THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *03/10/2017* **par :**

ROMARIC BREIL

Système multi-agents pour l'auto-structuration du trafic aérien

---

JURY

AMY R PRITCHETT	Pennsylvania State University	Rapporteur
GAUTHIER PICARD	École des Mines de Saint-Étienne	Rapporteur
DANIEL DELAHAYE	École Nationale de l'Aviation Civile	Directeur de thèse
ÉRIC FÉRON	Georgia Institute of Technology	Directeur de thèse
CHRISTIAN BÈS	Université Paul Sabatier	Président du Jury
JACCO HOEKSTRA	TU Delft	Examinateur

École doctorale et spécialité :
AA : Domaine Mathématiques : Mathématiques appliquées
Unité de Recherche :
École Nationale de l'Aviation Civile
Directeur(s) de Thèse :
Daniel DELAHAYE et *Éric FÉRON*
Rapporteurs :
Amy R PRITCHETT et *Gauthier PICARD*

Remerciements

Un doctorat est une entreprise de longue haleine. Durant plusieurs années, on effectue des recherches afin de répondre à une question complexe. Ce travail n'aurait pas pu être accompli sans le soutien de certaines personnes.

Tout d'abord, j'aimerais remercier les rapporteurs de cette thèse, Amy Pritchett et Gauthier Picard, qui ont relu ce manuscrit durant l'été, ainsi que les autres membres du jury : Christian Bès, Jacco Hoekstra, Éric Féron et Daniel Delahaye, qui ont accepté de venir, parfois de très loin, pour évaluer mes travaux.

Je remercie tout particulièrement mes deux directeurs de thèse, Daniel Delahaye et Éric Féron, qui m'ont donné envie de me lancer dans l'aventure, m'ont prodigué des conseils tout au long du doctorat, ont partagé leur savoir du contrôle aérien, et ont relu avec attention tout ce que j'ai pu écrire durant cette période. Je garde même un bon souvenir de la randonnée où Daniel m'a entrainé un jour d'automne 2016.

Puisqu'il s'agit d'une thèse CIFRE, j'ai également été plongé à temps partiel dans le monde de l'entreprise. Merci à Laurent Lapasset, qui a attentivement suivi mes travaux pour le compte de la société Capgemini Technology Services, de m'avoir aidé à gérer toutes les subtilités de l'organisation de ce doctorat (je crois que je n'ai plus de compte-rendu de réunion en retard).

Je voudrais également remercier les autres membres du laboratoire. Dans le désordre : Marcel Mongeau, qui gère avec efficacité le Master Recherche Opérationnelle de l'Université de Toulouse, et dont les cours m'ont été très profitables ; Stéphane Puechmorel, qui m'a initié aux arcanes de l'Entropie ; mais aussi Sonia Cafieri, Mohammed Sbihi, Pascal Lezaud, Ludovic D'Estampes, Florence Nicol, Catherine Mancel, et les autres, avec qui j'ai eu de nombreuses discussions enrichissantes.

Merci également à tous les étudiants du laboratoire. Quand j'ai commencé à travailler à l'ÉNAC, j'ai été accueilli et guidé par Laureline Guys et Brunilde Girardet, mes deux « grandes sœurs » autoproclamées, toutes les deux également en thèse CIFRE Capgemini-ÉNAC à ce moment-là. Votre combo crêpes-caramel au beurre salé me manque un peu !

Yohann Brenier, qui effectuait son stage de fin d'études en 2013, m'a fait découvrir la face cachée de l'ÉNAC, mais aussi la joie des pauses-café. Loïc Cellier, qui finissait sa thèse, a d'ailleurs participé à certaines d'entre elles.

Ces pauses se sont progressivement transformées en une institution, le « Romaric's Café », où les étudiants venaient discuter, ou débattre de manière enflammée, d'une grande variété de sujets, allant des travaux menés par chacun aux spécificités culturelles de leurs pays d'origine, en passant par les énigmes mathématiques et les astuces culinaires.

J'ai passé des moments très agréables avec les « clients » réguliers ou occasionnels de l'établissement : Vincent Courjault-Radé, qui fait de très bonnes pizzas, Florian Mitjana, qui m'a appris les règles de la belote, Hakim Djeriouat, qui adore animer des

débats philosophiques, Isabelle Santos, qui a élargi mes horizons musicaux, Clément Bouttier, qui m'a suggéré le nom du *Romaric's Café*, Jun Zhou et Man Liang, qui m'ont toutes les deux fait découvrir certains alcools chinois, Ning Wang, Olga Rodionova, Marina Sergeeva, Tambet Treimuth, Ji Ma, Daichi Toratani, François Lancelot, Zarrin Chua, et tous les autres que j'ai pu oublier.

Merci à Andrija Vidosavljevic, avec qui j'ai beaucoup apprécié partager un bureau pendant trois ans, et qui a toujours prêté une oreille attentive à mes questionnements.

Pour finir, j'aimerais remercier ma famille pour m'avoir encouragé et soutenu durant toute la durée du doctorat.

Table des matières

Introduction

La gestion des flux de trafic aérien, ou Air Traffic Flow Management (ATFM), cherche à structurer le trafic de manière à réduire la congestion dans l'espace aérien. La congestion étant causée par les avions volant dans les mêmes portions de l'espace aérien en même temps, l'ATFM organise le trafic dans les dimensions spatiales (ex. le réseau de routes) et dans la dimension temporelle (ex. séquencement et fusion de flux d'avions atterrissant ou décollant aux aéroports).

Le trafic aérien est en constante augmentation depuis plusieurs décennies, et l'Organisation de l'aviation civile internationale (OACI) prédit [ICA13a] que le nombre annuel d'avion doublera en 2030 par rapport à 2013. Les régions du monde où cette croissance devrait être la plus importante sont l'Asie-Pacifique et le Moyen-Orient [IAT16]. Les contrôleurs aériens assurent la sécurité et la fluidité du trafic en déviant temporairement les avions de leurs trajectoires initiales. De cette manière, la norme de séparation requise est maintenue entre tous les avions. Cette tâche est appelée détection et résolution de conflits.

Il est prévu qu'à terme, la capacité actuelle de l'espace aérien ne pourra pas absorber cette croissance. Pour gérer cette augmentation du trafic, les projets majeurs de recherche en gestion du trafic aérien à travers le monde [SES16 ; Nex16] cherchent à automatiser certaines tâches dévolues jusqu'à présent aux contrôleurs, pour leur permettre de gérer plus d'avions à la fois. Dans un effort plus importNomenclatureant, la gestion décentralisée des flux de trafic aérien, pour laquelle la gestion des flux est déléguée aux avions, est également envisagée.

L'objectif de cette thèse est de développer une méthodologie qui permet au trafic aérien de s'auto-structurer dans les dimensions spatiales et temporelle quand la demande est élevée. Cette structuration disparait quand la demande diminue. Pour remplir cet objectif, un système multi-agents a été développé, dans lequel les avions coopèrent pour structurer le trafic. Les systèmes multi-agents possèdent plusieurs avantages, incluant une bonne résilience aux perturbations.

Dans ce système, trois algorithmes sont implémentés, visant à réduire la complexité du trafic de trois manières différentes. Le premier algorithme, présenté dans le chapitre 3, permet aux agents avions volant sur un réseau de route de réguler leur vitesse de manière à réduire le nombre de conflits, un conflit se produisant quand deux avions ne respectent pas les normes de séparation. Le deuxième algorithme, décrit dans le chapitre 4, permet aux avions de résoudre les conflits quand le trafic n'est pas structuré par un réseau de routes. Le troisième algorithme, abordé dans le chapitre 5, crée des réseaux de routes locaux temporaires pour structurer le trafic.

Les trois algorithmes implémentés dans ce système multi-agents permettent de réduire la complexité globale du trafic, qui devient plus simple à gérer pour les contrôleurs aériens. Ces algorithmes sont appliqués à des exemples réalistes et sont capables de structurer le trafic de manière résiliente.

Le chapitre 1 présente le contexte opérationnel et scientifique dans lequel se positionne cette thèse. Le chapitre 2 décrit les algorithmes (systèmes multi-agents et recuit simulé) servant de base aux algorithmes décrits dans cette thèse.

Chapitre 1

État de l'art

Ce chapitre décrit le cadre général dans lequel s'inscrit cette thèse. La première section décrit l'organisation actuelle du trafic aérien, c'est-à-dire la structuration statique et dynamique du trafic, ainsi que les moyens de surveillance permettant aux contrôleurs aériens de connaitre les mouvements des avions.

La deuxième section décrit les améliorations prévues ou proposées pour faire face à l'augmentation prévue du trafic dans les prochaines décennies. Cette section décrit notamment des algorithmes permettant d'étudier la complexité du trafic, et d'autres permettant de structurer le trafic et de résoudre les conflits aériens. Certains des algorithmes présentés servent de base aux algorithmes développés pour cette thèse, et décrits dans les chapitres suivants.

1.1 Organisation du trafic aérien

Le trafic aérien est constitué de l'ensemble des aéronefs évoluant dans l'espace aérien. L'aviation civile, par opposition à l'aviation militaire, regroupe les activités liées au transport aérien (passagers, fret) ainsi qu'à l'aviation générale (tourisme, transport médical…), c'est une prérogative des États. En France, la Direction générale de l'Aviation civile (DGAC) est l'organisme public qui a pour rôle de structurer et de réguler le trafic aérien civil. Au niveau Européen, Eurocontrol est chargé d'uniformiser la gestion du trafic aérien opérée par les différents états membres.

On distingue deux catégories d'avions, ceux volant à vue, ou Visual Flight Rules (VFR), comme les avions de loisir ; et ceux volant aux instruments, ou Instrument Flight Rules (IFR), comme les avions de ligne. Comme leur nom l'indique, les IFR sont pourvus de systèmes leur permettant de voler sans visibilité, de décoller avec du brouillard et de voler à travers les nuages. Ces avions peuvent néanmoins être amenés à voler à vue, par exemple à l'approche de certains aéroports des États-Unis. Les avions IFR sont notamment équipés de Flight Management System (FMS), qui permet de gérer la navigation de l'avion. Ce sont donc les principaux concernés par l'automatisation de la gestion du trafic aérien en cours à travers le monde, et sujet de cette thèse. De manière générale, les avions IFR et VFR utilisent des portions différentes de l'espace aérien, séparées en altitude, sauf aux abords des aéroports.

Le trafic aérien est très structuré [ISD14]. Les contrôleurs aériens sont en charge d'en assurer la sécurité et la fluidité, notamment en maintenant les distances de séparation des avions supérieures aux minima réglementaires, de manière à éviter

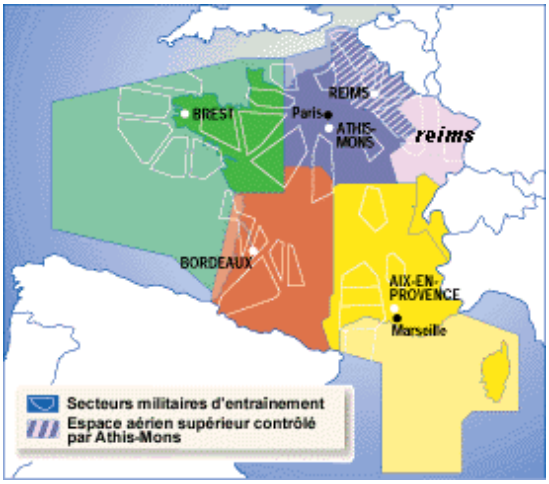


FIGURE 1.1 – Espaces aériens gérés par les cinq centres de contrôle en route de France.

les risques de collisions entre aéronefs. Le non respect de ces minima ou perte de séparation est appelé conflit aérien.

1.1.1 Structuration statique

La structuration statique permet d'organiser spatialement le trafic aérien, ce qui facilite la gestion du trafic pour les contrôleurs. L'espace aérien est divisé en secteurs plus petits, qui sont traversés par un réseau de routes que les avions doivent emprunter.

1.1.1.1 Secteurs

L'espace aérien de chaque pays suit habituellement les frontières de celui-ci. Quelques exceptions existent pour des raisons opérationnelles [ICA], notamment en Europe. Par exemple, la limite entre les espaces aériens Suisse et Allemand ne suit pas la frontière : certaines portions tortueuses ont été remplacées par des sections droites. Et du fait de sa petite taille, l'espace aérien Luxembourgeois est contrôlé par la Belgique.

Suivant la taille ou le nombre d'avions survolant chaque pays, son espace aérien peut être divisé en plusieurs régions appelées régions d'information de vol, ou Flight Information Region (FIR). La France en comporte cinq (figure 1.1), la Belgique, un seul, incluant le Luxembourg. Les avions en phase de croisière traversant un FIR sont contrôlés depuis un Centre de contrôle régional (CCR), en anglais Area Control Center (ACC), ou aux États-Unis Air Route Traffic Control Center (ARTCC) [FAA14a, PCG A–12]). Les CCR Français sont situés à Bordeaux, Brest, Marseille, Paris et Reims, comme indiqué sur la figure 1.1.

Au niveau d'un pays, l'organisme publique ou privé chargé d'assurer le service de navigation aérienne est appelé « Air Navigation Service Provider (ANSP) ». En France, il s'agit d'un service public fourni par la Direction des services de la Navigation aérienne (DSNA), appartenant à la DGAC.

Chaque FIR est à son tour divisé en secteurs, qui peuvent être représentés par des cylindres à section polygonale. Les avions traversant chaque secteur sont gérés par

FIGURE 1.2 – Réseau de routes de haute altitude en France.

une équipe de contrôleurs aériens.

1.1.1.2 Réseau de routes

Les avions doivent également suivre un réseau de routes. Le réseau de routes de l'espace aérien Français est représenté dans la figure 1.2. Celui-ci est défini à partir d'un ensemble de positions géographiques, en anglais waypoint, identifiées par un nom unique. Celles-ci peuvent être des balises radio installées au sol ou bien des positions GPS définies dans le FMS des avions.

Mathématiquement le réseau de routes peut être représenté par un graphe dont les nœuds sont les waypoints et les arcs définissent les routes que peuvent suivre les avions.

Sur le réseau de routes, les avions sont également organisés en altitude, comme indiqué dans la figure 1.3. L'altitude, mesurée grâce à la pression atmosphérique, est donnée en niveau de vol, ou Flight Level (FL), les FL étant séparés par 100 ft (foot, ou

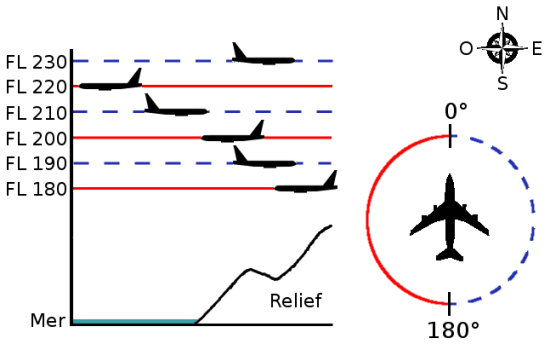


FIGURE 1.3 – Séparation des avions en niveaux de vol selon la règle semi-circulaire : les altitudes auxquelles les avions peuvent voler sont déterminées par leur cap.

pied). Les altitudes auxquelles peuvent évoluer les avions sont séparées par 10 niveaux de vol (1 000 ft), sauf bien sûr quand ils sont en phase de décollage ou d'atterrissage.

De plus, les niveaux de vol sont également répartis en deux groupes, les niveaux pairs (FL 100, FL 120, etc.) et impairs (FL 110, FL 130, etc.), selon le chiffre des dizaines. La règle internationale par défaut est que les avions IFR volant vers l'est (cap entre 0° et 179°) évoluent sur les niveaux impairs, les autres sur les niveaux pairs (figure 1.3). Les avions VFR (vol à vue) appliquent une règle similaire, mais volent 5 niveaux au dessus des IFR : les avions se dirigeant vers l'est volent aux FL 35, 55, etc., les autres aux FL 45, 65, etc.

La forme des secteurs est conditionnée par le réseau de routes [Ser+15]. Par exemple, les contrôleurs devant s'assurer de maintenir les avions séparés, les points de croisement du réseau de routes doivent être autant que possible éloignés de la bordure des secteurs, pour ne pas risquer de perte de séparation entre avions situés dans des secteurs voisins.

Le réseau de routes et la sectorisation de l'espace aérien permettent de limiter la charge de travail des contrôleurs. Il est plus facile de gérer un trafic structuré par un réseau de routes, où les points de croisement sont connus et les conflits plus faciles à anticiper [Hil04]. D'autre part, le nombre de secteurs est adapté au fil de la journée à la charge de trafic : quand le nombre d'avions diminue, les secteurs sont groupés, jusqu'à parfois former un seul secteur pour la totalité de la FIR, notamment la nuit quand le trafic est minimal.

1.1.1.3 Free Route Airspace, Free Flight Airspace

La structuration du trafic aérien sur un réseau de routes permet de réduire la complexité du trafic au bénéfice des contrôleurs aériens, notamment aux heures de pointe. Mais quand le niveau de trafic est faible, comme la nuit, imposer aux avions de suivre le réseau de routes ne fait qu'allonger leur temps de vol, et donc qu'augmenter leur consommation de carburant, sans avoir d'impact sur la charge de travail des contrôleurs aériens, qui n'ont qu'un petit nombre d'avions à gérer simultanément.

Le Free Route Airspace (FRA) [NMD15] est une méthode de gestion du trafic utilisée en Europe dans certains secteurs aériens. Dans ces zones, les avions peuvent suivre une trajectoire personnalisée entre les points d'entrée et de sortie du secteur, sans respecter le réseau de routes, dans des conditions de trafic peu dense. Cette

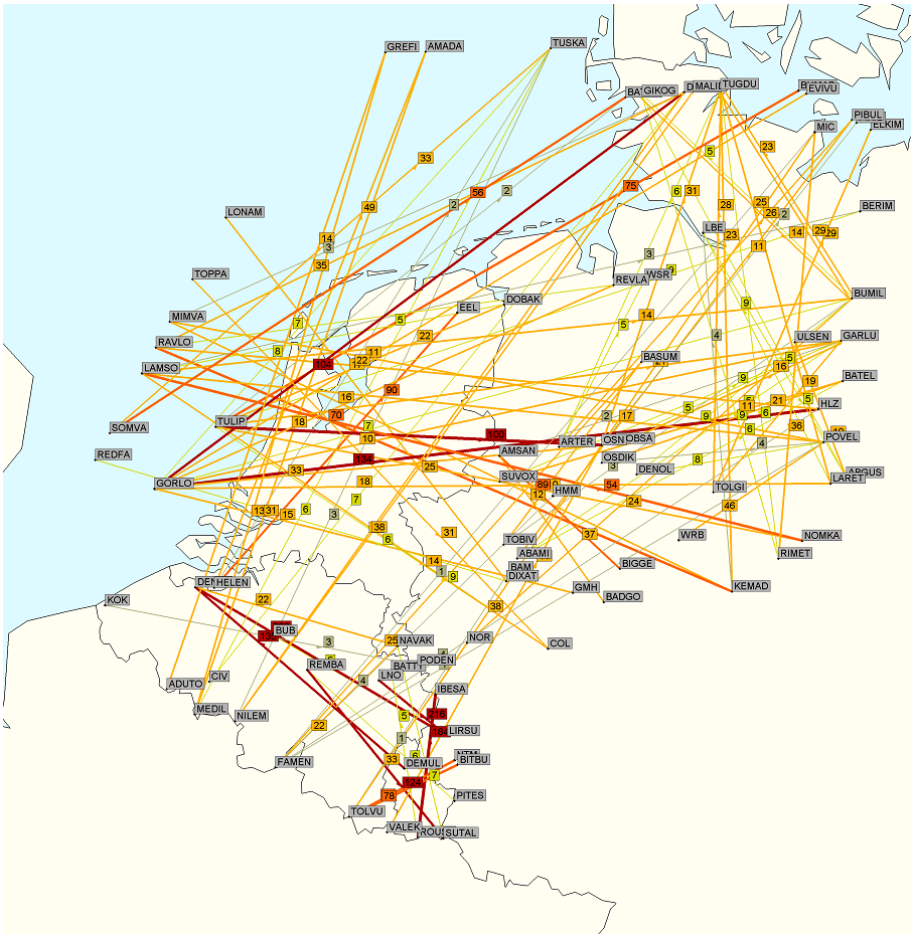


FIGURE 1.4 – Free Route Airspace : routes directes entre les points d'entrée et de sortie de l'espace aérien de Maastricht.

trajectoire est souvent une route directe, en ligne droite, comme le montre la figure 1.4 pour l'espace aérien contrôlé par le CCR de Maastricht. Mais l'avion peut également emprunter une route personnalisée, pour par exemple suivre une trajectoire optimale permettant de minimiser le temps de vol en fonction du vent.

Ces zones sont gérées par les contrôleurs aériens. Le FRA est actif uniquement pour des conditions de faible trafic ; cela veut dire la nuit dans certains secteurs, ou bien uniquement aux altitudes les plus élevées.

