MINISTÈRE DE l'ENSEIGNEMENT SUPÉRIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ MOULOUD MAMMERI, TIZI-OUZOU

FACULTÉ DES SCIENCES, DÉPARTEMENT DE MATHÉMATIQUES

Mémoire De Master

Filière : Mathématiques Appliquées

Spécialité : Recherche Opérationnelle

Présenté par :

Katia BELFODIL

Thème

Optimisation des niveaux de vol des drones pour éviter toute collision

Devant le jury d'examen composé de :

Mme. Leslous Fadila	Maître de conférences(B) UMMTO	**Encadrante**
Mme. Aumorassi Faroudja	Maître Assistant(A) UMMTO	**Présidente**
Me. Goubi Mouloud	Maître de conférences(A) UMMTO	**Examinateur**

Soutenue : le 30/06/2024

Table des matières

Table des matières **1**

Introduction générale **3**

1 Programmation Linéaire Mixte en Nombres Entiers (PLMNE) **7**
1.1 Introduction 7
1.2 Définition d'un problème linéaire en nombres entiers (PLNE) 8
1.3 Formulation 8
1.3.1 Forme standard 9
1.4 Modélisation d'un Problème Linéaire en Nombres Entiers (PLNE) . 9
1.4.1 Problème du voyageur de commerce 10
1.5 La Complexité 12
1.6 Relaxation linéaire continue 12
1.7 Méthodes de résolution 13
1.7.1 Méthodes Exactes 13
1.7.2 Méthodes Approchées 20
1.7.3 Les heuristiques 21
1.8 Méthode hybride pour la résolution des problèmes linéaires en nombres entiers 24
1.8.1 Les méta-heuristiques hybrides 25
1.8.2 Les méthodes hybrides 25
1.8.3 Présentation de la méthode hybride 25
1.8.4 Heuristique d'arrondissement 27
1.9 Définition d'un Problème Linéaire Mixte PLM 28
1.9.1 Formulation 28

2 Introduction à la Théorie des Graphes **31**
2.1 Introduction 31

2.2 Histoire 31
2.3 Définitions et Propriétés 33
2.3.1 Graphe 33
2.3.2 Graphe non-orienté 33
2.3.3 Graphe orienté 34
2.3.4 Voisins, Prédécesseurs, Successeurs 35
2.3.5 Le Degré 36
2.3.6 Chaine, Cycle, Chemin, Circuit 37
2.3.7 Cycle hamiltonien et cycle eulerien 39
2.4 Graphes particuliers 41
2.5 Les Sous graphes 45
2.5.1 graphe partiel 45
2.5.2 Sous graphe induit 45
2.6 La Connexité 46
2.6.1 Notion de connexité 46
2.6.2 Les composantes connexes 46
2.6.3 Connexité des graphes orientés 46
2.7 La représentation matricielle d'un graphe 47
2.7.1 Matrice d'adjacence 47
2.7.2 Matrice d'incidence 47
2.7.3 Les avantages et les inconvénients de la représentation matricielle 48
2.8 Arbre et arborecsence 48
2.9 Problème du plus court chemin 49
2.9.1 Algorithme de Dijkstra 51
2.9.2 Algorithme A* 53
2.9.3 Algorithme de Bellman-Ford 54

3 Optimisation des niveaux de vol des drones 57
3.1 Introduction 57
3.2 Préliminaires 58
3.3 Présentation du problème 59
3.4 Conception de trajectoire 4D 60
3.4.1 Génération des chemins horizontaux alternatifs 61
3.4.2 Détection des pertes de séparation potentielles 62
3.4.3 Calcul du temps de séparation 64

3.5 Modélisation du problème 68
3.6 Approche de solution 71
3.6.1 Optimisation des niveaux de vol 71
3.6.2 Relaxation des contraintes 72
3.7 Résultats Expérimentaux 73

Bibliographie **77**

Introduction générale

La **Recherche Opérationnelle** est une discipline scientifique qui s'appuie sur des méthodes et techniques rationnelles pour analyser, modéliser et résoudre des problèmes complexes rencontrés dans divers domaines, tels que l'économie, l'industrie, la logistique et la gestion. Son objectif principal est de trouver la meilleure solution possible parmi un ensemble de choix, permettant ainsi d'optimiser les processus et les prises de décision.

Apparue pendant la Seconde Guerre mondiale pour répondre aux besoins militaires d'optimisation des ressources, la Recherche Opérationnelle a connu un développement fulgurant depuis lors. Son champ d'application s'est considérablement élargi, englobant aujourd'hui une multitude de secteurs d'activité.

L'optimisation constitue l'un des domaines fondamentaux de la Recherche Opérationnelle. Elle vise à identifier la solution la plus optimale face à une situation donnée, en tenant compte de divers objectifs et contraintes. Pour ce faire, la Recherche Opérationnelle s'appuie sur des outils mathématiques puissants et des algorithmes sophistiqués, permettant de traiter des problèmes complexes avec une grande efficacité.

Allant de la livraison de marchandises à la surveillance, en passant par la recherche et le sauvetage, les drones, également appelés systèmes d'aéronefs sans pilote (UAS), sont de plus en plus utilisés en milieu urbain et constituent aujourd'hui l'un des défis majeurs du XXIe siècle. Grâce à leur autonomie et à leur capacité à accéder à des zones difficiles d'accès, ils sont particulièrement intéressants pour de nombreuses applications. Par exemple, le transport d'organes d'un hôpital vers un autre, la surveillance des feux de forêts, et la livraison de colis, pour le

grand public, les drones sont utilisés pour leurs applications de loisirs ainsi que pour leurs capacités à réaliser des prises de vues spectaculaires. Cependant, avec l'augmentation exponentielle de la densité du trafic aérien, le risque de collisions des drones s'accroît considérablement, rendant la prévention indispensable.
Pour garantir la sécurité des drones ainsi que celle des personnes et des biens au sol, diverses techniques sont en cours de développement et d'implémentation pour relever ce défi. Parmi ces stratégies clés figurent l'ajustement de la vitesse en vol, le retardement de décollage, ainsi que l'optimisation des niveaux de vol des drones afin d'assurer une séparation (spatiale et temporelle) adéquate entre les drones en vol. L'objectif est de créer des algorithmes et des systèmes intelligents capables d'assigner dynamiquement des altitudes appropriées à chaque drone, en prenant en considération des éléments tels que leurs trajectoires, les limitations d'espace aérien et les zones de conflit potentielles. Cette complexité a ouvert la voie à l'application de la recherche opérationnelle.
Grâce à sa capacité à apprendre à partir de données et à prendre des décisions intelligentes, l'intelligence artificielle renforce davantage les compétences de la Recherche Opérationnelle en matière de prévention des collisions. Les algorithmes d'intelligence artificielle ont la capacité d'analyser en temps réel les données de trafic, de prédire les mouvements des drones et d'ajuster dynamiquement les niveaux de vol afin d'éviter les différends. Cette méthode dynamique assure la sécurité de l'espace aérien même dans des circonstances imprévisibles.
La structure de ce mémoire est composée de trois chapitres : le premier aborde la programmation linéaire en nombres entiers et mixtes, où nous exposons les concepts fondamentaux de la PLNE, une technique d'optimisation mathématique particulièrement adaptée pour résoudre des problèmes complexes tels que le problème d'optimisation des niveaux de vol.

Ensuite, dans le deuxième chapitre nous nous intéresserons à quelques notions fondamentales de la théorie des graphes, un cadre mathématique qui fournit un outil puissant pour visualiser et analyser les conflits potentiels entre les drones. La théorie des graphes offre une manière structurée et intuitive de représenter

les interactions complexes entre les drones, les restrictions d'espace aérien et les zones de conflit. En modélisant le trafic de drones sous forme de graphe, nous pouvons identifier et résoudre efficacement les conflits potentiels, garantissant ainsi un fonctionnement sûr et efficace des drones dans l'espace aérien partagé.
Enfin, le troisième chapitre se consacre à une présentation détaillée du problème d'optimisation des niveaux de vol des drones. Ce chapitre aborde les objectifs spécifiques du problème, les contraintes à prendre en compte et une modélisation du problème. Il met l'accent sur l'utilisation de la programmation linéaire en nombres mixtes (PLM) et de la théorie des graphes pour développer des algorithmes efficaces permettant de résoudre efficacement ce problème complexe.

Chapitre 1

Programmation Linéaire Mixte en Nombres Entiers (PLMNE)

1.1 Introduction

La programmation linéaire (PL) est un domaine central de l'optimisation mathématique qui permet de résoudre une grande variété de problèmes du monde réel.
En effet, résoudre un problème linéaire consiste à trouver la solution optimale pour une fonction objectif linéaire soumise à un ensemble de contraintes linéaires (système d'équations ou d'inéquations). Les variables de décision dans un problème de PL sont continues, c'est-à-dire qu'elles peuvent prendre n'importe quelle valeur réelle dans leur domaine de définition.

Cependant, dans de nombreux problèmes pratiques, les variables de décision n'ont de sens que si elles prennent des valeurs entières. C'est le cas de la programmation linéaire en nombres entiers (PLNE), qui permet de résoudre des problèmes où les éléments ne peuvent pas être fractionnés, par exemple, lorsqu'il s'agit de déterminer le nombre d'unités à produire, le nombre d'employés à affecter à une tâche, etc.
Lorsqu'on est amené à utiliser des variables de décision entières et réelles dans le même problème, on parle alors de la programmation linéaire mixte (PLM) qui est une généralisation de la PLNE qui permet de traiter des problèmes où certaines

variables de décision peuvent être continues et d'autres doivent prendre des valeurs entières. Cette capacité à combiner des variables entières et réelles augmente considérablement le nombre et la complexité des problèmes qu'on peut modéliser et résoudre dans divers domaines, notamment la logistique, la production et le transport. Cependant, cette complexité accrue se traduit par une difficulté de résolution des problèmes PLM, les classant dans la catégorie des problèmes NP-difficiles. Cela signifie qu'il n'existe pas d'algorithme connu capable de les résoudre de manière optimale en un temps polynomial pour des problèmes de grande taille.
Dans ce chapitre, nous allons d'abord définir et modéliser un problème linéaire en nombres entiers (PLNE). Ensuite, nous illustrerons un exemple pratique pour mieux comprendre son application (Le Problème du voyageur de commerce). Nous nous concentrerons par la suite sur les différentes méthodes de résolution de ce type de problème. Dans cette partie, nous aborderons les méthodes directes, telles que la séparation et évaluation et la méthode des coupes, en passant par la relaxation linéaire. Puis, nous nous pencherons sur les méthodes indirectes, notamment les heuristiques et les méta-heuristiques. Enfin, nous terminerons cette partie par les méthodes hybrides qui combinent les approches directes et indirectes. Pour conclure ce chapitre, nous définirons la programmation linéaire mixte (PLM) et illustrerons un exemple pratique de son utilisation.

1.2 Définition d'un problème linéaire en nombres entiers (PLNE)

Un PLNE est un problème d'optimisation, où l'on cherche à optimiser (maximiser, minimiser) une fonction linéaire dite *Fonction Objectif*, sous un ensemble de contraintes affines qui peut inclure des inégalités et des égalités ; et où les variables de décision ne peuvent prendre que des valeurs entières. [1]

1.3 Formulation

Un PLNE peut être représenté sous forme matricielle de la manière suivante : [27]

$$(\text{PLNE})\begin{cases} \quad Optimiser & z = c^t x \\ \text{sous. contraintes :} & Ax \leq b \\ & x \in \mathbb{N}^n \end{cases}$$

Où :

- $c \in \mathbb{Z}^n$: vecteur couts.
- $A_{m \times n}$: matrice de rang $r(A) = m \leq n$ tel que : n est le nombre de variables et m est le nombre de contraintes..
- $b \in \mathbb{Z}^n$: vecteur du second membre.
- $x \in \mathbb{N}^n$: vecteur des variables de décision.

Remarque 1.1. Un cas particulier de la PLE ou les variables sont tenues de ne prendre que la valeur 0 ou 1, on parle alors de la programmation linéaire en variables booléennes "PLB".

$$(\text{PLB})\begin{cases} \quad Optimiser & z = c^t x \\ \text{sous. contraintes :} & Ax \leq b \\ & x \in \{0,1\} \end{cases}$$

1.3.1 Forme standard

On dit que le PLNE est sous sa forme standard si la *fonction objectif* est à maximiser et si tous les contraintes (sauf celles de positivité des variables) sont des égalités. [4]

$$\begin{cases} \quad Maximiser & z = c^t x \\ \text{sous. contraintes :} & Ax = b \\ & x \in \mathbb{N}^n \end{cases}$$

1.4 Modélisation d'un Problème Linéaire en Nombres Entiers (PLNE)

La modélisation d'un PLNE est un outil puissant pour résoudre des problèmes d'optimisation dans divers domaines. Afin d'établir le modèle mathématique d'un problème concret on doit le mettre en équations pour cela on effectue trois

étapes : [27]

1. Identification des variables
   - Représenter les éléments du problème par des variables.
   - Déterminer les diverses catégories de variables (d'état, de décision, etc.)
2. Formulation des Contraintes
   - Définir les relations mathématiques entre les variables sous forme d'équations ou d'inéquations.
   - Exprimer les limites et les restrictions du problème.
3. Définition de la fonction objectif.
   - Déterminer l'objectif à atteindre (minimisation ou maximisation).
   - Exprimer l'objectif à atteindre sous forme d'une fonction linéaire avec les variables déterminées préalablement.

1.4.1 Problème du voyageur de commerce

Le problème de voyageur de commerce également connu sous le nom "TSP" (Travelling Salesman Problem), consiste à trouver le chemin le plus court qu'un représentant de commerce doit emprunter pour visiter un ensemble donné de villes exactement une fois chacune, puis revenir à son point de départ. Ce problème est l'un des problèmes les plus anciens et les plus étudiés en optimisation combinatoire.

Dans ce problème, nous avons un graphe orienté G=(X,U), où X est l'ensemble des sommets représentant les villes à visiter, y compris la ville de départ, et U est l'ensemble des arcs représentant les chemins possibles entre les villes. Chaque arc (i,j)de U est associé à une distance $d_{i,j}$ de la ville i à la ville j. La longueur d'un chemin dans G est la somme des distances associées à ses arcs. Ainsi, le TSP revient à trouver un circuit hamiltonien (c'est-à-dire un chemin fermé passant exactement une fois par chaque sommet du graphe) de longueur minimale dans G.

Il convient de noter que dans le cas où les distances entre deux villes ne sont pas symétriques (c'est-à-dire $d_{i,j} \neq d_{j,i}$ pour certains arcs (i,j) de U), on parle de TSP

asymétrique.

Ce problème a de nombreuses applications pratiques, notamment dans le domaine de la planification de processus de fabrication, d'optimisation de trajets en robotique et de résolution de problèmes de transport. En effet, certains problèmes plus complexes, tels que les problèmes de transport, peuvent être modélisés comme des instances de TSP, même s'ils présentent une structure sous-jacente plus complexe.

Une manière équivalente de formuler le TSP consiste à associer à chaque paire de villes à visiter (notées i et j, avec i=1 à n et j=1 à n où $i \neq j$) une distance $\delta_{i,j}$ égale à $d_{i,j}$ s'il existe un chemin direct de i à j (c'est à dire si $(i,j) \in U$ dans G), et fixée à ∞ autrement. De plus, une variable binaire de succession $x_{i,j}$ est introduite. Cette variable prend la valeur 1 si la ville j est visitée immédiatement après la ville i dans la tournée, 0 sinon. Ainsi le TSP peut être de la manière suivante : [4]

$$\text{Minimiser} \quad \sum_{i=1}^{n}\sum_{j=1}^{n} \delta_{i,j} \cdot x_{i,j}$$

$$\text{s.c.} \quad \sum_{j=1}^{n} x_{i,j} = 1 \quad \forall i = 1..n$$

$$\sum_{i=1}^{n} x_{i,j} = 1 \quad \forall j = 1..n$$

$$\sum_{i \in S, j \notin S} x_{i,j} \geq 2 \quad \forall S \subset X, S \neq \emptyset$$

$$x_{i,j} \in \{0,1\} \quad \forall i = 1..n, \forall j = 1..n$$

Les deux premières contraintes traduisent le fait que chaque ville doit être visitée exactement une fois ; la troisième contrainte interdit les solutions composées de sous-tours disjoints, elle est généralement appelée contrainte d'élimination des sous-tours.

1.5 La Complexité

Résoudre un problème de la programmation linéaire en nombre entier (PLNE),mixte(PLM) ou binaire (PLVB) est bien plus ardu que résoudre un problème de programmation linéaire en variables continues (PL). [19]
L'analyse initiale du problème (PL) nous a rapidement révélé que :

1. Pour obtenir la solution optimale, il n'est pas nécessaire d'examiner un nombre infini de solutions. En effet, seuls les sommets du polyèdre $P = x \in \mathbb{R}^n : Ax \leq b, x \geq 0$ sont pertinents.
2. Ces sommets peuvent être caractérisés de manière simple, ce qui facilite leur détermination.

