Optimisation algorithmique de trajet de drone de livraison

**Plan :**

* Résumé + Abstract
* Mots clés
* Introduction
* État de l’art
  + Drones de livraisons
  + Algorithme d’optimisation
* Description du cadre de l’étude
* Contraintes et paramètres de l’étude
* Expérimentation
* Glossaire
* Références

Ressource pour le plan et les parties -> <https://journals.openedition.org/communiquer/2294>

**Ressources scientifiques :**

|  |
| --- |
| <https://www.researchgate.net/profile/Mohamed-Chakchouk/publication/358691791_Optimisation_de_plan_de_vol_d'un_drone_faisant_un_cycle_hamiltonien_entre_N_points_de_mesures_Probleme_du_voyageur_de_commerce_et_duree_de_vie_de_batterie/links/623c4f398f081a732fba5256/Optimisation-de-plan-de-vol-dun-drone-faisant-un-cycle-hamiltonien-entre-N-points-de-mesures-Probleme-du-voyageur-de-commerce-et-duree-de-vie-de-batterie.pdf> |
| <https://hal.inria.fr/hal-02160346/document> |
| <https://www.ifaamas.org/Proceedings/aamas2021/pdfs/p404.pdf> |
| <https://sci-hub.se/downloads/2019-11-14/04/10.1007@s10732-019-09431-y.pdf?download=true> |
| <https://zero.sci-hub.se/7010/8b4141ac94bc4fbb749d45769a0421b0/baker2003.pdf?download=true> |
| <http://www.macs.hw.ac.uk/~dwcorne/Teaching/bullnheimer-vr.pdf> |
| <https://moscow.sci-hub.se/2351/27772242e6b01ce3a40962728c522919/yanik2014.pdf?download=true> |
|  |

**Ressources VRP :**

|  |
| --- |
| <https://www.researchgate.net/profile/Hani-Guenoune/publication/337840545_Resolution_du_probleme_de_tournees_de_vehicules_avec_collecte_et_livraison_simultanees_avec_une_approche_cooperative_de_metaheuristiques/links/5dee6da4299bf10bc34e9bd1/Resolution-du-probleme-de-tournees-de-vehicules-avec-collecte-et-livraison-simultanees-avec-une-approche-cooperative-de-metaheuristiques.pdf?origin=publication_detail> |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

Prompt GPT :

Jouons a un jeu, tu es un expert en algorithmique et en résolution de problème NP-complet. Voici un contexte :

On dispose d'un nombre défini de drones (ici 2), d'un seul entrepôt et de 8 clients. Les clients ne sont pas reliés entre eux, chaque drone ne peut transporter qu'un seul colis à la fois (donc doit faire l'aller retour entrepôt-client à chaque fois), les distances entre l'entrepôt et les clients sont définies dans la matrice d'adjacence suivante :

matrice\_distance = np.array([

[0, 5.4, 3.2, 10.1, 7.6, 2.1, 14.4, 9.8, 4.6],

[5.4, 0, 0, 0, 0, 0, 0, 0, 0],

[3.2, 0, 0, 0, 0, 0, 0, 0, 0],

[10.1, 0, 0, 0, 0, 0, 0, 0, 0],

[7.6, 0, 0, 0, 0, 0, 0, 0, 0],

[2.1, 0, 0, 0, 0, 0, 0, 0, 0],

[14.4, 0, 0, 0, 0, 0, 0, 0, 0],

[9.8, 0, 0, 0, 0, 0, 0, 0, 0],

[4.6, 0, 0, 0, 0, 0, 0, 0, 0]

]).

Dans cette matrice, l'entrepôt porter l'indice 0 et les clients les indices entre 1 et 9. L'algorithme dispose d'un nombre d'itérations illimité mais doit s'arrêter quand le temps total d'exécution dépasse une minute (60 secondes). On a une fonction de génération d'une solution aléatoire appelée generate\_random\_solution et une fonction de calcul du fitness pour minimiser la différence de distance entre les drones (sur un total de 100km avec 2 drones, chaque drone doit avoir une distance totale parcourue la plus proche des 50km possible). Voici les deux fonctions et leurs définitions :

def generate\_random\_solution(distance\_matrix, num\_vehicles):

# Compute the total distance

total\_distance = distance\_matrix.sum()

# Compute the target distance per vehicle

target\_distance\_per\_vehicle = total\_distance / num\_vehicles

# Initialize the groups

groups = [[] for i in range(num\_vehicles)]

group\_distances = [0 for i in range(num\_vehicles)]

# Assign the customers to the groups

for i in range(1, distance\_matrix.shape[0]):

# Compute the customer distances to each group

distances = [abs(distance\_matrix[i,j]) for j in range(num\_vehicles)]

# Compute the weights for each group based on the distance already covered

weights = [1.0 - (group\_distances[j] / target\_distance\_per\_vehicle) for j in range(num\_vehicles)]

# Normalize the weights so they sum to 1

total\_weight = sum(weights)

weights = [w / total\_weight for w in weights]

# Choose the group with the highest weight

group\_idx = np.random.choice(range(num\_vehicles), p=weights)

# Add the customer to the group

groups[group\_idx].append(i)

group\_distances[group\_idx] += distances[group\_idx]

# Add 0 at the end, the beginning and between each group

for i in range(len(groups)):

for j in range(len(groups[i])-1):

groups[i].insert(2\*j+1,0)

j += 1

groups[i].insert(0,0)

groups[i].append(0)

return groups

def calculate\_fitness\_vrppd(distance\_matrix, solution, num\_vehicles):

# Calculate the total distance for each vehicle

total\_distances = []

for i in range(num\_vehicles):

total\_distance = 0

for j in range(len(solution[i])):

if j == 0:

# Add distance from depot to first node

total\_distance += distance\_matrix[0][solution[i][j]]

else:

# Add distance between consecutive nodes

total\_distance += distance\_matrix[solution[i][j-1]][solution[i][j]]

# Add distance from last node to depot

total\_distance += distance\_matrix[solution[i][-1]][0]

total\_distances.append(total\_distance)

# Calculate the variance of the total distances

variance = np.var(total\_distances)

# Calculate the mean of the total distances

mean\_distance = sum(total\_distances) / num\_vehicles

# Calculate the fitness score

target\_distance\_per\_vehicle = sum(total\_distances) / num\_vehicles

fitness\_score = 1 / (variance + 1) \* mean\_distance / target\_distance\_per\_vehicle

return total\_distances, fitness\_score

Créé moi un algorithme Particle Swarm Optimization (PSO) pour résoudre se problème. Il doit fonctionner du premier coup sans erreur !