Le Free Flight Airspace (FFA) [Eura], est un concept similaire, mais contrairement au FRA, le contrôle du trafic est délégué aux avions. Aucun contrôleur n'est donc responsable de la résolution des conflits aériens. Il a fait l'objet de nombreuses recherches depuis plusieurs décennies, comme le projet Mediterranean Free Flight mené par Eurocontrol [Eur08], mais n'est pas utilisé.

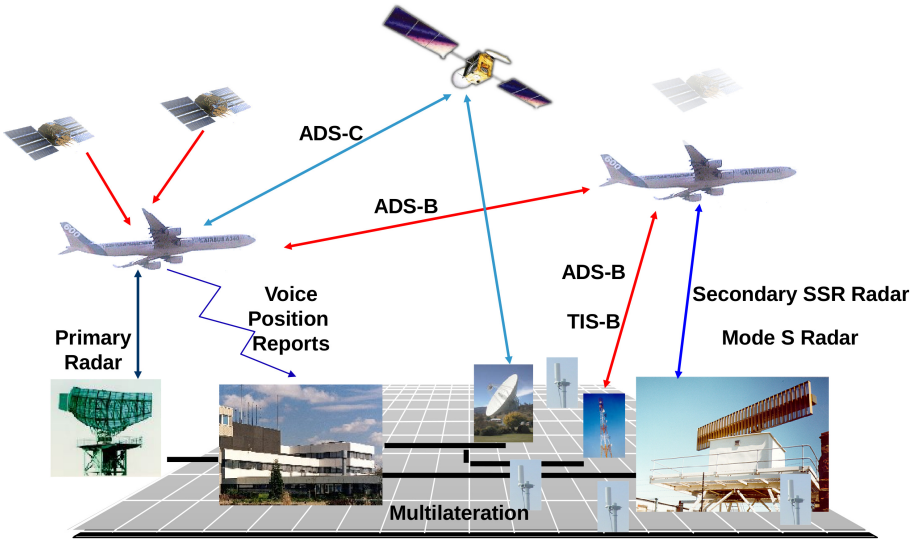


FIGURE 1.5 – Différents moyens de déterminer la position d'un avion : radars primaire et secondaire, multilatération, ADS-B et ADS-C, communications radio.

FIGURE 1.6 – Radar primaire d'un aéroport.

1.1.2 Moyens de surveillance du trafic

Un des rôles des contrôleurs aériens est de surveiller le trafic et de résoudre les conflits. On parle de détection et de résolution de conflits, ou Conflict Detection and Resolution (CDR). Pour obtenir la position des avions, les contrôleurs utilisent les informations provenant de différentes sources, comme le montre la figure 1.5. Ces informations sont combinées et présentées aux contrôleurs, qui peuvent alors connaitre l'état du trafic aérien et réguler la trajectoire des avions.

1.1.2.1 Radar

À l'origine, la position des avions était rapportée par les pilotes eux-mêmes par radio, d'abord en morse, puis par communication vocale.

Les radars sont apparus durant la seconde guerre mondiale [DGA07], et ont ensuite rapidement été utilisés pour surveiller le trafic aérien civil : le CCR de Paris a été équipé en 1952, et les autres ont rapidement suivi. La première génération de radars est appelée radars primaires, comme celui que l'on peut voir figure 1.6. Le principe est qu'une onde électromagnétique est envoyée par une antenne en rotation. Cette onde est réfléchie par les avions, puis captée par l'antenne. La position de l'antenne au moment de la réception de l'écho donne la direction de l'avion, et le temps que met l'onde à revenir à l'antenne donne sa distance. Afficher plusieurs positions successives permet de donner une idée de la route et de la vitesse de l'avion.

Ces informations, bien que très utiles, sont incomplètes : il manque notamment l'identifiant et l'altitude de l'avion. Le radar secondaire a été développé quelques années plus tard, et permet d'obtenir ces informations. Le fonctionnement est un peu différent de celui des radars primaires. Ici, l'antenne émet une requête qui est reçue par un système embarqué dans l'avion appelé *transpondeur*. Celui-ci renvoie un message contenant l'identifiant et le niveau de vol de l'avion. La position est obtenue par le radar en fonction de la direction de l'antenne et du temps séparant l'envoi de la requête et la réception de la réponse (ce qui donne la distance entre l'antenne et l'avion). Le premier radar secondaire installé en France a équipé le CCR de Paris en 1962, suivi en quelques années par les autres CCR.

L'augmentation de la précision des radars a permis en 50 ans de diminuer la norme de séparation des avions, qui est progressivement passée de 20 NM à 5 NM horizontalement et de 2 000 ft à 1 000 ft verticalement.

1.1.2.2 Multilatération

Cette technique, similaire au radar, consiste à utiliser plusieurs antennes omnidirectionnelles. Celles-ci émettent un signal à intervalle régulier. Le temps que met la réponse à revenir depuis les avions donne la distance entre l'avion et chaque antenne. Utiliser plusieurs antennes permet de trianguler la position de l'avion.

1.1.2.3 ADS-B

L'Automatic Dependant Surveillance – Broadcast (ADS-B) [FAA14a, p. 4–5–7] est un système visant à compléter la surveillance radar du trafic, et équipant un nombre croissant d'avions IFR (avions volant aux instruments). Comme le montre la figure 1.5, l'ADS-B complète les informations fournies par les radars primaires et secondaires. Grâce à l'ADS-B Out (émission), l'avion envoie régulièrement (jusqu'à plusieurs fois par seconde) un message numérique contenant son identifiant, sa position (latitude, longitude, altitude), et sa vitesse. Ces messages sont captés par des antennes au sol. Les informations collectées sont présentées aux contrôleurs aériens de la même manière que les informations radar.

Les avions peuvent également embarquer un système complémentaire appelé l'ADS-B In, permettant de recevoir les messages ADS-B émis par leurs voisins et permettre aux pilotes de connaitre les conditions de trafic environnantes (Airborne Traffic Situation Awareness).

L'ADS-B présente deux avantages. D'une part, les informations fournies par l'ADS-B sont similaires à celles obtenues par le radar, mais plus précises puisqu'elles sont fournies par le système GPS embarqué dans l'avion. D'autre part, le coût d'installation

d'un récepteur ADS-B au sol est très inférieur à celui d'un radar [1].

Le principal inconvénient est qu'une partie de la flotte aérienne n'est pas encore équipée. Mais ce système devrait se généraliser rapidement, notamment sous l'impulsion des États-Unis et de l'Europe : les avions IFR volant en Europe [Eur14] et aux États-Unis [FAA10] devront être équipés en ADS-B Out en 2020. D'autres pays l'ont rendu ou vont le rendre obligatoire, et les conditions d'application de la règle varient d'un pays à l'autre [ICA15]. Ce système est obligatoire en Europe pour les avions dont le poids ou la vitesse minimum dépasse un seuil. Cette obligation dépend aux États-Unis du secteur aérien traversé. En Australie l'ADS-B Out est obligatoire au delà d'une certaine altitude. Dans tous les cas, les avions de ligne et les avions cargo sont concernés dans ces régions, que ce soit par leur poids, leur vitesse ou leur altitude de croisière.

1.1.2.4 ADS-C

L'Automatic Dependant Surveillance – Contract (ADS-C) [ICA13b, p. 2.2.6] est un système similaire à l'ADS-B, mais fonctionnant selon un principe de requête-réponse. Une requête est envoyée depuis une station ADS-C au sol, ou par satellite dans les zones non équipées en stations ADS-C. L'avion répond en envoyant un message contenant les mêmes informations que l'ADS-B (identifiant, position, vitesse), et optionnellement la trajectoire prévue, sous forme d'un ensemble de coordonnées 3D associées à un temps ou un intervalle de temps, ainsi que des informations météorologiques (vent et température locaux). L'avion peut envoyer le message une seule fois ou de manière périodique.

L'ADS-B et l'ADS-C permettent donc aux avions de communiquer des informations précises au contrôle aérien au sol par rapport à leur situation courante et leurs intentions. D'une part, cela permet de contrôler le trafic aérien dans des zones qui ne sont pas couvertes actuellement par la surveillance radar. D'autre part, cela permet une certaine automatisation du trafic aérien, puisque les algorithmes de gestion du trafic ont besoin de ces informations (voir la section 1.2.4). Une partie des avions étant également équipés de récepteurs de messages ADS-B, des algorithmes de gestion automatisée du trafic pourraient même être embarqués à bord des avions.

1.1.3 Structuration dynamique

Comme le montre la figure 1.7, le vol d'un avion est divisé en cinq phases entre le moment où il quitte le terminal de l'aéroport de départ et le moment où il arrive au terminal de l'aéroport d'arrivée. Chaque phase est régie par des procédures de contrôle particulières. La phase de roulage sépare le *push-back*, le moment où l'avion quitte le terminal, de l'arrivée à la piste. Puis le décollage permet à l'avion d'atteindre son altitude de croisière. Le décollage se termine au point appelé Top of Climb (TOC). La phase de croisière, ou en-route [2] est la plus longue, et se termine au point appelé Top of Descent (TOD). Puis l'avion entame la phase d'atterrissage. Enfin, l'avion roule de nouveau pour atteindre le terminal d'arrivée.

Ce découpage en phases de vol correspond à la trajectoire optimale en terme de consommation de carburant. Quand l'avion vole à altitude et vitesse stabilisées, il consomme moins de carburant à haute altitude. La trajectoire optimale de ce point

1. Les récepteurs ADS-B destinés au grand public ne coûtent que quelques dizaines d'euros, et peuvent même être fabriqués avec un tuner TNT connecté à un ordinateur.

2. On dit *en-route* même en anglais.

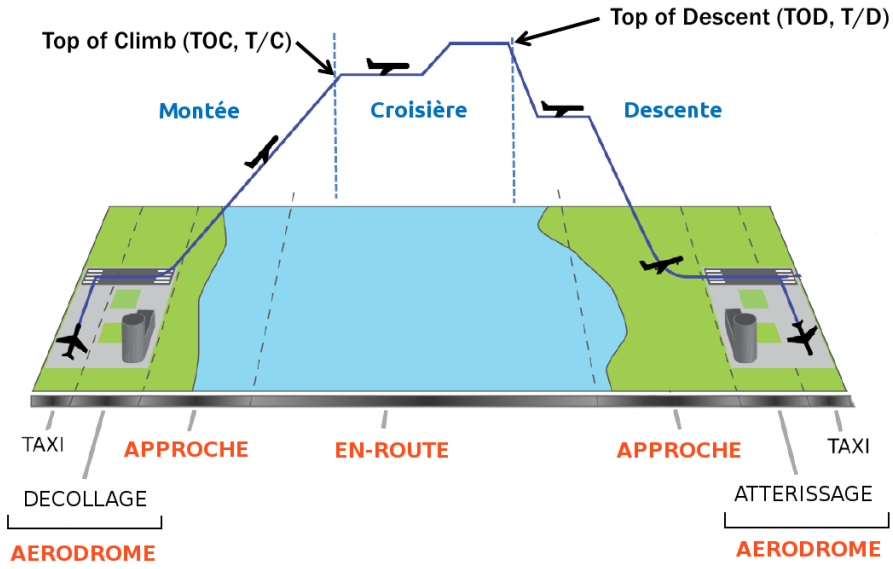


FIGURE 1.7 – Phases d'un vol du décollage à l'atterrissage.

de vue amène l'avion à altitude de croisière le plus rapidement possible, puis l'avion reste à cette altitude le plus longtemps possible, avant de redescendre pour atterrir.

Les contrôleurs opérant dans les tours de contrôle des aéroports sont chargés de contrôler le roulage. Le décollage et l'atterrissage sont gérés depuis les centres de contrôle d'approche, contrôlant l'espace aérien proche des aéroports, appelé en anglais Terminal Manoeuvring Area (TMA). La phase de croisière est gérée depuis les CCR. La phase de croisière sera abordée plus en détail par la suite.

La structuration du trafic permet de faciliter la tâche des contrôleurs pour atteindre les deux objectifs de sécurité et de fluidité. Cette structuration est effectuée en trois phases temporelles : la phase stratégique, la phase pré-tactique, et la phase tactique, correspondant à trois échelles de précisions, allant de l'organisation macroscopique des flux d'avions à la gestion microscopique de leurs trajectoires individuelles.

1.1.3.1 Phase stratégique

Du point de vue d'un vol, la phase stratégique commence plusieurs mois avant le décollage de l'avion, et se termine quelques heures avant celui-ci. Durant cette phase, le trafic est organisé de façon macroscopique par les compagnies aériennes : ouverture de lignes aériennes reliant des paires d'aéroports pour répondre à la demande prévue, planification des horaires de ces vols…

1.1.3.2 Phase pré-tactique

La phase pré-tactique commence quelques heures avant le décollage, et se poursuit durant le vol. Elle se base sur les décisions prises durant la phase stratégique, et permet de structurer les flux d'avions de manière à limiter la congestion du trafic, c'est-à-dire

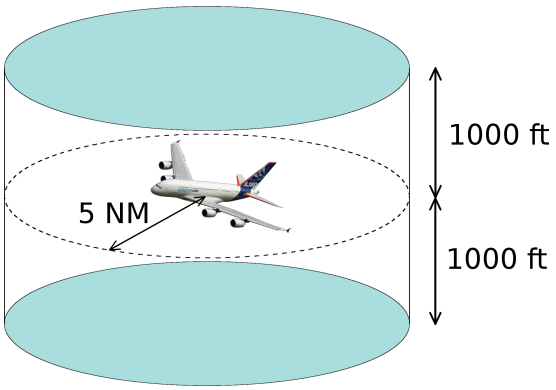


FIGURE 1.8 – Cylindre matérialisant la norme de séparation entre un avion et ses voisins.

la concentration locale instantanée d'avions. Pour ce faire, l'heure de décollage peut être retardée, ou bien le plan de vol de l'avion (son itinéraire prévu) modifié.

Avant le décollage, le pilote ou la compagnie aérienne doit déposer un plan de vol auprès des autorités de régulation du trafic aérien. Ce plan de vol contient entre autre les aéroports de départ et d'arrivée, l'itinéraire prévu, sous forme d'une liste d'identifiants de waypoints, ainsi que les heures de départ et d'arrivée prévues. Le plan de vol peut être modifié par les contrôleurs suivant les conditions de trafic ou météorologiques. Les pilotes peuvent également demander des modifications du plan de vol aux contrôleurs.

En Europe, un service d'Eurocontrol appelé Network Manager centralise la gestion des plans de vol. Suivant la demande de trafic estimée dans chaque secteur au cours de la journée, celui-ci peut refuser un plan de vol ou bien en retarder le départ, de manière à respecter la capacité maximale des secteurs et des aéroports.

1.1.3.3 Phase tactique

La phase tactique se déroule durant tout le vol, et comprend notamment toutes les activités permettant d'assurer la séparation des avions. Le trafic aérien a été structuré durant les phases stratégique et pré-tactique, de manière a ce que chaque contrôleur aérien ait un nombre limité d'avions à gérer simultanément, et puisse effectuer les derniers ajustements de leur trajectoire permettant de résoudre les conflits.

En croisière, les avions doivent être séparés d'au moins 5 milles nautiques horizontalement et 1 000 pieds verticalement. Un nautique, ou Nautical Mile (NM) étant égal à 1 852 m, et un pied à 30,48 cm, la norme de séparation des avions est de 9,26 km horizontalement et 304,8 m verticalement.

On peut se représenter cette norme de séparation par un cylindre d'environ 10 km de diamètre et 600 m de hauteur centré sur chaque avion (figure 1.8). Quand un autre avion pénètre dans ce cylindre, on parle de perte de séparation. Un conflit aérien est une situation dans laquelle une perte de séparation est prévue.

1.1.3.3.1 Vitesse de croisière La vitesse de croisière de l'avion est définie par les performances de ses moteurs, mais aussi par un indicateur appelé le Cost Index (CI). Celui-ci est le ratio du coût temps de vol, qui comprend le coût de l'équipage, sur le

coût du carburant. Il est fixé selon les préférences de la compagnie : augmenter le CI permet de raccourcir le temps de vol et de diminuer les retards, diminuer le CI permet de privilégier les économies de carburant.

À Cost Index fixe, la vitesse de croisière évolue au cours du vol. En consommant du carburant, l'avion s'allège, et son altitude de croisière s'élève légèrement de quelques centaines de mètres, ainsi que sa vitesse optimale.

1.1.3.3.2 Incertitude sur la vitesse de l'avion Un avion est capable de suivre un itinéraire très précisément grâce à son FMS [YK97] L'erreur de positionnement latéral suit une loi normale dont l'écart type vaut 50 m, ce qui veut dire que l'erreur de positionnement est inférieure à 100 m dans 95 % des cas (en comparaison un Airbus A380 fait 80 m d'envergure [Emi]). L'erreur de positionnement vertical suit aussi une loi normale dont le σ vaut 30 m, donc l'erreur d'altitude est inférieure à ±30 m dans 95 % des cas.

Par contre, l'incertitude sur la vitesse est beaucoup plus importante. Elle suit également une loi normale, mais de $\sigma = 15$ kt (un nœud (kt) valant 1 NM/h, soit 1,852 km/h). Cela veut dire qu'en laissant un avion voler seulement 20 minutes, l'erreur sur sa position le long de sa route est de ±10 NM dans 95 % des cas.

Cela est dû au fait que les avions doivent voler à vitesse constante pour limiter l'usure des réacteurs. En effet, changer de régime moteur a trois inconvénients : un changement de vitesse nécessite plusieurs dizaines de secondes pour prendre effet, cela accélère l'usure des réacteurs, ce qui réduit la longévité de l'avion, et cela augmente la consommation de carburant de manière importante, car contrairement aux moteurs de voiture, les réacteurs sont conçus pour fonctionner à régime constant.

Or, même si un avion vole à vitesse constante par rapport à l'air, il subit l'influence du vent, qui est souvent de quelques dizaines de km/h, et qui change au cours du vol. Cela augmente ou diminue donc la vitesse de l'avion par rapport au sol. De plus, la vitesse de l'avion en route n'est pas donnée en kilomètres par heure, mais en nombre de Mach. Il s'agit du rapport entre la vitesse de l'avion et la vitesse du son, qui dépend de l'altitude et de la température de l'air. Par exemple, un avion à Mach 0,89 vole à 89 % de la vitesse du son.

Le vent et la température sont deux facteurs d'incertitude de la position des avions qui rendent impossible la résolution des conflits à plus de quelques minutes d'avance, sauf en prenant des marges importantes, ce qui n'est pas toujours possible, notamment aux heures de pointe.

1.1.3.3.3 Résolution de conflits Pour résoudre les conflits, les contrôleurs aériens peuvent donner plusieurs types d'ordres aux pilotes de l'un ou des deux avions : changer de cap temporairement avant de revenir à la route initialement prévue, changer d'altitude, et changer de vitesse. Ces deux dernières manœuvres sont surtout utilisées en phase de décollage et d'atterrissage, et peu en croisière, puisqu'elles nécessitent de changer de régime moteur. Au contraire, changer de cap demande uniquement l'utilisation des ailerons (situés sur les ailes de l'avion), ce qui est rapide et économe en carburant.

Cette gestion des conflits n'est possible que dans les zones couvertes par les radars, ou d'autres systèmes comme l'ADS-B (voir la section 1.1.2). Dans les zones non surveillées, comme les déserts et les océans, il faut s'assurer que les avions ne rentreront pas en conflit. Avant de pénétrer dans ces zones, les contrôleurs séparent

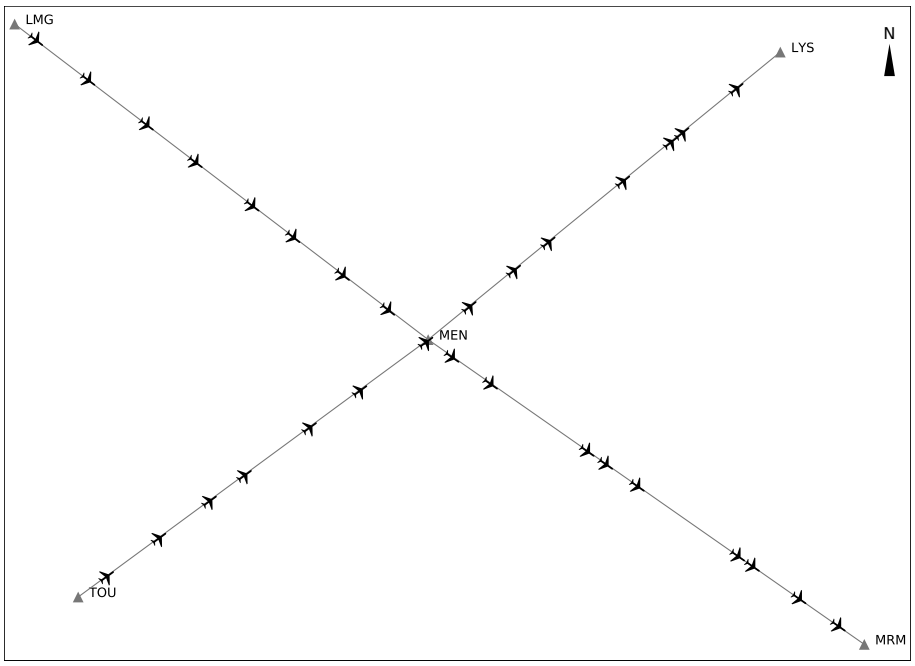


FIGURE 1.9 – Type de réseau de routes rencontré dans des configurations de Miles-in-Trail.

les avions avec des marges temporelles importantes tenant compte de l'incertitude sur la position des avions après plusieurs heures de vol non surveillées.