En revanche, dans les contextes de programmation linéaire en variables entières (PLNE), mixtes (PLM) ou binaires (PLVB), la solution optimale n'est généralement pas un sommet de P, mais plutôt un point intérieur ou situé n'importe où sur la frontière. Par conséquent, toute caractérisation spécifique de la solution optimale est perdue, ce qui rend sa détermination bien plus difficile. Le problème de la Programmation Linéaire en Nombres Entiers (PLNE) est un défi bien connu en informatique et en optimisation combinatoire. Classé parmi les problèmes NP-difficiles, sa résolution est réputée ardue même pour des instances de taille modérée. En effet, la nature combinatoire de ce problème rend la recherche d'une solution optimale complexe, car elle nécessite l'exploration d'un espace de solutions potentiellement vaste.

1.6 Relaxation linéaire continue

La relaxation linéaire est une technique utilisée en programmation mathématique. Elle consiste à relâcher les contraintes du problème linéaire en nombres entiers (PLNE) en permettant aux variables entières de prendre des valeurs fractionnaires, transformant ainsi le problème en un problème linéaire continu (PL) plus facile à résoudre. [16]

Remarque 1.2.

Toutes les solutions de (PLNE) sont des solutions de (PL). La réciproque est fausse La valeur optimale de (PL) constitue une borne inférieure pour le (PLNE) correspondant (cas de minimisation).
La relaxation linéaire ne donne pas seulement une borne supérieure mais par fois la solution optimale si celle ci est entière.

Lemme 1.1. *[11]*
Si x^* est une solution optimale de la relaxation linéaire et si on a de plus $x^* \in \mathbb{Z}^n$,alors x^* est une solution optimale du PLNE.

.

1.7 Méthodes de résolution

Contrairement à la programmation linéaire (PL), la solution optimale de la programmation linéaire en nombres entiers (PLNE) n'est pas nécessairement située sur la frontière de son polyèdre (D). Ainsi l'ensemble des solutions du PLNE est contenu dans (D). [17] Les algorithmes de PLNE sont basés sur l'exploitation du succès calculatoire considérable de la PL. La stratégie de ces algorithmes implique trois étapes

1. Relâcher l'espace des solutions du PLNE en supprimant la restriction entière sur toutes les variables entières et en remplaçant toute variable binaire y par la plage continue $0 \leq y \leq 1$. Le résultat du relâchement est un PL régulier.
2. Résoudre le PL et identifier son optimum continu.
3. À partir du point d'optimum continu, ajouter des contraintes spéciales qui modifient itérativement l'espace de solution du PL de manière à ce qu'elle rende finalement un point extrême optimal satisfaisant les exigences entières.

1.7.1 Méthodes Exactes

Il existe diverses méthodes pour obtenir une solution optimale à un PLNE. Parmi celles-ci, les techniques arborescentes sont souvent mentionnées, car elles

permettent d'explorer exhaustivement tout l'espace de recherche par énumération. Ce processus global devient particulièrement intéressant lorsqu'il est amélioré en utilisant des techniques de détection des régions sous-optimales, ce qui permet de les écarter de la recherche, améliorant ainsi l'efficacité de l'algorithme

1.7.1.1 Séparation et évaluation progressive (Branch and Bound)

"Diviser pour mieux régner"

La méthode Branch and Bound est une approche générique de résolution des problèmes d'optimisation combinatoire considérés comme difficiles à résoudre efficacement en réduisant l'effort de recherche. Cette méthode comporte deux phases : **le branchement :** qui consiste à séparer le problème original en sous problèmes plus simples séparant ainsi l'espace de recherche en plusieurs sous espaces, et la phase de **l'évaluation** des solutions des solutions des sous problèmes obtenues.
Cette méthode est aussi appelée méthode arborescente ou chaque problème corresponds à un noeud ou bien sommet de l'arbre des solutions et le problème initial représente sa racine.
Afin de résoudre un PLNE par la méthode de branch and bound on effectue les étapes suivantes :
On résout d'abord la relaxation linéaire du problème, si la solution optimale du problème relaxé satisfait les contraintes d'intégralité, alors elle est optimale pour le PLNE, sinon il doit exister au moins une variable x_j dont la valeure α est fractionnaire. On passe alors à l'étape de branchement ou l'on sépare le problème relaxé en deux sous problèmes, un contenant la contrainte $x_j \leq \alpha$ et un second $x_j \geq \alpha = \alpha+1$, en répétant le processus pour chacun des sous-problèmes on obtient l'arbre de solutions. Lorsqu'une solution est contradictoire avec les contraintes du problème initiale on tronque sa branche.
La performance d'un arbre de séparation et d'évaluation est directement liée à sa taille. Généralement, plus sa taille est réduite, plus l'approche est efficace. Cependant, même pour des arbres relativement grands, l'approche peut être efficace si les traitements effectués aux nœuds ne nécessitent pas de longs temps de calcul. Pour réduire la taille de l'arbre, des techniques d'évaluation de solutions permettent de

déduire que certaines branches sont non prometteuses. Un mécanisme d'élagage peut ensuite être appliqué pour couper les branches infructueuses. Le calcul de la borne inférieure est essentiel à chaque nœud de l'arbre. Il s'agit de la valeur du meilleur coût que pourrait avoir toute solution développée à partir de ce nœud. Lorsque la valeur de la borne inférieure atteint ou dépasse la valeur de la meilleure solution réalisable rencontrée auparavant, aucune solution obtenue à partir de ce nœud ne peut correspondre à la solution optimale. Le mécanisme d'élagage peut alors opérer en coupant cette branche infructueuse pour réduire l'effort de recherche. La qualité de la borne inférieure dépend des techniques de calcul utilisées, mais il est crucial que le temps de calcul nécessaire ne soit pas excessif, car cela pourrait ralentir la construction de l'arbre et diminuer les performances globales du schéma de résolution. Ainsi, il est important de trouver un équilibre entre l'effort nécessaire pour calculer la borne et sa qualité. [8, 5]

1.7.1.1.1 Avantages de la méthode Branch and Bound

1. Efficacité : La méthode Branch and Bound est capable de trouver des solutions optimales à des problèmes complexes en un temps raisonnable.
2. Flexibilité : La méthode peut être adaptée à une grande variété de problèmes d'optimisation combinatoire.
3. Garantie de l'optimalité : La méthode garantit de trouver la solution optimale si l'exploration est complète.

Algorithme de Branch and Bound [8]

1. Initialisation
   - (a) Poser $z^* = -\infty$
   - (b) Appliquer le calcul de borne et les critères d'élagage à la racine
   - (c)Critere d'arrêt : s'il n'y a plus de sous-problèmes non élagués, arrêter
2. Branchement
   - (a) Parmi les sous-problèmes non encore élagués, choisir celui qui a été créer le plus récemment (s'il y a égalité, choisir celui de plus grande borne supérieure).

- (b) Appliquer le Test 1 : si le sous-problème est élagué, retourner en 2.
- (c) Brancher sur la prochaine variable entière à valeur non entière dans la relaxation PL.

3. Calcul de la borne : résoudre la relaxation linéaire de chaque sous-problème.
4. Elagage : élaguer un sous-problème si
   - (a) La borne superieure est inférieure ou égale à z^*
   - (b) La relaxation PL n'a pas de solution réalisable.
   - (c) Dans la solution optimale de la relaxation linéaire, toutes les variables entières sont à valeurs entières : si la borne supérieure est strictement supérieure à z^*, z^* est mise à jour et la solution de la relaxation linéaire devient la meilleure solution courante.
5. Retourner en 2

1.7.1.2 Méthode des coupes

Les méthodes de coupes sont une famille d'algorithmes pour résoudre des problèmes linéaires à variables entières. Elles sont basées sur l'idée de couper le polyèdre des solutions en rajoutant des contraintes qui éliminent les solutions fractionnaires. [27]

1.7.1.2.1 Les étapes

1. Résoudre la relaxation linéaire : On obtient une solution optimale du problème linéaire relaxé (PLR).
   Si la solution optimale de PLR est entière, alors c'est la solution optimale du PLE et le problème est résolu.
   Sinon la solution est fractionnaire

2. Ajouter des coupes : Identifier et ajouter des contraintes valides " coupes " qui excluent la solution optimale non entière tout en conservant toutes les solutions entières possibles.

3. Répéter les étapes 1 et 2 : Résoudre à nouveau le problème linéaire augmenté de cette coupe en utilisant le dual du simplexe et itérer le processus jusqu'à obtenir une solution entière optimale.

1.7.1.2.2 Avantages

1. **Efficacité :** Les méthodes de coupes peuvent s'avérer très efficaces pour résoudre des PLNE de grande taille.
2. **Flexibilité :** Différents types de coupes peuvent être utilisés et adaptés à des problèmes spécifiques.
3. **Garantie d'optimalité :** La méthode converge toujours vers une solution optimale entière.

1.7.1.2.3 Limites

1. **Complexité :** La génération de coupes peut être un processus complexe et gourmand en calcul.
2. **Taille du problème :** L'efficacité des méthodes de coupes peut être limitée pour des problèmes de très grande taille.

1.7.1.2.4 Inégalité valide pour un Programme Linéaire [27]

Définition 1.1.
Une inégalité $\pi' \cdot x \leq \pi_0$ est une inégalité valide pour $\mathbf{X} \subseteq \mathbb{R}^n$ si $\pi' \cdot x \leq \pi_0 \ \forall x \in \mathbf{X}$

Proposition 1.1.
Si $\pi' \cdot x \leq \pi_0 \ \forall x \in \mathbf{X}$ *est une inégalité valide pour* $\mathbf{X} = \{x : x \geq 0, A \cdot x \leq b\} \neq \emptyset$, *alors* $\exists u \geq 0$ *tels que* $u' \cdot A \geq \pi'$ *et* $u' \cdot b \leq \pi_0$

1.7.1.2.5 Inégalités valides pour un problème de PLE

Inégalités valides de Chvàtal-Gomory Soient $u \in \mathbb{R}^m_+$, $S = \mathbf{X} \cap \mathbb{Z}^n$, où $\mathbf{X} = \{x \in \mathbb{R}^n_+ : A \cdot x \leq b\}$, $A = (a_1, a_2, \ldots, a_n)$ une matrice d'ordre $m \times n$:
(1) L'inégalité suivante est valide pour $\mathbf{X}$: $\sum_{j=1}^{n} u' a_j x_j \leq u'b$, car $u \geq 0$ et $\sum_{j=1}^{n} a_j x_j \leq b$.
(2) L'inégalité suivante est valide pour $\mathbf{X}$:

$$\sum_{j=1}^{n} [u' a_j] x_j \leq u'b$$

car $u \geq 0$.

(3) L'inégalité suivante est valide pour S :

$$\sum_{j=1}^{n} [u' a_j] x_j \leq [u'b],$$

car $x \in \mathbb{N}^n$ donc $\sum_{j=1}^{n} [u' a_j] x_j$ est entière.

Un résultat intéressant est que toute inégalité valide d'un problème de PLE peut être obtenue de cette façon.

Théorème. *Si $\pi' \cdot x \leq \pi_0$ est une inégalité valide pour S avec π et π_0 entiers, alors elle s'obtient en un nombre fini d'itérations par la procédure de Chvàtal-Gomory. Ce résultat est soumis au bon choix de l'inégalité valide à chaque itération.*

1.7.1.2.6 Coupes de Gomory [27]

La coupe de Gomory est l'une des coupes les plus analysées et employées, spécialement conçue pour s'adapter à la méthode du simplexe (et dual du simplexe).
Son processus de détermination peut être expliqué de manière concise comme suit.
Soit un problème PLE sous sa forme standard :

$$\begin{cases} \text{Max } z(x) = c'x \\ \text{S.C. } Ax = b \\ x \geq 0, \quad x_j \text{ entier} \quad \forall j = 1, \ldots, n. \end{cases}$$

Soit PLR la relaxation du PLE

$$\begin{cases} \text{Max } z(x) = c'x \\ \text{S.C. } Ax = b, \\ x_j \geq 0, \qquad \forall j = 1, \ldots, n. \end{cases}$$

Ponsons pour une base J_B (avec $|J_B| = m$) et une solution réalisable :

$$Ax = A_B x_B + A_N x_N = b$$

où $x_B = (x_j, j \in J_B)$ et $x_N = (x_j, j \in J_N)$, $J = J_B \cup J_N = \{1, 2, \ldots, n\}$
puisque A_B est régulière,

$$x_B + A_B^{-1} A_N x_N = A_B^{-1} b$$

Posons :
$\overline{A} = A_B^{-1} A_N$ et $\overline{b} = A_B^{-1} b$.
On a donc :

$$x_B + \overline{A} x_N = \overline{b}$$

où $x_i + \sum_{j \in J_N} [\overline{a}_{ij}] x_j = \overline{b}_i, (i \in J_B), \overline{A} = (\overline{a}_{ij}, i \in J_B, i \in J_N)$ et $\overline{a}_{ij} = [\overline{a}_{ij}] + \{\overline{a}_{ij}\}$,
tels que :
$[\overline{a}_{ij}]$: le plus grand entier inférieur à $\overline{a}_{ij}$
$\{\overline{a}_{ij}\}$: la partie fractionnaire de $\overline{a}_{ij}$ telle que $0 \leq \{\overline{a}_{ij}\} \leq 1$.
Il s'ensuit :

$$x_i + \sum_{j \in J_N} ([\overline{a}_{ij}]\{\overline{a}_{ij}\}) x_j = [\overline{b}_i] + \{\overline{b}_i\}, i \in J_B \Leftrightarrow x_i + \sum_{j \in J_N} [\overline{a}_{ij}] x_j - [\overline{b}_i] = \{\overline{b}_i\} - \sum_{j \in J_N} \{\overline{a}_{ij}\} x_j, i \in J_B \tag{1.1}$$

soit $\overline{x}$ une solution réalisable de (PLR) à valeurs entières. L'égalité (1.1) vaut pour $\overline{x}$ le coté gauche étant à valeurs entières. De plus :

$$\overline{x}_i + \sum_{j \in J_N} [\overline{a}_{ij}] \overline{x}_j \leq \overline{x}_i + \sum_{j \in J_N} \overline{a}_{ij} \overline{x}_j = \overline{b}_i, i \in J_B \tag{1.2}$$

Comme la partie gauche de (1.2) est à valeurs entières alors :

$$\overline{x}_i + \sum_{j \in J_N} [\overline{a}_{ij}]\overline{x}_j \leq [\overline{b}_i], i \in J_B$$

comme la formule (1.1) est valable pour toute solution réalisable, il s'ensuit que :

$$\{\overline{b}_i\} - \sum_{j \in J_N} \{\overline{a}_{ij}\}\overline{x}_j \leq 0, i \in J_B$$

d'où l'expression de la coupe de Gomory :

$$-\sum_{j \in J_N} \{\overline{a}_{ij}\}x_j \leq -\{\overline{b}_i\}, (i \in J_B).$$

Avec cette coupe, on n'élimine pas de solution réalisable à valeurs entières de l'ensemble des solutions réalisables.

1.7.2 Méthodes Approchées

On a vu précédement que les problèmes de programmation linéaire en nombres entiers (PLNE) appartiennent à la classe des problèmes NP-Difficiles, ce qui signifie qu'il est pratiquement impossible de trouver une solution optimale pour des problèmes de taille importante en un temps raisonnable. Dans la plupart des cas, il est plus important de trouver une solution acceptable rapidement que de trouver la solution optimale après un long délai. En effet, les besoins et les conditions peuvent changer rapidement dans le monde réel, et une solution optimale peut devenir obsolète avant même d'être trouvée.
Les algorithmes approchés sont donc largement utilisés dans la pratique pour résoudre les problèmes de la PLE. Ils peuvent être utilisés pour trouver une solution initiale acceptable, ou pour réduire l'espace de recherche et faciliter la recherche d'une solution optimale.

1.7.3 Les heuristiques

1.7.3.0.7 Définition Une heuristique est une méthode algorithmique de résolution de problèmes qui s'appuie sur des règles générales ou des procédures intuitives plutôt que sur un modèle mathématique rigoureux. Elle permet d'obtenir une solution approchée, mais souvent satisfaisante, à des problèmes complexes qui seraient trop difficiles à résoudre par des méthodes exactes.

Contrairement aux algorithmes classiques, les heuristiques ne garantissent pas toujours une solution optimale, c'est-à-dire la meilleure solution possible. Cependant, elles offrent l'avantage d'être souvent plus rapides et plus faciles à implémenter.

