

**CENTRALE
LYON**

ÉCOLE CENTRALE LYON

OPTION INFORMATIQUE - UE PROJET
RAPPORT

Optimisation du trafic de drones en milieu restreint

Élèves :

Emile BELOT
Alice DOBIECKI
Clémence GIRARD

Enseignant :

Charles-Edmond BICHOT

24 mars 2025

Table des matières

1	Introduction	6
2	Etat de l'art	8
2.1	Centralisation vs. Décentralisation dans la gestion du trafic de drones . . .	8
2.2	Planification de mission et allocation des tâches	8
2.3	Gestion des conflits et détection d'obstacles	8
2.4	Optimisation des trajectoires en milieu restreint	9
2.5	Conclusion et perspectives	9
3	Présentation du problème d'optimisation	10
3.1	Les contraintes	10
3.2	Un problème d'apparence NP-complet	10
3.3	Hypothèses simplificatrices ???	11
3.4	???	11
4	Modélisation	12
4.1	Modélisation d'entrepôt	12
4.2	Paramètres de la modélisation	12
4.2.1	Création de l'instance entrepôt	14
4.3	Modélisation des drones	15
4.4	Visualisation 3D des trajectoires	15
5	Planification	17
5.1	Paramètres de la planification	17
5.2	Graphe avec point de passage	18
5.3	Graphe sans point de passage	19
5.4	Calcul de distance minimale	19
5.4.1	Bellman-Ford	19
5.4.2	BFS	22
5.4.3	Comparaison des deux méthodes	22
5.5	Méthodologie de génération des tâches	23
5.6	Méthodologie d'attribution des tâches	25
5.7	Résultats de la planification sur les différents entrepôts	26
6	Évitement	29
6.1	Stratégie pour la prise en compte des collisions	29
6.1.1	Les collisions directes	30
6.1.2	Les collisions croisées	30
6.1.3	Les frôlements	31
6.2	Définition de la fonction coût	32
6.3	Recherche d'une solution optimale admissible	32
6.3.1	Parcours de solutions dans l'espace de recherche	32
6.4	Algorithme du recuit simulé	33
6.5	Élaboration des stratégies	34
6.5.1	Stratégie n°1	34
6.5.2	Résultat de la stratégie n°1	35

6.5.3	Visualisation de la stratégie n°1	35
6.5.4	Stratégie n°2	35
7	Perspectives du projet	37
8	Conclusion	38
9	Annexes	40

Table des figures

1	Photographie d'un drone dans un entrepôt	6
2	Résultat de la fonction <i>build_warehouse()</i> pour l'entrepôt <i>intermediate_warehouse</i>	15
3	Aperçu visualisation des trajectoires	16
4	Exemple d'entrepôt avec points de passage	19
5	Exemple d'entrepôt sans point de passage	19
6	Graphe des checkpoints	21
7	Type de collisions détectées par l'algorithme d'Evitement	29
8	Parcours de l'espace de recherche des solutions	33

Résumé

Summary

Remerciements

1 Introduction

Le concept de drone naît pendant la Seconde Guerre Mondiale, où les premiers prototypes d'avion sans pilote radio-commandés voient le jour. Bien que son essor est majoritairement dû à son utilisation au sein de l'armée pour la surveillance civile ou encore l'offensive criminelle, le drone ne s'y limite pas. Il est également utilisé par de nombreuses entreprises, pour le transport logistique ou même la livraison client. Ainsi, Amazon, UPS et La Poste sont des exemples d'entreprises qui exploitent ces technologies pour automatiser le stockage et la récupération des colis, améliorant ainsi considérablement l'efficacité logistique. En particulier, Amazon a conçu son propre type de drone de transport de colis, permettant le déplacement de petits colis à l'intérieur des entrepôts, appelé Amazon Prime Air drones.

Un enjeu majeur de l'utilisation des drones réside dans la planification d'un réseau de drones et donc d'une coordination optimisée et sécurisée des trajectoires. En effet, le risque de collisions dû à la forte densité du trafic aérien dans un milieu restreint peut engendrer la dégradation des drones, des infrastructures et peut représenter un danger pour les personnes au sol.



FIGURE 1 – Photographie d'un drone dans un entrepôt

Ce projet se concentre sur l'optimisation des trajectoires 4D des drones évoluant dans un entrepôt. L'objectif est de garantir le respect des contraintes de séparation entre drones à tout instant, tout en minimisant le temps total de vol. Le problème du trafic de drones en milieu restreint semble être NP-complet en raison de la complexité combinatoire associée à l'optimisation des trajectoires sous contraintes.

Du fait de ces caractéristiques, il est improbable d'obtenir une solution exacte à ce problème en un temps raisonnable. Il a donc été indispensable de mettre en place des heuristiques et des algorithmes d'optimisation approchés afin d'obtenir une solution optimale, mais non exacte, en un temps raisonnable.

Le projet est divisé en 2 grandes parties : la Planification permettant l'attribution des tâches à la flotte de drones en choisissant le chemin le plus court, et l'Évitement qui à partir de la solution de la première partie, applique un algorithme métaheuristique



pour obtenir une solution optimale de trajectoires sécurisées (sans risque de collision). On commencera par présenter l'objectif, les contraintes et la formulation mathématique du problème d'optimisation. Ensuite, on s'intéressera à la construction et à la modélisation de l'entrepôt ainsi qu'au déplacement des drones. Le déploiement de la planification et la gestion de l'évitement des collisions seront ensuite largement approfondis. Enfin, on discutera de la visualisation "en temps réel" des trajectoires optimales des drones obtenues grâce aux algorithmes développés.

2 Etat de l'art

L'optimisation du trafic de drones en milieu restreint, comme un entrepôt, est un domaine en pleine expansion, motivé par des applications telles que la logistique, la surveillance et la gestion des stocks. Ce sujet combine des défis liés à la planification de trajectoires, la gestion des conflits, et l'optimisation des ressources dans un environnement contraint. Cet état de l'art synthétise les travaux existants et identifie les pistes de recherche actuelles.

2.1 Centralisation vs. Décentralisation dans la gestion du trafic de drones

La gestion centralisée du trafic de drones, comme proposé dans **Metropolis II** [1], utilise une autorité unique pour concevoir des trajectoires 4D optimales évitant les conflits. Cette approche, basée sur un programme linéaire en nombres mixtes MILP (Mixed-Integer Linear Programming), permet de minimiser le temps de vol total tout en garantissant la sécurité. Cependant, elle peut atteindre ses limites en termes d'évolutivité et de réactivité dans des environnements dynamiques comme les entrepôts.

À l'inverse, des approches décentralisées, comme celle proposée par **Picard** [4], utilisent l'optimisation distribuée sous contraintes DCOP (Desktop COmmunication Protocol) pour permettre aux drones de coordonner leurs trajectoires localement. Cette méthode est plus adaptée aux environnements dynamiques mais peut souffrir de problèmes de convergence et de coordination dans des espaces très encombrés.

2.2 Planification de mission et allocation des tâches

Dans un contexte militaire, **Guillon et Farges** [2] mettent en avant une décomposition hiérarchique des tâches et l'utilisation de graphes de visibilité pour optimiser les trajectoires. Leur approche, bien que conçue pour des missions complexes, pourrait être adaptée à des entrepôts en intégrant des contraintes spécifiques comme les obstacles fixes et les zones interdites.

Une étude récente de **Smith et al.** [5] propose une méthode de planification de missions multi-drones dans des environnements intérieurs, utilisant des algorithmes génétiques pour optimiser les trajectoires tout en évitant les collisions. Cette approche est particulièrement pertinente pour les entrepôts, où les drones doivent naviguer entre des étagères et des obstacles dynamiques.

2.3 Gestion des conflits et détection d'obstacles

Le modèle de gestion des conflits proposé par la **FAA-NASA UTM** [6] distingue trois niveaux : stratégique, tactique et réactif. Les services comme la détection de conflits CDS (Conflict Detection System) et la mise à jour des trajectoires TUS (Trajectory Update System) sont essentiels pour garantir la sécurité. Cependant, en milieu restreint, la détection d'obstacles doit être plus fine et plus réactive.

Zhang et al. [7] proposent une méthode basée sur l'apprentissage profond pour la détection d'obstacles en temps réel dans des environnements intérieurs. Leur approche combine des capteurs embarqués (LiDAR, caméras) avec des algorithmes de prédiction de trajectoires, offrant une solution prometteuse pour les entrepôts.

2.4 Optimisation des trajectoires en milieu restreint

Les travaux de **Metropolis II** [1] et **Picard** [4] soulignent l'importance de modéliser les trajectoires en 4D (espace + temps) pour éviter les conflits. En milieu restreint, cette modélisation doit intégrer des contraintes supplémentaires, comme les limites physiques de l'entrepôt et la présence d'obstacles dynamiques (par exemple, des chariots élévateurs).

Une étude récente de **Lee et al.** [3] propose un algorithme d'optimisation basé sur des essaims de particules PSO (Particle Swarm Optimization) pour planifier des trajectoires dans des environnements intérieurs complexes. Leur méthode prend en compte les incertitudes liées aux capteurs et aux mouvements des obstacles, ce qui est crucial pour les applications en entrepôt.