1.1.3.3.4 Miles-in-Trail Le Miles-in-Trail (MIT) [Gre00] est une méthode de gestion du trafic développée aux États-Unis pour faire face à la congestion dans certaines zones de l'espace aérien. Quand celle-ci augmente, les contrôleurs utilisent le Miles-in-Trail pour structurer le trafic : les avions sont déroutés sur un nombre réduit d'axes principaux, ils sont séparés par une distance donnée, fixe (par exemple 20 NM), et volent à vitesse homogène. Les contrôleurs s'assurent alors que les avions respectent leur vitesse et la distance de séparation. Ils ajustent la vitesse si nécessaire, mais ne changent pas le cap des avions. Le trafic, ainsi organisé, est plus simple à gérer par les contrôleurs aériens.

Quand les flux d'avions doivent être fusionnés, les avions provenant de chaque flux passent alternativement le point de rencontre. Dans le cas de la fusion de deux MIT de 20 NM, le flux résultant sera un MIT de 10 NM dont les avions proviendront alternativement du premier et du deuxième flux. Les avions provenant de deux flux passeront l'intersection alternativement en respectant une distance de 10 NM entre avions des deux flux, comme indiqué dans la figure 1.9.

Ainsi, les avions sont organisés en flux homogènes, cette situation imposant une charge de travail moins importante aux contrôleurs que s'ils devaient gérer chaque avion indépendamment dans une situation de trafic important. De plus, cela permet de réguler la fréquence d'arrivée des avions aux aéroports.

L'inconvénient est que les avions doivent effectuer un détour pour atteindre ces

routes aériennes, ce qui rallonge leur temps de vol. Par ailleurs, les avions doivent voler à la même vitesse pour maintenir la distance requise avec leurs voisins. Cette vitesse peut être différente de leur vitesse de croisière. Ces deux contraintes entrainent une augmentation de la consommation de carburant, ce qui a des conséquences économiques pour les compagnies aériennes. C'est pourquoi le MIT n'est mis en place qu'en cas de fort trafic.

Une méthode similaire est appelée Minutes-in-Trail (MINIT) [NBA] : la séparation des avions n'est pas exprimée en distance, mais en temps. Cette méthode est utilisée dans les zones des États-Unis sans couverture radar.

Les algorithmes des chapitres 3 et 4 proposent deux méthodes pour réguler du trafic organisé en Miles-in-Trail. Le premier de ces algorithmes est dédié à ce type de trafic, tandis que le deuxième peut réguler n'importe quelle configuration de trafic, ce qui inclut le Miles-in-Trail.

1.2 Systèmes de contrôle du trafic aérien

1.2.1 Systèmes utilisé dans chaque pays

La gestion du trafic aérien repose sur un ensemble de systèmes permettant aux contrôleurs aériens de surveiller et de gérer le trafic, et de communiquer avec les pilotes. Ces systèmes sont chargés entre autre de collecter et d'agréger les informations issues des systèmes de surveillance du trafic évoqués dans la section 1.1.2 (radars, etc.), puis de les montrer aux contrôleurs. Tous les pays n'utilisent pas le même système.

En France, il s'agit du système de Coordination AUtomatique du TRafic Aérien (CAUTRA). Celui-ci comprend le module de visualisation du trafic Operational Display System (ODS), qui affiche l'état du trafic aux contrôleurs. ODS est actuellement développé par Capgemini Technology Services. Il sera remplacé d'ici 2018 par le système 4-Flight [Min15a] développé conjointement par Thales et la DGAC.

Dans le reste de l'Europe, la plupart des pays sont équipés par des systèmes développés par trois entreprises. Selex [Sel14] fournit le système utilisé en Italie. Indra [Ind] équipe l'Espagne, l'Allemagne et la Norvège. Les systèmes de Thales sont utilisés dans un grand nombre de pays [Tha], incluant l'Irlande, la Belgique, le Danemark, la Suède et la Grèce. Les systèmes de ces trois acteurs sont également déployés dans d'autres pays à travers le monde (notamment en Asie et en Afrique).

1.2.2 Limites des systèmes actuels

Le trafic aérien est en constante augmentation depuis ses origines [DGA07]. Cet accroissement a été rendu possible par l'amélioration des procédures et des technologies mises en œuvre dans la gestion du trafic aérien et par l'augmentation du nombre de contrôleurs aériens.

Comme indiqué dans la section 1.1, l'amélioration de la précision des radar a permis en 50 ans de réduire les normes de séparation de 20 à 5 NM horizontalement et de 2 000 à 1 000 ft verticalement. Cela a contribué à augmenter la capacité du réseau de routes aériennes, puisque d'une part, les avions à même altitude peuvent voler plus près les uns des autres, et d'autre part, cela double le nombre de niveaux de vols utilisables.

Comme le montre la figure 1.10, le nombre de contrôleurs aériens a également augmenté au fil des ans pour compenser l'augmentation du trafic. Comme évoqué précédemment, un contrôleur aérien ne peut gérer qu'un nombre limité d'avions

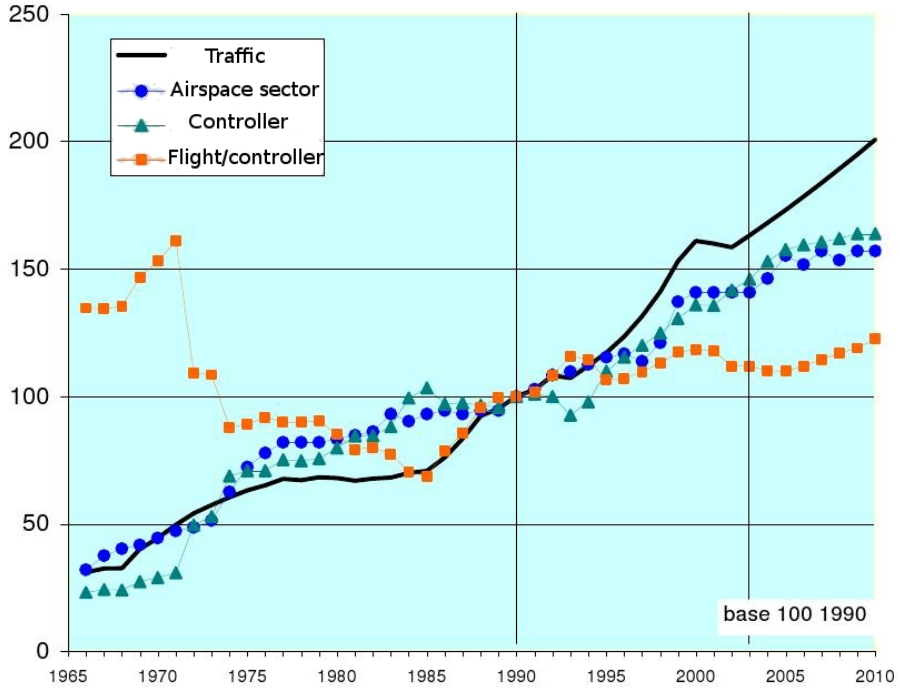


FIGURE 1.10 – Augmentation du trafic et du nombre de contrôleurs aériens entre 1965 et 2010 : le nombre d'avions par contrôleur est resté stable au cours des 50 dernières années.

simultanément (une dizaine environ). Ce nombre varie d'un secteur à l'autre, notamment en fonction de la configuration des routes qui le traversent. Chaque secteur est géré par une équipe de contrôleurs (deux en France). Pour augmenter le nombre de contrôleurs, il faut augmenter le nombre de secteurs, et donc réduire leur taille.

Un avion qui entre dans un secteur doit d'abord prendre contact avec le contrôleur par radio. Une fois le contact établi, le contrôleur peut gérer la trajectoire de l'avion pour résoudre les conflits. En sortant du secteur, le pilote clôture la communication et contacte le contrôleur du secteur suivant. Ces temps de transition (initiation et clôture des communications) sont incompressibles.

Le problème est qu'en réduisant la taille des secteurs, l'intervalle de temps pendant lequel le contrôleur peut agir sur un avion diminue. Cela, conjugué avec l'augmentation de la vitesse de croisière des avions, donne une limite inférieure à la taille d'un secteur, et donc une borne maximum au nombre de contrôleurs aériens [Ser+15]

Or, ces limites sont déjà atteintes dans beaucoup d'espaces aériens, et l'OACI estime que le trafic aérien doublera en 2030 par rapport à 2013 [ICA13a]. La figure 1.11 montre la croissance du trafic estimée par Eurocontrol en Europe à l'horizon 2050. Cette augmentation prévue du trafic ne peut plus être compensée par l'augmentation du nombre de contrôleurs.

Il est donc envisagé par les grands programmes de recherche en Air Traffic Management (ATM) à travers le monde d'automatiser certaines tâches jusque-là dévolues aux contrôleurs aériens, de manière à réduire leur charge de travail pour

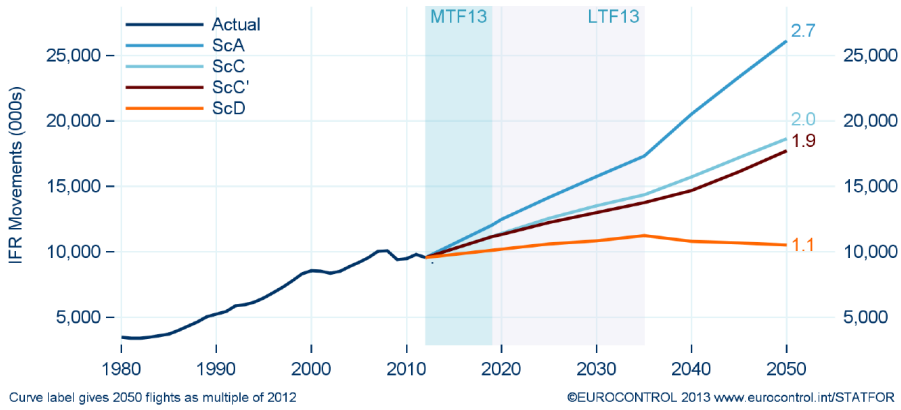


FIGURE 1.11 – Augmentation du trafic en Europe à l'horizon 2050 effectuée en 2013, selon différents scénarios.

augmenter le nombre d'avions qu'ils peuvent gérer simultanément. Les programmes Européen Single European Sky ATM Research (SESAR) [SES16] et Américain U.S. Next Generation Air Transportation System (NextGen) [Nex16] affichent leur volonté d'automatiser en partie le trafic aérien dans les prochaines décennies.

1.2.3 Aide à la décision

Un des moyens d'aider les contrôleurs aériens dans leur tâche est de leur fournir des systèmes d'aide à la décision.

1.2.3.1 Équilibrage de la demande du trafic en fonction de la capacité

Avant le décollage, un plan de vol est déposé et doit être validé par les autorités de contrôle aérien. Les plans de vol sont collectés par une division d'Eurocontrol appelée Network Manager (NM). Le Network Manager Operations Centre (NMOC) [Eurb], précédemment appelé CFMU (Central Flow Management Unit), est doté d'un système qui rassemble les plans de vol et estime la congestion des secteurs au cours de la journée, c'est-à-dire l'évolution du nombre d'avions les traversant. Quand ce nombre dépasse la capacité d'un secteur, ce système alerte les autorités concernées, qui peuvent retarder le décollage ou refuser le plan de vol.

1.2.3.2 Détection de conflits

Des systèmes de contrôle proposent des solutions à certains problèmes que doivent gérer les contrôleurs.

En-Route Air Traffic Organizer (ERATO) [DSN15 ; Min15b] est un système actuellement utilisé dans le CCR de Brest, et qui sera intégré au système 4-Flight de Thales. Il a pour but de détecter les conflits et de les présenter aux contrôleurs avec un ordre de priorité, mais aussi de leur permettre de tester différents scénarios de résolution (What-if), et de surveiller que les avions respectent les consignes données par les contrôleurs.

1.2.3.3 Mesure de la complexité du trafic

D'autres systèmes permettent de planifier la gestion du trafic à plus long terme. Pour assister les contrôleurs aériens, une des premières étapes est de mesurer la complexité du trafic dans l'espace aérien du point de vue des contrôleurs. L'établissement de cartes de complexité permet de chercher des axes d'amélioration dans la structuration actuelle du trafic, comme l'optimisation du découpage des secteurs aériens [Ser+15] Elle permet aussi d'estimer la complexité induite par la structuration du trafic proposée dans les travaux de recherche dédiés à la gestion du trafic aérien.

Comme indiqué dans la section 1.1, les contrôleurs aériens surveillent et régulent le trafic aérien durant la phase tactique. Chaque équipe de contrôleurs est en charge d'un secteur, et gère les trajectoires des avions qui le traversent. La charge de travail d'un contrôleur[Hil04], c'est-à-dire l'effort mental qu'il doit fournir pour accomplir ces tâches, dépend de plusieurs facteurs, incluant le nombre et la position des avions qu'il doit surveiller, ainsi que la configuration des routes aériennes (nombre de routes, position des points de croisement, etc.). La charge de travail maximale que peut fournir un contrôleur dépend également de plusieurs facteurs, comme son état de stress ou de fatigue. Elle conditionne aussi la capacité du réseau de routes aériennes, soit le nombre maximal d'avions pouvant traverser chaque région de l'espace aérien.

Il est difficile de mesurer la charge de travail. Certaines études dans le domaine des facteurs humains [LR12] se basent sur des questionnaires ou des mesures physiologiques [Bor+14], comme le rythme cardiaque (électrocardiogramme), l'activité du cerveau (électroencéphalogrammes), ou le mouvements des yeux grâce à l'eye-tracking [MCA11].

La charge de travail dans sa globalité est difficilement quantifiable. Cela implique de définir une métrique permettant de mesurer un phénomène physique. Or, les mesures physiques utilisées sont soit trop invasives pour être effectuées en centre de contrôle en situation opérationnelle (électrocardiogramme, électroencéphalogrammes), soit pas assez précises (eye-tracking).

S'il est difficile de mesurer la charge cognitive d'une tâche, il est par contre tout à fait possible de mesurer la complexité au sens mathématique d'une situation de trafic aérien [DP00]. Cette complexité permet d'estimer le risque d'apparition de conflits aériens. Plusieurs méthodes sont décrites dans la suite de cette section, et utilisent deux approches différentes pour estimer ce risque :

1. Plus la densité d'avions est importante dans un espace aérien, plus les avions sont proches en moyenne, et plus ils risquent d'entrer en conflit : c'est le principe de la mesure de la congestion du trafic (section 1.2.3.3.1).
2. Si les avions divergent ou suivent un même cap, le risque de conflit est plus faible que s'ils convergent les uns vers les autres : ce principe est utilisé par la mesure de la convergence (section 1.2.3.3.2) et celle de la complexité par exposants de Lyapunov (section 1.2.3.3.3).

La mesure de la complexité sert de base à l'élaboration de certains algorithmes, comme celui présenté dans le chapitre 5. Le but de celui-ci est la création de réseaux de routes locaux temporaires pour répondre à un accroissement local de la complexité du trafic. Le calcul de cartes de complexité lui permet de déterminer dans quelles zones l'algorithme devra créer ces réseaux de routes.

1.2.3.3.1 Congestion La métrique de complexité du trafic actuellement utilisée par le contrôle aérien est la congestion des secteurs aériens. Elle se base sur le principe

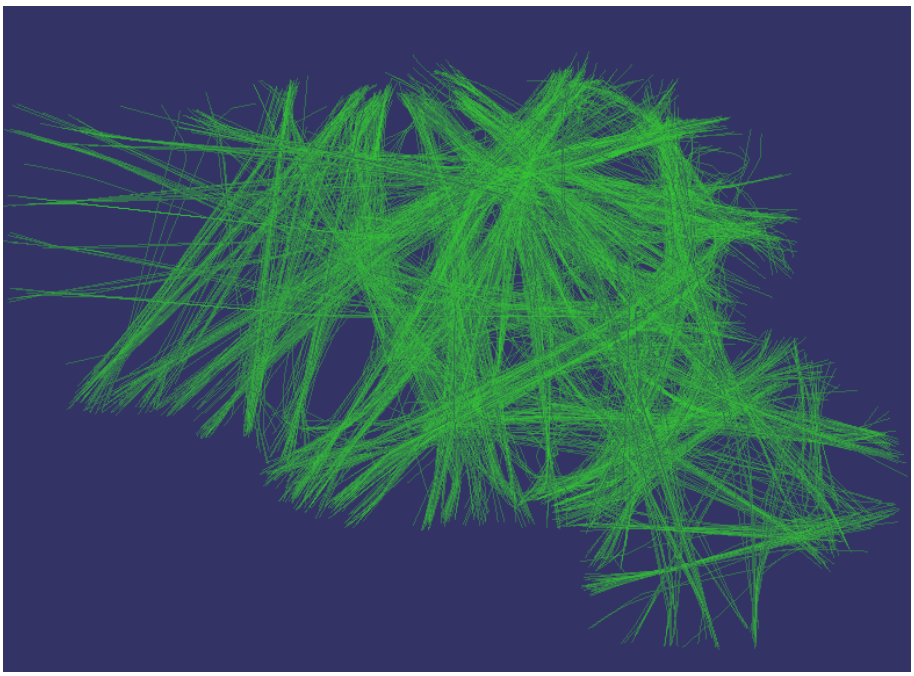


FIGURE 1.12 – Positions radar des avions enregistrées en France durant 24 heures. Les flux principaux d'avions sont visibles aux frontières et sur l'Atlantique, moins au dessus du territoire.

que plus le contrôleur aérien doit gérer d'avions en même temps, plus sa charge de travail est élevée. La congestion est utilisée notamment pour déterminer au cours de la journée si des secteurs doivent être groupés ou séparés. On compte le nombre d'avions traversant chaque secteur par tranche de 20 minutes. Quand ce nombre dépasse un seuil, les secteurs sont séparés, et quand il devient suffisamment bas, les secteurs sont regroupés. Ces seuils sont déterminés empiriquement pour chaque secteur par les contrôleurs.

Cette métrique permet aussi au Network Manager (le système qui centralise la gestion des plans de vol en Europe) de déterminer si le plan de vol d'un avion est accepté ou non : quand l'avion demande à traverser des secteurs trop chargés, le contrôle aérien lui demande de retarder son départ ou de changer sa route. Pour prendre en compte l'incertitude de la vitesse de l'avion au cours de son vol, l'avion est comptabilisé trois fois : une fois dans la période de 20 minutes où il est supposé traverser le secteur, mais aussi une fois dans la tranche horaire précédente et une fois dans la suivante.

Cette mesure de congestion des secteurs peut être affinée. Au lieu de compter le nombre d'avions traversant chaque secteur, on utilise un ensemble de positions radar [PN15] (figure 1.12). On lisse ensuite ces mesures ponctuelles pour obtenir une mesure continue. La figure 1.13 s'obtient en utilisant un estimateur par noyau, méthode utilisée en statistiques pour estimer une densité de probabilités à partir d'un ensemble de mesures ponctuelles [PN15].

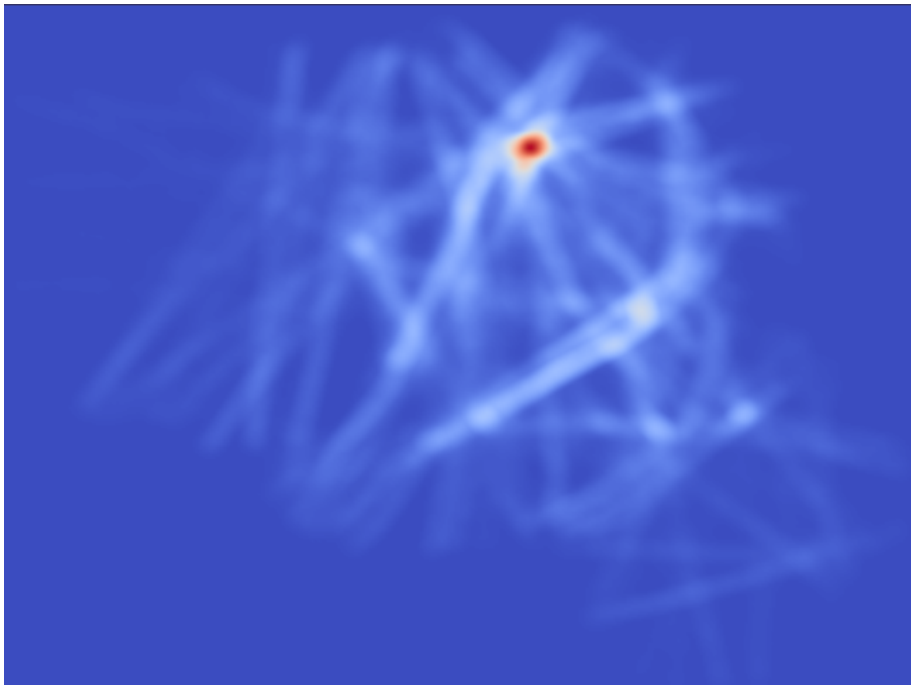


FIGURE 1.13 – Carte de densité calculée à partir des trajectoires de la figure 1.12. Certains flux intérieurs se détachent, comme sur les axes Toulouse-Lyon et Bordeaux-Paris. La région parisienne, qui concentre une grande partie du trafic français, se détache en rouge.