De nombreuses heuristiques existent et sont utilisées dans divers domaines, comme l'informatique, l'ingénierie et la gestion de projet. Certaines heuristiques sont conçues pour résoudre un problème spécifique en tenant compte de ses particularités, tandis que d'autres peuvent être appliquées à une large gamme de problèmes.

L'utilisation des heuristiques est un domaine de recherche actif, et de nouvelles heuristiques sont constamment développées pour améliorer l'efficacité de la résolution de problèmes complexes. [27, 26]

1.7.3.0.8 Enjeux des heuristiques L'enjeu central de la modélisation de problèmes réels réside dans la recherche d'un équilibre entre la précision du modèle et sa simplicité. Un modèle trop simpliste risque de ne pas capturer les nuances et les complexités du problème réel, conduisant à des solutions inefficaces ou erronées. À l'inverse, un modèle excessivement complexe peut devenir difficile à manier, voire impossible à résoudre, ce qui entrave l'obtention d'une solution concrète.
Face à ces défis, les heuristiques offrent une solution prometteuse. Les heuristiques présentent plusieurs avantages par rapport à la modélisation exacte :

– Simples à mettre en oeuvre : Les heuristiques sont généralement plus faciles

à comprendre et à implémenter que les méthodes d'optimisation exactes
- Amélioration des résultats : Bien que les heuristiques ne garantissent pas la solution optimale, elles peuvent souvent fournir des solutions de bonne qualité, proches de l'optimum.
- Résultat rapide : Les heuristiques peuvent trouver des solutions plus rapidement que les méthodes d'optimisation exactes, ce qui est particulièrement important pour les problèmes en temps réel.
- Robustesse : Les heuristiques sont généralement moins sensibles aux variations des données et des paramètres du problème que les méthodes d'optimisation exactes.
- Capacité à être combiné avec d'autres méthodes : Les heuristiques peuvent être facilement combinées avec des méthodes d'optimisation exactes pour améliorer les performances globales.

1.7.3.0.9 Les Types des Heuristiques [26]

Les heuristiques sont dépendantes du problème à résoudre. Principalement deux types sont utilisées :

1. Les heuristiques de construction (par exemple les méthodes gloutonnes), qui construisent itérativement une solution :
   - Des solutions générées aléatoirement.
   - Méthodes constructives.
2. Les heuristiques de descente, qui à partir d'une solution donnée cherchent un optimum local :
   - Méthodes de recherche locale.
   - Méthodes approximatives.

Il est important de noter que certaines heuristiques peuvent appartenir à plusieurs catégories, car elles combinent différentes techniques.
Lors de la résolution d'une classe spécifique de problèmes, il peut être bénifique de combiner plusieurs approches heuristique afin de trouver la meilleure solution.

Le choix des approches heuristiques dépends de :

- Le domaine de décision est stratégique, tactique ou opérationnel.
- La fréquence avec laquelle la décision est prise.
- Le temps de développement disponible.
- Les qualifications analytiques du décideur impliquées.
- La taille du problème.
- L'absence ou présence d'éléments stochastiques.

1.7.3.0.10 Evaluation des Heuristique Une heuristique efficace est conçue pour un problème précis et s'appuie sur les connaissances d'un expert dans le domaine. L'apprentissage de la construction d'heuristiques se fait par l'expérience accumulée sur de nombreux cas concrets. Les experts ont découvert plusieurs principes fondamentaux, connus sous le nom de méthodologie de construction heuristique, qui consistent principalement à développer des heuristiques et à vérifier progressivement leur adéquation avec le problème abordé. Cependant, il est difficile d'évaluer réellement une solution heuristique.

1.7.3.0.11 Méta-Heuristiques [26, 27]

Définition Les méta-heuristiques se distinguent comme des outils puissants et polyvalents. Elles représentent une méthode générique d'optimisation capable de traiter une large variété de problèmes sans nécessiter des modifications profondes de l'algorithme utilisé, permettant une adaptation aisée à des problèmes variés et une réutilisation efficace de modules d'algorithmes existants. En d'autres termes, les méta-heuristiques sont des algorithmes d'optimisation stochastique qui intègrent des techniques de recherche locale. Le terme "méta" souligne la capacité de ces algorithmes à combiner intelligemment les forces et faiblesses de plusieurs heuristiques pour résoudre un problème, orchestrant ainsi une optimisation globale. Les méta-heuristiques visent à trouver un optimum global en explorant l'espace de recherche et en identifiant les zones prometteuses, tout en évitant de rester

coincé dans des optima locaux. Les méta-heuristiques sont souvent hybridées avec d'autres méthodes de recherche opérationnelle pour améliorer leur efficacité et leur robustesse, en particulier pour des problèmes complexes ou stochastiques.

Méta-Heuristiques pour la PLNE Les principales méta-heuristiques utilisées pour résoudre des problèmes à variables discrètes sont :

- le recuit simulé : Etudier l'espace de recherche tout en acceptant la possibilité de dégrader la solution pour obtenir des optima locaux. Au cours de la procédure, le récuit tolère de moins en moins ces dégradations ce qui va le faire converger vers l'optimum global.
- la recherche avec tabous : qui est l'inverse du recuit simulé, elle est deterministe et a une notion de mémoire. Le choix du meilleur voisin d'une solution pousse l'algorithme à trouver les optima locaux ; et comme l'exploration de l'espace de recherche est effectué en limitant le voisinage de la solution en rendant « tabous » certains mouvements, l'algorithme doit théoriquement visiter l'optimum global.
- les algorithmes évolutionnaires(génétiques) : issus de la théorie de l'évolution de Darwin, qui manipulent plusieurs solutions en même temps, et qui en les combinant forment de nouvelles solutions. Le fait d'avoir une population de solutions facilite l'exploration de l'espace de recherche, et les meilleures solutions seront favorisées pour participer à la création de nouvelles solutions ce qui aura pour effet de favoriser les combinaisons des « bonnes caractéristiques », et donc de trouver un optimum global.

1.8 Méthode hybride pour la résolution des problèmes linéaires en nombres entiers

Au-delà des méthodes exactes et approchées, une troisième catégorie d'algorithmes de résolution des problèmes PLNE se distingue, c'est l'hybridation des méthodes. Cette approche consiste à combiner plusieurs méthodes pour en créer une nouvelle plus performante. L'objectif est d'exploiter les avantages de chaque approche afin d'améliorer les performances globales de l'algorithme. Par

exemple, une méthode hybride peut être formée en combinant des algorithmes exacts et/ou des algorithmes approchés. Une telle combinaison peut impliquer deux ou plusieurs méthodes exactes. Les méthodes hybridespeuvent être divisées en deux groupes : les métaheuristiques hybrides qui impliquent une combinaison de plusieurs métaheuristiques et les méthodes hybrides impliquant une combinaison d'une méthode exacte et d'une métaheuristique. [27]

1.8.1 Les méta-heuristiques hybrides

Le recuit simulé, la recherche avec tabous et les algorithmes évolutifs ont déjà été hybridés avec succès dans plusieurs applications.On peut classifier ces différentes hybridations selon la taxonomie, cette classification permet de comparer les métaheuristiques hybrides de façon qualitative. La taxonomie comporte deux aspects. Une classification hiérarchique permet d'abord d'identifier la structure de l'hybridation. Ensuite, une classification générale spécifie les détails des algorithmes impliqués dans l'hybridation.

1.8.2 Les méthodes hybrides

Il est possible d'hybrider de plusieurs façons une méthode exacte avec une métaheuristique.Une classification axée sur l'hybridation de méthodes exactes et approximatives a été proposée. On divise les méthodes hybrides en deux catégories : les hybridations collaboratives et celles intégratives. Les algorithmes qui échangent des informations de façon séquentielle, parallèle ou entrelacée entrent dans la catégorie des hybridations collaboratives. Les algorithmes à hybridation intégrative font en sorte qu'une technique est une composante incorporée à une autre technique. Autrement dit, il y a un algorithme maître et un algorithme esclave.

1.8.3 Présentation de la méthode hybride

Associer deux méthodes n'est pas une tâche simple. Il est nécessaire de déterminer le but de cette hybridation et de répondre à des questions telles que : Comment procéder à cette combinaison ? Quel objectif cherche-t-on à atteindre ? Cette démarche implique l'utilisation des avantages propres à chacune

des méthodes envisagées.
Nous allons présenter ici une hybridation entre l'algorithme Branch and bound et l'heuristique d'arrondissement. Si la méthode du Branch and Bound garantit une solution optimale, elle peut s'avérer lente et gourmande en ressources pour des problèmes complexes. De l'autre côté, les heuristiques d'arrondissement offrent une rapidité d'exécution mais ne garantissent pas toujours l'optimalité.

On considère le PLNE de maximisation à variables bornées suivant :

$$\begin{cases} Maxz = c'x \\ S.CAx \leq b \\ d^- \leq x \leq d^+, x \in \mathbb{N} \end{cases}$$

Et soit la relaxation linéaire du PLNE le PLR suivant :

$$\begin{cases} Maxz = c'x \\ S.CAx \leq b \\ d^- \leq x \leq d^+, x_s \in \mathbb{R} \end{cases}$$

Nous présentons cette méthode hybride comme suit :

1. **Initialisation :** On commence par initialiser le PLNE et lancer l'algorithme Branch and Bound
2. **Résolution par Branch and Bound :** Le Branch and bound explore l'espace de recherche en créant des sous-problèmes à partir du problème initial. A chaque étape, il utilise des techniques comme la relaxation et la propagation de contraintes pour éliminer des branches de l'arbre de recherche
3. **Heuristique d'arrondissement :**Arrondir les solutions fractionnaires obtenues dans l'étape 2.
4. **Comparaison et mise à jour :**On compare la solution arrondie de l'étape 3 à la meilleure solution connue jusqu'à présent. Si la solution arrondie est meilleure alors elle devient la nouvelle meilleure solution entière.

5. **Reprendre l'algorithme :** Le Branch and Bound continue son exploration de l'espace de recherche en utilisant la meilleure solution entière connue comme borne inférieure pour la valeur optimale.
6. **Arrêt :** Le Branch and Bound s'arrête lorsqu'il a exploré toutes les branches de l'arbre de recherche et que la meilleure solution entière connue est la solution optimale du PLNE.

1.8.4 Heuristique d'arrondissement

Soit $y^0 = (x^0, x_s^0)$ une solution optimale du PLR précédent, où x^0 est une solution optimale non entière et $x_s^0 = b - Ax^0$. [27]
On pose alors $x_j^0 = [x_j^0] + \alpha_j$ avec $0 \leq \alpha_j \leq 1, j \in J = 1, 2, \ldots, n$, où $[x_j^0]$ est la partie entière et $\alpha_j = x_j^0$ est la partie fractionnaire. Afin d'avoir une solution réalisable entière pour le PLNE précédent, un processus d'arrondissement a été proposé et qui est décrit comme suit : [27]
On pose $M = z(x^0)$ et $m = z(\widehat{x})$ où $\widehat{y} = (\widehat{x}, \widehat{x_s})$ vérifie $z(\widehat{x}) = min_{y \in S} z(x)$.
On définit les ensembles d'indices suivants :
$J_c^+ = j \in J : c_j \geq 0$ et $J_c^- = j \in J : c_j < 0$.

1. Arrondi de x^0
   - (A_1) : Arrondi x^1 par rapport à la variable tel que :

     $x^1 = (x_j^1, j \in J)$,

     $$x_j^1 = \begin{cases} [x_j^0], & \text{si } 0 \leq \alpha_j^0 \leq \frac{1}{2} \\ [x_j^0] + 1, & \text{si } \frac{1}{2} \leq \alpha_j^0 \leq 1 \end{cases}$$

     Si $z(x^1) > z(x^0)$ alors x^1 n'est pas réalisable ,
     Sinon, on vérifie alors l'admissibilité de la solution.

     Si x^1 n'est pas réalisable, on fait l'arrondissement par rapport à la fonction z.
   - (A_2) : Arrondi x^2 par rapport à la fonction z tel que :

$$x^2 = \begin{cases} [x^0], siz(\alpha_j^0) = \sum_{j\in J} c_j\alpha_j^0 \geq 0 \\ [x^0] + e, siz(e) = \sum_{j\in J} c_j \leq z(\alpha_j^0) < 0 \end{cases}$$

Où $e = (1, 1, \ldots, 1) \in \mathbb{R}$.

Dans le cas où cet arrondi n'est pas réalisable, on pose :

$$x^2 = (x_j^2, j \in J)\ x_j^2 = \begin{cases} [x_j^0], j \in J_c^+; \\ [x_j^0] + 1, j \in J_c^-; \end{cases}$$

Si x^2 n'est pas réalisable, on fait l'arrondissement du milieu x^m

2. (A_3) : Arrondi du milieu x^mtel que $x^m = 1 \div 2(x^0 + \widehat{x})$ et on arrondit x^m à x^3 comme suit :
$$x^3 = (x_j^3, j \in J),\ x_j^3 = \begin{cases} [x_j^m] + 1, & \text{si } j \in J_c^+ \\ [x_j^m], & \text{si } j \in J_c^- \end{cases}$$

Remarque 1.3. L'arrondi x^3 est plus susceptible d'être réalisable, car x^m se trouve soit au centre du polyèdre S soit au centre de l'une de ses faces.

1.9 Définition d'un Problème Linéaire Mixte PLM

Un PLM est un problème d'optimisation où l'objectif et les contraintes sont exprimés sous forme d'équations linéaires, et où certaines variables peuvent prendre des nombres entiers.[20]
Une partie des variables seulement sont astreintes à être entières, on parle de programme linéaire en variables mixtes (PL mixte).

1.9.1 Formulation

Le problème d'optimisation linéaire en nombres mixtes "PLNM" vise à optimiser (maximiser ou minimiser) une fonction linéaire dite Fonction Objectif sous un ensemble de contraintes linéaires qui peut inclure des inégalités et des égalités. Et des variables de décision pouvant prendre des valeurs réelles ou entières. [20]

$$\begin{cases} \text{Optimiser} & \sum_{i=1}^{n} c_i x_i, & \\ \text{sous. contraintes :} & \sum_{i=1}^{n} a_{ij} x_i \leq b_j, & j = 1, \ldots, m \\ & x_i \in \mathbb{N}, & i = 1, \ldots, q \\ & x_i \geq 0, & i = q+1, \ldots, n \end{cases}$$

Exemple 1.1.

