

Optimisation de tournées de drones à l'aide d'un GNN-PPO

Projet Drone Delivery Optimizer

Mahouna ____ (RTX 4050 / Intel i7)

May 16, 2025

Contents

1	Introduction	2
2	Modèle de réseau de livraison	2
2.1	Graphe orienté des k -plus-proches-voisins	2
2.2	Attributs nodaux X	2
3	Fonction de coût généralisée	3
3.1	Effet du vent sur les coûts d'arêtes	3
3.2	Consommation de batterie par arête	3
3.3	Objectif global d'une tournée E	3
4	Contraintes de début et fin de tournée	3
4.1	1. Algorithme génétique (GA)	3
4.2	2. PPO + GNN (RL)	4
4.3	Encodage de l'état s_t	4
4.4	Architecture du GNN	5
4.5	Readout et têtes acteur-critique	5
4.6	Récompense instantanée	5
5	Algorithme PPO	5
5.1	Avantage (GAE- λ)	5
5.2	Perte « clipped »	5
5.3	Pipeline d'entraînement	5
6	Conclusion	6

1 Introduction

Ce rapport présente la modélisation mathématique et l'implémentation d'une méthode hybride pour l'optimisation de tournées de drones :

1. un *Algorithme Génétique* (GA) classique corrigé pour garantir les points de départ et d'arrivée,
2. une approche **PPO** (Proximal Policy Optimization) alimentée par un **GNN** pour l'apprentissage par renforcement.

2 Modèle de réseau de livraison

2.1 Graphe orienté des k -plus-proches-voisins

Chaque nœud u est connecté vers ses $k = 10$ plus proches voisins *sortants*, formant un graphe orienté

$$G_t = (\mathcal{V}_t, \mathcal{E}_t), \quad \mathcal{E}_t \subset \{u \rightarrow v\}.$$

Chaque arête orientée $e = (u \rightarrow v)$ porte un coût $c_{uv} = d_{uv} \cdot \left(\underbrace{1}_{\text{base}} \underbrace{-\alpha \cdot w_{uv}}_{\text{effet du vent}} \underbrace{+\beta \cdot \eta_{uv}}_{\text{bruit aléatoire}} \right) \in [0, 600]$, où en général $c_{uv} \neq c_{vu}$.

Implémentation du graphe dans le code

Le graphe orienté des k -plus-proches-voisins est construit dans le code JavaScript (**app.js**) selon les étapes suivantes :

1. **Collecte des nœuds** : tous les points (hubs, charging, delivery, pickup) sont stockés dans un tableau **allNodes** avec leurs coordonnées et leur type.
2. **Recherche des voisins** : pour chaque nœud u , on calcule la distance de Haversine à tous les autres nœuds, puis on trie ces distances pour sélectionner les $k = 10$ plus proches voisins (hors lui-même).
3. **Création des arêtes orientées** : pour chaque voisin v sélectionné, une arête orientée ($u \rightarrow v$) est ajoutée à la liste des arêtes.
4. **Calcul du coût d'arête** : chaque arête reçoit un coût c_{uv} qui dépend de la distance, de l'effet du vent (calculé via l'angle entre l'arête et la direction du vent global), et d'un bruit aléatoire. Ceci est réalisé dans la fonction **annotateEdges**.
5. **Affichage** : le graphe est affiché sur la carte avec des couleurs et épaisseurs d'arêtes reflétant le coût, et chaque arête possède un tooltip interactif.

Ce procédé garantit que le graphe utilisé pour l'optimisation correspond exactement à la structure mathématique décrite précédemment.

2.2 Attributs nodaux X

Pour chaque sommet v , on définit

$$x_v = \left[\underbrace{\text{one-hot}(\text{type})}_{\in \{\text{hub}, \text{pickup}, \text{delivery}, \text{charging}\} (4)}, \underbrace{\lambda_v, \phi_v}_{\text{latitude/longitude} (2)}, \underbrace{\text{stock/demande}}_{\in \mathbb{R} (1)} \right] \in \mathbb{R}^7.$$

Le terme « stock/demande » reste nul ou constant ici car on part d'une hypothèse de demande unitaire illimitée.