2.5 Conclusion et perspectives

Les travaux existants offrent des solutions variées pour l'optimisation du trafic de drones, allant des approches centralisées aux méthodes décentralisées. Pour les entrepôts, une combinaison de ces approches semble nécessaire, intégrant :

- Une planification centralisée initiale pour optimiser les trajectoires globales
- Une coordination décentralisée pour gérer les événements dynamiques (obstacles, retards)
- Des méthodes avancées de détection d'obstacles et d'apprentissage automatique pour améliorer la réactivité

Les futures recherches pourraient explorer l'intégration de l'IA pour la prédiction des mouvements d'obstacles et l'optimisation en temps réel des trajectoires, ainsi que l'utilisation de simulations multi-agents pour valider ces approches dans des scénarios réalistes.

3 Présentation du problème d'optimisation

Notre problème consiste à optimiser la planification des trajectoires de drones autonomes évoluant dans un entrepôt restreint afin de minimiser la durée totale de vol.

L'objectif est donc le suivant :

Réduire la durée totale de vol des d drones sous contrainte d'une durée de travail fixée avec un nombre d'articles n à traiter.

On considère que l'entreprise possédant l'entrepôt est équipée d'une flotte de d drones homogènes. Les d drones débutent leur mission depuis des stations de recharge au sol. L'entrepôt est équipé de n tapis d'arrivée au sol pour les articles et de m tapis d'emballage au sol également. Il contient également de grandes étagères où les articles sont positionnés à des hauteurs précises. La liste des tâches à réaliser est définie en début de journée par un opérateur de l'entrepôt avec un nombre n d'articles à traiter. Il existe seulement deux types de tâches : déplacer un colis d'un tapis d'arrivée jusqu'à une case de l'étagère disponible et récupérer un colis d'une case pour le déposer sur un tapis de départ. A la fin de chaque tâche, les drones retournent à la station de charge. Selon la planification, le drone décolle verticalement de la station de recharge et se déplace à vitesse constante selon des mouvements verticaux et horizontaux en suivant le chemin préétabli. Il peut ajuster sa hauteur, mais la vitesse est conservée constante en première hypothèse simplificatrice. L'objectif est de minimiser la durée totale de vol des d drones tout en respectant une durée de travail imposée et un nombre d'articles à traiter donné.

3.1 Les contraintes

Plusieurs contraintes s'ajoutent à ce problème :

- Les drones doivent éviter les infrastructures et obstacles fixes de l'entrepôt (étagère, parois, tapis, station de recharge, ...).
- Les drones ne peuvent pas voler en continu et doivent respecter l'autonomie de leur batterie. Il doivent se recharger afin d'avoir un temps de vol suffisant pour réaliser leurs tâches.
- Les drones doivent suivre une trajectoire sûre en évitant de croiser (collisions) ou de frôler d'autres drones (collisions évitées de justesse selon une distance de sécurité inter-drone)

3.2 Un problème d'apparence NP-complet

Ce problème une fois défini s'apparente à un problème classique d'optimisation de trajets, comme le problème du voyageur de commerce (TSP) ou ses variantes multi-agents, qui sont connus pour être NP-difficiles. Chaque drone doit suivre un itinéraire efficace pour collecter ou déposer des articles tout en minimisant le temps de vol, ce qui implique un grand nombre de combinaisons possibles à explorer. De plus, ce problème inclut la contrainte dynamique forte de l'évitement des collisions entre drones, ce qui le rapproche des problèmes de planification sous contraintes et d'ordonnancement de tâches, eux aussi généralement NP-complets. Enfin, l'espace de recherche est exponentiel : chaque drone évolue dans un environnement tridimensionnel avec une contrainte temporelle, ce



qui complexifie encore davantage la recherche d'une solution optimale. Le problème semble donc NP-complet : il sera possible de trouver une solution optimale non-exacte et de la vérifier dans un temps polynomial. On adopte donc une approche algorithmique pour rechercher une solution approchée garantissant un compromis entre qualité et faisabilité opérationnelle.







3.3 Hypothèses simplificatrices ???

3.4 ???

4 Modélisation

4.1 Modélisation d'entrepôt

Pour pouvoir étudier les trajectoires des drones, on a effectué un quadrillage de l'entrepôt dans les trois dimensions (x, y, z). À chaque case dans l'entrepôt, on peut attribuer 7 valeurs possibles, associées à 7 états possibles différents dans notre modélisation :

- **0 - empty** : la case est un espace vide, de libre circulation pour les drones.
-  **1 - shelf** : la case est une partie de la structure de l'étagère non accessible pour un drone. Elle représente un obstacle fixe lors du calcul de la trajectoire des drones et permet de définir les rebords des étagères complètes.
-  **2 - storage line** : la case est un panier de l'étagère vide, où un objet peut être déposé par un drone. Ces cases doivent être définies sur une case où une étagère a déjà été définie.
-  **3 - object** : la case est un panier de l'étagère contenant un objet, qui peut être enlevé par un drone. Ces cases doivent être définies à un endroit où existe déjà une storage line.
-  **4 - start mat** : la case est un tapis de départ. Une case peut contenir plus d'un objet, et donc accueillir plus d'un drone : on définit donc un paramètre représentant cette capacité dans le code.
-  **5 - finish mat** : la case est un tapis d'arrivée. De même, une case peut accueillir plus d'un drone et possède une capacité. Par souci de simplicité, on considère les capacités des tapis d'arrivée et de départ égales.
-  **6 - charging station** : la case est une station de recharge, les drones commencent et finissent la journée sur la station. Ils peuvent s'y déposer toute la journée. Une case peut accueillir plus d'un drone.

Afin de pouvoir tester nos algorithmes de Planification et d'Évitement, plusieurs types d'entrepôts ont été générés. En s'appuyant sur des exemples réalistes, des entrepôts en rangées rectilignes, en rangées de U, en rangées de T, ont été générées avec des tailles et des hauteurs d'entrepôts variés, afin d'adapter la complexité de l'entrepôt pour les différents tests. Après avoir défini les différents objets entrant en jeu dans la définition d'un entrepôt pour notre projet, on implémente cette logique dans notre code.

4.2 Paramètres de la modélisation

Pour faciliter l'enregistrement de nouveaux entrepôts utilisés plus tard en phase de test, on introduit le fichier *config_warehouse.json* du dossier général **Warehouse**, contenant les différents scripts et fichiers relatif à la création de toute pièce d'un entrepôt. Ce fichier *config_warehouse* contient pour chaque entrepôt les détails des coordonnées (x, y, z) de la structure de l'entrepôt en incluant ses dimensions, les capacités de ses tapis de départ et d'arrivée, ainsi que la disposition des différents éléments logistiques.

Par exemple, la configuration propre à l'entrepôt *intermediate_warehouse*, comme affiché ci-dessous, représente un espace tridimensionnel de 15x15x3 unités. Le paramètre *mat_capacity* fixe la capacité maximale de drones que peut contenir un tapis d'arrivée ou un tapis de départ. Ainsi, pour *mat_capacity* = 10, cela signifie que 10 drones peuvent

se placer sur le tapis d'arrivée ou le tapis de départ, bien que le tapis ne soit représenté que par une case.

Les étagères sont positionnées sous forme de blocs aux coordonnées spécifiées : pour une hauteur fixée, les coordonnées (x, y) sont renseignées sous forme de liste de liste. Les lignes de stockage sont définies par une ligne horizontale et une ligne verticale pour structurer l'espace et optimiser le rangement des objets.

Les autres éléments définis dans la modélisation théorique sont intégrés sous forme de liste de liste, contenant les coordonnées (x, y, z) pour chaque instance introduite :

- Les **objets** placés dans l'entrepôt.
- Les **tapis de départ et d'arrivée des colis**, utilisés pour le transit des articles.
- Une **station de recharge** dédiée aux drones.
- Les **points de passage** utilisés pour la première stratégie de planification permettant de forcer la trajectoire des drones en certains points.

```
{
  "warehouses": {
    ...
    "intermediate_warehouse": {
      "dimensions": [15, 15, 3],
      "mat_capacity": [10],
      "shelves": [
        [2, [2, 2], [2, 12], [4, 2], [4, 12]],
        [2, [5, 2], [5, 4], [12, 2], [12, 4]],
        [2, [5, 10], [5, 12], [12, 10], [12, 12]]
      ],
      "storage_lines": {
        "horizontal": [
          [2, [2, 2], [2, 12]],
          [2, [4, 5], [4, 9]],
          [2, [12, 10], [12, 12]],
          [2, [12, 2], [12, 4]]
        ],
        "vertical": [
          [2, [3, 2], [11, 2]],
          [2, [5, 4], [11, 4]],
          [2, [3, 12], [11, 12]],
          [2, [5, 10], [11, 10]]
        ]
      },
      "objects": [],
      "start_mat": [[14,2,0]],
      "finish_mat": [[14,4,0]],
      "charging_station": [[14,0,0]],
      "checkpoints": [[[14,14,0],0]],
      "checkpoint_connection": []
    }
  }
}
```

Ce fichier JSON sert donc à initialiser un environnement logistique structuré, utilisé par la classe **Warehouse** pour générer un entrepôt et utilisé a posteriori dans le fichier principal *main.py* pour faire tourner nos algorithmes de Planification et d'Évitement et la circulation des robots dans l'entrepôt.