Soit (P) le PLM suivant :

$$(P)\begin{cases} max \quad z = 3x_1 + 2x_2 \\ S.c \quad x_1 + x_2 \leq 6 \\ \qquad 5x_1 + 2x_2 \leq 20 \\ \qquad x_1 \geq 0, \quad x_2 \in \mathbb{N} \end{cases}$$

Résolution du problème relaxé $(P^{'})$associé à (P) :

$$(P')\begin{cases} max\ Z = 3x_1 + 2x_2 \\ s.c. \\ x_1 + x_2 \leq 6 \\ 5x_1 + 2x_2 \leq 20 \\ x_1 \geq 0,\ x2 \geq 0 \end{cases}$$

La résolution graphique du problème relaxé (P') nous donne :

$x^{*'} = \left(\frac{8}{3}, \frac{10}{3}\right)$ *et* $z^{*'} = \frac{44}{3}$ *La valeur de* $x_2 \notin \mathbb{N}$*,donc la solution obtenue n'est pas optimale pour le PLM, on utilise alors la méthode Branch and Bound :*

Soit S_0 l'ensemble des solutions du problème relaxé (P'), on considère la partition suivante :

$S_1 = \{x \in S_0 : x_2 \leq 3\}$

$S_2 = \{x \in S_0 : x_2 \leq 4\}$

$S_0 = S_1 \cup S_2 \quad et \quad S_1 \cap S_2 = \emptyset$

On associe à S_1 et S_2 respectivement les PLM suivants :

$$(P_1)\begin{cases} max \quad z = 3x_1 + 2x_2 \\ S.c \quad x_1 + x_2 \leq 6 \\ \qquad 5x_1 + 2x_2 \leq 20 \\ \qquad x_2 \leq 3 \\ \qquad x_1 \geq 0, \quad x_2 \in \mathbb{N} \end{cases}$$

$$(P_2)\begin{cases} max \quad z = 3x_1 + 2x_2 \\ S.c \quad x_1 + x_2 \leq 6 \\ \qquad 5x_1 + 2x_2 \leq 20 \\ \qquad x_2 \geq 4 \\ \qquad x_1 \geq 0, \quad x_2 \in \mathbb{N} \end{cases}$$

On résout les problèmes relaxés $P_1^{'}$ *et* $P_2^{'}$ *se* P_1 *et* P_2 *respectivement :*

$$(P_1^{'})\begin{cases} max \quad z = 3x_1 + 2x_2 \\ S.c \quad x_1 + x_2 \leq 6 \\ \qquad 5x_1 + 2x_2 \leq 20 \\ \qquad x_2 \leq 3 \\ \qquad x_1 \geq 0, \quad x_2 \geq 0 \end{cases}$$

$$(P_2^{'})\begin{cases} max \quad z = 3x_1 + 2x_2 \\ S.c \quad x_1 + x_2 \leq 6 \\ \qquad 5x_1 + 2x_2 \leq 20 \\ \qquad x_2 \geq 4 \\ \qquad x_1 \geq 0, \quad x_2 \geq 0 \end{cases}$$

La résolution de $P_1^{'}$ *par la méthode graphique nous donne :*

$x^{*(P_1^{'})} = (2.8, 3)$ *et* $z^{*(P_1^{'})} = 14.4$ *(est une solution admissible car* $x_2 = 3 \in \mathbb{N}$*)*

La résolution de $P_2^{'}$ *par la méthode graphique nous donne :*

$x^{*(P_2^{'})} = (2, 4)$ *et* $z^{*(P_2^{'})} = 14$ *(est une solution admissible car* $x_2 = 4 \in \mathbb{N}$*)*

On a $z^{*(P_1^{'})} = 14.4 > z^{*(P_2^{'})} = 14$*, donc la solution optimale est donnée par x*=(2.8, 3) et valeur de la fonction objectif* $z = 3(2.8) + 2(3) = 14.4$

Chapitre 2

Introduction à la Théorie des Graphes

2.1 Introduction

La théorie des graphes est un domaine des mathématiques et de l'informatique qui étudie les structures constituées de points, appelés sommets ou nœuds, reliés par des paires de points, appelés arêtes. Les graphes permettent de modéliser des relations entre des objets de manière simple et intuitive, et ils trouvent des applications dans de nombreux domaines, tels que les réseaux sociaux, l'informatique, la biologie, la chimie, les transports et la logistique

2.2 Histoire

On ne peut étudier une théorie sans se poser la question de ses origines, pourquoi et dans quel contexte ces outils mathématiques ont-ils été élaborer, et quels sont les premiers fondateurs de ces théories.
L'origine de la théorie des graphes remonte au 26 août 1735, lorsque Euler a présenté un article intitulé "Solution d'un problème de géométrie de position", publié en 1741. Ce dernier traitait du célèbre problème des sept ponts de Königsberg. Le théorème d'Euler sur les graphes eulériens, bien que comptant parmi ses travaux les plus connus, ne fut démontré formellement qu'en 1873 par Carl Hierholzer. Il fallut attendre 1892 pour que Rouse Ball propose une modélisation du problème des sept ponts en utilisant le terme de "réseau".

La notion d'arbre a été introduite pour la première fois par le physicien Gustav Kirchhoff vers les années 1840. Cependant, c'est à la fin des années 1850 que la terminologie "arbre" a été formalisée par Arthur Cayley, alors qu'il travaillait sur un problème de calcul différentiel. Cayley a ensuite établi un lien entre les travaux du chimiste Alexander Crum Brown sur les structures moléculaires et les arbres, lui permettant de déduire une solution au problème de Crum Brown. En 1889, Cayley publia un article dans lequel il énonçait la célèbre formule : "il existe n^{n-2} arbres à n sommets étiquetés". En 1878, Sylvester utilise le terme "graphe" pour la première fois. Ce terme se répand rapidement et incite William Kingdon Clifford à s'intéresser au développement de la théorie des graphes. Cependant, les travaux de Clifford resteront inachevés suite à son décès prématuré en 1879. Le terme "graphe" s'impose définitivement grâce aux travaux de Julius Petersen, qui publie en 1891 un article fondateur intitulé "Die Theorie der regulären Graphen". Dans cet article, Petersen se concentre sur la théorie des graphes réguliers et leurs propriétés. Il reconnaît d'ailleurs l'influence de Sylvester dans ses travaux. En 1852, Francis Guthrie remarque qu'il peut colorier tous les comtés de la carte d'Angleterre avec seulement quatre couleurs, de sorte qu'aucun comté adjacent n'ait la même couleur. Désireux de généraliser ce résultat à n'importe quelle carte, il en parle à son frère, qui en informe son professeur Augustus De Morgan. Ce dernier, le 23 octobre 1852, écrit à William Rowan Hamilton pour lui présenter le problème, mais Hamilton, accaparé par ses propres travaux, ne donne pas suite. La première mention du problème des quatre couleurs dans un journal semble dater de 1860, dans la revue Athenaeum. Le problème reste alors sans solution jusqu'à ce que le philosophe américain Charles Sanders Peirce démontre qu'il faut six couleurs pour une carte dessinée sur un tore.

Vingt-cinq ans plus tard, Arthur Cayley s'intéresse au problème des quatre couleurs et lui donne une reconnaissance officielle en Grande-Bretagne. En 1879, Alfred Kempe publie une démonstration du théorème, modélisant le problème sous forme de graphe planaire. Malheureusement, onze ans plus tard, Percy Heawood démontre que la preuve de Kempe est erronée.

Avec l'avancée de l'informatique au XXe siècle, le mathématicien allemand Heinz

Heesch développe, en collaboration avec Wolfgang Haken, une technique de décharge entre 1950 et 1960. En 1976, Haken et Appel publient une preuve du théorème des quatre couleurs utilisant un ordinateur. Cette preuve sera ensuite simplifiée par Neil Robertson, Daniel Sanders, Paul Seymour et Robin Thomas en 1996.

Enfin, on ne peut pas clore cet historique sans mentionner Claude Berge et sa théorie des graphes parfaits, qui ont joué un rôle crucial dans l'essor de la théorie des graphes au cours des cinquante dernières années. Né en 1926, Claude Berge était un mathématicien français qui a d'abord mené des recherches en topologie, notamment en étendant un théorème de topologie dans l'espace de Banach et en publiant un ouvrage sur le sujet en 1959. Il s'est ensuite tourné vers la théorie des jeux sur les graphes, souvent appelés jeux de Nim, mais a rapidement constaté les limites de ce domaine. Son intérêt s'est alors porté vers d'autres problèmes liés à la théorie des graphes, en particulier un problème crucial pour la théorie de l'information. Berge a défini trois classes de graphes : G1 : les graphes α-parfaits, G2 : les graphes χ-parfaits, G3 : les graphes sans trou impair ni anti-trou impair, également appelés graphes de Berge Considéré comme le père des graphes parfaits, Berge a formulé plusieurs conjectures importantes dans ce domaine, notamment le théorème fort des graphes parfaits. Il a ainsi ouvert la voie à de nombreuses recherches ultérieures sur ce sujet fascinant. [9]

2.3 Définitions et Propriétés

2.3.1 Graphe

Définition 2.1. (**Graphe**)
Un Graphe est un schéma constitué par un ensemble qu'on suppose fini et non vide de points reliés entre eux par un ensemble de lignes ou de flèches. [2]

2.3.2 Graphe non-orienté

Définition 2.2. (**Graphe non-orienté**)
Un graphe non-orienté est défini comme un couple G=(X,E) tel que :

– X est l'ensemble de sommets de G. $X = \{x_1, x_2, \ldots, x_n\}$
– E est l'ensemble d'arêtes de G, qui sont des paires de sommets. $E = \{e_1, e_2, \ldots, e_n\}$.

Arête : on appelle arête une paire (x_i, x_j), et on dit que les sommets x_i et x_j sont **adjacents**.

Exemple 2.1. *Exemple de graphe non-orienté : G=(X,E), X={x_1,x_2,x_3,x_4} : l'ensemble des sommets, E=(e_1, e_2, e_3, e_4, e_5) : l'ensemble des arêtes.*

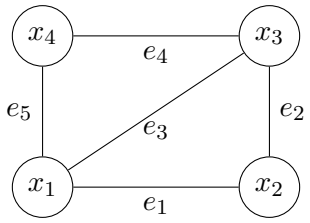


FIGURE 2.1 – Graphe non-orienté

2.3.3 Graphe orienté

Définition 2.3. Graphe orienté
un graphe orienté est défini comme un couple $G = (X, U)$ tel que :

– X est l'ensemble de sommets de G. $X = \{x_1, x_2, \ldots, x_n\}$.
– U est l'ensemble d'arcs de G, qui sont des paires de sommets. $U = \{u_1, u_2, \ldots, u_n\}$.

Exemple 2.2. *Exemple de graphe orienté : G=(X,U), X={x_1,x_2,x_3,x_4} : l'ensemble des sommets, U=(u_1,u_2,u_3,u_4,u_5) : l'ensemble des arcs.*

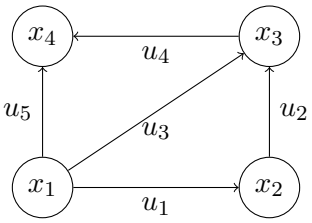


FIGURE 2.2 – Graphe orienté

Arc : On appelle arc un couple (x_i, x_j) tel que :

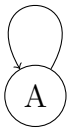
– u_i : Extrémité initiale de u. $I : U \to X$. $u \to I(u) = x_i$.

– u_j : Extrémité finale de u. $T : U \to X$. $u \to T(u) = x_j$.

Dans la figure 2.2 u_1 est un arc,avec $I(u_1) = a$,$T(u_1) = b$

Remarque 2.1.

– Lorsque $I(u) = T(u)$, u est appelé **boucle**.



– L'ordre du graphe est égal au nombre de ses sommets (n). [2]

– La taille du graphe est égal au nombre de ses arêtes (m).

2.3.4 Voisins, Prédécesseurs, Successeurs

Définition 2.4. Voisin [7]

Soit G=(X,E) un graphe non orienté, on dit de deux sommets x_i et x_j, qu'ils sont voisins s'ils forment les extrémité d'une même arête.

Dans la figure 2.1 les sommets a et b sont voisins

Définition 2.5. Prédécesseur [7]

Dans un graphe orienté G=(X,U), l'ensemble des prédécesseurs du sommet x est définit comme suit :

$\Gamma^-(x) = \{y \in X \quad / \quad \exists u \in U \quad t.q : I(u) = y \quad et \quad T(u) = x\}$

Dans la figure 2.2 $\Gamma^-(d) = a, c$

Définition 2.6. Successeur [7]

Dans un graphe orienté G=(X,U), l'ensemble des Successeurs du sommet x est définit comme suit :

$\Gamma^+(x) = \{y \in X \quad / \quad \exists u \in U \quad t.q : T(u) = y \quad et \quad I(u) = x\}$

Dans la figure 2.2 $\Gamma^+(c) = d$

Remarque 2.2.
L'ensemble des voisins d'un sommet x dans un graphe orienté est définit par :
$\Gamma(x) = \Gamma^+(x) \cup \Gamma^-(x)$

2.3.5 Le Degré

Définition 2.7. Degré d'un sommet [7]
Soit G=(X,E) un graphe non-orienté, le degré du sommet x est égal au nombre de ses voisins, et on note $d(x) = |N(x)|$, autrement d(x) est égal au nombre d'arêtes incidentes à ce sommet.

Dans la figure 2.1 d(a)=3.
Soit G=(X,U) un graphe orienté on a :

Définition 2.8. Demi-degré intérieur [7]
Le demi-degré intérieur ou bien entrant d'un sommet x est égal au nombre d'arcs ayant le sommet x comme extrémité finale, on le note $d_G^-(x) = |\{u \in U/T(u) = x\}|$.
Dans la figure 2.2 $d_G^-(d) = 2$

Définition 2.9. Demi-degré extérieur [7]
Le demi-degré extérieur ou bien sortant d'un sommet x est égal au nombre d'arcs ayant le sommet x comme extrémité initiale, on le note $d_G^+(x) = |\{u \in U/I(u) = x\}|$.
Dans la figure 2.2 $d_G^+(a) = 3$

Remarque 2.3.

- Une boucle sur un sommet compte double, ainsi elle apporte une contribution de 2 dans le calcul du degré de ce sommet.
- Dans le graphe orienté le degré d'un sommet x est le nombre d'arcs ayant x comme extrémité initiale ou finale, on le note : $d_G = d_G^+ + d_G^-$
- un sommet de *degré zéro* est dit *sommet isolé.*
- un sommet de *degré 1* est dit *sommet pendant.*

Définition 2.10. Le plus petit degré[7]
Le plus petit degré d'un graphe G est égal au degré du sommet ayant le moins de voisins et on le note $\delta(G)$.
Dans la figure 2.1 $\delta(G)= 2$.

Définition 2.11. Le plus grand degré[7]
Le plus grand degré d'un graphe G est égal au degré du sommet ayant le plus de voisins et on le note $\Delta(G)$.
Dans la figure 2.1 $\Delta(G)= 3$.

2.3.6 Chaine, Cycle, Chemin, Circuit

Soit G=(X,U) un graphe avec $X = (x_1, x_2, \ldots, x_n)$ et $U = (u_1, u_2, \ldots, u_n)$. [7]

Définition 2.12. La Chaine
Une chaine est une suite finie de sommets reliés entre eux par des arêtes. On la note $(x_0, x_1, \ldots, x_i)$, et on dit que x_0 et x_i sont les extrémité de la chaine.

Exemple 2.3. *G=(X,E), X=*$\{x_1,x_2,x_3,x_4,x_5\}$

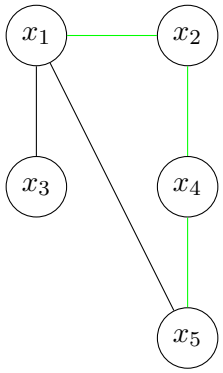


FIGURE 2.3 – Chaine

La chaine donnée par C=$\{\{x_1, x_2\},\{x_2, x_4\},\{x_4, x_5\}\}$ *a comme extrémité initiale* x_1 *et et comme extrémité finale* x_5

Définition 2.13. Le Cycle
Un cycle est une chaine ayant les extrémités confondues.

Exemple 2.4.

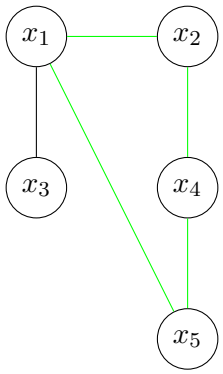


FIGURE 2.4 – Cycle

C=($\{x_1x_2\}$,$\{x_2x_4\}$,$\{x_4x_5\}$,$\{x_5x_1\}$) est un cycle.

Définition 2.14. Le Chemin

Un chemin est une suite finie de sommets reliés entre eux par des arcs orientés dans le même sens. On le note $(x_0, x_1, \ldots, x_i)$.

Exemple 2.5.

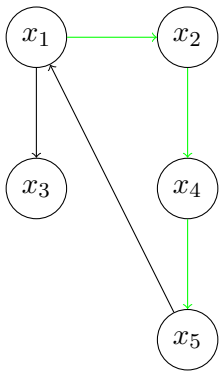


FIGURE 2.5 – Chemin

x_1 et x_5 sont respectivement les extrémités initiale et finale du chemin donné par C=((x_1x_2),(x_2x_4),(x_4x_5)).

Définition 2.15. Le Circuit

Un circuit est un chemin ayant les extrémités confondues.

Exemple 2.6.

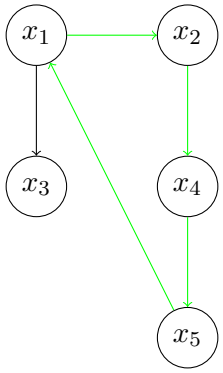


FIGURE 2.6 – Circuit

C=$((x_1x_2),(x_2x_4),(x_4x_5),(x_5x_1))$est un circuit.

Remarque 2.4.

Chaine et cycle sont des notions non-orientées.

2.3.7 Cycle hamiltonien et cycle eulerien

Définition 2.16. Cycle hamiltonien [14]

On appel cycle hamiltonien un parcours du graphe qui visite tous les sommets du graphe une, et une seule fois et revient au sommet de départ formant ainsi un cycle.

Exemple 2.7.

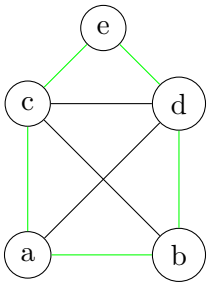


FIGURE 2.7 – Cycle hamiltonien

Remarque 2.5.

- un cycle hamiltonien n'a pas besoin de passer par toutes les arêtes du graphe, il suffit de passer par tous les sommets.
- On appel *graphe hamiltonien* un graphe contenant un cycle hamiltonien.
- Si un graphe possède un sommet pendant, alors il n'est pas hamiltonien

Définition 2.17. Cycle eulerien [7]
Un cycle eulerien est un cycle qui utilise chaque arête du graphe une, et une seule fois.