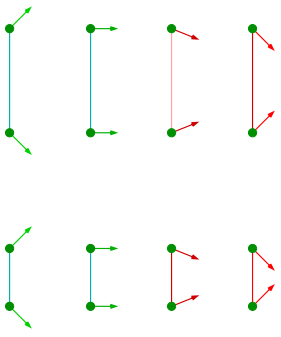


FIGURE 1.14 – Comparaison de plusieurs scénarios impliquant deux avions. Les avions divergents sont moins complexes à gérer que les convergents (situations de gauche à droite). La distance inter-avions influe aussi (de haut en bas).

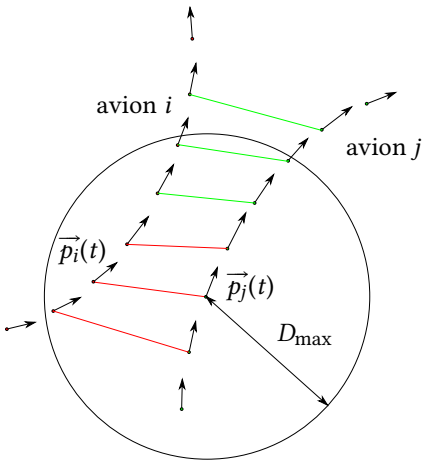


FIGURE 1.15 – Convergence calculée pour deux avions à un instant t.

1.2.3.3.2 Convergence La mesure de la densité est une première approche pour estimer la complexité du trafic. En effet, une situation où un grand nombre d'avions est concentré localement est plus difficile à gérer qu'une situation où peu d'avions sont présents. Par contre, comme le montre la figure 1.14, des situations similaires du point de vue de la congestion peuvent être perçues différemment par les contrôleurs. En raison du manque de finesse de la mesure de la densité pour évaluer la complexité du trafic, l'algorithme du chapitre 5 utilise à la place la méthode proposée par Delahaye et Puechmorel (2000) [DP00, Section 3] qui mesure le degré de convergence des avions.

En effet, des avions proches s'éloignant l'un de l'autre ne génèreront aucun conflit, et peuvent être ignorés sans risque par les contrôleurs aériens (qui les surveillent quand même pour s'assurer que ces avions ne convergeront pas de nouveau). Par contre, des avions moins proches mais convergeant vers une même position doivent être surveillés attentivement à cause du risque de conflit. La complexité induite est d'autant plus important que les avions convergent vite.

La convergence est calculée à un instant t pour chaque paire d'avions i et j, comme le montre la figure 1.15. À partir de la position des avions $\vec{p}_i(t)$ et $\vec{p}_j(t)$, on calcule le vecteur position relative :

$$\vec{p_{ij}}(t) = \vec{p}_j(t) - \vec{p}_i(t)\,. \tag{1.1}$$

De la même manière, à partir de leur vitesse $\vec{v}_i(t)$ et $\vec{v}_j(t)$, on calcule leur vecteur vitesse relative :

$$\vec{v_{ij}}(t) = \vec{v}_j(t) - \vec{v}_i(t)\,. \tag{1.2}$$

Le taux de divergence $r_{ij}(t)$ est la variation, ou la dérivée, de la position relative des avions :

$$\begin{aligned} r_{ij}(t) &= \frac{\mathrm{d}}{\mathrm{d}t}\|\vec{p_{ij}}(t)\| \\ &= \frac{\vec{p_{ij}}(t)\cdot\vec{v_{ij}}(t)}{\|\vec{p_{ij}}(t)\|} \\ &= \|\vec{v_{ij}}(t)\|\cos(\vec{p_{ij}}(t),\vec{v_{ij}}(t))\,. \end{aligned} \tag{1.3}$$

On considère que les avions divergents et éloignés de plus d'une distance $D_{\max}$ donnée n'apportent pas de complexité, et peuvent être ignorés. La métrique de conver-

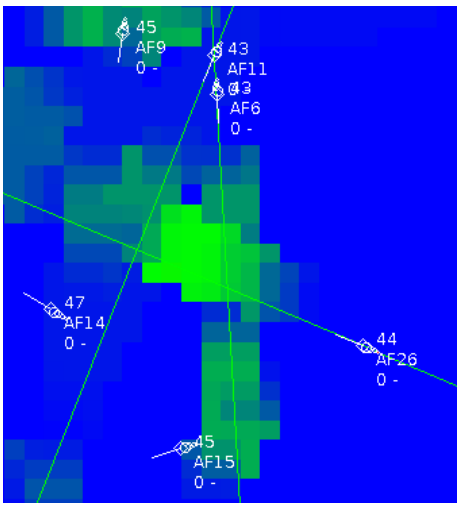


FIGURE 1.16 – Carte de convergence calculée à partir des positions futures estimées des avions. Les valeurs faibles sont en bleu, les fortes en vert.

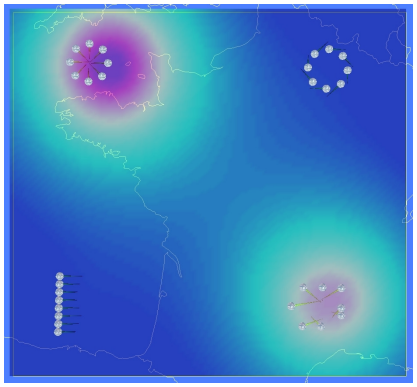


FIGURE 1.17 – Carte de convergence pour quatre groupes de huit avions

gence est donc mise à 0 quand une de ces deux conditions est vérifiée. Sinon elle est égale à $-r_{ij}(t)$, soit l'opposée de la divergence :

$$C_{ij}(t) = \begin{cases} -r_{ij}(t) & \text{si } r_{ij}(t) < 0 \text{ et } \|\vec{p_{ij}}(t)\| < D_{\max}, \\ 0 & \text{sinon.} \end{cases} \tag{1.4}$$

Ce calcul donne le taux de convergence de deux avions. On peut établir une carte de convergence de la même manière qu'une carte de densité, en considérant que la convergence mesurée est localisée à la position des avions (figure 1.15). On obtient un ensemble de mesures ponctuelles, qu'il faut alors lisser pour obtenir une représentation continue, comme la carte représentée figure 1.16.

1.2.3.3.3 Complexité par exposants de Lyapunov La mesure de la convergence est intéressante puisqu'elle permet d'obtenir un indice de la complexité du trafic avec

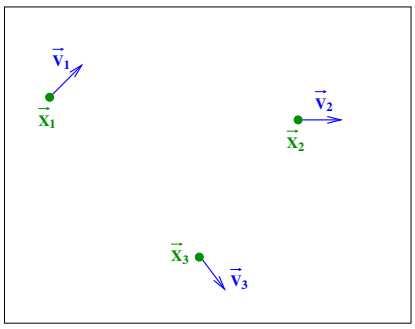


FIGURE 1.18 – Relevés radar associé à trois avions.

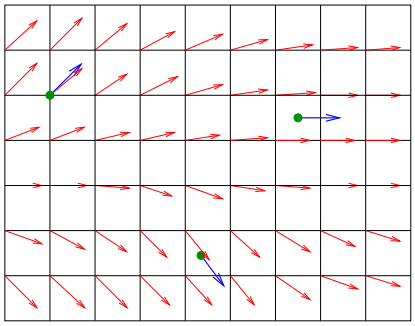


FIGURE 1.19 – Champ vectoriel induit par le système dynamique.

un calcul simple et rapide. C'est pourquoi elle est utilisée dans le chapitre 5. Elle souffre néanmoins de certains défauts. Comme le montre la figure 1.17, la convergence ne fait pas ressortir certaines situations complexes. Une situation où 8 avions convergent vers le même point (nord-ouest de la carte) génère beaucoup plus de convergence qu'une situation où 8 avions ont des caps aléatoires (sud-est de la carte), alors que la solution est plus simple à gérer dans le premier cas que dans le deuxième : dans le premier cas il « suffit » de demander à tous les avions de tourner à gauche (ou à droite). Le deuxième cas demande plus d'effort de réflexion.

Le scénario de la figure 1.17 soulève un autre problème : des avions tournant en cercle (comme dans un rond-point), au nord-est de la carte, ne sont même pas détectés par la métrique de convergence, alors que les contrôleurs doivent tout de même surveiller cette situation.

Delahaye et Puechmorel (2010) [DP10] proposent une meilleure manière de mesurer la complexité en déterminant le degré de désordre d'une situation de trafic aérien. Cette méthode est basée sur la mesure des exposants de Lyapunov associés aux vecteurs vitesse des avions.

Comme dans la mesure de la convergence, on considère un ensemble de positions radar associées au vecteur vitesse de l'avion (figure 1.18). On en déduit un modèle non-linéaire qui associe à chaque position $\vec{X}$ de l'espace un vecteur $\dot{\vec{X}}$ représentant

l'évolution du système en ce point. Le modèle non linéaire a la forme suivante :

$$\dot{\vec{X}} = \vec{f}(\vec{X}) \tag{1.5}$$

où f est une fonction d'évolution spatiale du modèle dynamique. On cherche alors une fonction $\vec{f}$ qui minimise le critère d'interpolation E_1 :

$$E_1 = \sum_{i=1}^{N} \|\vec{V}_i - \vec{f}(\vec{X}_i)\|^2 \tag{1.6}$$

Il faut noter que déterminer un système dynamique non linéaire respectant des données ponctuelles est toujours possible. En fait, il existe une infinité de fonction $\vec{f}$ permettant de minimiser le critère E_1 (avec $min(E_1) = 0$).

Pour obtenir une solution unique, il faut donc introduire un critère supplémentaire de régularité E_2 :

$$E_2 = \int_{\mathbb{R}^3} \left\{ \alpha \|\nabla \mathrm{div}\, \vec{f}(\vec{X})\|^2 + \beta \|\nabla \mathrm{rot}\, \vec{f}(\vec{X})\|^2 \right\} \mathrm{d}\vec{X} \tag{1.7}$$

La minimisation conjointe de E_1 et E_2 induit une fonction $\vec{f}$ unique [AB91] :

$$\vec{f}(\vec{X}) = \sum_{i=1}^{N} \Phi(\|\vec{X} - \vec{X}_i\|) \cdot \vec{a}_i + \boldsymbol{A} \cdot \vec{X} + \vec{B} \tag{1.8}$$

avec $\vec{a}_i$ vecteur paramètres. La matrice Φ (spline vectorielle associée) est donnée par :

$$\Phi(\|\vec{X} - \vec{X}_i\|) = \boldsymbol{Q}(\|\vec{X} - \vec{X}_i\|^3) \tag{1.9}$$

avec $\boldsymbol{Q}$ opérateur matriciel :

$$\boldsymbol{Q} = \begin{pmatrix} \frac{1}{\alpha}\partial^2_{xx} + \frac{1}{\beta}(\partial^2_{yy} + \partial^2_{zz}) & (\frac{1}{\alpha} - \frac{1}{\beta})\partial^2_{xy} & (\frac{1}{\alpha} - \frac{1}{\beta})\partial^2_{xz} \\ (\frac{1}{\alpha} - \frac{1}{\beta})\partial^2_{xy} & \frac{1}{\alpha}\partial^2_{yy} + \frac{1}{\beta}(\partial^2_{xx} + \partial^2_{zz}) & (\frac{1}{\alpha} - \frac{1}{\beta})\partial^2_{yz} \\ (\frac{1}{\alpha} - \frac{1}{\beta})\partial^2_{xz} & (\frac{1}{\alpha} - \frac{1}{\beta})\partial^2_{yz} & \frac{1}{\alpha}\partial^2_{zz} + \frac{1}{\beta}(\partial^2_{xx} + \partial^2_{yy}) \end{pmatrix} \tag{1.10}$$

De la même façon que dans le cadre linéaire, la régression du système dynamique non linéaire s'effectue par la méthode des moindres carrés à la différence que le nombre de paramètres à déterminer est beaucoup plus important ($\Longrightarrow \boldsymbol{A}$, $\vec{B}$, $\vec{a}_i$ $(i \in \{1, .., N\})$ soit un total de 3N+12 paramètres).

Ce modèle permet ainsi de construire un champ régulier qui s'ajuste parfaitement aux observations ($\min(E_1) = 0$). A partir de ce modèle, il est alors possible d'appliquer la théorie des exposants de Lyapunov afin de quantifier le niveau local d'organisation du champ de vecteur. Le principe des exposants de Lyapunov consiste à mesurer la sensibilité aux conditions initiales du champ de vecteur reconstitué. Pour cela on considère un point dans l'espace d'état ($\vec{x_0}$) et on observe sa trajectoire (γ), lorsqu'il est transporté par le champ (telle une poussière dans un champ de vent). On a alors :

$$\gamma(t, \vec{x_0}) = \vec{x_0} + \int_0^t \vec{f}(u, \gamma(u, \vec{x_0}))\mathrm{d}u \tag{1.11}$$

On considère ensuite une petite perturbation ($\vec{\epsilon}$) de la position initiale $\vec{x}_0$ pour laquelle on caractérise la trajectoire :

$$\begin{aligned} &\gamma(t, \vec{x}_0 + \vec{\epsilon}) = \gamma(t, \vec{x}_0) + \\ &\nabla_{\vec{x}} \vec{f}(\gamma(t, \vec{x})) \cdot \vec{\epsilon} + o(\|\vec{\epsilon}\|) \end{aligned} \tag{1.12}$$

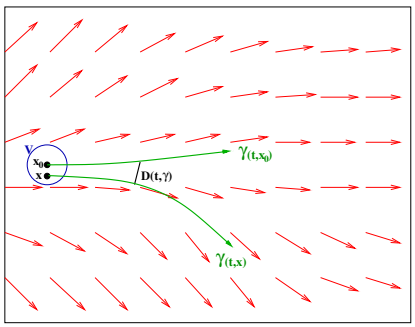


FIGURE 1.20 – Évolution temporelle de la trajectoire de référence et d'une trajectoire issue du voisinage en x_0.

où $\nabla_{\vec{x}}\vec{f}(t, \gamma(t, \vec{x}))$ est le gradient du champ $\vec{f}$ au point $\vec{x}$. On mesure ensuite la distance entre cette nouvelle trajectoire (issue de $\vec{x}$) et la trajectoire de référence en $\vec{x_0}$ (voir figure 1.20) :

$$\|\gamma(t, \vec{x_0}) - \gamma(t, \vec{x})\| = D(t, \vec{x}) \tag{1.13}$$

$\gamma(t, \vec{x})$ étant contrôlé par le champ vectoriel $\vec{f}$, on a :

$$\frac{\partial \gamma(t, \vec{x})}{\partial t} = \vec{f}(t, \gamma(t, \vec{x})) \quad \gamma(0, \vec{x}) = \vec{x} \tag{1.14}$$

il est alors possible de démontrer que la distance $D(t)$ est aussi régie par une équation différentielle :

$$\frac{\partial D(t, \vec{x})}{\partial t} = \nabla_{\vec{x}}(t, \gamma(t, \vec{x})).D(t, \vec{x}) \quad D(0, \vec{x}) = \|\vec{x} - \vec{x_0}\| \tag{1.15}$$

L'équation précédente étant linéaire et en considérant les trois dimensions de l'espace (x, y, z), il est possible d'effectuer son extension matricielle (équation variationnelle du champ) :

$$\frac{dM(t)}{dt} = \nabla_{\vec{x}}(t, \gamma(t, \vec{x})) \cdot M(t) \quad M(0) = \mathrm{Id} \tag{1.16}$$

Cette équation représente un système dynamique linéaire "tangent" au système initial (au point considéré). On calcule ensuite la décomposition en valeurs singulières de la matrice $M(t) = U^t(t)\Sigma(t)V(t)$. Les exposants de Lyapunov sont calculés en moyennant dans le temps les logarithmes de ces valeurs singulières (diagonale de la matrice $\Sigma(t)$) :

$$\kappa(\vec{x}) = -\frac{1}{T}\int_0^T \log(\Sigma_{ii}(t))\mathrm{d}t \quad \forall \Sigma_{ii}(t) \leq 1 \tag{1.17}$$

Lorsqu'un exposant a une valeur importante, il traduit une sensibilité importante aux conditions initiales (écart $\vec{\epsilon}$). Dans ce cas, les deux trajectoires $\gamma(t, \vec{x_0})$ et $\gamma(t, \vec{x_0} + \vec{\epsilon})$ suivies par deux particules en $\vec{x_0}$ et en $\vec{x_0} + \vec{\epsilon}$ sont très différentes. La situation future est donc très difficile à prédire dans la zone de calcul de cet exposant. A l'inverse, un exposant de Lyapunov ayant un valeur faible traduit une situation bien organisée facile à prédire. La carte des exposants de Lyapunov permet d'identifier les zones de l'espace aérien où le trafic est bien organisé (nécessite peu de surveillance) ainsi que les zones où le trafic est désordonné. Dans les zones ordonnées, les distances relatives

entre les avions restent inchangées au cours du temps ce qui traduit une situation stable sans modification dans un futur proche.

Les étapes de calculs d'une carte d'exposants de Lyapunov sont les suivantes :

1. Régression du système dynamique non linéaire à partir des N observations radar (position X_i et vitesse V_i). Ceci permet des fixer les N coefficients $\vec{a}_i$, ainsi que la matrice A et le vecteur $\vec{B}$.
2. Calcul sur chacun des points d'une grille 3D du gradient du champ de vecteur :$\nabla_{\vec{x}}\vec{f}$.
3. Calcul des exposants de Lyapunov en chaque point de grille à l'aide d'une intégration de Runge-Kutta :

$$\kappa(\vec{x}) = \frac{1}{L}\sum_{i=1}^{i=L} \|\nabla_{\vec{x}}(\gamma(t,\vec{x}))\|_2 \simeq \left(\frac{1}{L}\sum_{l=1}^{L} \log\left\{\max\left(\text{Sing Value}\left(\nabla_{\vec{x}}(\gamma(t,\vec{x}))\right)\right)\right\}\right) \quad (1.18)$$

où L représente le nombre de pas de temps d'intégration.

L'étape critique de ce calcul se situe au niveau de la régression du système dynamique non linéaire dont la complexité est $O[(3 \times (N+4))^3]$. En faisant $\alpha = \beta$, on remarque que l'opérateur différentiel matriciel se simplifie beaucoup :

$$\Phi(r) = Q(D)r^3 = \begin{bmatrix} \partial^2_{xx} + \partial^2_{yy} + \partial^2_{zz} & 0 & 0 \\ 0 & \partial^2_{xx} + \partial^2_{yy} + \partial^2_{zz} & 0 \\ 0 & 0 & \partial^2_{xx} + \partial^2_{yy} + \partial^2_{zz} \end{bmatrix} \quad (1.19)$$

Le problème devient alors :

$$\min\ E_1 = \sum_{i=1}^{i=N} \|\vec{V}_i - \vec{f}(\vec{X}_i)\|^2 \quad (1.20)$$

et

$$\min\ E_2 \int_{\mathbb{R}^3} \|\Delta\vec{f}(\vec{x})\|^2 \mathrm{d}\vec{x} \quad \text{avec} \quad \Delta\vec{f} = \begin{bmatrix} \frac{\partial^2 f_x}{\partial x^2} + \frac{\partial^2 f_x}{\partial y^2} + \frac{\partial^2 f_x}{\partial z^2} \\ \frac{\partial^2 f_y}{\partial x^2} + \frac{\partial^2 f_y}{\partial y^2} + \frac{\partial^2 f_y}{\partial z^2} \\ \frac{\partial^2 f_z}{\partial x^2} + \frac{\partial^2 f_z}{\partial y^2} + \frac{\partial^2 f_z}{\partial z^2} \end{bmatrix} \quad (1.21)$$

où $\Delta\vec{f}$ représente la Laplacien du champ vectoriel. On en déduit que

$$\Phi(r) = 12\begin{bmatrix} r & 0 & 0 \\ 0 & r & 0 \\ 0 & 0 & r \end{bmatrix} \quad (1.22)$$

Sous cette forme, la complexité se réduit en $O[3 \times (N+4)^3]$ et se trouve donc divisée par 9. De plus, il est possible de déterminer une forme analytique du gradient

spatial du champ :

$$\nabla_{\vec{X}}\vec{f}(\vec{X})\begin{bmatrix} \frac{\partial f_x}{\partial x} & \frac{\partial f_x}{\partial y} & \frac{\partial f_x}{\partial z} \\ \frac{\partial f_y}{\partial x} & \frac{\partial \Phi_y}{\partial y} & \frac{\partial \Phi_y}{\partial z} \\ \frac{\partial f_z}{\partial x} & \frac{\partial f_z}{\partial y} & \frac{\partial f_z}{\partial z} \end{bmatrix} = \boldsymbol{A} + \sum_{i=1}^{N} \begin{bmatrix} a_{ix} \\ a_{iy} \\ a_{iz} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial \Phi(r-r_i)}{\partial x} & \frac{\partial \Phi(r-r-i)}{\partial y} & \frac{\partial \Phi(r-r_i)}{\partial z} \end{bmatrix} \quad (1.23)$$

($\Phi(r)$ fonction scalaire dépendant de $r = \sqrt{x^2 + y^2 + z^2}$

$$\frac{\partial \Phi(r - r_i)}{\partial x} = 12.\frac{x - x_i}{\|\vec{X} - \vec{X_i}\|} \quad (1.24)$$

$$\frac{\partial \Phi(r - r_i)}{\partial y} = 12.\frac{y - y_i}{\|\vec{X} - \vec{X_i}\|} \quad (1.25)$$

$$\frac{\partial \Phi(r - r_i)}{\partial z} = 12 \cdot \frac{z - z_i}{\|\vec{X} - \vec{X_i}\|}. \quad (1.26)$$

La complexité globale de l'algorithme est alors la suivante :

— Régression $3 \times (N + 4)^3$ (point le plus critique)

— Reconstruction $N \times M$ avec M nombre de points de grille.

