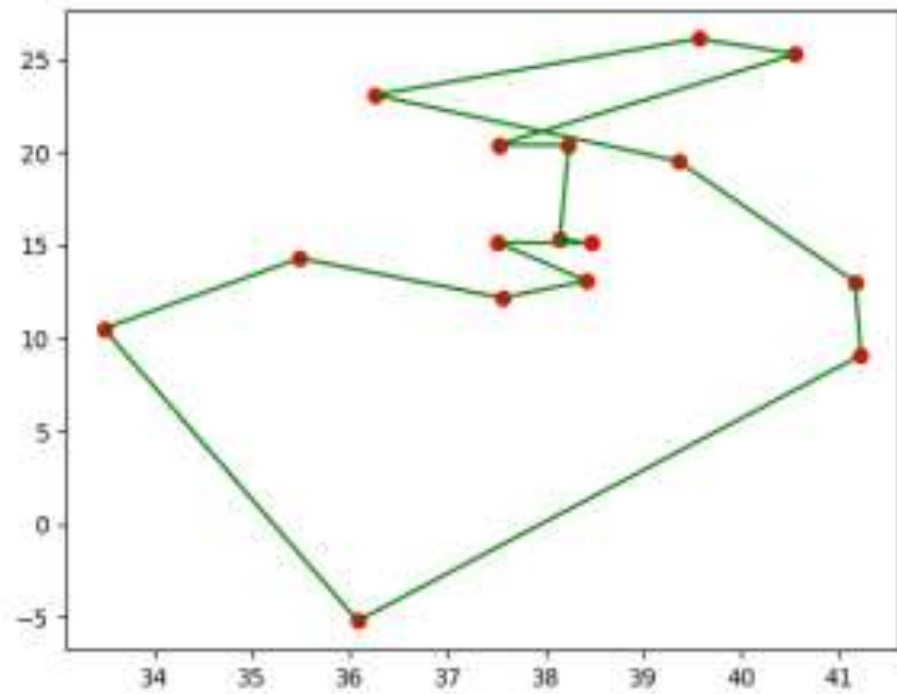
pso TSP iter 0



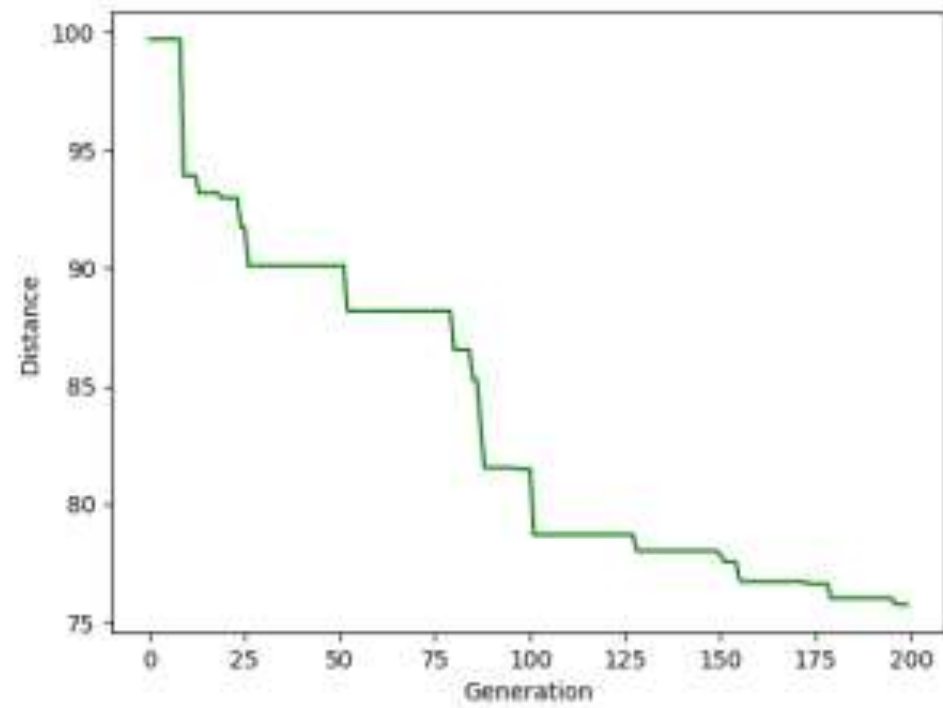
pso iter



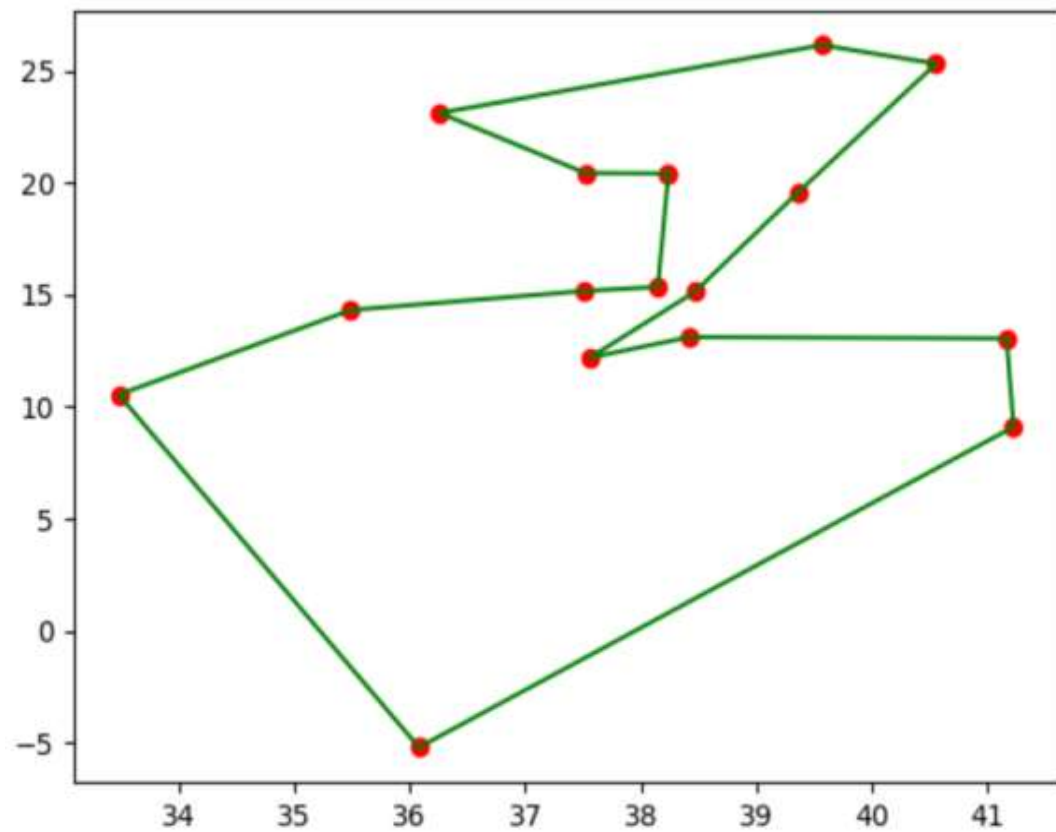
pso TSP iter 180



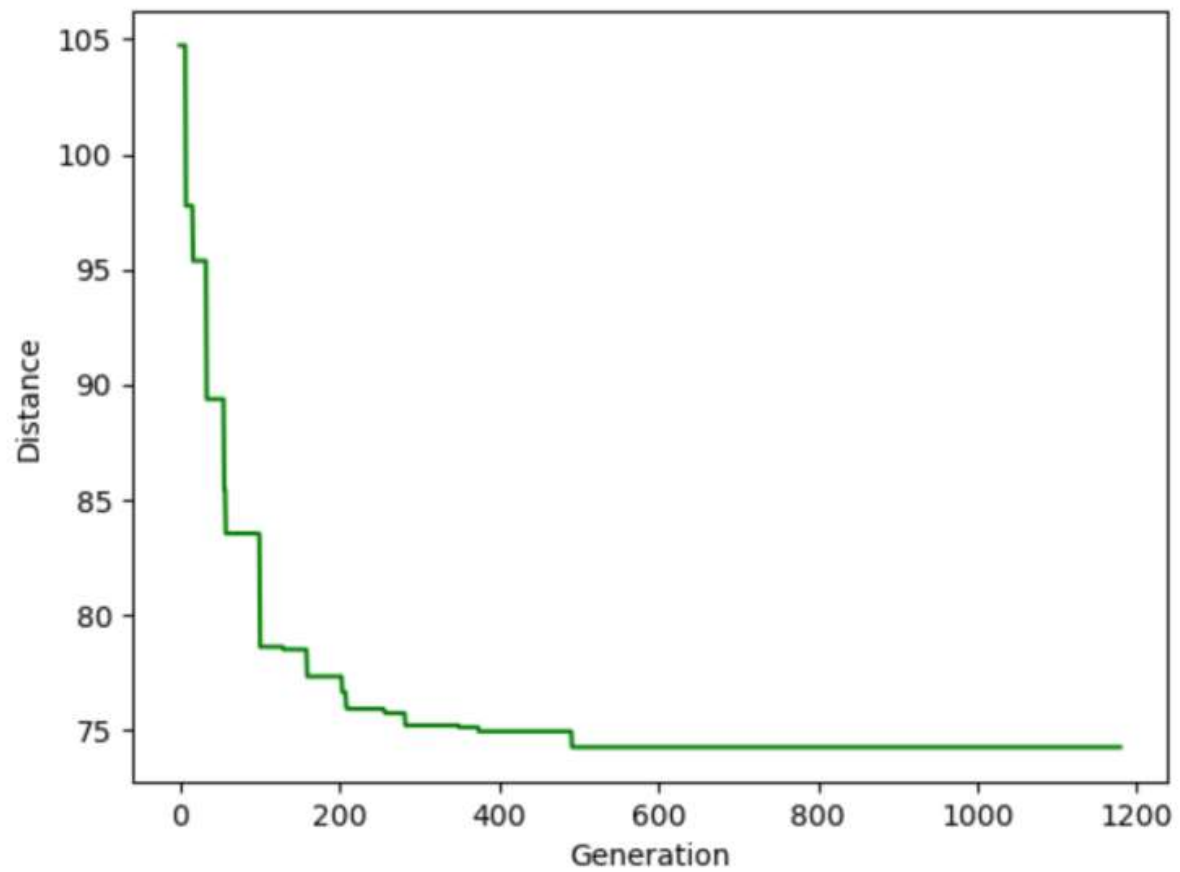
pso iter



pso TSP iter 1160



pso iter



# Avantages et Limitations

## Avantages

**Simplicité** d'utilisation, **capacité** à explorer de grandes régions de recherche, **adaptabilité** à différents problèmes, pas de dépendance sur les gradients.

## Limitations

Sensibilité aux paramètres, convergence vers des optima locaux, performances variables selon les problèmes, nécessité de connaissances préliminaires.



Il est important de noter que le PSO peut être amélioré et ajusté pour atténuer certaines limitations et améliorer ses performances dans des contextes spécifiques.



A blurred background image of a desk with a laptop and a mouse. The laptop is silver with an Apple logo, and the mouse is white. The desk is dark wood. The background is a bookshelf filled with books.

CONCLUSION





**Merci  
pour votre  
attention!**