Théorème 2.1. ***Théorème d'Euler*** *[2]*
G=(X,E) un graphe simple connexe, il est dit eulerien *si et seulement si :*

$$\forall x_i \in X \quad i = (1, \ldots, n) \quad d(x_i) = 2k, \quad k \in \mathbb{N}$$

Exemple 2.8.

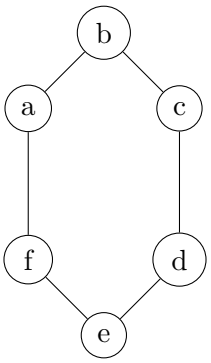


FIGURE 2.8 – Graphe eulerien

Le cycle donné par : ab,bc,cd,de,ea est eulerien, donc le graphe G est eulerien.

Définition 2.18. Cycle simple
Un cycle est dit simple s'il n'utilise pas une arête plus d'une fois.

Définition 2.19. Cycle élémentaire [14]
Un cycle est dit élémentaire s'il n'utilise pas un sommet plus d'une fois.

2.4 Graphes particuliers

Définition 2.20. Graphe simple [13]
Un graphe G=(X,E) est simple s'il est sans boucle et entre deux sommets il y a au plus une arête.

Exemple 2.9.

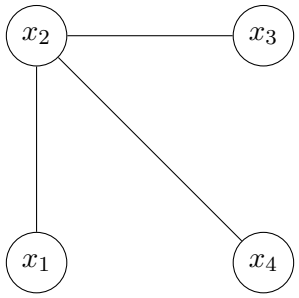


FIGURE 2.9 – Graphe simple

G=(X,E), X=$\{x_1,x_2,x_3,x_4\}$, E=$(\{x_1x_2\},\{x_2x_3\},\{x_2x_4\})$.

Définition 2.21. Graphe complet [13]
Un graphe complet d'ordre n est un graphe dans lequel tous les sommets sont deux à deux adjacents, on le note K_n, avec n est le nombre de sommets.

Exemple 2.10. *Graphe complet, K_4*

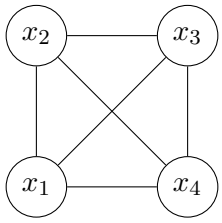


FIGURE 2.10 – Graphe complet

G=(X,E), X=$\{x_1,x_2,x_3,x_4\}$, E=$(\{x_1x_2\},\{x_2x_3\},\{x_3x_4\},\{x_1x_4\},\{x_1x_3\},\{x_2x_4\})$

Définition 2.22. Graphe biparti [13]
Un graphe biparti est un graphe où l'on peut partitionner l'ensemble des sommets X en deux sous-ensembles X_1 et X_2 tel que :

1. $X_1 \neq \emptyset$, $X_2 \neq \emptyset$.
2. $X_1 \cap X_2 = \emptyset$.
3. $X_1 \cup X_2 = X$.
4. Deux sommets du meme sous ensemble X_1 ou X_2 ne sont pas adjacents.

Exemple 2.11.

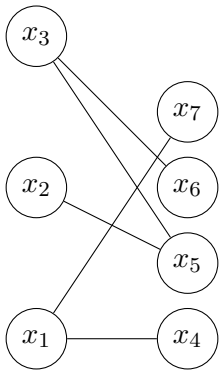


FIGURE 2.11 – Graphe biparti

G=(X,E), X=$\{x_1,x_2,x_3,x_4,x_5,x_6,x_7\}$, E=$(\{x_1x_7\},\{x_2x_5\},\{x_3x_6\},\{x_3x_5\},\{x_1x_4\})$
X_1=$\{x_1,x_2,x_3\}$, X_2=$\{x_4,x_5,x_6,x_7\}$.

Définition 2.23. Graphe pondéré [14]
Un graphe pondéré est un graphe où on associe à chaque arête un *poids*, on le note (X,E,p). Avec $p : E \to \mathbb{R}^+$ une application sur les arêtes.
Le poids d'un ensemble d'arêtes A est donné par $p(A) = \sum_{e \in A} p(e)$.

Exemple 2.12.

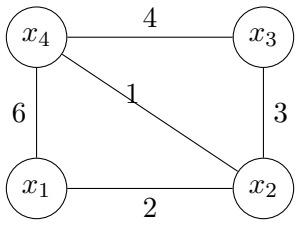


FIGURE 2.12 – G=(X,E,p)

X=$\{x_1, x_2, x_3, x_4\}$, E=($\{x_1x_2\}$,$\{x_2x_3\}$,$\{x_3x_4\}$,$\{x_1x_4\}$,$\{x_2x_4\}$)

Matrice de pondération [14]

On appelle matrice de pondération d'un graphe G=(X,E) la matrice dont les coefficients correspondant aux sommets x_i, x_j vaut :

$$\begin{cases} 0, & si \quad x_i = x_j \\ \infty, & \{x_i, x_j\}\text{n'est pas une arête} \\ p, & si \quad \{x_i, x_j\} \text{ est une arête de poids emphp} \end{cases}$$

La matrice de pondération de la figure 2.12 est :

$$\begin{pmatrix} 0 & 2 & \infty & 6 \\ 2 & 0 & 3 & 1 \\ \infty & 3 & 0 & 4 \\ 6 & 1 & 4 & 0 \end{pmatrix}$$

Le poids de l'ensemble des arêtes de la figure 2.12 est : $p(E)=\sum_{e\in E} p(e)=16$

Définition 2.24. Graphe planaire [14, 13]

Soit G=(X,E)un graphe non orienté, on dit qu'il est planaire si on peut le dessiner sur le plan et que les seuls points d'intersection des arêtes sont les sommets et que ces derniers sont des points distincts.

Ainsi on dit que G est *plongé* dans le plan.

Face d'un graphe planaire : Etant donné un graphe planaire G, le plan se retrouve divisé en un certain nombre de régions qu'on appelle *les faces de la représentation planaire.*

Remarque 2.6.
Le graphe planaire possède des faces internes et une face externe.

Théorème 2.2. ***Formule d'Euler 1752***
Soit G un graphe planaire on a :
n-m+f=2 , avec n :nombre de sommets , m :nombre d'arêtes, f : nombre de faces.

Définition 2.25. Graphe dual [7]
On appelle *dual* d'un graphe planaire -appelé primal- le graphe obtenu de la manière suivante :

- On dessine un sommet du dual dans toute face du primal.
- On trace une arête entre deux sommets du graphe dual si et seulement si les deux faces correspondantes dans le primal ont une arête frontière commune.

Théorème 2.3. ***Théorème de dualité***
La relation de dualité est symétrique, en effet, si un graphe est dual d'un autre, alors ce dernier est aussi le dual du premier.

Définition 2.26. Graphe complémentaire
Le complémentaire d'un graphe G=(X,E) est le graphe noté $\overline{G}$ défini par :
- $X_{\overline{G}} = X_G$.
- L'arête $x_i, x_j (x_i \neq x_j) \in E_{\overline{G}}$ si et seulement si $\{x_i, x_j\} \notin E_G$.

Exemple 2.13.

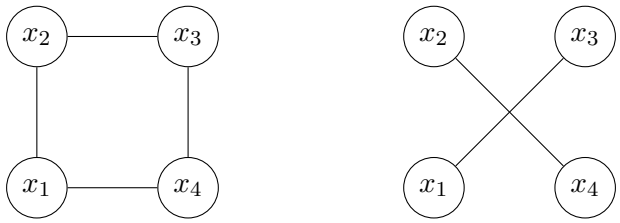


FIGURE 2.13 – Graphe et son complémentaire

G=(X,E),un graphe avec X=$\{x_1,x_2,x_3,x_4\}$, E=$((x_1x_2),(x_2x_3),(x_3x_4),(x_1x_4))$ et $\overline{G} = (X, E_{\overline{G}})$ son complémentaire tel que $E_{\overline{G}}$=$((x_1x_3),(x_2x_4))$.

2.5 Les Sous graphes

2.5.1 graphe partiel

Un graphe partiel de G=(X,E) est un sous-graphe obtenu en supprimant des arêtes de G, on le note $G^{'} = (X, E^{'})$.[12, 13]

Exemple 2.14. *: Le graphe partiel de la figure 2.1 est* $G^{'} = (X, E^{'})$*, avec* $E^{'} = \{e_1, e_2, e_5\}$.

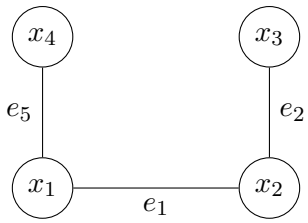


FIGURE 2.14 – Graphe partiel

2.5.2 Sous graphe induit

Le sous graphe de G=(X,E) *induit* par $X^{'} \subseteq X$, est le sous graphe ayant pour ensemble de sommets les éléménts de $X^{'}$ et pour ensemble d'arêtes toutes les arêtes ayant leurs deux extrémités dans $X^{'}$. [12, 13]

Exemple 2.15. *: Le sous graphe induit* $G^{'} = (X^{'}, E^{'})$ *de la figure 2.1 avec* $X^{'}$=$\{x_1,x_2,x_3\}$ *et* $E^{'}$=(e_1,e_2,e_3) *est le suivant :*

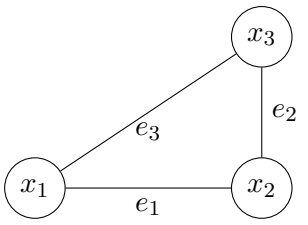


FIGURE 2.15 – sous graphe induit

2.6 La Connexité

Un graphe non-orienté est connexe si chaque sommet x_i est accessible à partir de n'importe quel autre sommet x_j. Autrement $\forall$ ($x_i x_j \in X$ avec $i \neq j$, il existe une chaine entre x_i et x_j. [12]

2.6.1 Notion de connexité

On définit la connexité dans un graphe par la relation entre les sommets du graphe considéré comme suit :
Pour $x, \quad y \in X, \quad xR \quad y \Leftrightarrow x = y$ ou bien il existe une chaine entre x et y.

2.6.2 Les composantes connexes

On appelle composante connexe un sous-graphe G' de G qui est connexe et maximal.

Remarque 2.7.
On appelle *graphe connexe* un graphe ayant une seule composante connexe.

Exemple 2.16.

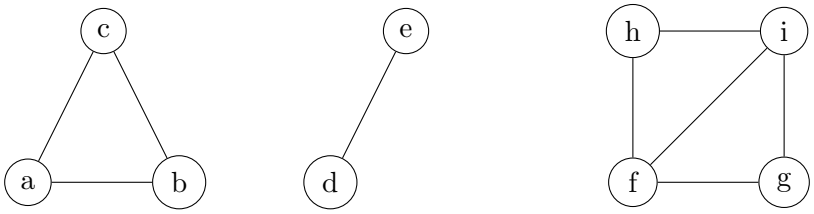


FIGURE 2.16 – composantes connexes

2.6.3 Connexité des graphes orientés

Soit G=(X,U) un graphe orienté, on dit que G est *fortement connexe* si $\forall x_i, x_j \in X$, il existe un chemin de x_i à x_j et un chemin de x_j à x_i pour toute paire $\{x_i, x_j\}$ du graphe [14].

Exemple 2.17.

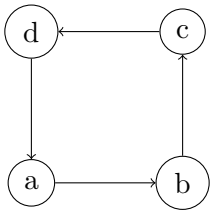


FIGURE 2.17 – Graphe fortement connexe

2.7 La représentation matricielle d'un graphe

2.7.1 Matrice d'adjacence

La matrice d'adjacence d'un graphe G=(X,U) d'ordre n est la matrice carrée $A_{(n\times n)} = a_{ij}$, i=1,. . . , n j=1, . . . , n , [2] tel que :

$a_{ij}= \begin{cases} 1, & si \quad (ij) \in U \\ 0, & sinon \end{cases}$

Exemple 2.18. *La matrice d'adjacence du graphe de la figure 2.1 est la suivante :*

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

2.7.2 Matrice d'incidence

La matrice d'incidence d'un graphe G=(X,U) d'ordre n est la matrice carrée $M_{(n\times m)} = m_{ij}$, i=1,. . . , n j=1, . . . , m. [6] (avec m est le nombre d'arcs du graphe) tel que :

$m_{ij}= \begin{cases} 1, & si \quad n_i = I(u_j) \\ -1, & si \quad n_i = T(u_j) \\ 0, & sinon \end{cases}$

Exemple 2.19. *La matrice d'incidence du graphe orienté de la figure 2.2 est la suivante :*

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & -1 \end{pmatrix}$$

2.7.3 Les avantages et les inconvénients de la représentation matricielle

Les avantages

◇ Rapidité des recherches.

◇ Compacité de la représentation.

◇ Simpliité des algorithmes de calcul.

Les inconvénients

◇ Représentation ne convenant qu'aux graphes simples.

◇ Stockage inutile de cas inintéressants (les zéros de la matrice)

◇ Le nombre d'aretes E est à remplacer par X^2

Afin d'éviter le stockage inutile des zéros de la matrice, on peut utiliser la représentation par *liste d'adjacence*.

Définition 2.27. Liste d'adjacence [13]
Soit g=(X,U) un graphe orienté, une liste d'adjacence est une structure de données dans laquelle on associe à chaque sommet sa liste de successeurs.

Dans le cas de graphes non-orientés, pour chaque arête $\{x_i, x_j\}$ on aura x_i qui appartient à la liste de x_j et x_j qui appartient à la liste de x_i

Remarque 2.8. Si le graphe est pondéré, on peut stocker le poids des arcs (arêtes) dans la liste d'adjacence avec les sommets.

2.8 Arbre et arborecsence

Définition 2.28. Arbre [13]
On dit que le graphe $T = (X, E)$ est un arbre s'il est connexe sans cycle.

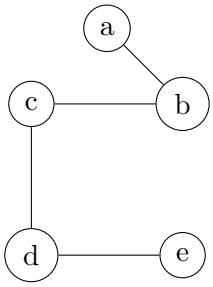


FIGURE 2.18 – Arbre

Définition 2.29.

Forêt : une forêt est un graphe dont les composantes connexes sont des arbres.

feuille : une feuille est un sommet pendant ou bien de degré 1.

Théorème 2.4.

Tout arbre fini avec au moins deux sommets comporte au moins deux sommets pendants.

Définition 2.30. Arborescence [3]

Une arborescence est un graphe orienté sans circuit admettant une racine $x_0 \in X$ telle que $\forall x_i \in X$ il existe un chemin unique allant de x_0 à x_i avec $x_0 \neq x_i$.
Si l'arborescence comporte n sommets alors elle comporte exactement (n-1) arcs.

Exemple 2.20.

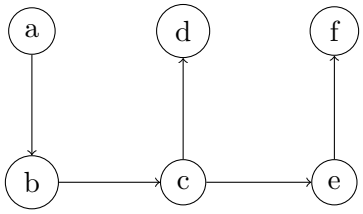


FIGURE 2.19 – Arborescence

2.9 Problème du plus court chemin

Le problème du plus court chemin ou bien problème de cheminement consiste à trouver le chemin de coût minimal entre deux sommets du graphe orienté, pondéré

G=(X,U, p).
Dans le cas de graphe non-orienté on remplace les arêtes par des arcs afin de le transformer en un graphe orienté. [6]

Remarque 2.9. Dans un graphe non-orienté une arête {xy} est vu comme l'arc (xy) et l'arc (yx).

Définition 2.31. coût d'un chemin [12]
Le coût du chemin $c = (x_0, x_1, \ldots, x_k)$ est égale à la somme des poids des arcs composants ce chemin :
$L(c) = \sum_{(x,y)\in c} p(x, y)$
Le coût du plus court chemin[12] entre deux sommets x,y est noté par $\delta(x, y)$ et définit comme suit :
$\delta(x_i, x_j) = \begin{cases} minL(c), & s'il\ existe\ au\ moins\ un\ chemin\ entre\ x_i\ et\ x_j \\ +\infty, & sinon \end{cases}$

Définition 2.32. Circuit absorbant [12]
On appelle *circuit absorbant* un circuit C de valeur strictement négative.

Exemple 2.21. *:*

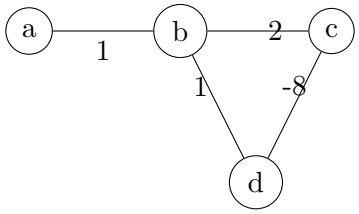


FIGURE 2.20 – circuit absorbant

Remarque 2.10.

- On dit que le sommet x est une **source (racine)** s'il n'a pas de prédécesseur, alors $d^-(x) = 0$
  Dans le graphe de la figure 2.19 {a} est une source.
- On dit que le sommet y est un **puit (anti-racine)** s'il n'a pas de successeur, alors $d^+(x) = 0$
  Dans le graphe de la figure 2.19 {d} et {f} sont des puits.

– Un graphe orienté peut posséder plusieurs sources et plusieurs puits.