4.2.1 Création de l'instance entrepôt

La classe **Warehouse3D**, définie dans le fichier python *warehouse.py* du dossier **Warehouse**, est l'élément central permettant de générer un entrepôt sur mesure à partir du fichier de configuration présenté plus tôt. Cette classe contient des fonctions offrant la possibilité d'ajouter à l'entrepôt des objets, des étagères, des points de passage, des tapis, ... selon la disposition dans la configuration. La fonction *compute_manhattan_distance_with_BFS* (Breadth-First Search) permet, elle, de calculer les distances de Manhattan entre deux

points de l'entrepôt. On expliquera plus tard, dans la partie 5, en quoi consiste cette méthode de calcul de distance minimale. De plus, la classe intègre deux fonctions permettant une première version de la visualisation de l'entrepôt. La fonction `display()` affiche l'entrepôt en 2D avec la librairie **matplotlib**, avec une grille d'axe (x, y) par hauteur (si 3 hauteurs on été définies dans la configuration, il y aura trois graphiques de l'entrepôt renvoyé par la fonction, avec $z = 0$, $z = 1$ et $z = 2$) qui permettent la représentation graphique en 2D de la structure.

Finalement, pour l'entrepôt *intermediate_warehouse*, on obtient l'entrepôt représenté sur la figure 2 ci-dessous.

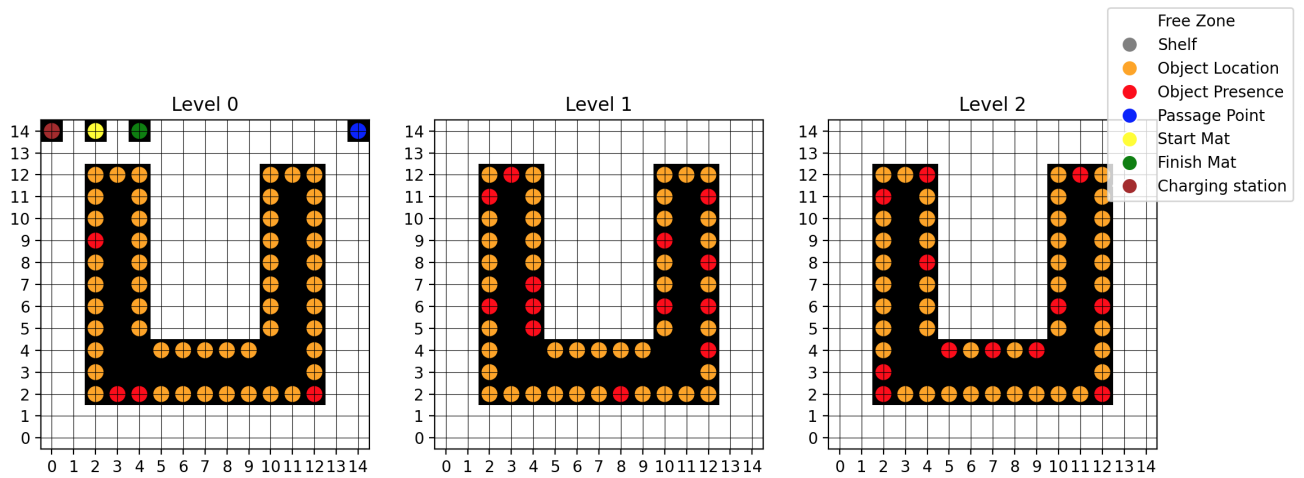


FIGURE 2 – Résultat de la fonction `build_warehouse()` pour l'entrepôt *intermediate_warehouse*

La fonction `show_graph()` affiche le graphe des connections entre les points de passage grâce à la librairie **networks**, montrant les passages autorisés pour la trajectoire des drones.

4.3 Modélisation des drones

Pour ce qui concerne la modélisation des drones, ces derniers sont soumis à de nombreuses simplifications.

4.4 Visualisation 3D des trajectoires

La partie visualisation des trajectoires en 3D offre un aperçu original du projet et permet de vérifier la qualité du travail d'optimisation.

L'utilisation de `plotly` permet de tracer assez aisément chaque itérations du scénario de déplacement.

On propose au lecteur d'avoir un aperçu de ce que renvoie le navigateur au moment de la visualisation dans la Figure 3

Animation des drones - Temps: 2025-03-24 08:19:30s

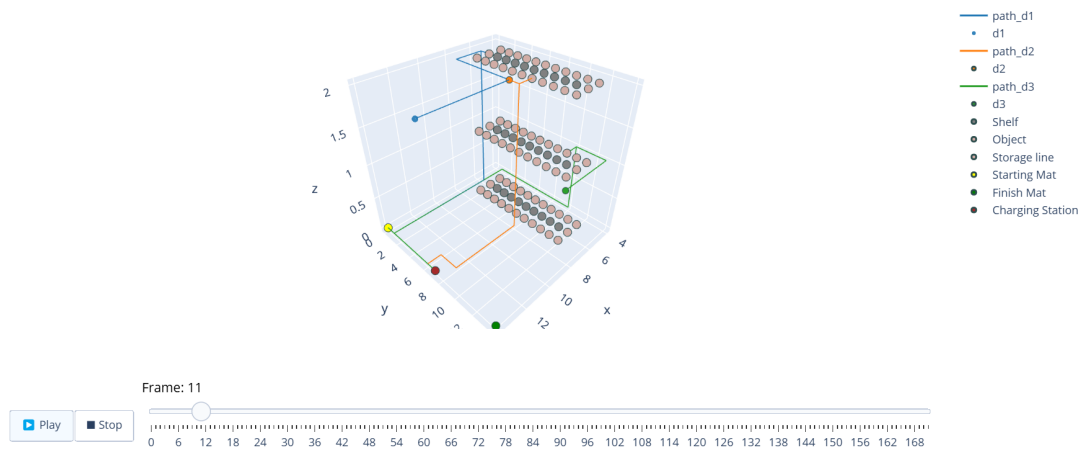


FIGURE 3 – Aperçu visualisation des trajectoires

5 Planification

La planification est une des sections majeures du projet. L'algorithme de planification s'appuie sur un grand nombre d'inputs et donne une première solution au problème général. Cette première solution proposée n'est en général pas admissible. C'est à dire que la contrainte majeure d'absence de collision n'est pas respectée.

Pour construire un planning de 0, de nombreux éléments sont à calculer. La distance et le parcours empruntés par chaque drone pour chaque tâche en est un. La stratégie de répartition des tâches ou encore les stratégies de charge en sont d'autres. Cette section a donc pour but d'introduire les différents algorithmes et intelligences programmés au lecteur et de détailler les différents paramètres utilisés pour une meilleure compréhension globale de la tâche.

5.1 Paramètres de la planification

Comme pour l'entrepôt, la planification dispose d'un fichier de configuration propre, nommé *config_planning.json*. Ce fichier au format JSON permet de conserver facilement différentes configurations de planning et facilite donc les tests et l'utilisation de l'algorithme générale.

Le fichier de configuration contient 2 sous listes. La première `config['planning']`, contient diverses configurations de planning où chacune d'elle reprend les éléments de la structure ci-dessous :

```
"planning_test_1": {
  "drone_quantity": 3,
  "object_time_treatment": 30,
  "drone_speed": 2,
  "battery": {
    "max_capacity": 3600,
    "initial_charge": 3600,
    "upper_threshold": 0.8,
    "lower_threshold": 0.2,
    "charger_speed": 2
  }
}
```

On proposera au lecteur de découvrir la signification de ces paramètres dans la Table 1.

Dénomination	Description	Unité
Name	Nom de la configuration	
Drone quantity	Nombre de drones en circulation	
Object time treatment	Temps d'interaction avec un objet	s
Drone speed	Vitesse uniforme de déplacement	u/min
Max capacity	Temps de vol maximum	s
Initial charge	Charge en début d'algorithme	s
Upper threshold	Limite de charge conseillée	%
Lower threshold	Limite de décharge conseillée	%
Charger speed	Vitesse de charge	sans unité*

TABLE 1 – Description des paramètres de la configuration de planning

**pour une vitesse de charge paramétrée à 2 : 1 min de charge permet 2 min de vol*

La deuxième section accessible, en appelant `config['task_name_mapping']` permet de nommer les typologies de tâches qui seront affectés dans le planning et de les coder en dure à un seul endroit :

```
"task_name_mapping": {
    "positioning": "PO",
    "arrival": "A",
    "departure": "D",
    "return_charge": "RC"
}
```

On proposera au lecteur de découvrir la signification de ces paramètres dans la Table 2.

Dénomination	Abréviation	Description tâche
positioning	PO	Positionnement du drone vers une nouvelle tâche
arrival	A	Tâche d'arrivée (le drone va poser un objet)
departure	D	Tâche de départ (le drone va récupérer un objet)
return_charge	RC	Le drone retourne à la charge

TABLE 2 – Description des paramètres de la configuration de planning

5.2 Graphe avec point de passage

Dans la première configuration, différents points de passage ont été disséminés au travers de l'entrepôt. Ces derniers sont des points stratégiques obligatoires pour une bonne circulation. Dans le fichier de configuration, l'opérateur spécifie quels sont les points de passages qui seront reliés entre eux. En général, ce sont ceux proches ou en trajectoire directe.