— Estimation du gradient M

— Calcul des exposants de Lyapunov $M \times L$ avec L nombre de pas de temps utilisés

Il faut préciser que la spline vectorielle solution de ce problème ($\Phi(r)$) n'est pas localisée en espace. Ainsi, la contribution des observations lointaines est plus importantes que la contribution des observations proches. Dans le cadre de notre application, et à titre d'exemple, cela veut dire que pour reconstituer le champ de vecteur au dessus de New-York, il nous faut prendre en compte le trafic lointain situé à San-Francisco par exemple. De fait, le nombre d'observations à prendre compte dans la régression des moindres carrés ne peut être réduite par des considérations de proximité spatiale.

Comme le montre la carte de la figure 1.21, cette méthode est plus précise que la mesure de la convergence sur les cas de la figure 1.16, mais au prix d'un temps de calcul beaucoup plus élevé. Il est néanmoins possible d'améliorer les performances en calculant les exposants de Lyapunov sur carte graphique [Tre+15].

1.2.3.4 Sectorisation

L'élaboration de cartes de complexité du trafic peut servir de base à des algorithmes de sectorisation automatique de l'espace aérien. Treimuth, Delahaye et Ngueveu [TDN14] proposent une méthode qui contrôle le groupement de secteurs en fonction de la complexité calculée par exposants de Lyapunov décrite dans [DP10 ; Tre+15].

Comme décrit dans la section 1.1.1.1, l'espace aérien est découpé en secteurs élémentaires. Quand la charge de travail des contrôleurs est faible, c'est-à-dire quand il y a peu d'avions, les secteurs élémentaires sont groupés, ce qui nécessite moins de contrôleurs. Quand le trafic augmente, les secteurs sont dégroupés.

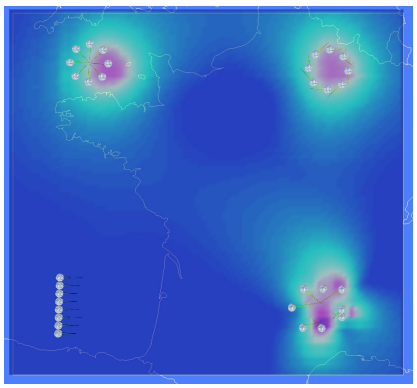


FIGURE 1.21 – Carte des exposants de Lyapunov pour quatre situations de trafic artificielles.

Ces décisions sont actuellement basées uniquement sur le nombre d'avions devant traverser les secteurs dans les prochaines minutes. La méthode décrite dans [TDN14] permet à la place de grouper les secteurs en fonction de la complexité mesurée par [DP10], cette métrique étant plus précise que celle actuellement en usage.

La méthode de [TDN14] gère le groupement de secteurs préétablis. Mais il peut également être intéressant de redéfinir la forme des secteurs. Dans le cadre du projet SESAR, Sergeeva *et al.* [Ser+15] proposent un outil d'aide à la décision permettant d'optimiser la sectorisation d'un espace aérien en respectant les contraintes liées à la forme des secteurs, notamment la durée minimale durant laquelle un avion doit le traverser, et la minimisation du nombre de ré-entrée d'un avion (un avion sort d'un secteur puis y entre de nouveau).

1.2.3.4.1 Étude statistique du trafic aérien Salaün *et al.* (2011) [Sal+12] étudient la structure du trafic aérien aux États-Unis d'un point de vue statistique. Les trajectoires des avions sont regroupées par similitude par un partitionnement en k-moyennes. Ce partitionnement permet d'obtenir l'ensemble des avions appartenant à ces flux principaux, ainsi que ceux qui dévient trop de ces flux pour y être intégrés. Ces données sont alors utilisées pour calculer une carte de proximité, donnant la probabilité que plusieurs avions se retrouvent à proximité, et augmentent le risque de conflit.

Les cartes résultantes sont similaires aux cartes de congestion. Par contre, cette étude donne un certain nombre d'informations permettant de générer un trafic aérien aléatoire réaliste, en donnant des indications sur la répartition spatiale et temporelle des avions dans les flux : décalage vertical et horizontal par rapport à la route, intervalle temporel séparant deux avions, etc.

1.2.3.5 Agrégation de trajectoires

Il est difficile d'étudier le trafic aérien en observant les trajectoires radar « brutes », comme celles de la figure 1.12. Ce type de représentation du trafic ne permet pas d'en visualiser l'organisation. Une manière de faire émerger les flux principaux d'avions consiste à agréger les trajectoires. En anglais, on parle de méthodes de bundling.

Certaines de ces méthodes consistent à maximiser localement la densité [GSF11 ; MHF12 ; Hur14]. Ces méthodes partent des positions radar enregistrées qui permettent de calculer une carte de densité, comme celle de la figure 1.13. Ces algorithmes déplacent alors chacun de ces points dans la direction des pics locaux de densité du trafic, tout en maintenant la cohésion des points appartenant à la même trajectoire.

Comme évoqué précédemment, l'algorithme présenté dans le chapitre 5 mesure la complexité du trafic en calculant des cartes de convergence. Il en déduit des zones où la complexité est localement élevée. Il crée alors des réseaux locaux temporaires de routes adaptés au trafic qui traverse ces zones. Pour créer un réseau de routes adapté à chaque situation, l'algorithme part des trajectoires des avions traversant la zone, et les agrège pour faire émerger le réseau.

Bien que les méthodes basées sur la maximisation de la densité soient relativement simples et efficaces, elles n'offrent pas de garantie sur la forme finale des trajectoires agrégées. La courbure de ces trajectoires peut être localement trop importante, et donc non compatibles avec les capacités des avions. Or, le réseau de routes créé par l'algorithme du chapitre 5 *doit* pouvoir être emprunté par les avions.

Puechmorel et Nicol [PN15 ; PN16 ; NP16] proposent une méthode de minimisation de l'entropie de la densité du trafic. Elle a l'avantage de générer des groupements de trajectoires dont la forme est conforme aux performances des avions, sans courbure excessive.

Dans le domaine de la théorie de l'information, l'entropie est le degré de désordre d'un système. Elle est généralement notée H (la lettre grecque Êta), et elle est calculée à partir d'une distribution de probabilité $d(\vec{x})$, $\vec{x}$ appartenant à un espace Ω :

$$H(\Omega) = -\int_{\Omega} d(\vec{x}) \log(d(\vec{x}))\, \mathrm{d}\vec{x}\,. \tag{1.27}$$

La densité de probabilité peut être calculée à partir d'un ensemble de positions radar grâce à un estimateur par noyau. Chaque trajectoire i peut être vue comme une fonction $\vec{p}_i(t)$ associant une position à chaque instant t. La valeur de t doit être normalisée sur l'intervalle $[0, 1]$, 0 correspondant à la première position de l'avion, et 1 à la dernière. L'estimateur par noyau permet de calculer en tout point $\vec{x}$ de l'espace une valeur de densité proportionnelle à la distance entre ce point $\vec{x}$ et l'ensemble des positions $\vec{p}_i(t)$. La valeur de la densité au point $\vec{x}$ est la moyenne des valeurs retournées par la fonction K_h, et proportionnelles à la distance entre chaque point $\vec{p}_i(t)$ et $\vec{x}$:

$$d(\vec{x}) = \frac{1}{N}\sum_{i=1}^{N}\int_0^1 K_h(\|\vec{x} - \vec{p}_i(t)\|)\, \mathrm{d}t \tag{1.28}$$

$$K_h(u) = \frac{2}{\pi h^2}\left(1 - \left(\frac{u}{h}\right)^2\right)\mathbf{1}_{\{|u|<h\}}\,. \tag{1.29}$$

$K_h(u)$ est la fonction noyau, qui associe à chaque distance u une valeur de densité, qui décroit avec la distance. Elle est paramétrée par la valeur de la fenêtre h, qui détermine sa taille. Dans le cas du noyau d'Epanechnikov, équation (1.29), la valeur retournée est de $1 - \left(\frac{u}{h}\right)^2$ quand u est inférieur à h, 0 sinon. Le coefficient $\frac{2}{\pi h^2}$ permet de s'assurer que l'intégrale du noyau soit égale à 1 dans l'espace à deux dimensions des positions radar, ce qui est un prérequis pour être utilisée dans un estimateur par noyau.

La taille de la fenêtre h doit être choisie avec soin. Si h est trop petit, les noyaux issus des points d'une même trajectoire ne seront pas en intersection, et la densité de

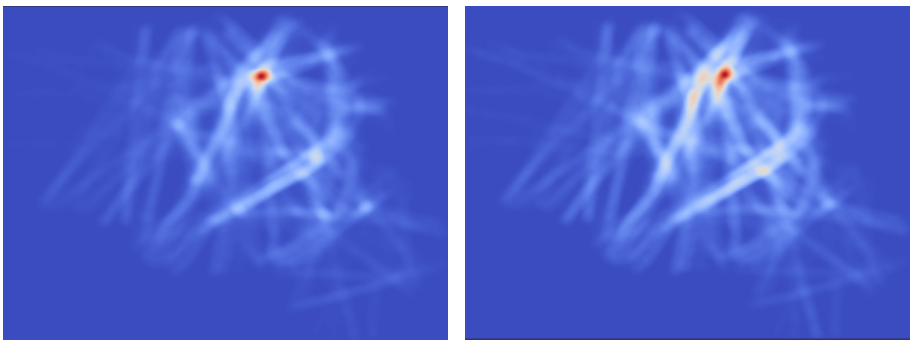


FIGURE 1.22 – Carte de densité issue d'une journée de trafic aérien Français. La densité n'est pas pondérée à gauche, et elle est pondérée par la vitesse à droite.

probabilité résultante sera discontinue. Et si h est trop grand, la densité résultante sera trop lisse, et il sera impossible de discerner les trajectoires individuelles. [PN16] recommande d'utiliser un noyau dont la taille est de 3 fois la distance moyenne qui sépare deux positions radar d'une même trajectoire.

Le calcul de la densité de probabilité par cette méthode souffre néanmoins d'un défaut. Au cours du processus, la trajectoire continue est échantillonnée avec un pas de temps fixe, par exemple en extrayant une position toutes les 10 secondes. Or, l'avion ne vole pas à vitesse constante : au décollage et à l'atterrissage, il est plus lent qu'en croisière. Cela implique que les positions aux environs des aéroports seront rapprochés, ce qui augmente artificiellement la densité.

Pour résoudre ce problème, la densité issue de chaque position $\vec{p}_i(t)$ doit être pondérée par la vitesse $\vec{v}_i(t)$ de l'avion :

$$d(\vec{x}) = \frac{1}{\sum_{i=1}^{N} l_i} \sum_{i=1}^{N} \int_0^1 K_h(\|\vec{x} - \vec{p}_i(t)\|) \|\vec{v}_i(t)\| \, \mathrm{d}t \,, \tag{1.30}$$

l_i étant la longueur totale de la trajectoire de l'avion i. Le résultat est visible dans la figure 1.22. À gauche, la densité de probabilité est calculée avec les équations (1.28) et (1.29). À droite, la densité est pondérée par la vitesse, conformément à l'équation (1.30). Le trafic aérien Parisien n'est plus le seul visiblement dense, et certains flux deviennent visibles, comme au dessus de l'Atlantique.

Une fois que la densité de probabilité est déterminée, il devient possible de calculer l'entropie liée à cette densité grâce à l'équation (1.27). L'entropie indique le degré d'organisation des trajectoires. Elle sera plus élevée quand la densité de probabilité est uniforme, ce qui veut dire que les trajectoires sont réparties uniformément dans l'espace aérien. L'entropie sera plus faible quand les trajectoires sont concentrées localement, par exemple quand elles sont organisées en flux, comme quand les avions suivent le réseau de routes aériennes.

Minimiser l'entropie d'un ensemble de trajectoires revient à les agréger, ce qui maximise localement la densité. Mais contrairement aux méthodes de bundling basées sur la maximisation de la densité, la minimisation de l'entropie garantit que les trajectoires résultant du processus d'agrégation sont toujours géométriquement correctes et peuvent être empruntées par des avions, comme le montre la figure 1.23.

[PN15] propose une méthode de minimisation de l'entropie par descente de gradient. La densité étant calculée à partir d'un ensemble de positions $\vec{p}_i(t)$ obtenues

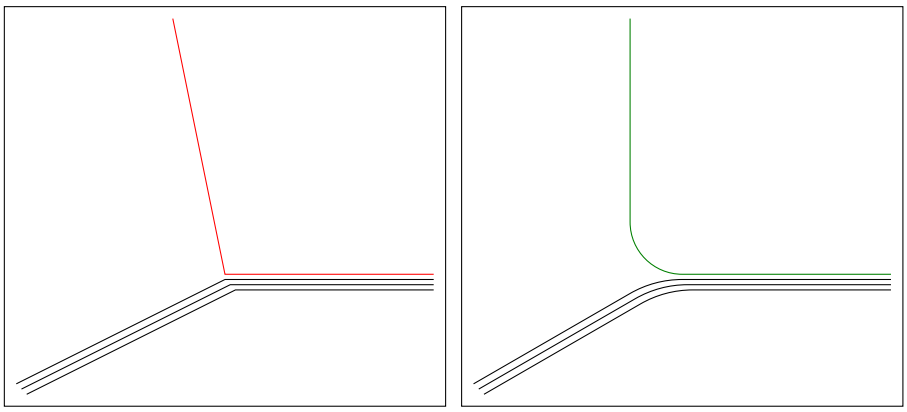


FIGURE 1.23 – À gauche : l'agrégation de trajectoires basée sur la densité peut en rendre certaines géométriquement incorrectes, non conformes aux performances des avions à cause de courbures excessives. À droite : l'agrégation basée sur l'entropie de la densité minimise la courbure, ce qui rend les trajectoires résultantes conformes aux capacités des avions.

par échantillonnage de la trajectoire de chaque avions i, il est possible de calculer la dérivée de l'entropie en chacun de ces points. Cette dérivée, notée $\vec{\eta}_i(t)$ est donnée par :

$$\vec{\eta}_i(t) = \int_\Omega \left(\frac{\vec{p}_i(t) - \vec{x}}{\|\vec{p}_i(t) - \vec{x}\|} \right)_{\mathcal{N}} K_h'(\|\vec{p}_i(t) - \vec{x}\|) \log d(\vec{x}) \, \mathrm{d}\vec{x} \|\vec{v}_i(t)\| \tag{1.31a}$$

$$- \left(\int_\Omega K_h(\|\vec{p}_i(t) - \vec{x}\|) \log d(\vec{x}) \, \mathrm{d}\vec{x} \right) \left(\frac{\vec{a}_i(t)}{\|\vec{v}_i(t)\|} \right)_{\mathcal{N}} \tag{1.31b}$$

$$+ \left(\int_\Omega d(\vec{x}) \log d(\vec{x}) \, \mathrm{d}\vec{x} \right) \left(\frac{\vec{a}_i(t)}{\|\vec{v}_i(t)\|} \right)_{\mathcal{N}} . \tag{1.31c}$$

avec $K_h'(u)$ la dérivée du noyau, $\vec{p}_i(t)$, $\vec{v}_i(t)$ et $\vec{a}_i(t)$ les vecteurs position, vitesse et accélération, et $()_{\mathcal{N}}$ la composante d'un vecteur normale à la trajectoire.

L'équation (1.31) permet de calculer le déplacement à appliquer en chaque point des trajectoires pour minimiser l'entropie. Une explication détaillée de son fonctionnement peut être trouvé dans [PN16]. La ligne (1.31a) déplace le point de manière à s'éloigner des minima de densité, ce qui les déplace vers les maxima locaux de densité. Les lignes (1.31b) et (1.31c) déplacent le point de manière à minimiser la courbure de la trajectoire. Le résultat obtenu à partir des trajectoires de la figure 1.12 est visible figure 1.24.

Dans sa forme la plus simple, comme exposée dans [PN15], cette méthode ne prend pas en compte la direction des flux. Par exemple, elle groupera les trajectoires des vols Toulouse-Paris avec les vols Paris-Toulouse. Or, il est important de ségréguer ces groupes, que ce soit pour la visualisation de données ou pour la création de réseaux de routes temporaires, comme dans le chapitre 5. En effet, dans cette application, grouper des flux de sens contraire créerait des conflits en face-à-face au lieu de simplifier le trafic.

Pour empêcher l'apparition de topologies de route où les avions volent en sens contraire, il faut prendre en compte le cap des avions dans le calcul de la densité.

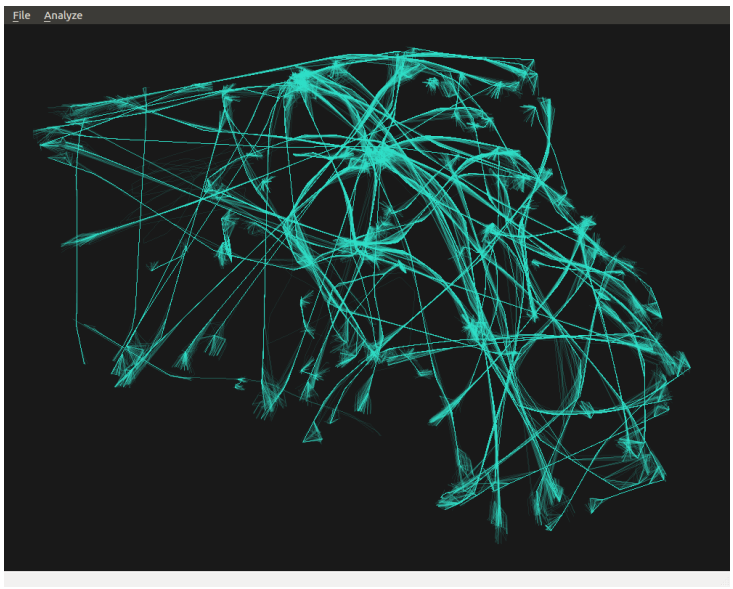


FIGURE 1.24 – Trajectoires de la figure 1.12 agrégées par la méthode de minimisation de l'entropie de la densité du trafic aérien.

Nicol et Puechmorel (2016) [NP16] proposent de calculer la densité dans un espace à quatre dimensions. Chaque point d'une trajectoire peut être représenté par un vecteur contenant la position et la vitesse de l'avion, soit (x, y, v_x, v_y). En utilisant l'algèbre de Lie, le triplet d'opérations [translation, rotation, mise à l'échelle] est calculé, qui permet de transformer un vecteur unitaire en le vecteur (position, vitesse) de départ.

Pour des trajectoires en deux dimensions, cela revient à transformer le quadruplet (x, y, v_x, v_y) en un quadruplet $\big(x, y, \theta, \log \|(v_x, v_y)\|\big)$, avec θ le cap de l'avion, et $\log \|(v_x, v_y)\|$ le logarithme de sa vitesse. La densité de probabilité est calculée à partir de ce quadruplet. Les coordonnées (x, y) sont représentées dans la densité de probabilité grâce à un noyau d'Epanechnokov, l'angle par une densité de probabilité circulaire (loi de von Mises), et la mise à l'échelle par une loi log-normale. Ainsi, minimiser l'entropie de cette densité de probabilité en 4D permet de tenir compte du cap des avions.