Proposition 2.1.
*Dans un graphe orienté valué G=(X,U,*p*), il existe un plus court chemin entre tout couple de sommet si et seulement si le graphe G ne possède pas de circuit absorbant. En effet, à chaque fois que l'on passe par ce circuit on diminue le coût total du chemin, par conséquant* $\delta(x, y) = -\infty$ *donc il n'existe pas de plus court chemin entre x et y.*

Afin de résoudre le problème du plus court chemin, nous allons étudier trois algorithmes efficaces. L'algorithme de Dijkstra dans le cas de graphes valués positivement, l'algorithme de Bellman-Ford pour un graphe quelconque et l'algorithme A* .

2.9.1 Algorithme de Dijkstra

Edgser Wybe Dijkstra(1930-2002). [13] C'est en 1959 que Dijkstra propose un algorithme qui calcule le plus court chemin entre un sommets donné (*racine*) et tous les autres.
Soit G=(X,E,p) un graphe valué (orienté ou non) et dont le poids est positif, le principe de cet algorithme est le suivant : [2]

– Numérotons tous les sommets de G de 1 à n.
– Supposons qu'on veut calculer le plus court chemin partant du sommet 1 à tous les aures sommets du graphe.
– construisons un vecteur $\Pi = (\pi(1), \pi(2), \ldots, \pi(n))$ ayant n composantes tel que $\pi(i)$ tel que $\pi(i)$ soit égal à la longuer du plus court chemin allant de 1 à i. On initialise ce vecteur à $c_{1,i}$
– On initialise ce vecteur à $c_{1,i}$ c'est-à-dire à la première ligne de la matrice des coûts du graphe.
Matrice des coûts de G [13]

$$c_{i,j}= \begin{cases} 0, & si \quad i = j \\ \infty, & si \quad i \neq j \quad et \quad (i,j) \notin E \\ \gamma(i,j), & si \quad i \neq j \quad et \quad (i,j) \in E \end{cases}$$

avec $\gamma(i,j) > 0$ est le poids de l'arcs (i,j)

– Considérons deux ensembles de sommets S et son complémentaire dans G $\overline{S}$. S est initialisé à 1 à chaque pas de l'algorithme on ajoute à S des sommets de $\overline{S}$ jusqu'à ce que S=X de telle sorte que le vecteur Π donne à chaque étape le coût minimal des chemins de 1 aux sommets de S.

Algorithme de Dijkstra [14]

Données : G=(X,E,p)graphe pondéré de longueur d'aretes non-négative, et une source s.

Résultat : le plus court chemin de la source s vers tous les autres sommts (X-s).

Initialisations :

– $\pi = (c_{1,i})$ pour i={1,2,...,n}

– S={1} ; $\overline{S} = \{2, 3, \ldots, n\}$

Itérations :

– Tant que $\overline{S} \neq \emptyset$

– Choisir $i \in \overline{S}$ tel que $\pi(i)$ est minimum.

– Retirer i de $\overline{S}$ et l'ajouter à S.

– Pour chaque successeur j de i dans $\overline{S}$ **Faire**

– $\pi(j) = min(\pi(j), \pi(i) + \gamma(i,j))$.

Exemple 2.22. *Appliquons l'algorithme de Dijkstra au graphe suivant :*

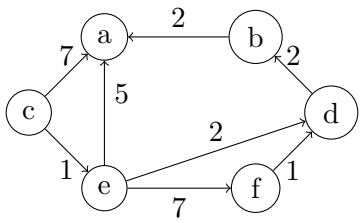


FIGURE 2.21 – Exemple dijkstra

Initialisation :

S={c}, $\overline{S} = \{a, b, d, e, f\}$, $\pi = (0, 7, \infty, \infty, 1, \infty)$ $1^{\textbf{ère}}$ **itération :** i=e, car $\pi(e) = min(7, \infty, \infty, 1, \infty) = 1$

S={c,e}, $\overline{S} = \{a, b, d, f\}$; les successeurs de e dans $\overline{S}$ sont : a,d et f.

- $\pi(a)$ prends la valeur $min(7, \pi(e) + \gamma(e, a)) = min(7, 1 + 5) = 6$
- $\pi(d)$ prends la valeur $min(\infty, \pi(e) + \gamma(e, d)) = min(\infty, 1 + 2) = 3$
- $\pi(f)$ prends la valeur $min(\infty, \pi(e) + \gamma(e, f)) = min(\infty, 1 + 7) = 8$

Le nouveau vecteur est $\pi = (0, 6, \infty, 3, 1, 8)$

$2^{\text{ème}}$ **itération :** i=d, $\pi(d) = 3$

S={c,e,d}, $\overline{S} = \{a, b, f\}$; d'où $\pi = (0, 6, 5, 3, 1, 8)$.

$3^{\text{ème}}$**itération :** i=b, $\pi(b) = 5$

S={c,e,d,b}, $\overline{S} = \{a, f\}$; et $\pi = (0, 6, 5, 3, 1, 8)$

$4^{\text{ème}}$**itération :** i=a, $\pi(a) = 6$.

S={c,e,d,b,a}, $\overline{S} = \{f\}$; $\pi = (0, 6, 5, 3, 1, 8)$

$5^{\text{ème}}$**itération :** i=f, $\pi(f) = 8$.

S={c,e,d,b,a,f}, $\overline{S} = \emptyset$; $\pi = (0, 6, 5, 3, 1, 8)$

Le coût minimal du chemin de c à a est égal à 6.

2.9.2 Algorithme A*

Peter Hart, Nilson, Bertram Raphael,1968 (prononcé A-Star) [18]

L'algorithme A* est un algorithme de recherche informé de l'intelligence artificielle qui effectue une recherche heuristique, c'est une extension de l'algorithme de Dijkstra qui utilise une heuristique pour améliorer l'efficacité de la recherche du plus court chemin entre une source (s) et une destination unique (d). Contrairement à Dijkstra qui explore tous les chemins possibles de manière exhaustive, A* se concentre sur les chemins les plus prometteurs en se basant sur une estimation de la distance restante jusqu'à la destination, ce qui lui permet de réduire considérablement le temps de calcul..

Et si ces chemins n'aboutissent pas ou bien s'avèrent mauvais par la suite, il examinera les solutions mises de côté. C'est ce retour en arrière pour examiner les solutions mises de côté qui nous garantit que l'algorithme trouvera toujours une solution (si elle existe, bien sûr).

L'heuristique H associe à chaque sommet x une estimation H_x de la distance qu'il reste à parcourir entre x et la destination (d).

A* choisit ensuite le sommet non visité pour lequel la valeur F(x) est minimale. Cette valeur F(x) représente la somme de la distance parcourue depuis le départ (G(x)) et de l'estimation de la distance restante jusqu'à la destination fournie par l'heuristique (H(x)) : F(x) = G(x) + H(x).

Algorithm 1 Recherche du plus court chemin dans un graphe pondéré

Entrées : Un graphe pondéré G (où $G_{u,v} \geq 0$), une paire de sommets (o, d), une heuristique admissible H

Sorties : La longueur du plus court chemin du sommet o au sommet d

```
pour u = 1 to n faire
    D_u ← +∞
fin pour
Q ← {o}
D_o ← 0
tant que Q n'est pas vide faire
    u ← sommet de Q ayant le plus petit D_u + H_u
    si u = d alors
        Retourner D_d
    fin si
    Enlever u de Q
    pour tous les v voisins de u faire
        si D_u + G_{u,v} < D_v alors
            D_v ← D_u + G_{u,v}
            Q ← Q ∪ {v}
        fin si
    fin pour
fin tant que
Retourner "pas de chemin entre o et d"
```

2.9.3 Algorithme de Bellman-Ford

L'algorithme de Bellman-Ford permet de trouver le plus court chemin à partir d'une source unique dans un graphe orienté contenant des arcs valués négativement à condition qu'il n'y ait pas de circuit absorbant (dans ce cas l'algorithme retourne une valeure booléenne indiquant l'existance d'un circuit absorbant accessible depuis la source). son principe est le suivant :

- On initialise S={s} et $A = \emptyset$

– On choisit un sommet $y \notin S$ et dont les prédécesseurs sont dans S.
– On calcule la distance de s à y.
– On ajoute le sommet y à S et l'arc (x,y) à A.

A la fin de cet algorithme on aura une arboressence de plus court chemin issus de la source unique s.

Algorithme de Bellman

Entrée : un graphe G=(X,U,p)sans circuit et une source s, $|U| = n$.

Sortie : Arborescence des plus courts chemins de source s T=(X,A).

1. $S = \{s\}$, $\pi(s) = d(s,s) = 0$, $\pi(y) = d(s,y) = \infty$ pour $y \neq s$, $A = \emptyset$.
2. Tant que $|S| \neq n$ faire
3. Choisir $y \in U - S$ ayant tous ses prédécesseurs dans S.
4. Si un tel sommet n'existe pas, terminer et afficher s n'est pas une racine.
5. Sinon poser $\pi(y) = min\{\pi(s) + p(s,y), (s,y) \in U\}$=$\pi(x) + p(x,y)$
6. Fin Si
7. $S = S \cup x$, $A = A \cup (x,y)$
8. Fin Tant que
9. Retourner A
10. Fin.

Exemple 2.23. *Appliquons l'algorithme de Bellman-ford au graphe suivant :*

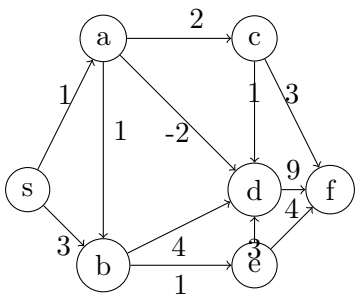


FIGURE 2.22 – Exemple Bellman-ford

S={s}, {a} est le seul sommet ayant tous ses prédécesseurs dans S. Donc $\pi(a) = 1$
S={s,a}, A={(sa)}, {b} et {c} ont tous leurs prédécesseurs dans S, donc

- $\pi(b) = min(\pi(a) + \gamma(a,b), \gamma(s,b)) = min(1+1, 3) = 2$
- $\pi(c) = \pi(a) + \gamma(a,c) = 1 + 2 = 3$

S={s,a,b,c}, A={(sa),(ab),(bc)}, {e} est le seul sommet ayant tous ses prédécesseurs dans S. Donc $\pi(e) = \pi(b) + \gamma(b,e) = 2 + 1 = 3$
S={s,a,b,c,e}, A={(sa),(ab),(bc),(be)}, {d} est le seul sommet ayant tous ses prédécesseurs dans S. Donc $\pi(d) = min((\pi(c) + \gamma(c,d)), \pi(a) + \gamma(a,d), \pi(b) + \gamma(b,d), \pi(e)+\gamma(e,d)) = min((3+1), (1-2), (2+4), (3+3)) = min(4, -1, 6, 6) = -1$
S={s,a,b,c,e,d}, A={(sa),(ab),(bc),(be),(ad)}, {f} est le seul sommet ayant tous ses prédécesseurs dans S. Donc $\pi(f) = min((\pi(d) + \gamma(d,f)), ((\pi(c) + \gamma(c,f)), ((\pi(e) + \gamma(e,f))) = min((9-1), (3+3), (3+4)) = min(8, 6, 7) = 6$
S={s,a,b,c,e,d,f}, A={(sa),(ab),(bc),(be),(ad),(cf)}, $|S| = n = 7$ (le nombre de sommets).
L'arborescenece des plus court chemins du graphe G de la figure 2.22 est donnée par le graphe suivant :

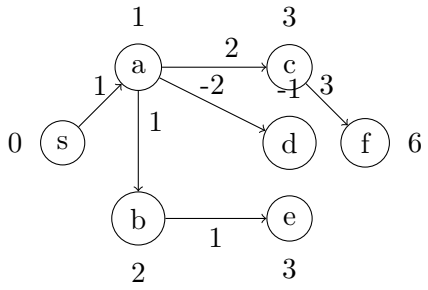


FIGURE 2.23 – Arborescence des plus court chemins

Chapitre 3

Optimisation des niveaux de vol des drones

3.1 Introduction

Ce chapitre présente une approche centralisée et stratégique pour la gestion du trafic des systèmes d'aéronefs sans pilote, visant à concevoir des trajectoires 4D optimales. L'objectif est de minimiser le temps de vol total de tous les véhicules sur une fenêtre de temps donnée. Pour cela, les pertes potentielles de séparation entre les véhicules sont modélisées et résolues.
Une trajectoire 4D est définie par le choix d'un chemin horizontal, associé à un profil de vitesse nominal, une piste de départ, et un niveau de vol de croisière. Le problème est formulé comme un programme linéaire à variables mixtes. Une approche de solution en deux étapes est proposée, prenant en compte les exigences opérationnelles telles que les dépôts d'intention de vol tardif ou les géofences statiques et dynamiques. Cette méthode est capable de gérer une densité de trafic très élevée.
Les résultats expérimentaux démontrent qu'en retardant les vols au départ ou en modifiant leur route 4D (verticalement ou horizontalement), il est possible d'obtenir des trajectoires viables pour les systèmes d'aéronefs sans pilote. Ces trajectoires permettent d'éviter les pertes de séparation tout en minimisant le temps de vol total.
Dans ce chapitre, nous nous basons essentiellement sur l'article "Metropolis II" de

Sonia Cafieri et son équipe [23] pour aborder l'optimisation des niveaux de vol pour les systèmes de gestion du trafic aérien. Les notions fondamentales nécessaires à la compréhension du problème seront d'abord définies. Ensuite, le problème étudié et la conception des trajectoires 4D seront présentés, en détaillant les phases d'une trajectoire 4D et la génération de chemins alternatifs en utilisant les notions de théories des graphes vues dans le chapitre 2, notamment le graphe dual et la recherche du plus court chemin avec l'algorithme A*.

La détection d'une perte potentielle de séparation sera ensuite examinée, suivie par le calcul du temps de séparation requis. La modélisation du problème sous forme d'un programme linéaire mixte (PLM) sera ensuite abordée. Par la suite, une approche de solution sera proposée en résolvant directement le problème PLM avec le solveur mathématique Gurobi pour les instances de petite à moyenne taille. Pour les instances de grande taille, une approche en deux étapes sera proposée : optimisation des niveaux de vol et relaxation des contraintes de perte de séparation.

3.2 Préliminaires

Définition 3.1. Trajectoire 4D [21]
la trajectoire 4D comprends la position tridimentionnelle (longitude, latitude, altitude) et le temps qui est la 4^{ieme} dimension.
Une trajectoire 4D est un ensemble $W \subset \mathbb{R}^4$, w=(x,y,z,t), avec (x,y) sont les coordonnées sur le plan 2D qui représentes la longitude et la latitude, z représente l'altitude, et t est le temps de vol.

Définition 3.2. Longitude
Ce sont les coordonnées géographiques qui représentent la position Est-Ouest d'un point sur terre, exprimées en degrés et varies entre 0° et 180°.

Définition 3.3. Latitude
Ce sont les coordonnées géographiques qui représentent la position Nord-Sud d'un point sur terre, exprimées en degrés et varies entre 0° et 90°.

Définition 3.4. Altitude

Distance verticale d'un points par rapport au niveau de la mer. Exprimée en mètres.

Définition 3.5. UAS
Unmained Aircraft System, qui veut dire système d'aéronerf sans pilote ou bien drone.

Définition 3.6. UTM
Unmained Traffic Management, qui veut dire système de gestion du trafic des UAS.

Définition 3.7. Géofences [23]
Ce sont des barrières virtuelles autour d'un emplacement physique, elle limitent les zones de vol autorisées.

Définition 3.8. Vertipot [23]
un vertipot est un emplacement oubien point de départ ou d'arrivée d'un UAS.

Définition 3.9. Niveau de vol [23]
C'est l'altitude spécifique à laquelle un UAS vole.

Définition 3.10. LoS (Loss of Separation) [23]
Une perte de séparation fait référence à une situation où deux UAS se rapproche à une distance inférieure à la distance minimale de séparation de sécurité imposée par la réglementation du controle aérien qui se traduit par une collision.

Définition 3.11. PLoS(Potential Loss of Separation) [23]
Une Perte de Séparation Potentielle fait référence à une situation où l'on peut avoir une perte de séparation.

Définition 3.12. Temps de séparation
Est le temps nécessaire pour éviter une perte de séparation.