Ces points de passages sont symbolisés par un cercle bleu dans la légende et prennent la valeur 4 dans la matrice d'entrepôt. La Figure 4 ci-dessous illustre un exemple d'entrepôt avec points de passage :

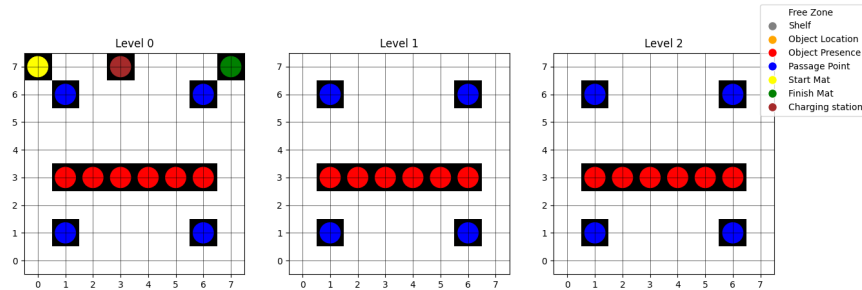


FIGURE 4 – Exemple d'entrepôt avec points de passage

5.3 Graphe sans point de passage

Dans cette deuxième configuration, il n'existe plus de points de passage. Toutes les cases de circulations deviennent des sommets du graphe. Le graphe sera donc beaucoup plus gros. La Figure 5 ci-dessous illustre un exemple d'entrepôt sans point de passage :

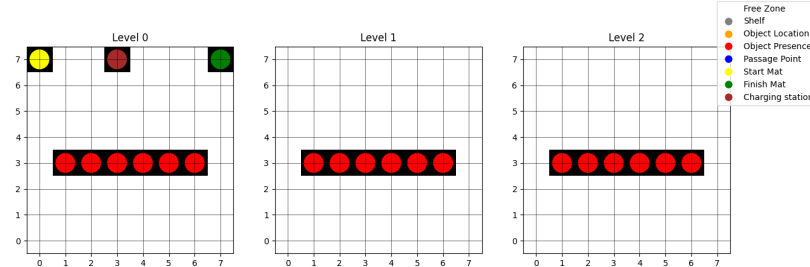


FIGURE 5 – Exemple d'entrepôt sans point de passage

5.4 Calcul de distance minimale

Avant toute répartition des tâches, il est important de pouvoir calculer la distance de chaque tâches et le déplacement de chaque drone à chaque instant. Puisque l'objectif est de minimiser le temps de vol, des algorithmes sont nécessaires pour assurer que la distance est minimale à chaque instant. Deux approches ont alors été déployées et comparées.

5.4.1 Bellman-Ford

Matrice d'adjacence : On propose au lecteur quelques rappels concernant la matrice d'adjacence en théorie des graphes. Soit un graphe $G = (V, E)$ avec n sommets numérotés de 1 à n . La **matrice d'adjacence** A associée à G est une matrice carrée $n \times n$ définie par :

$$A[i, j] = \begin{cases} 1 & \text{si une arête existe entre le sommet } i \text{ et le sommet } j \\ 0 & \text{sinon} \end{cases}$$

Notre projet répond à plusieurs cas particuliers qui ont un impact direct sur la matrice d'adjacence :

- **Le graphe est pondéré** : $A[i, j]$ représente la distance de l'arête entre i et j .
- **Le graphe est non orienté** : La matrice A est **symétrique** ($A[i, j] = A[j, i]$).

Pour construire la matrice d'adjacence de l'entrepôt, il est nécessaire de définir quels seront les sommets du graphe et lesquels seront reliés entre eux. On propose donc une version de la matrice de transition simplifiée sous forme de blocs :

	Objet	Panier	Checkpoint	Tapis arrivé	Tapis départ	Borne de charge
Objet	∞	∞	M1	∞	∞	∞
Panier	∞	∞	M2	∞	∞	∞
Checkpoint	M1	M2	∞	M3	M4	M5
Tapis arrivé	∞	∞	M3	∞	∞	M6
Tapis départ	∞	∞	M4	∞	∞	M7
Borne de charge	∞	∞	M5	M6	M7	∞

TABLE 3 – Matrice des connexions entre les différents éléments

Un symbole ∞ signifie qu'il n'est pas possible de transiter d'un élément i à j . Par exemple, il n'est pas possible d'aller d'un objet à un panier. Un objet transite uniquement vers un/depuis un tapis par l'intermédiaire d'un point de passage.

Une fois les sommets définis et reliés entre eux, il est possible d'utiliser la librairie `networkx` de Python pour visualiser les graphes. La Figure 6 ci-dessous illustre l'exemple du sous-graphe des points de passages de l'entrepôt présenté en 5.2. :

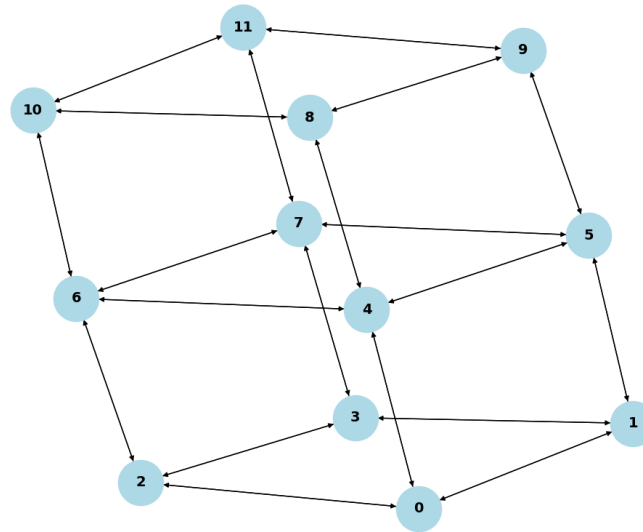


FIGURE 6 – Graphe des checkpoints

Algorithme : Une fois la matrice d'adjacence réalisée, le déploiement de l'algorithme est possible. Pour rappel, l'algorithme de Bellman-Ford permet de trouver les plus courts chemins depuis une source unique dans un graphe orienté, même si certains arcs ont des poids négatifs. Il fonctionne en effectuant plusieurs passes sur l'ensemble des arêtes du graphe pour progressivement améliorer les estimations de distances.

On propose au lecteur de découvrir l'algorithme en pseudo code, rédigé ci-dessous :

Algorithm 1 Algorithme de Bellman-Ford pour un graphe orienté pondéré

Graphe orienté $G = (V, E)$ avec des poids $w(u, v)$, sommet source s Distances minimales $d[v]$ depuis s , prédécesseurs $p[v]$

Étape 1 : Initialisation

for chaque sommet $v \in V$ **do** $d[v] \leftarrow +\infty$, $p[v] \leftarrow \text{nil}$ $d[s] \leftarrow 0$

Étape 2 : Relaxation des arêtes

for $i \leftarrow 1$ à $|V| - 1$ **do**

for chaque arête $(u, v) \in E$ **do**

if $d[u] + w(u, v) < d[v]$ **then** $d[v] \leftarrow d[u] + w(u, v)$ $p[v] \leftarrow u$

Étape 3 : Détection des cycles négatifs

for chaque arête $(u, v) \in E$ **do**

if $d[u] + w(u, v) < d[v]$ **then** **Retourner** "Cycle négatif détecté"

Retourner $d[v]$ et $p[v]$

Après avoir lancé l'algorithme, on obtient une matrice des distances minimum D :

$D_{i,j}$ = distance minimale qui sépare le sommet i et j dans le graph

On peut donc à présent relier chaque sommet du graphe par le chemin le plus court.

5.4.2 BFS

La deuxième méthode, pour calculer les distances entre 2 sommets est la méthode de parcours en largeur (BFS pour Breadth First Search). Cette méthode permet d'explorer un graphe de manière systématique en visitant d'abord tous les voisins d'un sommet avant de passer aux niveaux suivants. Il est souvent utilisé pour trouver le plus court chemin dans un graphe non pondéré, explorer un graphe connexe, ou résoudre des problèmes comme celui du labyrinthe. L'avantage du BFS appliqué à la grille est qu'il ne nécessite pas le calcul d'une matrice d'adjacence complète pour fonctionner.

On propose au lecteur de découvrir l'algorithme en pseudo code, rédigé ci-dessous :

Algorithm 2 Algorithme de parcours en largeur (BFS)

Graphe $G = (V, E)$ sous forme de liste d'adjacence, sommet source s Distances $d[v]$ depuis s , prédécesseurs $p[v]$

Étape 1 : Initialisation

for chaque sommet $v \in V$ **do** $d[v] \leftarrow +\infty$, $p[v] \leftarrow \text{nil}$ $d[s] \leftarrow 0$ Créer une file vide F et y ajouter s

Étape 2 : Parcours en largeur

while F n'est pas vide **do** Extraire un sommet u de F

for chaque voisin v de u **do**

if $d[v] = +\infty$ **then** $d[v] \leftarrow d[u] + 1$ $p[v] \leftarrow u$ Ajouter v à F

Retourner $d[v]$ et $p[v]$

5.4.3 Comparaison des deux méthodes

Ces deux algorithmes ont chacun des avantages et des inconvénients.