Cette méthode peut être utilisée dans plusieurs cas. Tout d'abord, elle permet de visualiser les flux principaux d'avions pour faciliter l'analyse des données par un humain. Elle peut également être utilisée pour classifier automatiquement les trajectoires en groupes aux caractéristiques similaires. Enfin, les trajectoires peuvent être agrégées pour créer automatiquement un réseau de routes, puisque les trajectoires, une fois agrégées, sont géométriquement correctes. Cette dernière possibilité est décrite plus en détail dans le chapitre 5.

1.2.4 Automatisation de la planification de la trajectoire des avions

À l'heure actuelle, aucun algorithme de résolution automatique de conflits aériens n'a été déployé dans les systèmes opérationnels. Seul le TCAS est déployé, mais il s'agit d'un système d'anti-collision, et pas de résolution de conflit, et il s'active uniquement dans des situations d'extrême d'urgence. Un obstacle majeur s'oppose

à l'utilisation d'algorithmes de résolution de conflit. Les standards de sécurité aéronautiques imposent que les algorithmes soient certifiés pour fonctionner dans toutes les situations. Or, il est extrêmement difficile de prouver le bon fonctionnement d'un algorithme dans toutes les situations, et cela dès que l'algorithme dépasse un niveau de complexité assez faible.

Par contre, les systèmes d'aide à la décision tels que ceux présentés précédemment permettent d'assister les contrôleurs dans leur prise de décision. D'autres outils agissent également sans qu'ils en aient besoin explicitement d'y faire appel. Ces outils d'automatisation agissent sans que les contrôleurs en aient conscience, soit pour simplifier le trafic à leur bénéfice, soit pour renforcer la sûreté du trafic aérien.

Une automatisation complète de la résolution de conflits permettrait d'augmenter la capacité du trafic dans les zones non contrôlées, comme les déserts et les océans. C'est également un prérequis pour implémenter le concept de Free Flight Airspace, décrit dans la section 1.1.1.3.

Des dizaines d'algorithmes de détection et de résolution automatique de conflits ont été développés ces dernières décennies. L'article de Kuchar et Yang (2000) [KY00] en dénombre 60 créés entre 1976 et 2000. Ils se différencient par le problème résolu (détection ou résolution de conflits, ou les deux), la méthode mathématique utilisée, ainsi que les types de manœuvres permis (changement de cap, d'altitude ou de vitesse) et la capacité ou non à résoudre des conflits impliquant plus de deux avions.

Plusieurs problèmes s'opposent au déploiement de ces outils dans les systèmes opérationnels. Le premier, évoqué précédemment, est le problème de la responsabilité. Quand le trafic aérien est géré par les contrôleurs aériens et leurs ordres exécutés par les pilotes, un accident peut arriver soit parce qu'un contrôleur a mal évalué la situation, soit parce que le pilote a mal exécuté la manœuvre (soit à cause d'une panne mécanique, ce qui est un autre problème). La faute peut donc être imputée à un acteur identifié. Or, en remplaçant les contrôleurs par des algorithmes, en cas d'accident, les responsabilités sont plus délicates à établir : est-ce la faute de la personne qui a rédigé les spécifications du logiciel, celle qui l'a programmé, celle qui a mené le processus de certification… ?

Une des manières de résoudre ce problème consiste à prouver que les algorithmes ne provoqueront pas d'erreur, ce qui est actuellement impossible au vu de leur complexité, bien plus élevée que le TCAS par exemple. Cela est dû notamment au fait que le problème de résolution de conflits est hautement combinatoire. En théorie, dans les modèles mathématiques développés, il faut détecter tous les conflits possibles entre toutes les paires d'avions. De manière générale, le nombre d'interactions entre avions correspond au carré du nombre d'avions (doubler le nombre d'avions quadruple le nombre d'interactions). Ce nombre est encore augmenté par l'incertitude de la position de l'avion, liée notamment à la vitesse du vent.

Cette complexité peut être abordée de différentes manières. L'idéal serait de modéliser fidèlement le problème et de le résoudre avec une méthode d'optimisation globale. L'optimisation mathématique consiste à modéliser un problème sous forme d'un ensemble de fonctions mathématiques, puis d'utiliser un solveur pour trouver la solution optimale du problème. Souvent on essaie de minimiser un coût ou de maximiser un profit, en terme de temps ou de quantité de carburant, par exemple.

Avec les méthodes exactes, comme la programmation linéaire, il est impossible de gérer plus de quelques avions à la fois. Suivant le problème, pour gérer plus d'avions, les temps de calculs nécessaires peuvent atteindre plusieurs siècles. Pour augmenter le nombre maximal d'avions pouvant être gérés simultanément, il faut donc simplifier le problème, soit utiliser des méthodes associées. Cela peut être fait soit en simplifiant le

modèle mathématique, soit en développant des heuristiques de résolution. Simplifier le modèle mathématique revient à faire des approximations de la réalité. Par exemple on peut considérer que les avions accélèrent instantanément, sans prendre en compte le temps nécessaire pour passer d'une vitesse à une autre. Il faut noter que cette simplification peut être valide à des échelles macroscopiques, quelques dizaines de NM ou de minutes en amont du conflit, mais pas à des échelles microscopiques, à échéance plus brève.

D'autre part, utiliser une heuristique permet de résoudre le problème (qu'il soit fidèle à la réalité ou simplifié) dans des temps de calcul compatibles avec les échéances opérationnelles, mais au prix de la qualité des résultats, qui peuvent être éloignés du résultat optimal théorique.

On qualifie d'heuristique (au sens large) un algorithme conçu pour trouver rapidement une solution à un problème. Cette solution est considérée comme valide, mais peut être éloignée de l'optimum. Les heuristiques sont souvent adaptées à la résolution d'un type de problème particulier. Par exemple, l'algorithme A* permet de trouver le chemin le plus court dans un graphe, comme un réseau de routes. La notion de distance varie d'un problème à l'autre : par exemple, pour planifier un trajet en voiture, on peut vouloir minimiser la distance parcourue ou le temps nécessaire au trajet. Pour construire le plus court chemin, l'algorithme A* se base sur des règles adaptées à la métrique choisie, qui permettent d'estimer à chaque étape de la résolution quel est le meilleur choix de route. Ces règles sont liées à un type de problème, et peuvent ne pas être applicable à un autre.

Une métaheuristique est une heuristique capable de résoudre n'importe quel problème d'optimisation, en trouvant une solution proche de l'optimum, sans forcément l'atteindre. Parmi les métaheuristiques couramment employées, on trouve notamment le recuit simulé (simulated annealing), les algorithmes évolutionnaires (comprenant les algorithmes génétiques), et l'optimisation par essaims de particules (particle swarm optimization, ou PSO).

Ces algorithmes sont basés sur des processus stochastiques : il ne peut pas être prouvé que l'algorithme trouvera la solution optimale de tous les problèmes qui lui sont soumis. Ils disposent simplement de preuves de convergence stochastique : sur un grand nombre d'exécutions, la solution trouvée est en moyenne située sur l'optimum, mais pour une exécution donnée, il n'y a aucune certitude.

Les méthodes de gestion automatisées du trafic aérien doivent donc trouver le bon équilibre entre modèle suffisamment précis et heuristique donnant des résultats de qualité suffisante dans des temps de calcul suffisamment courts. Cet équilibre dépend bien évidemment du problème à résoudre.

Les sections suivantes décrivent certains algorithmes qui ont été développés pour structurer le trafic aérien. Ceux-ci sont classés selon la phase de planification à laquelle ils s'appliquent (stratégique, pré-tactique, tactique, anti-collision). Ces phases successives de planification permettent d'organiser progressivement le trafic, de manière à en assurer la fluidité et la sécurité en maintenant les distances de séparation entre avions.

La phase stratégique se déroule de plusieurs mois à quelques heures avant le vol, et permet la structuration macroscopique du trafic : élaboration du réseau de routes, sectorisation de l'espace aérien, ouverture des lignes aériennes par les compagnies.

La phase pré-tactique commence quelques heures avant le décollage, et permet de structurer le trafic plus finement pour diminuer la probabilité d'interaction entre les trajectoires, afin de réduire le nombre de conflits aériens que les contrôleurs devront résoudre durant la phase tactique.

Durant la phase tactique, les contrôleurs (ou les algorithmes) surveillent le trafic et résolvent les conflits aériens.

Enfin, si les conflits ne sont pas résolus, les algorithmes d'anti-collision s'assurent en dernier recours que les avions restent séparés spatialement.

1.2.4.1 Phase stratégique

Delahaye *et al* (1996) [Del+96] proposent un algorithme dédié à la sectorisation de l'espace aérien, c'est-à-dire la définition de la forme des secteurs. La sectorisation est réalisée grâce à un diagramme de Voronoï, l'algorithme génétique déterminant la position du centre de chaque cellule en fonction des contraintes du problème. Cette méthode sert de base à [Ser+15], déjà évoqué dans la section 1.2.3.4.

Durant le décollage et l'atterrissage, les avions doivent suivre certaines procédures particulières, notamment en respectant strictement un réseau de routes qui évite par exemple les zones résidentielles ou les infrastructures stratégiques, comme les centrales nucléaires. On parle de Standard Instrument Departure (SID) au décollage, et de Standard Terminal Arrival Routes (STAR) à l'atterrissage. Zhou *et al.* (2016) [Zho+16] proposent une méthode d'optimisation de ces réseaux de routes en utilisant une hybridation de branch-and-bound et de recuit simulé.

Chaimatanan (2014) [Cha14] propose une méthode de résolution de conflits durant la phase stratégique à l'échelle d'un continent (±30 000 vols par jour en Europe), grâce à un algorithme de recuit simulé. Les avions peuvent retarder leur départ, changer de route, ou changer d'altitude. L'algorithme prend en compte l'incertitude temporelle le long des trajectoires.

1.2.4.2 Phase pré-tactique

Delahaye *et al.* (1996) [Del+96] proposent une méthode d'allocation des flux d'avions sur un réseau de routes. Chaque flux d'avion est défini par une paire d'aéroports origine-destination appartenant au réseau de route. L'assignation des flux permet de définir sur quelles routes du réseau chaque flux doit circuler en minimisant les interactions entre les flux.

Bertsimas et Patterson (1998) [BP98] ont développé un algorithme d'allocation de créneaux de décollage, dont le principe ressemble au fonctionnement du Network Manager en Europe, qui peut retarder le décollage d'un avion pour ajuster la demande à la capacité des secteurs que son plan de vol lui fait traverser.

Allignol (2011) [All11] résout les conflits en ajustant les retards au décollage des avions ainsi que leur niveau de vol. Deux méthodes d'optimisation globale sont proposées : un algorithme génétique, et une programmation par contrainte.

1.2.4.2.1 Projet ERASMUS La plupart des algorithmes de résolution de conflits ont été pensés pour fonctionner de manière complètement autonome, sans intervention des contrôleurs. [KY00] dresse une liste représentative de ces algorithmes. Ceux-ci appliquent les mêmes types de manœuvres que les contrôleurs : principalement des changements de cap pour de courtes périodes, mais aussi de vitesse et d'altitude.

Or, il est plutôt envisagé à l'heure actuelle de faire collaborer les algorithmes et les contrôleurs aériens. Selon ces derniers, ces algorithmes peuvent interférer avec leurs propres décisions puisque ces deux acteurs prennent les mêmes types de décisions dans le même espace aérien, au même moment [Vil04].

En 2004, une nouvelle approche a été proposée pour résoudre ce problème, dans le cadre du projet ERASMUS (En Route Air traffic Soft Management Ultimate System). D'après Jacques Villiers [Vil04], à l'origine du projet, ces algorithmes ne devraient pas essayer de reproduire ce que les contrôleurs font, mais supprimer des conflits avant la phase tactique sans en informer les contrôleurs, de manière « subliminale ». De petits changements de vitesse imposés aux avions permettent à ce système automatisé d'organiser le trafic de manière à réduire le nombre de conflits, et crée des conditions de trafic « chanceuses », plus faciles à gérer pour les contrôleurs. Ce concept a été validé par des expérimentations [BDG09] et des études de facteurs humains [CL07 ; CCL10]. La méthode de régulation de la vitesse a été implémentée sous forme d'algorithme génétique [BDG09].

L'algorithme décrit dans le chapitre 4 propose une implémentation du projet ERASMUS sous forme de système multi-agents.

1.2.4.2.2 Optimisation d'une seule trajectoire en-route En phase de croisière, d'autres articles traitent de l'optimisation d'une seule trajectoire. Dougui *et al.* (2012) [Dou+12] proposent une méthode de génération de trajectoire basée sur un modèle de propagation d'un rayon lumineux. En effet, la lumière se propage en suivant des géodésiques (trajectoires les plus courtes en temps), c'est-à-dire en ligne droite dans le vide, et en contournant les matériaux à fort indice optique qui la ralentissent. Par analogie, l'algorithme construit des trajectoires qui contournent des zones marquées comme plus « denses », comme les zones congestionnées, ou des obstacles fixes et mobiles. L'algorithme est itératif : à chaque étape, la source lumineuse émet plusieurs ondelettes, qui sont plus ou moins ralentis par le milieu qu'ils traversent. L'ondelette la plus proche de la destination devient alors la nouvelle source, et le processus recommence jusqu'à atteindre la destination.

Girardet *et al.* (2013) [Gir+13] proposent une méthode similaire de planification de trajectoire d'avion qui optimise le temps de parcours en tenant compte du vent, des zones à éviter et des zones congestionnées. Il s'agit d'une méthode de propagation de front d'onde de type « ordered upwind » [SV01] qui a été étendue au modèle sphérique. Un front d'onde part de la destination. À chaque étape, le front s'agrandit, la distance parcourue par le front variant localement selon le vent rencontré : le front parcourt plus de distance aux endroit où l'avion rencontrera un vent arrière, et moins de distance en cas de vent de face et de zones à éviter ou congestionnées. Quand le front atteint le point de départ de l'avion, il suffit de remonter de proche en proche les fronts successifs pour déterminer la trajectoire.

Cette méthode de propagation de front d'onde permet d'optimiser la trajectoire d'un avion, en prenant uniquement en compte les autres pour déterminer les zones congestionnées. Girardet [Gir14] propose également de coupler cette méthode à un recuit simulé pour optimiser les trajectoires de plusieurs avions. À chaque étape du recuit simulé, une trajectoire est tirée aléatoirement, puis optimisée. Le recuit simulé accepte ou rejette cette modification en fonction de son impact sur le trafic.

La planification de la trajectoire des drones fait également l'objet de nombreuses recherches. Par exemple, Anderson, Beard et McLain (2005) [ABM05] décrivent une méthode de planification de trajectoires de drone suivant un plan de vol.

De manière générale, l'optimisation des trajectoires cherche à minimiser un coût associé au vol. Les méthodes développées cherchent souvent à minimiser le temps de vol, tout en évitant les conflits ou en minimisant la congestion, selon l'horizon temporel étudié. Mais une autre source de coût est plus rarement prise en compte.

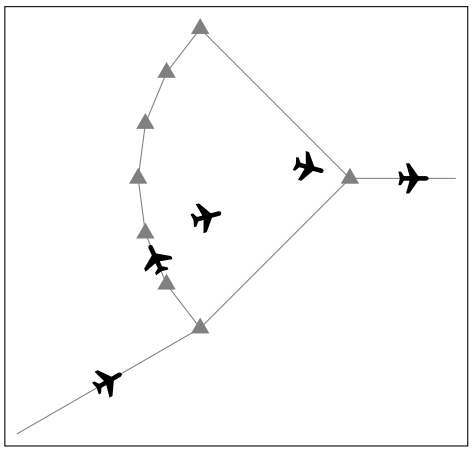


FIGURE 1.25 – Topologie Point Merge.

Il s'agit de la redevance de navigation aérienne [Min11]. À chaque fois qu'un avion traverse un secteur, il doit verser une taxe à l'ANSP concerné. Le montant de la redevance permet d'ailleurs dans certains cas d'inciter les compagnies aériennes à traverser un pays plutôt qu'un autre. Bonami *et al.* (2013) [Bon+13] proposent une méthode d'optimisation d'une trajectoire par contrôle optimal linéaire qui prend en compte les coûts liés au survol des secteurs aériens.

1.2.4.3 Phase tactique

Durant la phase tactique, les contrôleurs aériens ou les algorithmes doivent détecter et résoudre les conflits aériens durant toutes les phases du vol : roulage, décollage, atterrissage et croisière.

1.2.4.3.1 Optimisation de trajectoires au roulage, au décollage et à l'atterrissage

Avant de décoller et après avoir atterri, un avion doit emprunter un réseau de routes au sol, les *taxiways*. Ils sont dirigé depuis la tour de contrôle par des contrôleurs qui s'assurent que les avions au roulage n'entrent pas en collision. Deau, Gotteland et Durand (2009) [DGD09] proposent une méthode de résolution de conflits au sol qui prend en compte l'heure de départ prévue. L'optimisation de la trajectoire des avions est effectuée par un algorithme génétique.

D'autres travaux de recherche se consacrent à de nouvelles topologies de routes à l'approche des aéroports. Une de ces topologies, nommée Point Merge, visible dans la figure 1.25, est utilisée pour l'approche de certaines zones aéroportuaires très congestionnés, comme Paris. Cette topologie permet de rallonger le temps de vol de manière à espacer suffisamment les avions avant l'atterrissage. Les avions empruntent une route en arc de cercle centré sur le point de sortie de la topologie. Au moment opportun, le contrôleur aérien dirige l'avion directement vers le point de sortie, avant que celui-ci n'arrive à l'extrémité de l'arc de cercle. Dans des conditions de trafic peu dense, les avions se dirigent en ligne droite du point d'entrée vers le point de sortie sans passer par l'arc de cercle. Quand le trafic est dense, les avions sont redirigés vers l'arc de cercle, qui sert de zone tampon pour s'assurer que les distances de séparation sont respectées.

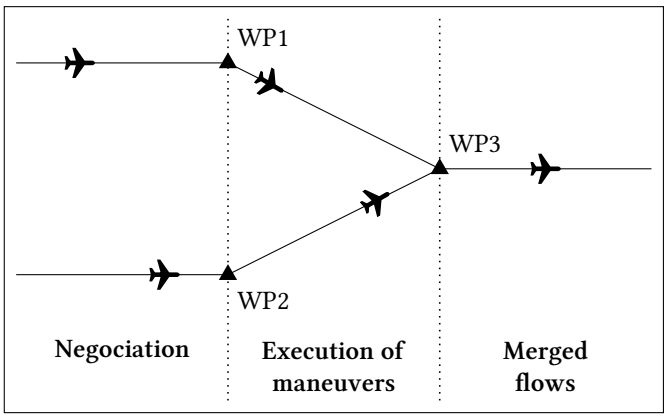


FIGURE 1.26 – Topologie de route décrite par Chipalkatty *et al.* [Chi+12].

Liang *et al.* (2016) [Lia+16] proposent une méthode de gestion automatique des avions dans une configuration de Point Merge multi-niveaux, les avions de différentes catégories (léger, moyen, lourd, super lourd) étant séparés verticalement dans la topologie. La topologie possède aussi plusieurs points de sortie, ce qui permet de desservir plusieurs pistes d'atterrissage.

Certains articles décrivent des méthodes de résolution de conflits à l'atterrissage, dans les TMA. Par exemple, Ma *et al.* (2016) [Ma+16b ; Ma+16a] proposent une méthode de résolution de conflits par régulation de la vitesse, de l'heure de décollage et du choix de la piste de décollage ou d'atterrissage. Le problème est résolu par un recuit simulé à fenêtre temporelle glissante : l'algorithme est optimise la trajectoire des avions volant durant une tranche horaire de deux heures, puis la fenêtre glisse d'une demi-heure, et l'algorithme recommence en intégrant les nouveaux vols sans modifier les vols terminés.

Chipalkatty *et al.* (2012) [Chi+12] proposent une méthode de résolution de conflits collaborative à l'atterrissage, qui comporte quelques points communs avec la méthode décrite dans la section 3. Dans cet article, la vitesse et la date d'arrivée sont régulées dans une TMA, sur un réseau de routes où plusieurs flux d'avions sont fusionnés avant l'atterrissage, comme le montre la figure 1.26. Les avions proviennent de deux routes se terminant aux waypoints WP1 et WP2. Les avions se dirigent ensuite vers le point de jonction WP3 où les deux flux sont fusionnés sur une seule route.

Les avions peuvent exécuter deux types de manœuvre entre le point WP1 ou WP2 et le point WP3 : changer de vitesse ou rallonger leur trajectoire en changeant temporairement de cap, de manière à retarder l'heure d'arrivée au point WP3. Pour ce faire, les avions négocient deux à deux avant d'atteindre les points WP1 et WP2 : le premier avion qui atteindra WP3 négocie avec le suivant. Quand une solution est trouvée, les décisions du premier avion sont figées. Le second avion négocie alors avec le troisième, et ainsi de suite jusqu'à ce que tous les conflits soient résolus. Les décisions sont prises de manière qu'après WP3, tous les avions volent à la même vitesse et respectent les normes de séparation.

Durant la phase de négociation, les deux avions exécutent itérativement un processus d'optimisation qui minimise le rallongement de trajectoire, la variation de vitesse et le retard à l'arrivée, jusqu'à ce que le conflit soit résolu.