3 Fonction de coût généralisée

3.1 Effet du vent sur les coûts d'arêtes

Le vent est considéré comme un paramètre global (même direction partout), et son effet est calculé au niveau de chaque arête orientée ($u \rightarrow v$) via:

$$\text{windEffect}_{uv} = \cos(\phi_{uv} - \text{windAngle})$$

où ϕ_{uv} est l'angle de la direction $u \rightarrow v$ et windAngle est la direction globale du vent. Cette valeur est ensuite utilisée dans le calcul du coût de l'arête présenté précédemment.

3.2 Consommation de batterie par arête

Pour une arête e_i de coût c_i , la batterie consommée est

$$\Delta b_i = \frac{c_i}{k} (1 + \alpha (p_i - 1)), \quad \underbrace{k = 10.8}_{\text{normalisation}}, \quad \underbrace{\alpha = 0.2}_{\substack{\text{facteur de surcharge} \\ \text{(colis multiple)}}}, \quad \underbrace{p_i}_{\text{nombre de colis embarqués}}.$$

Plus p_i est grand, plus la consommation augmente.

3.3 Objectif global d'une tournée E

Pour une séquence d'arêtes $E = (e_1, \dots, e_T)$, on définit

$$J(E) = \sum_{e_i \in E} c_i + \lambda \sum_S \left[\max(0, \sum_{e_i \in S} \Delta b_i - B_{\max}) \right]^2 + \mu \# \{\text{recharges}\},$$

où

- $B_{\max} = 100$ est la capacité maximale de batterie,
- $\lambda \gg 1$ pénalise fortement tout dépassement de batterie,
- $\mu \ll \lambda$ pénalise légèrement chaque recharge,
- S parcourt chaque segment consécutif entre deux recharges.

4 Contraintes de début et fin de tournée

4.1 1. Algorithme génétique (GA)

But Générer des chromosomes (tours) garantissant la séquence :

$$H_{\text{start}} \rightarrow D \rightarrow L \rightarrow H_{\text{end}},$$

avec H , hubs et D, L points pickup/delivery.

Initialisation Pour chaque individu :

1. Choix aléatoire d'un hub H_{start} .
2. Construction de $\text{shortest_path}(H_{\text{start}} \rightarrow D)$.
3. Ajout de $\text{shortest_path}(D \rightarrow L)$.
4. Ajout de $\text{shortest_path}(L \rightarrow H_{\text{end}})$ avec H_{end} choisi aléatoirement.

Le chromosome est la concaténation, en omettant les doublons consécutifs.

Réparation (repair) Après crossover/mutation :

- On repère les indices de D et L .
- On sectionne pour forcer $\dots \rightarrow D \rightarrow L \rightarrow \dots$
- Si la fin n'est pas un hub, on y greffe `shortest_path(dernier $\rightarrow H_{\text{rand}}$)`.

Mutation spécifique Échanger deux sous-chemins internes tout en maintenant la séquence $H \rightarrow D \rightarrow L \rightarrow H$.

4.2 2. PPO + GNN (RL)

Intégration des contraintes (mise à jour)

- *Pas 0 (téléportation)* : comme avant,

$$a_0 = (\text{téléportation vers hub } H_i), \quad H_i \sim \{H \mid d(\text{pickup}, H) \leq d_{\max}\}.$$

- *Étapes intermédiaires* ($1 \leq t \leq T$) : on suit une arête ($v_t \rightarrow v_{t+1}$).
- *Termination* : l'épisode se termine dès que

$$\text{picked} = \text{delivered} = 1 \quad \text{et} \quad v_t \in \mathcal{H} \quad \text{ou} \quad t = T_{\max}.$$

L'agent doit naturellement rejoindre un hub après avoir desservi tous les points. Cela rend la terminaison plus réaliste et évite d'introduire une action artificielle de retour. Un critère temporel optionnel (T_{\max}) permet aussi de forcer la fin de l'épisode si besoin.