3.3 Présentation du problème

Dans ce chapitre, le problème étudié concerne la conception des trajectoires 4D optimale pour une flotte de drones, qui respectent deux à deux de normes de

séparation à chaque instant de temps. Pour chaque drone une intention de vol est définie par un vertipot de départ et d'arrivée et un horaire de départ préféré.
L'objectif est de minimiser la durée totale des vols d'un ensemble de drones donné sous un horizon temporel donné.
Un modèle de programmation linéaire mixte en nombres entiers (PLM) est proposé dans lequel les contraintes principales visent l'évitement des pertes de séparation. [23, 25]

3.4 Conception de trajectoire 4D

Afin de concevoir un modèle 4D, on doit prendre en considération les hypothèses suivantes :

- Un drone monte/descend verticalement sur son vertipot de départ/arrivée.
- Un drone est autorisé à voler à un niveau de vol de croisière donné et aucun changement de niveau supplémentaire n'est autorisé.
- Pendant la croisière, le drone suit une structure de route prédéfinie modélisée sous forme d'un graphe.
- Un drone vole à différentes vitesses en fonction de la phase de vol :
  - Vitesse de croisière nominale : vitesse de vol maintenue en ligne droite sur de longues distances.
  - Vitesse de virage nominale : vitesse réduite adoptée lors des virages pour maintenir la stabilité de l'appareil. Les angles de virage plus importants peuvent nécessiter une vitesse encore plus lente.
  - Vitesse nominale de montée/descente : vitesse utilisée pour prendre de l'altitude ou en perdre de manière contrôlée.

  L'ensemble de ces vitesse compose **le profil de vitesse nominale**

Une trajectoire de vol est composée de trois phases :

- La phase de décollage : est une montée verticale jusqu'à atteinte du niveau de vol souhaité.
- La phase de croisière : Le drone maintien son niveau de vol et une vitesse vitesse constante en suivant une trajectoire horizontale.

– La phase d'attérissage : est une descente verticale jusqu'à atteindre le vertipot d'arrivée.

La conception de la trajectoire repose sur le choix de :

– Maintien au sol avant le décollage (retard au départ).
– Niveau de vol.
– Trajectoire horizontale (chemin en croisière), donnée sous forme d'une liste de sommets du graphe représentant la structure de la route.

Un temps de passage à n'importe quel point p de la trajectoire f peut être calculé pour chaque trajectoire. [23]

$$t_{p,f} = t^*_{p,f} + d_f + \delta(_{p,f})$$

Où :

$t^*_{p,f}$ est le temps de passage au point p de la trajectoire f sur sa trajectoire horizontale assignée, calculée en utilisant le profil de vitesse nominal et la dynamique du drone.

d_f est le retard au départ assigné à f.

$\delta(_{p,f})$ est le temps nécessaire pour monter ou monter et descendre jusqu'au niveau de vol du point p, qui dépend de la phase du vol (montée, croisière, descente).

En particulier, pour un point p en phase de croisière, $\delta(_{p,f})$ correspond au temps nécessaire pour monter jusqu'au niveau de vol choisi.

Afin d'obtenir des trajectoires 4D évitant les pertes de séparation, plusieurs chemins horizontaux alternatifs sont pré-calculés pour chaque intention de vol. Ensuite, pour chaque paire de ces chemins horizontaux alternatifs, les intersections spatiales sont détectées et le temps nécessaire pour garantir la séparation est calculé. [23]

3.4.1 Génération des chemins horizontaux alternatifs

Dans ce problème, le réseau routier est modélisé sous forme de graphe, où les arcs représentent les rues et les sommets représentent les intersections de rues et les

vertiports. Cette représentation permet de formaliser les déplacements des drones comme des chemins sur ce graphe.[23]
On utilise l'algorithme A^* pour trouver le chemin optimal entre le vertiport de départ et le vertiport d'arrivée. Ce chemin minimise une fonction de coût définie, dans ce cas précis, le temps de vol total.
Le choix du chemin a un impact direct sur le temps de vol, car la vitesse du drone est réduite en fonction de l'angle de virage entre les segments de la trajectoire. Cet angle de virage dépend de la forme du chemin emprunté.
Afin de mieux prendre en compte les angles de virage et d'optimiser le temps de vol, le graphe dual est utilisé à la place du graphe primal. Le graphe dual permet de représenter les angles de virage comme des arêtes et les segments de trajectoire comme des sommets.
L'algorithme A* peut être utilisé pour générer plusieurs chemins horizontaux alternatifs entre les vertiports de départ et d'arrivée. En explorant différentes routes, il est possible de trouver des chemins qui minimisent le temps de vol tout en tenant compte des contraintes de l'environnement, telles que les obstacles et les zones d'exclusion aérienne.
Plusieurs trajectoires alternatives sont générées pour chaque intention de vol. Une fois que les chemins horizontaux alternatifs sont générés, les intersections spatiales entre eux peuvent être détectées et les séparations requises calculées.

3.4.2 Détection des pertes de séparation potentielles

Une perte de séparation potentielle (PLoS) entre deux trajectoires horizontales de véhicules est une perte de séparation (LoS) qui pourrait se produire entre leurs trajectoires à quatre dimensions (4D), en considérant tous les choix possibles de niveaux de vol et de retard de départ. Selon nos hypothèses de modélisation, une PLoS peut se produire à un sommet du graphe. Par conséquent, une condition nécessaire pour une PLoS est que deux trajectoires partagent un sommet commun. [23]
Les catégories de pertes de séparation potentielles :

- Entre deux drones en croisière.(voir B de la figure LA REF) : les deux drones doivent etre associés au meme niveau de vol.
- Entre un drone en montée/descente et un drone en croisière(voir respct. A et C de la figure LA REF) : le drone en évolution doit etre affecté à un niveau de vol supérieur à celui du drone en croisière.
- Entre deux drones en décollage ou deux drones en attérissage : les deux drones doivent partager le meme vertipot.

Remarque 3.1.

Même si le partage d'un sommet commun est une condition nécessaire pour une perte de séparation potentielle entre deux trajectoires, une condition temporelle supplémentaire doit également être satisfaite.

Pour chaque trajectoire horizontale, le temps nominal de passage sur les sommets de cette trajectoire peut être calculé en utilisant l'heure de départ nominale (souhaitée) et le profil de vitesse nominal. Il est également possible de déterminer l'heure de passage la plus haute et la plus basse, en tenant compte des choix possibles de niveau de vol et de retard au départ.

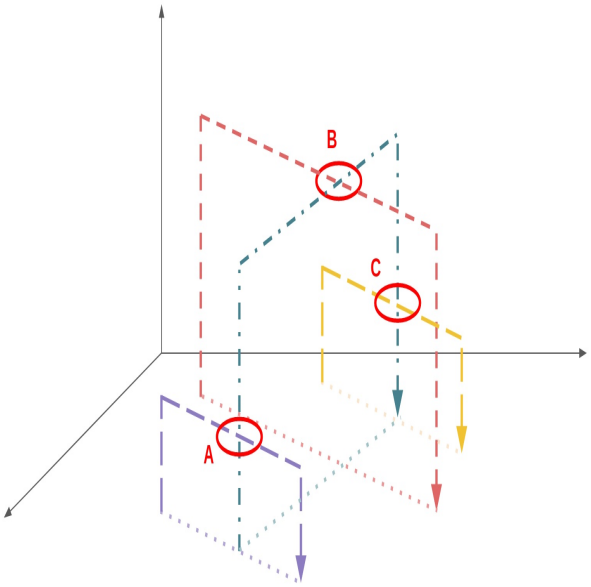


FIGURE 3.1 – PLoS aux intersections entre trajectoires

3.4.3 Calcul du temps de séparation

Aprés avoir détecté une perte de séparation potentielle, il est crucial de calculer le *temps de séparation* nécessaire pour éviter que la perte de séparation potentielle se traduise en une perte de séparation, et ainsi éviter une collision entre les deux drones. [23]
Plus précisément, le temps de séparation requis corresponds au temps nécessaire à garantir entre le passage des deux drones sur le sommet correspondant à la perte de séparation potentielle (c'est le sommet en commun).

Exemple 3.1.
cet exemple illustre une situation générique d'une perte de séparation potentielle entre deux trajectoires.

- D_1 *est la trajectoire du drone (1).*
- D_2 *est la trajectoire du drone (2).*
- I *est le point d'intersection des deux trajectoires,* D_1 *et* D_2*.*
- A_1/B_1 *sont les points avant/aprés intersection sur* D_1*.*
- A_2/B_2 *sont les points avant/aprés intersection sur* D_2*.*
- *Les points* A_1*,* B_1 *pour* D_1 *et* A_2*,* B_2 *pour* D_2 *représentent les points dont la distance par rapport au point d'intersection (I) est* la distance minimale requise.

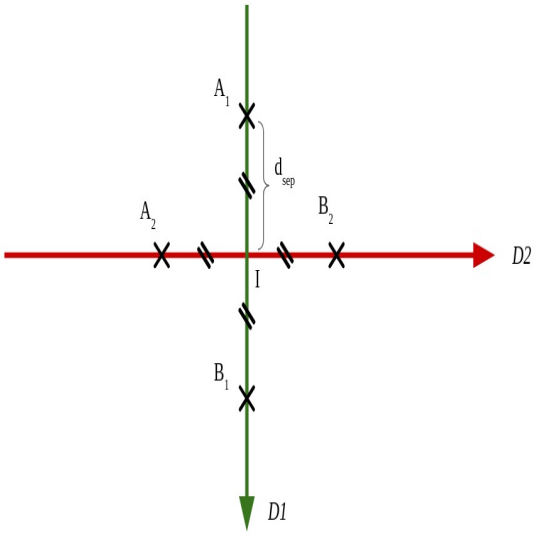


FIGURE 3.2 – Illustration d'une PLoS

Afin d'assurer la séparation des drones se déplaçant respectivement sur les trajectoires D_1 et D_2, si le drone sur D_1 passe avant celui sur D_2 il faut que :

- *Le drone sur D_2 ne doit pas franchir le point A_2 avant que le drone sur D_1 n'atteigne le point d'intersection (I).*
- *Le drone sur D_1 doit dépasser le point B_1 avant que le drone sur D_2 n'atteigne le point d'intersection (I).*

Par conséquent, le temps de séparation requis entre le moment de passage du drone sur D_1 et celui sur D_2 est donné par : le maximum du temps nécessaire au drone sur D_2 pour voler de A_2 au point d'intersection (I) et le temps nécessaire au drone sur D_1 pour voler du point d'intersection (I) à B_1, on le note :
$TSR = \max_t(t_{A_2 \text{ à } I}, t_{I \text{ à } B_1})$

Le même raisonnement si D_2 passe avant D_1.

Deux trajectoires peuvent parfois partager une portion commune, ce qui signifie qu'elles partagent une séquence de sommets consécutifs du graphe. Cette séquence peut être parcourue dans le même sens (Perte de séparation potentielle en queue), ou dans le sens inverse (Perte de séparation potentielle face à face).

Considérons l'exemple suivant :

D_1 et D_2 deux trajectoires qui se coupent.
I_1 et I_2 sont les extrimités de la portion partagée.
A_1 et A_2 sont les points qui précèdent l'intersection de D_1 et D_2 respectivement.
B_1 et B_2 sont les points qui succèdent l'intersection de D_1 et D_2 respectivement.

3.4.3.1 Perte de séparation potentielle en queue

Ce cas est similaire au cas général d'intersection décrit précédement, où les deux conditions garantissant la séparation sont appliquées aux deux extrémités de la portion de chemin partagée I_1 et I_2, cela permet de maintenir l'ordre de passage sur cette portion commune. [23]
Dans le cas où le drone sur D_1 passe avant celui sur D_2 le temps de séparation requis est calculé en suivant les étapes suivantes :

- Déterminer le temps nécessaire au drone sur D_2 pour voler de A_2 à I_1. On le note t_1
- Déterminer le temps nécessaire au drone sur D_1 pour voler de I_2 à B_1. On le note t_2
- Calculer la différence entre les temps de vol des drones D_1 et D_2 de I_1 à I_2. On le note t_3

Ainsi le temps requis pour garantir la séparation et l'ordre des deux drones est donné par : $max(t_1, t_2 + t_3)$
Le calcul du temps de séparation dans le cas où D_2 passe avant D_1 se fait de manière symétrique.

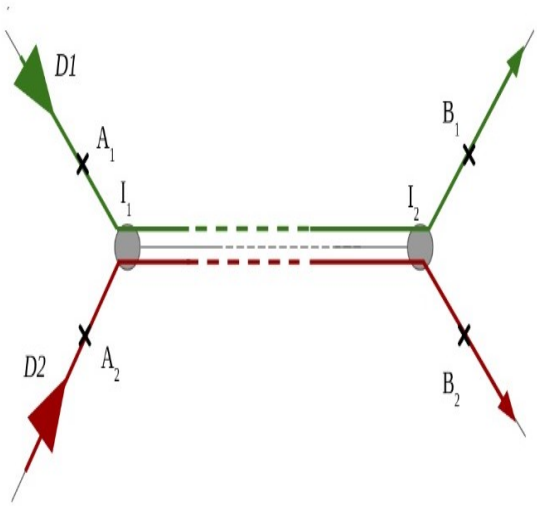


FIGURE 3.3 – PLoS en queue

3.4.3.2 Perte de séparation potentielle face-à-face

Si le drone sur D_1 passe avant celui sur D_2 sur le point d'intersection I_1, il doit aussi passer avant au point d'intersection I_2. Par conséquent, le drone sur D_1 doit avoir franchi B_1 avant que celui sur D_2 n'atteigne le point d'intersection I_2. [23] le temps de séparation requis est calculé en suivant les étapes suivantes :

- Déterminer le temps nécessaire au drone sur D_1 pour voler de I_2 à B_1. On le note t_1
- Déterminer le temps nécessaire au drone sur D_1 pour voler de I_1 à I_2. On le note t_2
- Déterminer le temps nécessaire au drone sur D_2 pour voler de I_1 à I_2. On le note t_3

Ainsi le temps requis pour garantir la séparation et l'ordre des deux drones est donné par : $t_1 + max(t_2, t_3)$

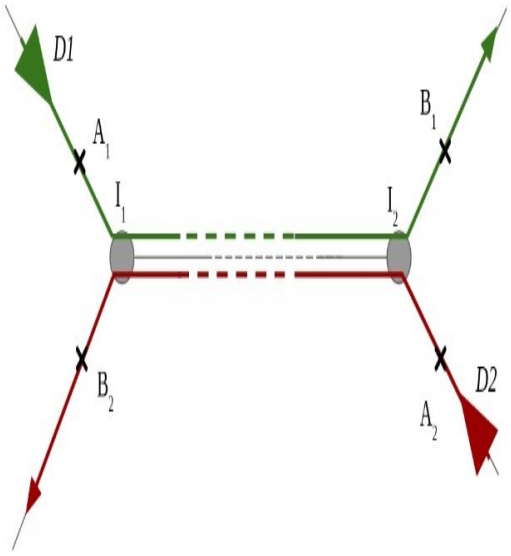


FIGURE 3.4 – PLoS face à face

Un dernier cas particulier concerne les situations de perte de séparation potentielle, celui où deux drones décollent ou atterrissent sur le même vertiport. Pour garantir la sécurité dans ces situations, il est nécessaire de définir un temps de séparation minimal que chaque drone doit respecter. Ce temps de séparation correspond à l'intervalle de temps minimum qui doit s'écouler entre le passage du premier drone et celui du second pour éviter tout risque de collision.

3.5 Modélisation du problème

Nous allons présenter un modèle de programmation en nombres entiers mixtes utilisé pour la conception de trajectoires 4D. Ce modèle attribue à chaque vol un niveau de vol (altitude), une attente au sol (retard au départ) et une trajectoire horizontale. [23]
Les contraintes du modèle garantissent le respect des temps de séparation établis (vu précédement). Ces temps de séparation assurent à leur tour la distance physique entre les UAS, empêchant ainsi toute perte de séparation. L'objectif du modèle est de minimiser le temps de vol total (y compris les attentes au sol) pour

l'ensemble des vols.

1. Les paramètres du problème

   – F : ensemble de vols.
   – $K_f, \forall f \in F$: ensemble de tous les chemins possibles pour le vol f.
   – K=$\cup_{f \in F} K_f$: ensemble de tous les chemins.
   – $b_k, \forall k \in K$: temps de déplacement associé au chemin k.
   – Y : ensemble de niveaux de vol.
   – P : ensemble de PLoS.
   – $\delta(i)$: temps de montée et de déscente vers le niveau de vol i.
   – $D_f, \forall f \in F$: délai d'attente au sol maximal pour le vol f.

   Pour chaque PLoS $p \in P$:

   – k_p^1, $k_p^2 \in K$: le premier et le deuxième chemin de la PLoS p.
   – f_p^1, $f_p^2 \in F$: le premier et le deuxième vol associés aux chemin k_p^1 et k_p^2.
   – t_p^1, t_p^2 : temps de passage nominal au point p du chemin k_p^1 et k_p^2 respectivement.
   – s_p^{12}, s_p^{21} : temps de séparation à respecter si f_p^1 est le premier à passer sur un point d'intersection, et respectivement si f_p^2 est le premier.