Même si la méthode décrite dans [Chi+12] et l'algorithme décrit dans le chapitre 3

partagent quelques points communs, ils divergent sur certains points. Tout d'abord, le problème auquel ils s'appliquent est différent : [Chi+12] régule le trafic dans les TMA, où les avions ralentissent avant d'atterrir, les algorithmes décrits dans cette thèse s'appliquent au trafic en route. Les contraintes de ces deux problèmes sont différents : dans les TMA, les avions volent à altitude plus faible, ce qui augmente leur intervalle de vitesses admissibles. Ce détail garantit que tous les avions sont capables de voler à la vitesse requise après le waypoint WP3. Les distances de séparation sont aussi différentes et dépendent de la catégorie des avions : par exemple cette distance sera plus courte quand un avion lourd (Airbus A350) suit un avion léger (A320) que dans la situation inverse. En route, la distance de séparation minimale est de 5 NM quelque soit la catégorie des avions.

Les deux méthodes régulent la vitesse en calculant les dates d'arrivée au plus tôt et au plus tard au waypoint suivant. Mais le scénario considéré dans [Chi+12] permet un second type de manœuvre, l'allongement de trajectoire, qui augment le degré de liberté du processus de décision. D'un autre côté, l'algorithme décrit dans le chapitre 3 actualise les décisions en permanence, même après qu'un conflit entre deux avions ait été résolu, de manière à prendre en compte un troisième aéronef qui forcerait les deux premiers à adapter leur plan.

Dans [Chi+12], un avion peut prendre jusqu'à trois décisions pour résoudre un conflit (deux changements de vitesse entre WP1/2 et WP3 et un rallongement de trajectoire), cela donne de meilleurs résultats en terme de nombre de manœuvres nécessaires à la résolution des conflits (voir la section 3.3). Mais une fois prise la décision ne peut plus être modifiée. Au contraire, les algorithmes décrits dans cette thèse adaptent constamment leurs décisions pour prendre en compte les perturbations (la panne d'un avion, un nouvel avion entrant dans la zone, etc.).

1.2.4.3.2 Résolution de conflits en phase de croisière Dans cette section, un ensemble d'algorithmes de détection et de résolution des conflits aériens est présenté. Ils sont divisés en deux groupes. Le premier présente des méthodes exactes de résolution de conflits. Ces algorithmes utilisent des méthodes d'optimisation globale (programmation linéaire et non-linéaire). Cela garantit que ces algorithmes trouveront la solution optimale au problème posé si elle existe. Par contre, à cause de la combinatoire élevée du problème de résolution de conflits, ces algorithmes ne peuvent gérer que quelques trajectoires à la fois.

Le deuxième groupe rassemble les algorithmes basés sur des méthodes heuristiques ou méta-heuristiques. Ici, il n'y a pas de preuve théorique que les algorithmes trouveront la solution optimale, mais on observe expérimentalement que les solutions trouvées s'en approchent. Ces méthodes ont cependant l'avantage de pouvoir gérer un grand nombre d'avions simultanément, avec un temps de calcul compatible avec les contraintes opérationnelles.

Méthodes exactes Une fois qu'une trajectoire a été planifiée, il faut l'intégrer au trafic aérien environnant. Il faut donc la modifier, de manière à prendre en compte les autres avions, entre autre pour éviter les conflits. Certains algorithmes de régulation de trajectoires font appel à des méthodes d'optimisation globale déterministes (programmation linéaire et non-linéaire, programmation par contrainte, etc.). Comme cela a déjà été dit, ces méthodes étant très gourmandes en temps de calcul, les scénarios d'expérimentation impliquent souvent un petit nombre d'avions.

Niedringhaus (1995) [Nie95] propose une méthode de résolution de conflits en résolvant un problème de programmation linéaire. Des changements de cap sont imposés aux avions. Les contraintes du problème sont linéarisées : par exemple, la distance minimale de séparation n'est pas matérialisée par un cercle, mais par un polygone circonscrit à ce cercle. La puissance des ordinateurs de l'époque permettait de réguler une vingtaine d'avions.

Frazzoli *et al.* (1999) [Fra+99] formulent un problème quadratique pour résoudre les conflits, en minimisant l'écart à la vitesse optimale des avions. Cette méthode permet de réguler la vitesse et le cap des avions. Le scénario étudié implique deux flux de quatre avions en intersection.

Bicchi et Pallottino (2001) [BP01] proposent de résoudre les conflits entre deux avions grâce à une méthode de contrôle optimal.

L'algorithme de Cobano *et al.* (2012) [Cob+12] permet la détection et la résolution de conflits par régulation de la vitesse des avions. Les conflits sont détectés à l'aide d'une grille spatiale. Les avions positionnés sur des cases proches sont considérés en conflit, sur la même case en collision. Une fois les conflits détectés, les changements de vitesse sont planifiés par optimisation d'un problème quadratique.

Cafieri et Durand (2014) [CD14] décrivent une méthode de résolution de conflits basée sur un problème d'optimisation linéaire en nombres entiers, qui régule la vitesse des avions. Cellier, Cafieri et Messine (2013) [CCM13] proposent une méthode similaire, mais basée sur la résolution d'un problème de contrôle optimale, pour résoudre un conflit entre deux avions.

Ehrmanntraut (2005) [Ehr05] propose une méthode de résolution de conflits par application d'un décalage (offset) latéral à la trajectoire de l'avion par rapport à son plan de vol. L'avion vole donc à quelques miles à gauche ou à droite de sa trajectoire initiale durant une partie de son vol.

Méthodes heuristiques et méta-heuristiques Les méthodes d'optimisation globale déterministes ne peuvent pas gérer plus de quelques avions (ou quelques dizaines) à la fois. Pour en gérer plus, d'autres méthodes sont basées sur des heuristiques ou des méta-heuristiques.

Alliot *et al.* (1998) [ADG98 ; AGD04] proposent un algorithme de résolution de conflits de manière séquentielle. Les avions sont classés par ordre de priorité. Le premier avion suit sa trajectoire prévue. Le deuxième adapte alors sa trajectoire pour éviter les conflits avec le premier. Le troisième adapte la sienne pour résoudre les conflits avec les deux premiers, et ainsi de suite jusqu'à ce que toutes les trajectoires aient été intégrées.

Durand, Alliot *et al* (1996–1997) [Del+96 ; DAN96 ; DA97 ; GDA97] proposent une méthode de résolution des conflits aériens grâce à un algorithme génétique qui dévie temporairement les avions ou change leur altitude.

La méthode de planification de trajectoire par propagation de lumière de Dougui *et al.* (2012) [Dou+12] permet de résoudre les conflits en utilisant une approche séquentielle : la trajectoire du premier avion est planifiée, puis celle du deuxième est calculée en considérant la première comme un obstacle, etc.

Kosecka *et al.* (1997-1998) [Kos+97 ; Kos+98] modélisent les avions et leurs destinations comme des particules chargées électriquement. Les avions sont attirés par leur destination, et se repoussent entre eux. Si les avions ne peuvent pas voler selon la trajectoire obtenue, ils changent de niveau de vol de manière à permettre à l'algorithme de calculer une trajectoire valide.

Hoekstra, Ruigrok *et al.* (1998–2016) [HVR98] proposent une méthode d'auto-séparation des avions appelée Modified Voltage Potential (MVP). Un avion qui connaît la position et le cap de ses voisins détecte les conflits et calcule une manœuvre minimale d'évitement. Cette méthode détermine, à partir des vecteurs vitesse des deux avions, l'accélération à appliquer pour éviter le cylindre de séparation situé autour du voisin au moment où les deux avions sont les plus proches. Ce vecteur accélération permet de déterminer le changement de cap (composante perpendiculaire à la trajectoire) et le changement de vitesse (composante longitudinale).

Les articles suivants décrivent l'application de la méthode MVP à la résolution de conflits par les avions, sans intervention d'un contrôleur aérien (Airborne Separation Assurance System), comme dans les Free Flight Areas. [RVH99] décrit une expérimentation dans laquelle des pilotes contrôlant chacun un des avions d'une simulation résolvent les conflits aériens en appliquant les consignes données par un algorithme implémentant la méthode MVP, décrit dans [Cla+01]. Des simulations informatiques[HRV01] permettent d'expérimenter le comportement de cet algorithme dans des zones FFA à fort trafic. Plusieurs articles [Bal+02 ; IJt+15] présentent différentes distributions des responsabilités de la gestion du trafic aérien entre le sol et l'avion.

Cet algorithme est également utilisé pour résoudre les conflits aériens dans le cadre du projet Metropolis [Sun+15], dont le but est de contrôler un trafic aérien urbain dans lequel évolueraient des véhicules volants automatisés.

Enfin, [Maa+16] propose une hybridation de la méthode MVP avec un algorithme de création d'essaims (Boids), qui coordonne la trajectoire de plusieurs avions pour former des groupes cohérents.

Hill *et al.* (2005) [Hil+05] et Archibald *et al.* (2008) [Arc+08] ont développé une méthode de résolution de conflits qui utilise la théorie des jeux. Les avions changent de cap de manière à éviter les conflits sans gêner leurs voisins plus que nécessaire.

La méthode proposée par Rodionova *et al.* (2014) [Rod+14] permet la résolution de conflits des vols transatlantiques. Dans cet espace aérien, les avions empruntent des routes parallèles suivant le trajet du jet-stream. Ces courants aériens rapides de quelques centaines de kilomètres de large soufflent d'ouest en est tout autour de la Terre. Il en existe plusieurs, au niveau des tropiques et des cercles polaires. Les vols entre l'Europe et les États-Unis bénéficient du jet-stream polaire. Les avions volant sur ces routes sont séparés en temps : ils doivent survoler les waypoints avec 15 minutes de séparation. [Rod+14] décrit une méthode basée sur un algorithme génétique qui régule automatiquement le trafic aérien transatlantique. Cette méthode prend en compte les améliorations techniques de ces dernières années permettant de réduire les temps de séparation.

Un ensemble d'articles écrits par Mao *et al.* étudie la manière de croiser plusieurs flux infinis d'avions. Les articles [DFB02 ; MFB00 ; MFB01 ; MF01 ; Mao+05 ; MDF07] explorent le croisement de deux flux, en imposant aux avions soit un décalage pour qu'ils volent parallèlement à l'axe du flux, soit un changement de cap. Le décalage est privilégié dans [MFB01 ; MF01 ; Mao+05 ; MDF07]. Les articles soulignent que la manière optimale de résoudre les conflits est la formation de groupes d'avions qui se croisent à la manière d'un engrenage, comme le montre la figure 1.27.

Les articles suivants enrichissent ce concept en l'élargissant à trois flux se croisant en un point [TM08], à plusieurs flux se croisant deux à deux [Hua+14], ou bien en permettant aux avions de voler à des vitesses différentes [LFM10 ; LFM11].

En se basant sur ces travaux, Yoo, Devasia *et al.* (2011–2014) [Dev+11 ; YD11 ; YD12 ; YD13c ; YD13b ; YD13a ; YD14] proposent une topologie de route permettant de croiser

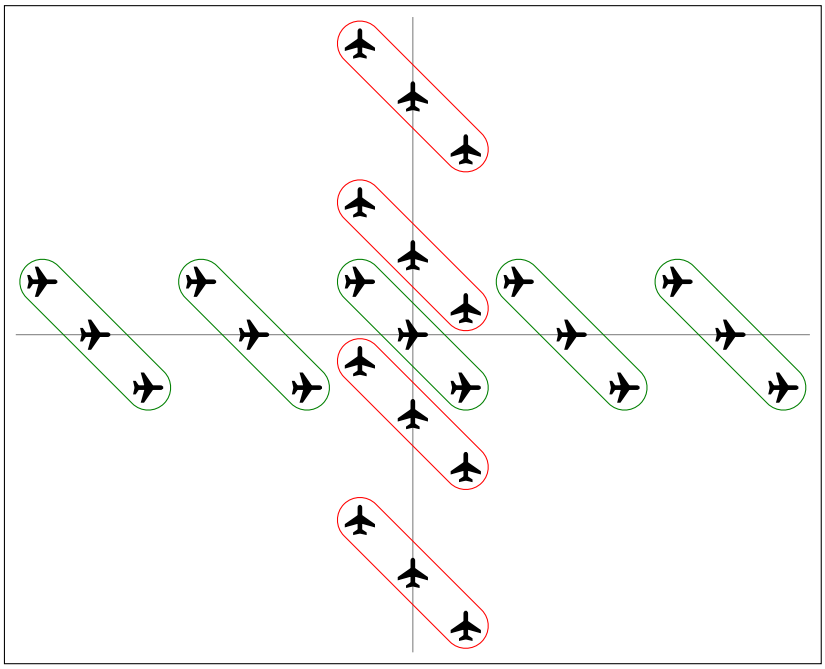


FIGURE 1.27 – Croisement de flux d'avions décrit par Mao *et al.*.

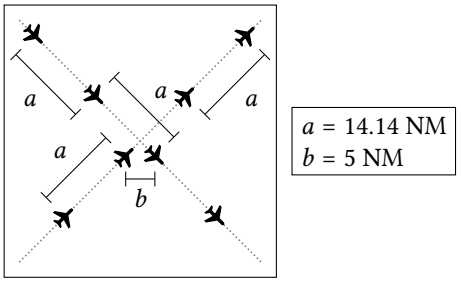


FIGURE 1.28 – Intersection de deux flux d'avions perpendiculaires dans une configuration Miles-in-Trail. Si les avions volent à la même vitesse, ils doivent être séparés par 14,14 NM sur chaque route. En passant alternativement le point de croisement, la distance minimale de séparation est de 5 NM.

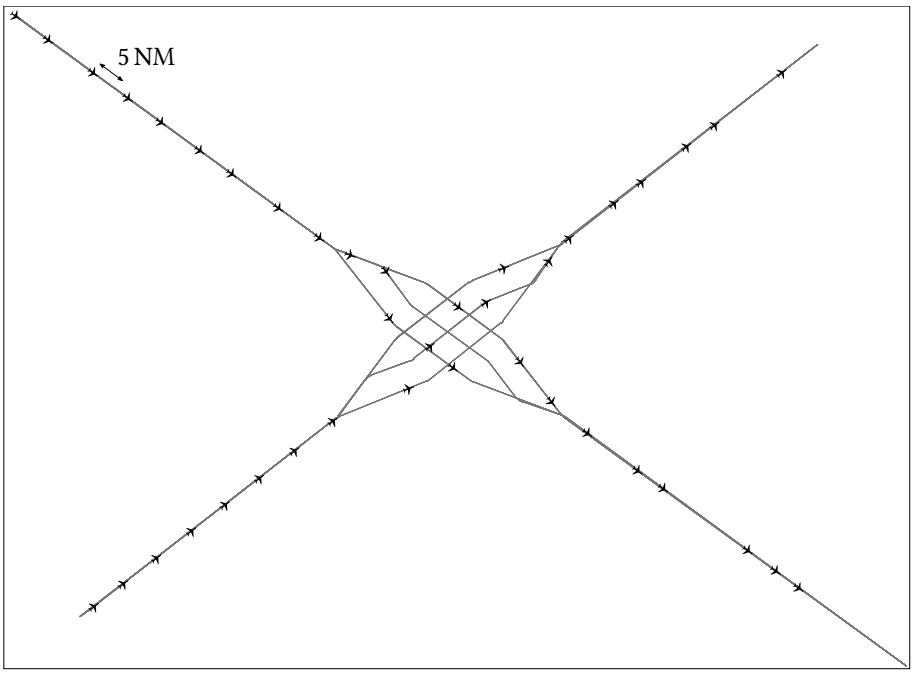


FIGURE 1.29 – Topologie de croisement à trois voies proposée dans [YD11] permettant d'augmenter la capacité du trafic organisé en Miles-in-Trail décrit dans la figure 1.9. Les routes parallèles sont séparés par 5 NM. La capacité maximale théorique de ce réseau permet de placer un avion tous les 5 NM sur chacune des deux routes.

plusieurs flux d'avions en Miles-in-Trail (voir la section 1.1.3.3.4). En effet, le Miles-in-Trail permet de croiser des flux perpendiculaires dont les avions sont séparés par au moins 14,14 NM (voir la figure 1.28). Comme le montre la figure 1.29, la topologie proposée par Yoo et Devasia sépare chaque flux en trois routes parallèles de même longueur. Les avions de chaque flux sont séparés par 5 NM avant la séparation, puis la topologie les positionne en engrenage, de la même manière que dans la figure 1.27, avant de fusionner les trois sous-flux.

L'algorithme proposé dans le chapitre 4 intègre cette topologie de routes, ce qui permet d'augmenter la capacité du trafic dans les scénarios où cet algorithme est utilisé pour réguler du trafic organisé en Miles-in-Trail.

1.2.4.4 Anti-collision

Il n'existe pas d'algorithme de résolution de conflits entièrement automatisé actuellement en cours de déploiement ou d'utilisation, ERASMUS ne faisant que diminuer le nombre de conflits, et ne les résout pas tous. Par contre, un autre système embarqué dans les avions permet d'éviter les collisions. Il s'agit du Traffic Collision Avoidance System (TCAS) [Eur16]. Ce problème est plus simple que la résolution de conflits. L'algorithme utilisé est également suffisamment simple pour pouvoir être certifié.

Pour être exact, Airborne Collision Avoidance System (ACAS) est la norme décri-

vant le système d'anti-collision, et TCAS est le nom commercial de l'implémentation de ce système. Il s'agit d'un filet de sécurité qui permet d'éviter les collisions essentiellement utilisé dans les espaces non contrôlés. Le TCAS communique avec le transpondeur des autres avions pour estimer les distances de séparation et les vitesses relatives entre avions. À partir d'un ensemble de règles, il estime le risque de collision, et quand il prévoit que les avions vont s'approcher à moins de 2 NM horizontalement et 600 ft verticalement, ce qui représente un horizon de quelques dizaines de secondes, il ordonne au pilote de monter ou de descendre. Le taux de montée permet à l'avion d'éviter la collision même si l'autre n'entreprend aucune action. Si les deux avions sont équipés de TCAS, ils se coordonnent pour adapter leur réponse (l'un des avions monte, l'autre descend).

Le TCAS est un filet de sécurité. Il se déclenche le plus souvent pour deux avions qui se rapprocheraient trop, plus rarement pour trois avions ou plus. L'ACAS X est une évolution du TCAS en cours de développement. Le principe est le même, mais les manœuvres possibles sont plus variées : changement de cap, d'altitude et de vitesse.

Gariel, Kunzi et Hansman (2011) [GKH11] proposent une méthode de détection de risque de collision similaire au TCAS (voir la section 1.2.4.4), mais reposant sur l'ADS-B au lieu du transpondeur. La différence est que l'ADS-B donne la position et la vitesse des avions, alors que le transpondeur, utilisé par les radars secondaires, ne permet que d'obtenir une distance et une vitesse relative entre les deux avions. Leur algorithme est donc plus précis, et permet de réduire le nombre de fausses alertes.

1.3 Conclusion

La première section de ce chapitre a présenté la structuration statique et dynamique actuelle du trafic aérien. La structuration statique comprend notamment la sectorisation de l'espace aérien et la création du réseau de routes. La structuration dynamique consiste à organiser le trafic à plusieurs échelles de temps, de la phase stratégique, qui structure les flux d'avions, à la phase tactique, qui évite les collisions entre aéronefs.

La deuxième section a décrit les différents systèmes actuellement utilisés et en cours de développement pour aider à la gestion du trafic par les contrôleurs aériens. Certains d'entre eux sont des outils d'aide à la décision à destination des contrôleurs. D'autres visent à remplacer ceux-ci pour certaines tâches.

Les chapitres 3, 4 et 5 proposent différents algorithmes de gestion du trafic aérien reprenant certains des concepts décrits dans le présent chapitre. L'algorithme du chapitre 3 propose une régulation automatique d'un trafic Miles-in-Trail grâce à un système multi-agents. Le chapitre 4 propose une implémentation du concept ERASMUS sous forme de système multi-agents, puis sous forme d'un algorithme de recuit simulé à fin de comparaison. Enfin, le chapitre 5 s'appuie sur la mesure de complexité du trafic basée sur la convergence des avions, ainsi que sur l'agrégation de trajectoires par minimisation de l'entropie de la densité pour créer des réseaux de routes temporaires.

Le chapitre 2 décrit le fonctionnement des systèmes multi-agents et de l'algorithme du recuit simulé, utilisés dans les différents algorithmes proposés dans cette thèse.

Chapitre 2

Base des algorithmes

Les algorithmes des chapitres 3, 4 et 5 cherchent à réguler le trafic aérien de manière coopérative, c'est-à-dire qu'une partie des décisions sont déléguées aux avions. Ces algorithmes ont été implémentés sous forme de système multi-agents. Afin de comparer l'algorithme du chapitre 4 avec une méthode d'optimisation globale, cet algorithme a également été implémenté sous forme de recuit simulé.