Critère	BFS	Bellman-Ford
Type de graphe	Non pondéré	Pondéré (poids positifs et négatifs)
Objectif	Plus court chemin en nombre de pas	Plus court chemin en coût total
Complexité	$O(V + E)$	$O(VE)$
Gestion des poids	Tous les poids égaux	Poids variables, négatifs inclus
Adapté aux grandes grilles	Oui	Non
Détection des cycles négatifs	Non	Oui
Facilité d'implémentation	Simple	Plus complexe
Cas d'usage	Labyrinthes, chemins dans une grille	Réseau routier avec coûts variables, finance

TABLE 4 – Comparaison entre BFS et Bellman-Ford

Pour comparer les solutions, on va donc utiliser BFS sur le graphe sans point de passage car il est rapide, efficace et simple pour un graphe sous forme de grille.

Pour le graphe avec checkpoint, on utilisera en revanche Bellman-Ford qui sera approprié pour le graphe pondéré avec un poids différent pour chaque déplacement.

En appelant les deux algorithmes pour le même entrepôt, voici un extrait des résultats que l'on peut obtenir :

task_type	id	departure	arrival	time	distance Bellman	distance BFS
A	HB9321LC	[2, 0, 0]	[2, 4, 0]	08 :00	8	6
A	WLFYA1B5	[3, 0, 0]	[5, 3, 0]	08 :00	9	7
A	A8NSJ0A2	[3, 0, 0]	[5, 5, 0]	08 :00	11	9
A	8E83IHSK	[3, 0, 0]	[2, 3, 0]	08 :00	10	6
A	V51AFNFQ	[3, 0, 0]	[2, 2, 0]	08 :00	9	3
A	X2DR7T4H	[3, 0, 0]	[4, 2, 0]	08 :00	9	3
A	FXH4Y7XB	[3, 0, 0]	[2, 4, 0]	08 :00	11	7
A	TFNFB0D0	[3, 0, 0]	[5, 2, 0]	08 :00	8	4
A	ZT2AK242	[3, 0, 0]	[2, 4, 0]	08 :00	11	7
A	12VXLD00	[2, 0, 0]	[2, 2, 0]	08 :00	6	2
A	0HMM2HLK	[3, 0, 0]	[5, 2, 0]	08 :00	9	5
A	MK1YUUT2	[3, 0, 0]	[2, 3, 0]	08 :00	10	6
A	0NS3H7QM	[3, 0, 0]	[5, 2, 0]	08 :00	8	4
A	9A76EF68	[2, 0, 0]	[5, 5, 0]	08 :00	14	10

TABLE 5 – Tableau des tâches avec distances calculées

On constate et c'est sans appel, que les distances calculées sur le graphe associé à l'algorithme de Bellman sont toutes supérieures à celle calculées par le BFS. Et c'est compréhensible. En effet, le faible nombre de point de passage (relativement bas dans l'exemple pour l'illustrer) rend la trajectoire des drones désoptimisées. Le BFS lui, n'est pas contraint dans son itinéraire et assure qu'on se déplacera de la façon la courte à chaque fois. On peut longer les parois et changer de direction tout en gardant l'objectif en vue. Les points de passages imposent des détours inutiles.

Et c'est partant de ce constat que pour la suite du projet, seul le calcul par BFS sera conservé et implémenté.

5.5 Méthodologie de génération des tâches

Dans l'entrepôt, les drones réalisent des tâches prédéfinies qui permettent de traiter les objets. En réalité, ces tâches sont données à l'algorithme en paramètre par l'utilisateur, mais les besoins du projet impliquent de générer une telle liste, qui servira de donnée d'entrée pour tester l'algorithme d'optimisation du trafic des drones. Chaque objet est représenté par un identifiant unique généré aléatoirement, composé de lettres et de chiffres. Les objets peuvent être initialement placés soit sur des étagères, soit sur des tapis d'arrivée. Les positions disponibles pour les objets sont identifiées dans l'entrepôt en fonction de la matrice de stockage. Un objet est assigné aléatoirement à une étagère ou à un emplacement d'arrivée. Si aucune position libre n'est disponible, l'objet n'est pas

placé. Pour chaque mouvement d'objet, un horaire et un emplacement cible sont déterminés. Les plages horaires disponibles pour les arrivées et les départs sont organisées dans des dictionnaires associant des capacités d'accueil à chaque emplacement. Pour chaque tâche, l'algorithme sélectionne un horaire et un emplacement en fonction des disponibilités restantes. Si aucun emplacement n'est disponible pour une catégorie de tâches (arrivée ou départ), l'objet est exclu de la simulation.

Une liste de tâches est créée sous forme de fichier CSV contenant les déplacements des objets selon le format de la Table 6 ci dessous :

task_type	id	pos0	pos1	time
A	HB9321LC	[14, 2, 0]	[4, 2, 1]	08 :00 :00
A	9A76EF68	[14, 2, 0]	[12, 2, 0]	08 :00 :00
A	TM4KSMO9	[14, 2, 0]	[6, 4, 0]	08 :00 :00
D	9BD31A7U	[9, 2, 0]	[14, 4, 0]	14 :00 :00
A	ZT2AK242	[14, 2, 0]	[4, 9, 0]	08 :00 :00
D	MOKCOC0A	[2, 12, 2]	[14, 4, 0]	14 :00 :00
D	IHG4FH2L	[2, 2, 2]	[14, 4, 0]	14 :00 :00
A	8RFLKZQ3	[14, 2, 0]	[8, 4, 1]	08 :00 :00
D	FG3P7JTL	[6, 10, 1]	[14, 4, 0]	14 :00 :00
D	JI9VJPPJ	[5, 4, 1]	[14, 4, 0]	14 :00 :00
D	4JY2TFWP	[12, 11, 2]	[14, 4, 0]	14 :00 :00
D	HB9321LC	[4, 2, 1]	[14, 4, 0]	14 :00 :00
A	62GAI81Z	[14, 2, 0]	[4, 8, 0]	08 :00 :00
A	CYSBZII2	[14, 2, 0]	[12, 12, 2]	08 :00 :00
D	YQR38U9J	[8, 4, 2]	[14, 4, 0]	14 :00 :00
D	A8NSJ0A2	[4, 7, 2]	[14, 4, 0]	14 :00 :00
D	CWE5IZM5	[4, 9, 2]	[14, 4, 0]	14 :00 :00
A	XDxD5AJT	[14, 2, 0]	[2, 5, 2]	08 :00 :00
A	0HMM2HLK	[14, 2, 0]	[2, 10, 0]	08 :00 :00
D	Z12VXLD0	[2, 4, 0]	[14, 4, 0]	17 :00 :00
D	9JFFEXPG	[11, 2, 1]	[14, 4, 0]	17 :00 :00
D	CYSBZII2	[12, 12, 2]	[14, 4, 0]	17 :00 :00
A	ULRNT01J	[14, 2, 0]	[4, 5, 1]	08 :00 :00

TABLE 6 – Tableau des tâches à effectuer

Le processus suit ces étapes : sélection aléatoire d'un objet mobile, détermination de son type de tâche en fonction de sa position actuelle, attribution d'un horaire et d'un emplacement en tenant compte des contraintes d'espace et de temps, enregistrement de la tâche dans le fichier CSV, mise à jour de l'état de l'objet et de l'entrepôt pour refléter le mouvement effectué. Une fois les tâches générées, le fichier CSV est sauvegardé dans un répertoire dédié sous le nom TL_<nom_entrepot>.csv. Un message de confirmation est affiché indiquant le nombre de tâches effectivement générées. Cette méthodologie garantit une répartition aléatoire et réaliste des mouvements d'objets dans l'entrepôt, permettant de tester l'efficacité de l'algorithme d'optimisation du trafic des drones.

5.6 Méthodologie d'attribution des tâches

Puisqu'on est à présent capable d'évoluer partout dans l'entrepôt via les plus courtes distances, intéressons nous à l'attribution des tâches. Pour ce faire, plusieurs règles définies viennent cadrer le problème :

- **Règle de priorité horaire** : Les tâches du lot N-1 doivent être terminées avant le démarrage des tâches du lot N.
- **Règle de la tâche la plus courte** : Si 2 tâches proviennent du même lot et peuvent être réalisées par le même drone, alors la tâche permettant une fin d'exécution la plus proche sera réalisée en première (désengorgement des tapis)
- **Règle de retour à la charge** : Après chaque tâche effectuée, le drone se repositionne par défaut à la charge.
- **Règle concernant la batterie des drones** : Un drone ne peut voler que si sa batterie est suffisamment chargée pour assurer la bonne réalisation de la tâche impliquant (positionnement à la tâche + réalisation de la tâche + retour à la charge)
- **Règle concernant l'autonomie de la batterie** : pour assurer la préservation de la batterie sur le long terme, le drone doit s'assurer de revenir à la station de charge avec x% de batterie et sa charge optimisée s'arrêtera lorsque le niveau y% sera atteint. Les niveaux recommandés pour les batteries de type lithium-ion se situent entre 20 et 80%

Avec l'ensemble de ces règles, on est capable de produire un algorithme de planification complet. On propose au lecteur de découvrir l'algorithme en pseudo code, rédigé ci-dessous :

Algorithm 3 Création du planning des drones

Require: *csv* (DataFrame des tâches), *warehouse* (entrepôt 3D), *config* (paramètres de planification), *mapping_config* (config de nommage)