2. Les variables de décision :

   Pour chaque vol $f \in F$:

   – $y_f \in Y$: niveau de vol attribué au vol f.
   – $d_f \in [0, D_f]$: retard au décollage attribué au vol f.

   Pour chaque $k \in K$:

   – x_k=$\begin{cases} 1, \text{si le chemin k est attribué à son vol correspondant.} \\ 0, \text{sinon} \end{cases}$

3. La fonction objectif :

$$z = \sum_{f \in F} (d_f + \delta(y_f) + \sum_{k \in K_f} b_k x_k) \tag{3.1}$$

4. Les contraintes :

$$\sum_{k \in K_f} x_k = 1, \quad \forall f \in F \tag{3.2}$$

$$(t_p^1 + d_{f_p^1}) - (t_p^2 + d_{f_p^2}) + \Delta_{p,y_{f_p^1},y_{f_p^2}} + s_p^{12} \leq 0, \quad \forall p \in P \tag{3.3}$$

ou

$$(t_p^2 + d_{f_p^2}) - (t_p^1 + d_{f_p^1}) + \Delta_{p,y_{f_p^1},y_{f_p^2}} + s_p^{21} \leq 0, \quad \forall p \in P \tag{3.4}$$

Avec :

$\Delta_{p,y_{f_p^1},y_{f_p^2}}$ représente le temps de décalage dépendant de la PLoS et du niveau de vol f_p^1 et f_p^2 au point d'intersection.

La contrainte (3.2) garrantit q'un chemin est attribué à chaque vol.

Les contraintes (3.3) et (3.4) sont des contraintes d'évitement des pertes de séparation.

Si le drone sur D_1 passe avant celui sur D_2 alors le problème linéaire en variables mixtes (PLM) est donné par :

$$\begin{cases} min \quad z = \sum_{f \in F} \left(d_f + \delta(y_f) + \sum_{k \in K_f} b_k x_k \right) \\ S.C \\ \quad \sum_{k \in K_f} x_k = 1, \quad \forall f \in F \\ \quad (t_p^1 + d_{f_p^1}) - (t_p^2 + d_{f_p^2}) + \Delta_{p,y_{f_p^1},y_{f_p^2}} + s_p^{12} \leq 0, \quad \forall p \in P \\ \quad y_f \in Y \\ \quad d_f \in [0, D_f] \\ \quad x_k \in \{0, 1\} \end{cases}$$

Lorsque c'est D_2 qui passe avant D_1, on remplace la contrainte (3.3) par la contrainte (3.4) dans le PLM.

3.6 Approche de solution

Le modèle de Programmation Linéaire en nombres Mixtes (PLM), décrit précédement peut être directement résolu par un solveur mathématique de pointe pour les problèmes à instances petites à moyennes. [23]
En effet, les solveurs de PLM modernes tels que : CPlEX, Gurobi,etc. Sont dotés d'algorithmes puissants et optimisés qui leur permettent de traiter de manière efficace et précise les problèmes.
Cependant, l'UTM est particulièrement associé à des instances de forte densité de trafic. Cela implique des problèmes à grande échelle, pour lesquelles il est nécessaire de déterminer une approche de solution adéquate.
L'approche proposée ici est composée de deux étapes :

3.6.1 Optimisation des niveaux de vol

Cette étape vise à optimiser le niveau de vol attribué à chaque drone (UAS) en utilisant un modèle de programmation linéaire mixte (PLM). Ce modèle assigne des niveaux de vol et minimise le nombre total de pertes de séparation entre les drones. [23]

Le nombre de niveaux de vol disponibles influence considérablement la complexité du modèle PLM de conception de trajectoire 4D. Pour simplifier le problème, les niveaux de vol peuvent être prédéterminés.

Un modèle PLM est utilisé pour assigner un niveau de vol à chaque intention de vol. L'objectif est de minimiser le nombre total de lignes de vue (LoS) en considérant que les vols empruntent leur trajectoire la plus courte sans maintien au sol. Les niveaux de vol obtenus à l'issue de cette étape serviront de contraintes dans l'étape suivante de l'optimisation.
La contrainte de niveau de vol pour chaque vol f est donnée par : [23]

$$y_f = y_f^s, \quad \forall f \in F \tag{3.5}$$

Avec y_f^s est le niveau de vol optimal obtenu de la première étape d'optimisation. Cette pré-optimisation des niveaux de vol réduit le nombre de variables dans la prochaine étape de l'optimisation. Bien que cela réduise le temps de calcul, les solutions ne sont pas garanties d'être optimales.

3.6.2 Relaxation des contraintes

Cette étape consiste à résoudre le modèle PLM décrit précédemment en considérant une relaxation des contraintes de perte de séparation.

Plus précisément, les niveaux de vol du modèle PLM sont fixés aux valeurs calculées lors de la première étape, et les contraintes de perte de séparation sont relaxées. Cela signifie que les séparations temporelles ne sont plus garanties, et des situations de perte de séparation peuvent se produire.

Pour gérer ces situations, une variable de violation est introduite pour chaque contrainte de séparation. Cette variable est proportionnelle à la valeur de la violation, c'est-à-dire au temps passé en ligne de vue (LoS) par les drones. [23]
Variable de violation : variable de violation de contrainte pour la contrainte de séparation associée à p.

$$v_p \in \mathbb{R}^+, \quad \forall p \in P$$

Les contraintes relaxées :

$$(t_p^1 + d_{f_p^1}) - (t_p^2 + d_{f_p^2}) + \Delta_{p,y_{f_p^1},y_{f_p^2}} + s_p^{12} \leq v_p, \quad \forall p \in P \tag{3.6}$$

$$(t_p^2 + d_{f_p^2}) - (t_p^1 + d_{f_p^1}) + \Delta_{p,y_{f_p^1},y_{f_p^2}} + s_p^{21} \leq v_p, \quad \forall p \in P \tag{3.7}$$

La nouvelle fonction objectif :

$$V \sum_{p \in P} v_p + \sum_{f \in F} (d_f + \delta(y_f) + \sum_{k \in K_f} b_k x_k) \tag{3.8}$$

Elle représente la somme du temps de vol total et de la violation totale des contraintes.
V : représente le facteur de pénalité de violation défini par l'utilisateur.

Ainsi le problème obtenu après le calcul des niveaux de vol optimaux et de la relaxation des contraintes est le suivants : [23]

$$\begin{cases} min \quad V \sum_{p\in P} v_p + \sum_{f\in F} \left(d_f + \delta(y_f) + \sum_{k\in K_f} b_k x_k\right) \\ Sous \quad contraintes : \\ \qquad \sum_{k\in K_f} x_k = 1, \quad \forall f \in F \\ Si \quad le \quad vol \quad f_1^p \quad passe \quad en \quad premier : \\ \qquad (t_p^1 + d_{f_p^1}) - (t_p^2 + d_{f_p^2}) + \Delta_{p,y_{f_p^1},y_{f_p^2}} + s_p^{12} \leq v_p, \quad \forall p \in P \\ Si \quad le \quad vol \quad f_2^p \quad passe \quad en \quad premier : \\ \qquad (t_p^2 + d_{f_p^2}) - (t_p^1 + d_{f_p^1}) + \Delta_{p,y_{f_p^1},y_{f_p^2}} + s_p^{21} \leq v_p, \quad \forall p \in P \\ \qquad y_f = y_f^s, \quad \forall f \in F \\ \qquad y_f \in Y \\ \qquad d_f \in [0, D_f] \\ \qquad x_k \in \{0, 1\} \\ \qquad v_p \in \mathbb{R}^+, \quad \forall p \in P \end{cases}$$

3.7 Résultats Expérimentaux

Afin de valider les approches élaborées précédemment, nous allons les appliquer à un problème concret : l'optimisation des trajectoires de drones pour la livraison de colis dans la ville de Vienne. La ville de Vienne prévoit une augmentation significative de la demande de livraison de colis par drones dans les années à venir. Il est nécessaire de développer des stratégies efficaces pour gérer le trafic aérien des drones afin de garantir la sécurité et l'efficacité des livraisons.
Cette étude propose une approche en deux étapes pour optimiser les trajectoires 4D des drones en tenant compte des contraintes de sécurité et d'efficacité. L'approche utilise d'abord un modèle de programmation en nombres entiers mixtes (PLM) pour générer des trajectoires préliminaires, puis un simulateur de trafic pour évaluer les performances des trajectoires dans un environnement réaliste. La ville est modélisée par un graphe dont les sommets représentent

les intersections et les arêtes représentent les segments de rue, comme illustré dans la Figure (3.5). Les formes rouges dans cette Figure représentent des zones géo-clôturées interdites au trafic.

Les résultats des simulations montrent que l'approche proposée réduit considérablement le nombre et la gravité des situations de perte de séparation (LoS) entre les drones, tout en augmentant légèrement la durée de vol totale.

Cette étude démontre le potentiel de l'approche proposée pour optimiser les trajectoires de drones dans des environnements urbains complexes. L'approche peut être appliquée à d'autres scénarios de livraison de colis par drones, ainsi qu'à d'autres applications de drones nécessitant une gestion efficace du trafic aérien. [23]

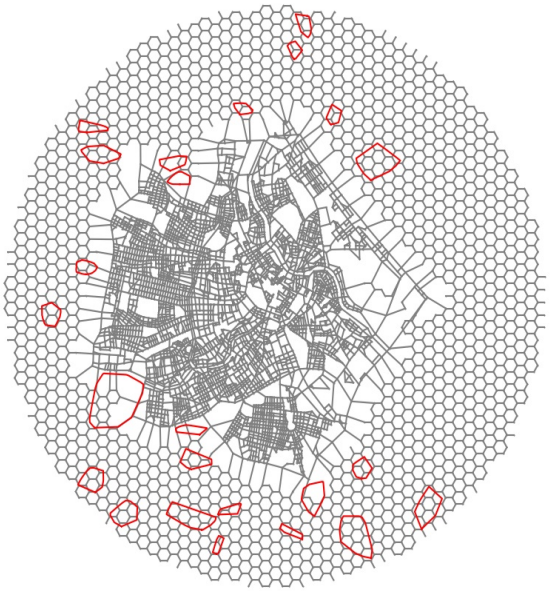


FIGURE 3.5 – Graphe représentant de la ville de Vienne

Conclusion générale

Ce mémoire de master a abordé le problème crucial de la sécurité aérienne dans le contexte des systèmes d'aéronefs sans pilote (UAS), en particulier la prévention des collisions. En s'appuyant sur les outils mathématiques de la programmation linéaire en nombres entiers mixtes (PLM) et de la théorie des graphes, une approche innovante d'optimisation des niveaux de vol a été développée pour garantir une séparation sécurisée entre les UAS.

D'abord, on a vu des concepts clés, des formulations mathématiques et des algorithmes de résolution (directes et approchés) pour les problèmes de Programmation linéaire Mixte, qui ont fourni un cadre solide pour la modélisation du problème d'optimisation des niveaux de vol des UAS.

Ensuite, on a exploré les fondements de la théorie des graphes, en s'attardant sur les structures graphes, les algorithmes de recherche de chemins et les concepts de connectivité. Ces outils mathématiques se sont révélés essentiels pour représenter les réseaux de trajectoires UAS et identifier les conflits potentiels.

Enfin, Le troisième chapitre a constitué le cœur de ce mémoire qui a été l'objet de l'article [23], en appliquant les connaissances acquises en PLM et en théorie des graphes à l'optimisation des niveaux de vol des UAS. Le problème a été formulé comme un PLM avec des contraintes de séparation minimale et d'efficacité de trajectoire. Deux approches de résolution ont été proposées : une approche directe pour les instances de petite taille et une approche heuristique en deux étapes pour les instances de grande taille.

En conclusion, ce mémoire démontre que l'application de la programmation linéaire en nombres entiers mixtes et de la théorie des graphes permet de développer des solutions robustes pour l'optimisation des niveaux de vol des drones, contribuant

ainsi à la sécurité et à l'efficacité des opérations aériennes dans un contexte de trafic croissant.

Comme perspectives, nous proposons un modèle de conception d'un réseau aérien de surveillance des feux de forêts dans la Wilaya de Tizi-Ouzou. Ce modèle vise à optimiser les trajectoires de vol et à améliorer l'efficacité de la détection et de la gestion des incendies.

Bibliographie

[1] Alexader Schrijver, Theory of Linear and Integer Programming, 1986, John Wiley Sons, 0-471-98232-6.

[2] Eric Sigward, Introduction à la théorie des graphes, Metz, France, Mars 2002.

[3] Mohammed Charkani Elhassani, Introduction à la Théorie des Graphes, Cours, ResearchGate, Université Sidi Mohamed Ben Abdellah, Fes, Maroc, 2002/2003.

[4] Catherine Mancel, Modelisation et resolution de problemes d'optimisation combinatoire issus d'applications spatiales, Thèse, INSA, Toulouse, France, 2005.

[5] Sana Belmokhtar, Lignes d'usinage avec équipements standard : modélisation, configuration et optimisation, Thèse, Ecole Nationale Supérieure des Mines, Saint-Etienne, France, 2007.

[6] Pierre Lopez, Graphes, Cours, LAAS–CNRS, France, 2 avril 2008.

[7] Solnon Christine, Théorie des graphes et optimisation dans les graphes, livre, Presses Universitaires de France, paris, France, 2010.

[8] Fabian Bastin, Modèles de Recherche Opérationnelle, Thèse, Université de Montréal, Canada, 2010.

[9] Gregory Morel, Stabilité et coloration des graphes sans P5, Thèse, Grenoble, France, 2011.

[10] Maria Ayala, Programmation linéaire en nombres entiers pour l'ordonnancement cyclique sous contraintes de ressources, Thèse, Toulouse, France, 2011.

[11] Friedrich Eisenbrand, programmation linéaire en nombres entiers, cours, EPFL, 14 avril 2011.

[12] Théorie des graphes, cours, IUT Lyon, France, 2011/2012.

[13] Didier Müller, Introduction à la théorie des graphes, 2012, Pahud et Cie, Cahiers de la CRM N°6.

[14] Guillaume Connan, Graphes, Cours, IUT Nantes, France, 23 septembre 2013.

[15] Souar Hamid, Résolution du problème de multi-flot compatible à coût minimal par l'approche génération de colonnes, Thèse, USTOMB, Oran, Algérie, 2015/2016.

[16] Amélie Lambert, Les algorithmes de Branch-and-Bound pour la PLNE, cours, cnam, France, 2016/2017.

[17] Hamdy A. Taha, Operations Research An Introduction, 2017, Pearson Education, University of Arkansas, Fayetteville, Angleterre, 10, 1-292-16554-5.

[18] Chennoufi Mohammed, La décentralisation des systèmes complexes par les systèmes multi-agents et l'algorithme A*, Thèse, USTOMB, Oran, Algérie, 2017-2018.

[19] Rachid Belgacem, Approche Sous-gradient pour le problème du TSP " Travelling Salesman Problem ", Thèse, Université Abdelhamid Ibn Badis, Mostaganem, Algérie, 2018.

[20] AMANI Sabrina, Programmation Linéaire Mixte en Nombres Entiers, Mémoire de master, UMMTO, Tizi-Ouzou, Algérie, 2020/2021.

[21] Gauthier Picard, Coordination de trajectoires 4D par optimisation distribuée dans la gestion du trafic aérien sans pilote, Article, ONERA/DTIS, Université de Toulouse, France, 2022.

[22] A new approach for non convex optimization problems applied to Hump and Benchmark functions, Fadila Leslous, Mouloud Goubi, Mohand Ouanes, Article, International Journal of Mathematics in Operational Research, Vol. 25, N°.3, UMMTO, Tizi-Ouzou, Algérie, 2023.

[23] Denis Bereziat, Sonia Cafieri, Andrija Vidosavljevic, Metropolis II : Centralised and strategical separation management of UAS in urban environment, Article, ENAC, Université de Toulouse, Toulouse, France, 18 Jan 2023.

[24] Fadila Leslous, Problème d'optimisation non convexe et optimisation DC, Thèse, UMMTO, Tizi-Ouzou, Algérie, 19/04/2023

[25] Zahraa Asfour, Sonia Cafieri, Andrija Vidosavljevic, Optimisation des trajectoires de drones en milieu urbain, Article, Amiens, France, 2024.

[26] Eurodecision, Heuristiques et Méta-Heuristiques, https ://www.eurodecision.com/algorithmes/recherche-operationnelle-optimisation/heuristiques-meta-heuristiques, 2024.

[27] Dihya Aissaoui, Souad Benhama, Méthode hybride pour la résolution des problèmes de programmation linéaire en nombres entiers, Mémoire de master, Université Abderrahmane Mira, Béjaia, Algérie.