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

Projet Drone Delivery Optimizer

Mahouna ___ (RTX 4050 / Intel i7)

June 2, 2025

Contents

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.

Annexe A
Comprendre l'agent GNN–PPO « en une page »

1. Le décor :

- Tous les points (hubs, stations de recharge, pickup, delivery) sont placés sur la carte ;
- Chaque point est relié à ses $k = 10$ voisins les plus proches ; chaque flèche possède un coût (distance + vent + bruit) ;
- Ce réseau orienté est le terrain de jeu : le drone ne peut avancer qu'en suivant les flèches.

2. La mission d'un épisode :

1. Se téléporter dans *un* hub de départ ;
2. Rejoindre le pickup et charger ;
3. Livrer au point delivery ;
4. Finir dans *n'importe quel* hub ;
5. Respecter batterie, capacité colis et recharges possibles.

3. Prise de décision :

a) Le drone observe le graphe complet, sa position, sa batterie ;

b) Un GNN condense cela en un vecteur-résumé ;

c) Un réseau « acteur » en déduit les probabilités de chaque flèche sortante ;

d) Un réseau « critique » estime la valeur de la situation.

4. Apprentissage (PPO) : Des dizaines de drones virtuels explorent en parallèle ; leurs trajectoires alimentent PPO qui met à jour les poids GNN + acteur/critique toutes les quelques itérations, en favorisant les tournées les moins coûteuses et conformes aux contraintes.

5. Pourquoi ça marche : Le GNN capture la structure du réseau (vent, distances, hubs), tandis que le RL gère la séquence pickup → delivery → hub sous contraintes. Ensemble, ils découvrent des itinéraires faisables et économes, là où un simple plus-court-chemin échouerait.

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

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

Chaque arête orientée $e = (u \to v)$ porte un coût $c_{uv} = d_{uv} \cdot \Big(\underbrace{1}_{\text{base}} \underbrace{-\alpha \cdot w_{uv}}_{\text{effet du vent}} \underbrace{+\beta \cdot \eta_{uv}}_{\text{bruit aléatoire}} \Big) \in [0, 600]$, 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 $(u \to v)$ 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

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

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 $(u \to v)$ via:

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

où ϕ_{uv} est l'angle de la direction $u \to v$ 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

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

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 :

$$H_{\text{start}} \;\to\; D \;\to\; L \;\to\; H_{\text{end}},$$

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

Initialisation Pour chaque individu :

1. Choix aléatoire d'un hub H_{start}.
2. Construction de shortest_path($H_{\text{start}} \to D$).
3. Ajout de shortest_path($D \to L$).
4. Ajout de shortest_path($L \to H_{\text{end}}$) 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 $\dots \to D \to L \to \dots$.
- Si la fin n'est pas un hub, on y greffe shortest_path(dernier $\to H_{\text{rand}}$).

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

4.2 2. PPO + GNN (RL)

4.3 Modélisation RL détaillée

Cadre MDP. Le problème d'optimisation est formulé comme un *processus de décision de Markov* $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$:

Ensemble	Définition précise
États s_t	$(A,\ X,\ E,\ b_t,\ p_t,\ v_t,\ f_{\text{pick}},\ f_{\text{deliv}})$ avec : • $A \in \{0,1\}^{N\times N}$: matrice d'adjacence orientée ; • $X \in \mathbb{R}^{N\times 7}$: features nodales (one-hot(type), lat, lon, stock); • $E \in \mathbb{R}^{\lvert\mathcal{E}\rvert\times 3}$: features d'arête (distance, cos(angle_vent), bruit); • $b_t \in [0, B_{\max}]$: batterie restante ; • $p_t \in \{0, \ldots, p_{\max}\}$: charge à bord ; • v_t : index du nœud courant ; • $f_{\text{pick}}, f_{\text{deliv}} \in \{0,1\}$: flags « pickup fait / delivery fait ».
Actions a_t	**Pas 0** : choisir un hub $h_i \in \mathcal{H}$ (téléportation). **Pas** $t \geq 1$: choisir l'une des arêtes sortantes $\mathcal{N}^{\text{out}}(v_t)$ du nœud courant. Optionnel : action `RECHARGE` disponible uniquement si v_t est une station.
Transition	1. Si $t = 0$: $v_{t+1} \leftarrow h_i$ (hub choisi). 2. Sinon : v_{t+1} = voisin choisi. 3. Mise à jour batterie : $b_{t+1} = b_t - \Delta b_t$ (cf. § 4.3). 4. Recharge automatique si $v_{t+1} \in \mathcal{C}$ (*charging*). 5. Flags : pickup $\rightarrow f_{\text{pick}}{:=}1$ | delivery $\rightarrow f_{\text{deliv}}{:=}1,\ p_{t+1}{:=}0$.
Récompense r_t	$-\,c_{v_t v_{t+1}} - \lambda\,[\max(0,\ b_{t+1}{<}0)]^2 - \mu\,\mathbb{1}_{\{\text{recharge}\}}$
Condition d'arrêt	$(f_{\text{pick}}{=}1\ \wedge\ f_{\text{deliv}}{=}1\ \wedge\ v_t \in \mathcal{H})\ \vee\ t = T_{\max}$

Table 1: Résumé des composantes du MDP utilisé pour PPO.

Masque d'action dynamique. À chaque pas, la distribution de politique est calculée *uniquement* sur les arêtes autorisées ; les autres logits sont fixés à -10^9 avant la softmax.

Réseau de politique. La combinaison *Edge-conditioned GraphSAGE + MLP* décrite dans la sous-section précédente réalise la paire acteur–critique. Le vecteur d'entrée final est :

$$\big[g_t,\ b_t,\ p_t,\ f_{\text{pick}},\ f_{\text{deliv}},\ \text{one-hot}(v_t)\big].$$

Pseudo-code environnement (Gym).

```
class DroneDeliveryEnv(gym.Env):
    def reset(self):
        self.v = 0              # noeud virtuel
        self.batt = B_max
        self.colis, self.picked, self.deliv = p_max, 0, 0
        self.t = 0
        return self._obs()

    def step(self, action):
        if self.t == 0:
            self.v = hubs[action]
        else:
            self.v = out_neighbors[self.v][action]
        cost = edge_cost(prev, self.v)
        self.batt -= batt_cost(cost)
        if self.v == pickup and not self.picked:
            self.picked = 1
```

```
        if self.v == delivery and self.picked and not self.deliv:
            self.deliv, self.colis = 1, 0
        if node_type[self.v] == CHARGING:
            self.batt = B_max
        done = (self.picked and self.deliv and node_type[self.v]==HUB) \
               or self.t >= T_max
        reward = -cost - lam * (self.batt < 0)**2 - mu * (node_type[self.v]==CHARGIN
        self.t += 1
        return self._obs(), reward, done, False, {}
```

Pipeline d'entraînement. Le modèle `PolicyGNN` est entraîné via PPO avec $|\mathcal{E}|$ environnements parallèles (cf. listing 1).

Listing 1: Extraction minimale du script d'entraînement

```
envs = SubprocVecEnv([make_env(i) for i in range(num_envs)])
model = PPO(PolicyGNN, envs,
            n_steps=2048, batch_size=256,
            learning_rate=3e-4, gamma=0.99,
            gae_lambda=0.95, clip_range=0.2)
model.learn(total_timesteps=5_000_000)
```

Intégration des contraintes (mise à jour)

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

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

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

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

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.

4.4 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*,

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

  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 $e = (u \to v)$ 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,
- $v_t \in \{1, \ldots, N\}$: index du nœud courant.

4.5 Architecture du GNN

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

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

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

4.6 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)

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

Critique

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

4.7 Récompense instantanée

Pour tout pas t,

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

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

5 Algorithme PPO

5.1 Avantage (GAE-λ)

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

5.2 Perte « clipped »

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

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 :

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

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.