Ensure: Dictionnaire de DataFrames représentant le planning de chaque drone

```

1: while toutes les tâches ne sont pas terminées do
2:   undone_tasks  $\leftarrow$  récupérer les tâches non complétées dans csv
3:   for chaque tâche task dans undone_tasks do
4:     (start_time, drone_nb)  $\leftarrow$  DRONE_INITIAL_SELECTION(task,
drone_battery, drone_time_follow, ...)
5:     if aucun drone disponible (batterie insuffisante) then
6:        $\triangleright$  Sélection du meilleur drone à recharger
7:       (best_task, best_task_type, time_for_charge, battery_gain)  $\leftarrow$ 
DRONE_EMPTY_BATTERY_SELECTION(csv, ...)
8:        $\triangleright$  Mise à jour de la charge des drones
9:       (drone_time_follow, drone_battery)  $\leftarrow$  DRONE_CHARGING_PROCESS(battery_gain,
time_for_charge, ...)
10:      for chaque drone d do
11:        if best_task[d]  $\neq$  None then
12:          Mettre à jour le planning via FULL_TASK_TREATMENT(d,
best_task[d], ...)
13:        end if
14:      end for
15:    else
16:      if le drone est disponible avant la tâche then
17:         $\triangleright$  Calcul du temps de charge possible avant la tâche
18:        (drone_time_follow, drone_battery)  $\leftarrow$ 
DRONE_CHARGING_PROCESS(drone_nb, ...)
19:      end if
20:      Mettre à jour le planning via FULL_TASK_TREATMENT(drone_nb, task,
...)
21:    end if
22:  end for
23: end while
24: return planning = 0

```

5.7 Résultats de la planification sur les différents entrepôts

On propose au lecteur de découvrir l'output du planning au travers de la Table 7. Ce planning comprend 6 colonnes :

- **Time** : suivi horaire des mouvements drone
- **Task_type** : donne le type de mouvement qu'est en train de réaliser le drone
- **ID** : ID de l'objet qui est en train d'être manipulé
- **Position** : Tuple qui contient les coordonnées du drone
- **Battery_time** : Temps de vol restant
- **Battery_percentage** : Temps de vol restant en % du temps total

time	task_type	id	position	battery_time	battery_percentage
08 :00 :00	PO		(14, 7, 0)	0 days 01 :00 :00	100
08 :03 :30	PO		(14, 0, 0)	0 days 00 :56 :30	94.17
08 :04 :00	A	0HMM2HLK	(14, 0, 0)	0 days 00 :56 :00	93.33
08 :04 :30	A	0HMM2HLK	(14, 1, 0)	0 days 00 :55 :30	92.5
08 :08 :30	A	0HMM2HLK	(6, 1, 0)	0 days 00 :51 :30	85.83
08 :09 :30	A	0HMM2HLK	(6, 1, 2)	0 days 00 :50 :30	84.17
08 :10 :00	A	0HMM2HLK	(6, 2, 2)	0 days 00 :50 :00	83.33
08 :10 :30	RC		(6, 2, 2)	0 days 00 :49 :30	82.5
08 :11 :00	RC		(6, 1, 2)	0 days 00 :49 :00	81.67
08 :12 :00	RC		(8, 1, 2)	0 days 00 :48 :00	80
08 :15 :00	RC		(8, 7, 2)	0 days 00 :45 :00	75
08 :18 :00	RC		(14, 7, 2)	0 days 00 :42 :00	70
08 :19 :00	RC		(14, 7, 0)	0 days 00 :41 :00	68.33
08 :26 :00	PO		(14, 7, 0)	0 days 00 :41 :00	68.33
08 :29 :30	PO		(14, 0, 0)	0 days 00 :37 :30	62.5
08 :30 :00	A	HKQGRT9S	(14, 0, 0)	0 days 00 :37 :00	61.67
08 :30 :30	A	HKQGRT9S	(14, 1, 0)	0 days 00 :36 :30	60.83
08 :35 :30	A	HKQGRT9S	(4, 1, 0)	0 days 00 :31 :30	52.5
08 :40 :30	A	HKQGRT9S	(4, 11, 0)	0 days 00 :26 :30	44.17
08 :41 :00	A	HKQGRT9S	(4, 11, 1)	0 days 00 :26 :00	43.33
08 :41 :30	A	HKQGRT9S	(5, 11, 1)	0 days 00 :25 :30	42.5
08 :42 :00	RC		(5, 11, 1)	0 days 00 :25 :00	41.67
08 :42 :30	RC		(4, 11, 1)	0 days 00 :24 :30	40.83
08 :43 :30	RC		(4, 13, 1)	0 days 00 :23 :30	39.17
08 :45 :30	RC		(8, 13, 1)	0 days 00 :21 :30	35.83
08 :48 :30	RC		(8, 7, 1)	0 days 00 :18 :30	30.83
08 :51 :30	RC		(14, 7, 1)	0 days 00 :15 :30	25.83
08 :52 :00	RC		(14, 7, 0)	0 days 00 :15 :00	25
08 :59 :00	PO		(14, 7, 0)	0 days 00 :29 :00	48.33
09 :02 :30	PO		(14, 0, 0)	0 days 00 :25 :30	42.5
09 :03 :00	A	8RFLKZQ3	(14, 0, 0)	0 days 00 :25 :00	41.67
09 :05 :30	A	8RFLKZQ3	(14, 5, 0)	0 days 00 :22 :30	37.5
09 :08 :30	A	8RFLKZQ3	(8, 5, 0)	0 days 00 :19 :30	32.5
09 :09 :30	A	8RFLKZQ3	(8, 5, 2)	0 days 00 :18 :30	30.83
09 :10 :00	A	8RFLKZQ3	(7, 5, 2)	0 days 00 :18 :00	30
09 :10 :30	RC		(7, 5, 2)	0 days 00 :17 :30	29.17
09 :11 :00	RC		(8, 5, 2)	0 days 00 :17 :00	28.33
09 :12 :00	RC		(8, 7, 2)	0 days 00 :16 :00	26.67
09 :15 :00	RC		(14, 7, 2)	0 days 00 :13 :00	21.67
09 :16 :00	RC		(14, 7, 0)	0 days 00 :12 :00	20

TABLE 7 – Extrait output planning drone 1

On constate que le drone enchaîne ses tâches le plus rapidement possible. On attirera l'attention du lecteur sur la charge réalisée entre 8h52 et 9h02. Elle dure 7 min et permet au drone d'augmenter son temps de vol de 14 min (ici la vitesse de charge est égale à 2).

Elle est parfaitement ajustée pour que la tâche suivante entraîne un retour charge au seuil de 20%. C'est ce qu'on pourra constater sur la dernière ligne de l'extrait (fin de retour charge en $(14,7,0)$).

6 Évitement

Cette partie est consacrée à la recherche algorithmique d'un trafic de drones optimisé en prenant en compte les collisions et les frôlements entre drones, ignorés jusqu'à présent. Le but est de trouver une solution admissible et optimale.

6.1 Stratégie pour la prise en compte des collisions

Précédemment, les collisions arrivant entre plusieurs drones n'ont pas été prises en compte lors de la création du planning pour chaque drone. La première étape avant d'écrire l'algorithme d'Évitement a donc été d'identifier les collisions entre drones à partir du planning fourni en fin de l'algorithme de Planification. Le planning est un dictionnaire contenant toutes les positions de chaque drone. Une ligne contient comme information le temps t , les coordonnées de la position du drone à t , la tâche en cours, l'id de l'objet traité et le pourcentage de la batterie. On ne possède pas la position de tous les drones à chaque instant directement à partir du planning. En effet, l'algorithme de planification décompose les trajectoires d'un drone et sous-mouvements rectilignes, chaque ligne du planning correspondant alors à la position à la fin d'une trajectoire rectiligne.

Sachant cela, on distingue deux types de collisions et on prends en compte les frôlement entre drones, que l'on extrait du planning. Les méthodes de détection sont implémentées dans le fichier Python *avoidance.py* du dossier Évitement. La figure ?? illustre les 3 phénomènes que l'algorithme d'évitement va détecter : **1- les collisions directes**, **2 - les collisions calculées**, **3- les frôlements**.

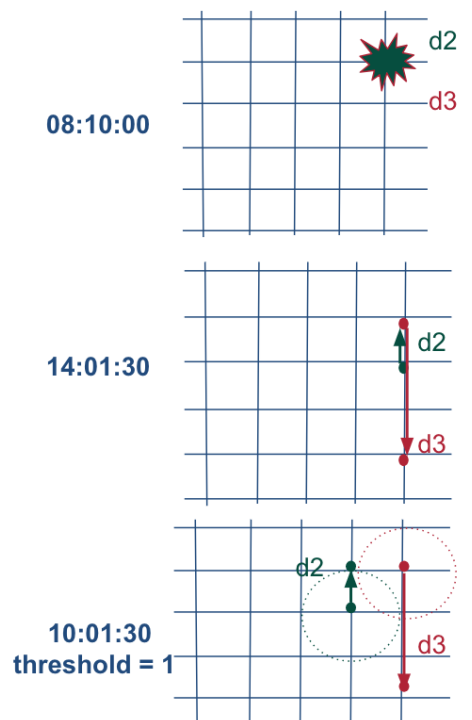


FIGURE 7 – Type de collisions détectées par l'algorithme d'Évitement

6.1.1 Les collisions directes

Lorsqu'un ou plusieurs drones se retrouvent exactement au même endroit et au même moment dans le planning et donc dans l'entrepôt, on parle de **collision directe**. Ces collisions sont identifiées par la fonction *count_direct_collisions*.