4.3 Encodage de l'état s_t

L'état est un tuple

$$s_t = (A_t, X_t, E_t, b_t, p_t, v_t),$$

- $A_t \in \{0, 1\}^{N \times N}$: matrice d'adjacence orientée du graphe,
- $X_t \in \mathbb{R}^{N \times d_0}$: matrice des *features nodales*,

$$x_v = [\text{one-hot}(\text{type}), \lambda_v, \phi_v, \text{stock/demande}]$$

où :

- `one-hot(type)` : vecteur indicateur (hub, pickup, delivery, charging), dimension 4,
- λ_v : latitude,
- ϕ_v : longitude,
- `stock/demande` : valeur de stock ou demande (ici constante).
- $E_t \in \mathbb{R}^{|\mathcal{E}| \times f_e}$: matrice des *features d'arête*, pour chaque $e = (u \rightarrow v)$ on encode
$$e_{uv} = [d_{uv}, \cos(\phi_{uv} - \omega), \eta_{uv}],$$
- $b_t \in \mathbb{R}$: batterie restante,
- $p_t \in \mathbb{N}$: nombre de colis à bord,
- $v_t \in \{1, \dots, N\}$: index du nœud courant.

4.4 Architecture du GNN

On utilise un schéma *Message Passing* enrichi par les features d'arête (Edge-conditioned Graph-SAGE) :

$$H^{(\ell+1)} = \sigma \left(W_1^{(\ell)} H^{(\ell)} + W_2^{(\ell)} \sum_{u \rightarrow v} \phi_\theta(e_{uv}) H_u^{(\ell)} \right), \quad H^{(0)} = X_t,$$

avec ϕ_θ un réseau MLP qui transforme la feature d'arête e_{uv} en un poids d'agrégation.

4.5 Readout et têtes acteur-critique

Après L couches GNN, on agrège par

$$g_t = \frac{1}{N} \sum_{v=1}^N H_v^{(L)} \in \mathbb{R}^d.$$

Politique (acteur)

$$\pi_\theta(a_t \mid s_t) = \text{Softmax} \left(W_\pi g_t + U_\pi [b_t, p_t] \right).$$

Critique

$$V_\psi(s_t) = w_v^\top g_t + u_v^\top [b_t, p_t].$$

4.6 Récompense instantanée

Pour tout pas t ,

$$r_t = -c_t - \lambda \left[\max(0, b_t - \Delta b_t) \right]^2 - \mu \mathbb{1}_{\{\text{recharge}\}} - \kappa \mathbb{1}_{\{\text{téléport. hors hub}\}},$$

avec $\kappa \gg 1$ pour interdire toute téléportation hors hub.

5 Algorithmes PPO

5.1 Avantage (GAE- λ)

$$\hat{A}_t = \sum_{k \geq 0} (\gamma \lambda)^k \left(r_{t+k} + \gamma V_\psi(s_{t+k+1}) - V_\psi(s_{t+k}) \right).$$

5.2 Perte « clipped »

$$\mathcal{L} = \mathbb{E}_t \left[\min(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 \pm \varepsilon) \hat{A}_t) \right] + c_1 \|R_t - V_\psi\|^2 - c_2 \mathcal{H}[\pi_\theta].$$

5.3 Pipeline d'entraînement

1. **Collecte de trajectoires** sur N environnements parallèles.
2. **Calcul des avantages** $\{\hat{A}_t\}$ via GAE- λ .
3. **Optimisation conjointe** des paramètres θ, ψ sur plusieurs epochs :

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}, \quad \psi \leftarrow \psi - \eta \nabla_\psi \|R_t - V_\psi\|^2.$$

4. Boucler jusqu'à convergence.

Hyperparamètres typiques

- $\gamma = 0.99$, $\lambda_{\text{GAE}} = 0.95$, $\varepsilon = 0.2$.
- GNN : $L = 2$ couches, $d = 128$, ReLU, dropout 0.1.
- Optimiseur : Adam, LR 3×10^{-4} , FP16 sur GPU.

6 Conclusion

L'intégration explicite des contraintes de téléportation via un hub garantit la validité des tournées tant dans GA (par génération/réparation) que dans PPO+GNN (par encapsulation dans l'environnement). Le GNN-PPO exploite les coûts orientés c_{uv} et converge plus rapidement qu'un GA classique tout en respectant les contraintes de batterie grâce aux pénalités λ et μ .