La sortie contient une ligne par collision avec le temps de la collision **time**, la position du lieu de la collision **position**, ainsi que les drones en jeu **drones** et leur nombre **drone_count**, comme on le voit avec cette sortie de terminal ?? :

Direct collisions:

	time	position	drone_count	drones
1	2025-03-24 08:01:00	(14, 2, 0)	3	[d1, d2, d3]
2	2025-03-24 08:01:30	(14, 2, 0)	3	[d1, d2, d3]
4	2025-03-24 08:02:00	(14, 3, 0)	2	[d2, d3]
5	2025-03-24 08:02:30	(13, 3, 0)	2	[d2, d3]
29	2025-03-24 08:18:00	(14, 2, 0)	3	[d1, d2, d3]
30	2025-03-24 08:18:30	(14, 2, 0)	3	[d1, d2, d3]
32	2025-03-24 08:19:00	(14, 3, 0)	2	[d1, d3]
33	2025-03-24 08:19:30	(13, 3, 0)	2	[d1, d3]
121	2025-03-24 10:01:00	(14, 2, 0)	2	[d1, d2]
123	2025-03-24 10:01:30	(14, 2, 0)	2	[d1, d2]
147	2025-03-24 10:10:00	(14, 2, 0)	2	[d1, d2]
149	2025-03-24 10:10:30	(14, 2, 0)	2	[d1, d2]
386	2025-03-24 12:40:00	(13, 4, 2)	2	[d1, d3]

6.1.2 Les collisions croisées

Il se peut que des drones entrent en collision lorsqu'ils effectuent leur trajectoires rectilignes, et donc qu'elle ne soit pas directement visibles dans le planning. La fonction *count_calculated_collisions* identifie ces collisions grâce à plusieurs étapes :

- Extraction des segments de trajectoires obtenues entre deux points spatio-temporels d'un drone.
- Filtrage des points spatio-temporels de 2 drones évoluant dans le même intervalle temporel et dont les trajectoires se croisent (croisement détecté grâce à la librairie **shapely**).
- Interpolation à chaque instant pour les point spatio-temporels filtrés pour connaître la position en utilisant notamment la vitesse du drone en paramètre.

Cette fonction fournit en sortie de terminal :

Calculated collision:

	start_time1	start_time2	collision_time	collision_position	drone1	drone2
2	2025-03-24 08:00:00	2025-03-24 08:00:30	(14, 1, 0)	d1	d2	
6	2025-03-24 08:17:00	2025-03-24 08:17:30	(14, 1, 0)	d1	d2	
14	2025-03-24 10:00:00	2025-03-24 10:00:30	(14, 1, 0)	d1	d2	
19	2025-03-24 10:09:00	2025-03-24 10:09:30	(14, 1, 0)	d1	d2	
27	2025-03-24 11:44:30	2025-03-24 11:45:20	(13, 2, 0)	d1	d3	
32	2025-03-24 13:51:30	2025-03-24 13:53:30	(9, 5, 0)	d1	d2	
37	2025-03-24 08:00:00	2025-03-24 08:00:30	(14, 1, 0)	d1	d3	
42	2025-03-24 08:17:00	2025-03-24 08:17:30	(14, 1, 0)	d2	d3	

79	2025-03-24	08:00:00	2025-03-24	08:00:30	(14, 1, 0)	d2	d3
93	2025-03-24	08:17:00	2025-03-24	08:17:30	(14, 0, 1)	d2	d3
97	2025-03-24	08:45:30	2025-03-24	08:48:20	(11, 5, 0)	d2	d3
110	2025-03-24	12:10:00	2025-03-24	12:10:30	(13, 2, 0)	d2	d3
112	2025-03-24	12:24:30	2025-03-24	12:25:00	(13, 3, 0)	d2	d3

En effet, les collisions sont totalement inenvisageables en pratique car elles mettent en péril le matériel (drones, colis), les infrastructures (étagères, tapis, ...) et la main d'œuvre sur site. De plus, les trajectoires des drones ne sont pas totalement contrôlées dans la réalité, et peuvent varier pour plusieurs raisons : déviation d'un drone endommagé/usé, ralentissement de la vitesse ou baisse d'altitude à cause d'un colis lourd, variations associées à la défaillance du contrôleur de vol du drone, ... On cherchera donc également à éviter à tout prix les frôlements entre drones, qui peuvent générer de collisions dans des conditions réelles.

6.1.3 Les frôlements

Pour cela, on introduit un paramètre **threshold** qui définit un rayon de distance de sécurité que chaque drone doit respecter pendant ses trajets avec les autres drones. La détection des frôlements a été implémentée avec la fonction `detect_near_misses()`. On suit la même logique que pour les collisions calculées, en interpolant les trajectoires rectilignes pour les segments de même intervalle temporel. Si la distance entre les points interpolés est inférieure à notre seuil de sécurité **threshold**, on identifie un frôlement. On obtient en sortie les positions des drones au moment du frôlement et le temps au moment du frôlement :

Near misses:

	time	drone1	drone2	pos1	pos2
0	2025-03-24 08:16:20	d1	d2	(14, 0, 0)	(14, 0, 1)
4	2025-03-24 08:40:20	d1	d2	(13, 0, 0)	(14, 0, 0)
7	2025-03-24 08:43:50	d1	d2	(14, 0, 0)	(14, 1, 0)
10	2025-03-24 08:57:50	d1	d2	(14, 0, 0)	(13, 0, 0)
14	2025-03-24 10:03:50	d1	d2	(12, 1, 1)	(11, 1, 1)
...
239	2025-03-24 10:41:20	d2	d3	(14, 0, 0)	(13, 0, 0)
243	2025-03-24 10:49:50	d2	d3	(14, 1, 0)	(14, 0, 0)
246	2025-03-24 11:13:50	d2	d3	(14, 0, 0)	(14, 3, 0)
250	2025-03-24 11:35:20	d2	d3	(13, 0, 0)	(14, 0, 0)
254	2025-03-24 11:44:20	d2	d3	(14, 1, 0)	(14, 0, 0)

NB : On a pris soin à chaque fois d'ignorer les collisions et frôlements ayant lieu exactement sur la station de recharge.

On peut maintenant identifier ces dangers dans le planning initial construit dans la première partie du projet. Il est crucial de les prendre en compte pour pénaliser les solutions initiales de notre problème d'optimisation. Pour cela, on introduit une **fonction coût** les prenant en compte. En effet, actuellement, les solutions initiales possèdent un bon temps d'exécution de la totalité des tâches, alors qu'elles ne respectent pas les contraintes de sécurité appliquées aux drones. Ce ne sont donc des solutions non admissibles.

6.2 Définition de la fonction coût

L'objectif de la fonction coût C est d'évaluer la qualité d'une solution de planning en prenant en compte les critères essentiels pour l'optimisation du trafic des drones dans l'entrepôt. L'objectif est de minimiser cette fonction afin d'obtenir une planification optimale du déplacement des drones. On la définit comme suit :

$$C = \sum_d \sum_i end_time_{d,i} - start_time_{d,i} + \lambda_1 * nb_collisions + \lambda_2 * nb_frolement + \lambda_3 * temps_total$$

Elle comprend quatre termes principaux :

- 1. **Durée d'exécution des tâches des drones** $\sum_d \sum_i end_time_{d,i} - start_time_{d,i}$: La somme des durées individuelles pour chaque tâche i réalisée par chaque drone d . L'objectif est de minimiser ce temps afin d'améliorer l'efficacité du processus.
- 2. **Nombre de collisions** $nb_collisions$: Ce terme pénalise fortement toute situation où deux drones se retrouvent au même endroit au même moment, ce qui est à éviter absolument.
- 3. **Nombre de frôlements** $nb_frolements$: Une pénalité est appliquée lorsque des drones se retrouvent trop proches les uns des autres, même sans collision, pour assurer la sécurité du système.
- 4. **Temps total d'exécution** $temps_total$: Ce terme mesure la durée totale de la journée pour traiter tous les colis. L'objectif est de minimiser cette durée pour maximiser l'efficacité du processus.

Les coefficients λ_1 , λ_2 , λ_3 permettent d'ajuster l'importance relative de chaque critère dans l'optimisation. Dans notre cas, on choisit λ_1 très élevé par rapport au terme $\sum_d \sum_i end_time_{d,i} - start_time_{d,i}$ (au moins 10 fois plus élevé) pour fortement pénaliser les collisions, qui sont totalement rédhibitoires. Ainsi, l'algorithme de recuit a de fortes chances d'éliminer les solutions non-admissibles contenant des solutions. De même, on choisira λ_2 assez élevé.

En résumé, on prendra :

$$\lambda_1 \gg \lambda_2 \gg \lambda_3 > 1$$

Ainsi, l'objectif global est d'obtenir un planning optimal des mouvements des drones, garantissant sécurité, fluidité et efficacité dans l'entrepôt.

6.3 Recherche d'une solution optimale admissible

6.3.1 Parcours de solutions dans l'espace de recherche

L'objectif de la partie Évitement au sein de ce projet est d'améliorer progressivement la solution initiale afin d'**éliminer les collisions et les frôlements** tout en **minimisant le temps total nécessaire au traitement des colis**. Pour cela, l'algorithme d'évitement explore l'espace des solutions de manière efficace. Au sein de celui-ci, des **stratégies**

sont implémentées pour réajuster la planification et se rapprocher de l'espace des solutions admissibles. Ces stratégies seront présentées ultérieurement.

Un **algorithme méta-heuristique** est une méthode d'optimisation approximative conçue pour explorer efficacement un large espace de solutions afin de trouver une solution optimale ou quasi-optimale à un problème complexe. Contrairement aux algorithmes exacts, qui garantissent de trouver la meilleure solution possible mais peuvent être trop coûteux en temps de calcul, les méta-heuristiques privilégient une recherche exploratoire et adaptative, souvent basée sur des stratégies inspirées de phénomènes naturels. Ici, l'algorithme de **recuit simulé**, une méta-heuristique de recherche locale, sera utilisé pour la recherche de solution répondant au problème d'Évitement. La figure 8 schématise le parcours de l'espace des solutions par l'algorithme méta-heuristique.

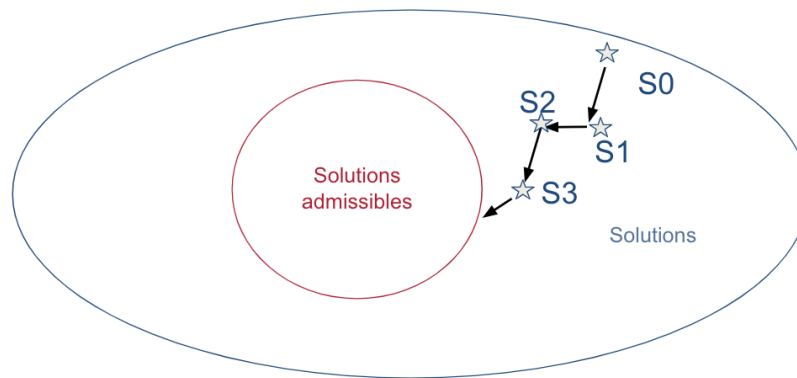


FIGURE 8 – Parcours de l'espace de recherche des solutions

On implémente dans cette optique un **algorithme méta-heuristique** capable de rechercher une solution optimale par itération tout en évitant les pièges des minima locaux.

6.4 Algorithme du recuit simulé

L'algorithme du recuit simulé est une méthode d'optimisation inspirée du processus physique de recuit, qui consiste à chauffer un matériau puis à le refroidir progressivement afin d'atteindre un état d'énergie minimale. Cette approche permet d'explorer efficacement l'espace des solutions en acceptant temporairement des solutions sous-optimales pour éviter les minima locaux.

Dans notre cas, l'objectif est de minimiser le coût total des déplacements des drones tout en réduisant les collisions et en optimisant le temps de complétion des tâches. L'algorithme suit les étapes suivantes :

Algorithm 4 Algorithme du recuit simulé appliqué à l'optimisation du planning des drones

Température initiale T_0 ,température finale T_f ,coefficient de refroidissement α ,

planning initial

Planning optimisé avec un coût minimal

Étape 1 : Initialisation ; $T \leftarrow T_0$; $currentPlanning \leftarrow planning$; $currentCost \leftarrow computeCost(currentPlanning)$; $bestPlanning \leftarrow currentPlanning$;**Étape 2 : Boucle de refroidissement ;****while** $T > T_f$ **do****Exploration des solutions voisines ;****for** $i = 1$ to $iterationsPerTemp$ **do** $newPlanning \leftarrow makeNewPlanning(currentPlanning)$; $newCost \leftarrow computeCost(newPlanning)$;**Critère d'acceptation****if** $newCost < currentCost$ **ou** $e^{-\frac{(newCost - currentCost)}{T}} > rand(0, 1)$ **then** $currentPlanning \leftarrow newPlanning$ $currentCost \leftarrow newCost$ **if** $newCost < computeCost(bestPlanning)$ **then** $bestPlanning \leftarrow newPlanning$ **Refroidissement** $T \leftarrow \alpha T$ **return** $bestPlanning$;

L'implémentation de cet algorithme suit cette logique dans la fonction `simulated_annealing`, qui assure la recherche d'un planning optimisé tout en explorant l'espace des solutions de manière efficace.

6.5 Élaboration des stratégies

6.5.1 Stratégie n°1

Dans cette première stratégie d'optimisation, l'idée principale est d'ajuster les temps de passage des drones pour éviter les collisions sans bouleverser l'organisation générale du planning. Pour ce faire, l'algorithme commence par analyser les conflits présents dans le planning actuel et sélectionne aléatoirement un type de collision à résoudre, qu'il s'agisse de collisions directes, de croisements de trajectoires ou de frôlements.

Une fois un conflit identifié, le système doit choisir quel drone modifier. Cette sélection se fait en prenant en compte le niveau de batterie des drones concernés. Plutôt que d'imposer un retard à un drone au hasard, on privilégie celui ayant la batterie la plus faible, de manière à éviter d'aggraver inutilement la gestion énergétique de la flotte.

Une fois le drone à modifier sélectionné, l'ajustement du planning est réalisé en utilisant la fonction `push_back_transit_times`. L'algorithme commence par identifier les deux

dernières visites du drone à la station de charge : l'une avant la collision et l'autre après. Tous les déplacements intermédiaires du drone entre ces deux points sont ensuite retardés de quelques minutes. Ce décalage permet d'éviter la collision tout en garantissant que le drone ne se retrouve pas hors de sa capacité énergétique prévue.

Après cette modification, le coût du planning ajusté est recalculé. Si le nouvel arrangement est plus optimal que l'ancien, il est directement adopté. Dans le cas contraire, il peut encore être accepté selon le critère de Métropolis, permettant ainsi d'explorer des solutions variées et d'éviter de rester coincé dans un optimum local inefficace. Cette stratégie apporte une flexibilité intéressante à l'optimisation et permet d'améliorer progressivement la fluidité du trafic des drones tout en minimisant les retards et les perturbations globales.

6.5.2 Résultat de la stratégie n°1

Drone Speed	Collision Penalty	Avoidance Penalty	T_init	T_freeze	alpha_T	k
1.0	100	50	1000	0.1	0.95	1

TABLE 8 – Paramètres et résultats pour `intermediate_warehouse_v2`

6.5.3 Visualisation de la stratégie n°1

6.5.4 Stratégie n°2

Une seconde approche a été explorée pour éviter les conflits entre drones : l'utilisation de la fonction `bypass_obstacle`. Contrairement à la stratégie précédente, qui agit principalement sur le décalage temporel des trajectoires, cette approche cherche à modifier directement les itinéraires des drones en leur faisant contourner l'obstacle responsable de la collision.

Dans cette stratégie, lorsqu'une collision est détectée, l'algorithme sélectionne un des drones impliqués et lui fait suivre un itinéraire alternatif temporaire. Cette modification consiste généralement à insérer de nouveaux points de passage dans la trajectoire du drone, en déplaçant légèrement sa trajectoire latéralement avant de lui permettre de reprendre son chemin normal après avoir évité l'obstacle. L'idée sous-jacente est d'introduire une marge de sécurité dans l'espace pour empêcher l'interférence avec un autre drone.

Cependant, après quelques tests, cette approche a montré des limites en termes de performance. En effet, bien qu'elle permette de résoudre certains conflits, elle allonge considérablement les temps de trajet des drones, ce qui peut entraîner une détérioration de l'efficacité globale du système. De plus, en introduisant des détours, elle peut créer de nouveaux conflits imprévus avec d'autres drones, nécessitant alors des ajustements supplémentaires.

Pour ces raisons, cette méthode n'est pas actuellement utilisée dans notre optimisation principale. Toutefois, elle pourrait être améliorée dans le futur, notamment en explorant des heuristiques plus intelligentes pour choisir les détours ou en combinant cette stratégie

avec une gestion plus fine des priorités des drones. Avec une implémentation plus efficace, elle pourrait devenir une alternative intéressante pour certaines configurations spécifiques où les autres méthodes de résolution ne seraient pas adaptées.

7 Perspectives du projet

8 Conclusion

Références

- [1] Denis BEREZIAT, Sonia CAFERI et Andrija VIDOSAVLJEVIC. “Metropolis II : Centralised and strategical separation management of UAS in urban environment”. In : *SESAR Innovation Days* (2022).
- [2] Julien GUITTON et Jean-Loup FARGES. *Planification de mission pour un ensemble de drones de combat aériens*. Rapp. tech. ONERA, 2007.
- [3] H. LEE, S. KIM et J. PARK. “PSO-Based Trajectory Optimization for Drones in Dynamic Indoor Environments”. In : *International Journal of Advanced Robotic Systems* (2023).
- [4] Gauthier PICARD. “Coordination de trajectoires 4D par optimisation distribuée dans la gestion du trafic aérien sans pilote”. In : *Conférence Nationale en Intelligence Artificielle*. 2022.
- [5] A. SMITH, B. BROWN et D. CLARK. “Multi-Drone Mission Planning in Indoor Environments Using Genetic Algorithms”. In : *Journal of Autonomous Systems* (2023).
- [6] FAA-NASA UTM Research Transition TEAM. *UAS Traffic Management Conflict Management Model*. Rapp. tech. 2022.
- [7] Y. ZHANG, L. WANG et X. CHEN. “Deep Learning-Based Obstacle Detection for Indoor Drone Navigation”. In : *IEEE Robotics and Automation Letters* (2023).

9 Annexes