Ces algorithmes présentent des caractéristiques de robustesse et de résilience, dont voici la définition utilisée dans cette thèse :

La robustesse est la capacité d'un algorithme à fournir une solution qui *reste valide* en cas de perturbation dans les données initiales. Par exemple, un algorithme robuste de résolution de conflits aériens applique une marge pour prendre en compte l'incertitude sur la position des avions de manière à ce que le conflit soit résolu même si les avions volent légèrement plus rapidement ou plus lentement que prévu ;

La résilience est la capacité d'un algorithme à *mettre à jour* sa solution en cas de perturbation dans les données initiales. Par exemple, un algorithme résilient de résolution de conflits aériens modifie sa décision pour résoudre un conflit qui apparaitrait parce que les avions volent légèrement plus vite ou plus lentement que prévu.

La section 2.1 de ce chapitre est une introduction aux SMA. La section 2.2 décrit le fonctionnement d'un algorithme de recuit simulé.

2.1 Système multi-agents

2.1.1 Description

Le but de cette thèse est de développer des méthodes permettant de déléguer une partie du processus de planification des trajectoires aux avions, qui coopèrent pour élaborer une solution permettant de structurer les trajectoires dans les dimensions spatiales et temporelle. Pour ce faire, il a été décidé de développer un système multi-agents (SMA).

Ce paradigme est souvent considéré comme une sorte d'intelligence artificielle distribuée. Les SMA sont composés d'agents autonomes interagissant entre eux et avec leur environnement [Fer99]. Habituellement, les agents perçoivent leur environnement de manière limitée, et ils ont une connaissance partielle de l'état interne de leurs voisins en échangeant des messages.

L'auto-organisation [Cap+03] est un des aspects principaux d'un système multi-agents. Quand les règles de comportement des agents sont correctement choisies, un comportement complexe peut émerger au niveau du système, résultant des interactions locales et du comportement des agents. Quand un SMA est utilisé pour résoudre des problèmes de recherche opérationnelle, des règles de comportement des agents soigneusement choisies peuvent permettre au système de trouver une solution au problème (au niveau du système) en utilisant uniquement des règles locales (au niveau des agents).

Les agents peuvent avoir un comportement plus ou moins complexe. Ceux dont le comportement est le plus simple sont les agents réactifs. À chaque instant, ils perçoivent l'état de leur environnement immédiat, et déterminent leur action en fonction de règles prédéterminées. Ils n'ont pas de mémoire, et aucune notion de but à atteindre. À l'autre extrémité du spectre, les agents cognitifs déterminent leurs actions grâce à des règles d'intelligence artificielle, cherchent à remplir un objectif, et ont une mémoire leur permettant d'adapter leur comportement aux actions passées.

Les systèmes impliquant des agents réactifs sont en général composés de très nombreux agents effectuant des tâches simples. Les algorithmes de colonies de fourmis [DBS06] appartiennent à cette catégorie. Dans ces systèmes, utilisés pour planifier des trajectoires, des dizaines de fourmis au comportement simple explorent aléatoirement un espace à la recherche de « nouriture » (la destination), et quand elles en trouvent, reviennent à la fourmilière (l'origine), en laissant derrière elles des traces de phéromones. Le fonctionnement du système fait que les traces de phéromones les plus courtes sont renforcées au fil du temps, et les autres disparaissent. Au final, la trace restante matérialise le chemin le plus court.

Dans les systèmes impliquant des agents cognitifs, chacun des agents a un rôle déterminé et un comportement spécifique. Le nombre d'agents est petit en comparaison avec les systèmes comportant des agents réactifs. Ils entretiennent des relations sociales : en échangeant des messages, ils connaissent partiellement les intentions de leurs voisins. Le modèle Belief-Desire-Intention [Fer99] est le plus couramment rencontré. Chaque agent agit en fonction de ce qu'il sait de son environnement (croyances), possède un but (désir), et élabore un plan qui lui permet d'atteindre son but (intentions). Avoir des croyances et un plan implique que les agents ont une mémoire qui leur permet d'en garder la trace.

Un système multi-agent peut être implémenté comme une simulation informatique, ou bien comme un système physique composé de robots capables de communiquer et d'interagir physiquement avec leur environnement.

2.1.2 Avantages et limites

Bien que de nombreuses expérimentations tendent à montrer que les SMA de résolutions de problèmes de recherche opérationnelle convergent vers un état stable indiquant la solution, il est difficile de prouver que cette convergence survient dans tous les cas. Certains chercheurs ont étudié ce problème, et ont trouvé une preuve de convergence pour des systèmes simples. Par exemple, Jadbabaie, Lin et Morse (2003) [JLM03] prouvent la convergence de l'algorithme *Boids*. Ce programme a été développé pour simuler les vols d'oiseaux en essaim (comme les vols d'étourneaux). Le nom « Boids » vient d'ailleurs de la contraction de l'anglais *bird-oid*, « qui ressemble à un oiseau ».

Les systèmes multi-agents ont toutefois plusieurs avantages dans le cadre de l'implémentation de méthodes de décision décentralisées. Tout d'abord, quand ils sont

correctement conçus, ils possèdent une bonne résilience aux évènements perturbateurs [RMB13]. Les agents essaient d'atteindre un objectif, et agissent de manière à se rapprocher de leur but. Quand ils sont confrontés à une perturbation locale dans leur environnement, ils adaptent leurs actions pour prendre en compte ces changements. Cela permet au système de revenir à un nouvel état stable, qui peut être différent de l'état observé avant la perturbation.

De plus, les décisions sont décentralisées au niveau des agents. Un agent arrêtant de fonctionner ne compromet pas le système dans son ensemble, les autres agents s'adaptant à ce changement. Dans un processus de décision centralisé, la panne d'une entité de régulation centrale peut empêcher tout le système de fonctionner. En informatique, un tel point central est qualifié de point individuel de défaillance, ou SPOF (Single Point of Failure).

Ensuite, quand le système multi-agents est implémenté comme une simulation informatique, les calculs des agents peuvent être effectués en parallèle, et exploiter les architectures matérielles modernes (processeurs multi-cœurs, calcul sur cartes graphiques). Un système multi-agents peut également fonctionner sur une grappe de serveurs.

Enfin, dans le cas de d'un système multi-agents dédié à la gestion du trafic aérien, celui-ci permet d'expérimenter l'implémentation progressive de systèmes coopératifs embarqués à bord des avions. Dans ces simulations, les avions équipés et non-équipés peuvent interagir dans le même espace aérien, ce qui permet de valider la capacité de ces systèmes collaboratifs embarqués à gérer ce type de trafic mixte. Les avions équipés coopèrent entre eux, et ont plus de liberté dans leur décisions que les avions qui ne sont pas équipés. Ces derniers peuvent par exemple être contraints de suivre des couloirs aériens prédéfinis.

Cependant, les avions n'embarquent pas encore de systèmes leur permettant d'échanger des trajectoires 4D (des trajectoires décrites à la fois par leur forme géométrique et leur déroulement temporel). Ils sont toutefois capables d'échanger ces informations avec le sol, notamment grâce à l'ADS-C (voir la section 1.1.2.4). À l'heure actuelle, l'implémentation d'un système multi-agents suppose donc que les trajectoires soient collectées par un système au sol, puis régulées par un logiciel, avant que les nouvelles trajectoires soient renvoyées aux avions.

Néanmoins, le premier algorithme décrit dans cette thèse, dans le chapitre 3 est basé sur l'ADS-B, et les avions peuvent recevoir ces messages. Il est donc théoriquement possible d'implémenter cet algorithme à bord des avions.

2.1.3 Optimisation et systèmes multi-agents

Dans le cadre de la recherche opérationnelle, les systèmes multi-agents peuvent être utilisés pour résoudre des problèmes d'optimisation [Pic14]. Celui-ci peut avoir déjà été formulé sous forme de fonction objectif et de contraintes. Si ce problème est trop complexe pour être résolu par une méthode d'optimisation globale, utiliser un SMA peut permettre de le subdiviser en entités plus petites, afin de le résoudre.

Ce processus d'*agentification* consiste à regrouper les variables et les contraintes, chaque groupe étant pris en charge par un agent. Celui-ci doit optimiser un sous-problème, et ainsi contribuer au processus global d'optimisation.

Jorquera (2013) [Jor13] propose une méthode qui permet de résoudre des problèmes d'optimisation globale grâce à un système multi-agents. Il crée un SMA dans lequel chaque variable et chaque contrainte, ainsi que la fonction objectif, sont représentés par un agent. Les agents sont reliés selon leur lien d'interdépendance :

les variables sont reliées aux contraintes qui impliquent celles-ci, ainsi qu'à la fonction objectif. Les agents *variables* modifient leur valeur en fonction des feedbacks des agents *contraintes* auxquels ils sont reliés, et de l'agent *fonction objectif*, dont ils tentent de maximiser ou de minimiser le résultat.

Plus généralement, le regroupement des variables et des contraintes au sein des agents se fait de manière plus macroscopique. Par exemple, dans le problème décrit dans le chapitre 4, chaque avion est représenté par un agent du système. Les variables et les contraintes relatives à chaque avion sont donc gérés par l'agent correspondant.

L'enjeu principal de l'élaboration d'un SMA à partir d'un problème d'optimisation consiste donc à transposer la fonction objectif. Celle-ci peut être prise en charge par un agent central, mais cela revient à re-centraliser le processus de décision. La plupart des travaux de recherche visant à résoudre un problème d'optimisation par un SMA n'implémentent pas explicitement la fonction objectif dans un agent. Celle-ci émerge à partir des règles d'interaction locales entre les agents, qui font émerger l'organisation du système, ainsi que la solution du problème [Cap+03].

2.1.4 Exemples de systèmes multi-agents

Les systèmes multi-agents ont été utilisés pour résoudre de nombreux problèmes en recherche opérationnelle, comme la régulation de réseaux de transports urbains [BP08], la conception de systèmes mécaniques [CGG04], ou l'optimisation de trajectoires [DBS06 ; LZG04].

Van Dyke Parunak et Brueckner (2001) [VB01] étudient l'évolution de l'entropie d'un système multi-agents simple. Ils décrivent un système composé d'une fourmi remontant une trace de phéromones émise depuis une source. Au niveau macroscopique, ils prouvent que l'entropie du système diminue : la fourmi atteint toujours la source.

Ünsal et Bay (1994) [ÜB94] décrivent des systèmes dans lesquels des robots mobiles se placent dans différentes formations, notamment en cercle et en parabole, uniquement à partir de règles locales. Les agents se déplacent de manière à être à une certaine distance du centre du cercle, et à égale distance de leurs voisins les plus proches.

Olfati-Saber (2006) [Olf06] décrit un système de flocking (simulation de troupeau). Les agents se mettant en formation dans laquelle ils sont espacés régulièrement. Ils suivent des leaders, et les groupes formés sont capables d'éviter des obstacles de manière coordonnée

Picard (2005) [Pic05] décrit un système dans lequel des robots transportent des caisses entre deux pièces reliées par deux couloirs trop étroits pour que ceux-ci puissent se croiser. À partir des règles d'interaction locales des robots, un sens de circulation émerge, les robots qui se déplacent de la première pièce à la deuxième passent tous par un couloir, ceux allant dans le sens contraire utilisent l'autre.

Mui, Mohtashemi et Halberstadt (2002) [MMH02] étudient la notion de réputation dans des groupes grâce à un système multi-agents, utilisée par exemple par les plateformes de e-commerce. Pour cela, ils étudient une variante du dilemme du prisonnier répété, issu de la théorie des jeux, qui intègre la réputation.

Legras (2003) [Leg03] propose une méthode permettant de faire émerger des équipes de robots. Il s'appuie sur un processus d'écoute flottante. Les agents diffusent en permanence des messages radio, et déterminent automatiquement la composition de groupes qui effectueront des tâches en équipe.

Welcomme (2008) [Wel08] décrit un système dédié à la conception préliminaire d'avion. Il s'agit de la première étape de la conception d'un avion, durant laquelle ses grandes dimensions sont définies : taille des ailes, volume des réservoirs, poids maximal, etc. Dans ce système, les grandeurs principales sont des agents qui ont une influence les uns sur les autres. Par exemple, l'augmentation de la taille des réservoirs permet à l'avion de voler plus longtemps, mais augmente sa masse au décollage, ce qui oblige à augmenter la taille de ses ailes.

Georgé *et al.* (2003) [Geo+03] décrivent un système permettant d'estimer en temps réel le risque et la hauteur des crues. Un réseau de capteurs est installé dans les bassins versants. Ceux-ci déterminent automatiquement quel capteur est situé en aval de quel autre, et quelle est l'influence d'un niveau d'eau mesuré sur les niveaux mesurés en aval. Un modèle émerge dans lequel la contribution de chaque capteur est intégrée. Ce modèle permet de prédire en temps réel l'évolution du niveau des cours d'eau et les risques de crues.

2.1.5 Systèmes multi-agents appliqués à la gestion du trafic aérien

Les systèmes multi-agents ont déjà été utilisés pour résoudre des problèmes liés à la gestion du trafic aérien. Certains SMA sont utilisés pour simuler le trafic aérien et tester l'impact de certaines évolutions des procédures de contrôle du trafic. Parmi ces travaux, Canino *et al.* (2012) [Can+12] proposent un SMA permettant de simuler le trafic pour valider les améliorations techniques introduites par les projets SESAR et NextGen permettant la communication et le partage d'informations entre les avions et le sol.

Pritchett *et al.* (2012) [Pri+02] utilisent un SMA pour évaluer les problèmes de sécurité en ATM. Cet algorithme intègre un modèle des interactions entre les systèmes informatisés et les humains. Feigh *et al.* (2005) [Fei+05] simulent le système de transport aérien et étudient l'émergence de comportements complexes au niveau du système.

Wolfe *et al.* (2007–2009) [Wol+07 ; WSJ08 ; Wol+09] étudient à l'aide d'un SMA les négociations entre l'ATCSCC (Air Traffic Control System Command Center) [FAA14b], l'équivalent Américain du Network Manager, et les compagnies aériennes dans le processus de validation des plans de vol. Les différentes stratégies que peuvent utiliser les compagnies (agressive, coopérative) sont comparées au niveau du bénéfice des compagnies et de la complexité résultante pour l'ensemble du trafic.

D'autres systèmes multi-agents sont des algorithmes dédiés à la gestion automatisée du trafic aérien.

Le projet Modern Taxiing (MoTa), financé par le projet SESAR, est consacré à la modernisation des phases de roulage (dans les aéroports). Les avions seraient acheminés par des tracteurs entièrement automatiques entre les terminaux et les pistes de décollage et d'atterrissage, leur trajectoire étant établie par les contrôleurs. À l'heure actuelle, la phase de roulage est effectuée en partie par des véhicules qui font reculer les avions depuis le terminal, et par les avions eux-mêmes qui avancent en utilisant leurs réacteurs. Dans ce cadre, Lancelot *et al.* (2015) [Lan+15] proposent un système multi-agents qui planifie les trajectoires des avions sur les taxiways en collaboration avec les contrôleurs. Ceux-ci interagissent avec l'algorithme pour corriger les trajectoires proposées, ce qui permet à l'algorithme d'adapter ses réponses futures en apprenant les préférences des contrôleurs.

D'autres projets sont consacrés à la régulation du trafic en route. Tomlin, Pappas et Sastry (1997) [TPS97] proposent un système multi-agents dont les règles d'interactions

entre avions sont basées sur la théorie des jeux. Les avions tentent d'abord de négocier. Si la négociation est impossible, par exemple parce qu'un des avions n'est pas équipé du système de négociation, l'autre avion planifie une trajectoire suivant une règle du type poursuivant-évadé : la trajectoire est élaborée de manière à éviter un conflit quelque soit la trajectoire de l'autre avion, comme si ce dernier poursuivait volontairement le premier. Cette contrainte est ensuite relaxée, en considérant que le deuxième avion suit une trajectoire rectiligne, ce qui est plus réaliste.

Nguyen-Duc, Briot et Drogoul (2003) [NBD03] proposent un SMA hiérarchique reprenant le fonctionnement du système de gestion du trafic actuel. Les avions interagissent avec les contrôleurs aériens, qui eux-mêmes négocient avec des « local flow managers », qui dialoguent avec le « central flow manager » (l'équivalent du Network Manager). Les négociations se déroulent à l'intérieur de chaque groupe, un contrôleur avec ses avions, le local flow manager avec ses contrôleurs, etc.

Agogino et Tumer (2007 [TA07], 2012 [AT12]) décrivent un algorithme dans lequel chaque agent contrôle un secteur aérien. Les avions suivent un réseau de route. Les agents peuvent réguler la distance de séparation des avions, changer leur route, et les faire attendre au sol. Chaque « contrôleur » est évalué en fonction du retard qu'il impose et de la congestion globale du trafic.

D'autres projets sont consacrés à la régulation du trafic dans les espaces Free Flight et Free Route. Dans ces zones, les avions doivent trouver une trajectoire sans conflit (qui respectent les normes de séparation entre avions). Plusieurs systèmes multi-agents ont été développés dans cette optique, dont celui de Wollkind, Valasek and Ioerger [WVI04], ou celui de Sislak, Volf and Pechoucek [Pec+06 ; PS09 ; SSP08 ; SVP09 ; ŠVP09 ; ŠVP11 ; Sis13]. La résolution de conflits proposée par ces deux algorithmes est basée sur une négociation pair-à-pair entre avions. Ceux-ci peuvent utiliser plusieurs types de manœuvres, comme le changement de cap ou d'altitude.

Les systèmes multi-agents de Wollkind *et al.* et de Pechoucek *et al.* sont les plus proches de ceux présentés dans les chapitres 3 et 4, puisqu'ils cherchent à résoudre des conflits aériens. Cependant, ils ne résolvent pas les mêmes problèmes : [WVI04] et [Pec+06] régulent du trafic Free Flight, alors que le SMA du chapitre 3 régule un trafic très structuré (Miles-in-Trail), et que le SMA du chapitre 4 ne fait que réduire le nombre de conflits au bénéfice des contrôleurs. De plus, les deux algorithmes de cette thèse ne régulent que la vitesse des avions, alors que [WVI04] et [Pec+06] régulent les trajectoires 4D.

2.1.6 Système multi-agent développé durant ce doctorat

Cette thèse propose trois algorithmes permettant de résoudre différents problèmes de gestion du trafic aérien. Ces trois systèmes multi-agents sont basés sur la même architecture générale et partagent le même cycle de vie, qui est visible dans la figure 2.1.

Le système est régulé par une horloge globale qui égrène les secondes. À chaque impulsion de l'horloge, un nouveau cycle de la vie des agents a lieu. Chacun commence, durant la phase de perception, par recevoir les messages contenus dans leur boîte aux lettres. Il actualise alors sa connaissance de son environnement. Ensuite, durant la phase de décision, il planifie ses prochaines actions. Enfin, la phase d'action lui permet d'effectuer les actions planifiées pour cet instant, et de poster les messages destinés à ses voisins.

Le cycle de vie des agents est donc synchronisé. À la fin d'une itération, les agents envoient leurs messages, et au début de l'itération suivante, les messages sont remis aux destinataires. Les processus de décision des agents sont donc logiquement exécutés

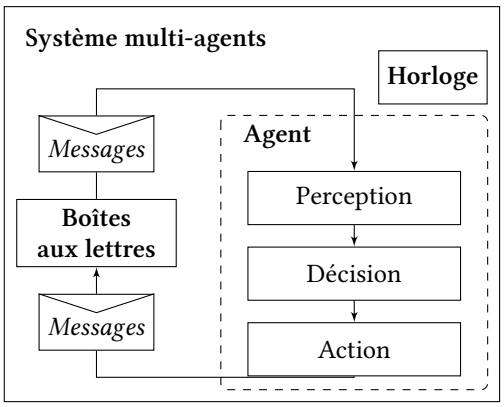


FIGURE 2.1 – Cycle de vie du système multi-agents utilisé décrit dans cette thèse.

en parallèle. L'ordre dans lequel ces processus sont exécutés n'a pas d'importance, et ceux-ci peuvent même être exécutés séquentiellement. Cela permet d'éliminer les problèmes liés à l'ordre de la prise des décisions.

Cela peut également amener d'autres problèmes, comme des phénomènes d'oscillation : deux agents peuvent prendre des décisions incompatibles, puis les « corriger » à l'itération suivante en prenant de nouveau des décisions incompatibles, avant de revenir à la première décision, etc.

L'implémentation de ces algorithmes est néanmoins parallélisée, grâce au multithreading, pour bénéficier de la capacité des processeurs récents de distribuer des calculs sur plusieurs cœurs.

Les systèmes multi-agents sont sensés faire émerger une solution globale à un problème grâce aux interactions locales entre les agents. Pour étudier les différences entre un processus d'optimisation globale et un SMA, il faut comparer les résultats obtenus avec les deux méthodes appliquées aux mêmes problèmes. Cela permet de déterminer les différences de qualité des résultats, en terme de proximité avec l'optimum, et de temps de calcul. Il a été décidé d'utiliser l'algorithme du recuit simulé.

2.2 Recuit simulé

En métallurgie, le recuit est le processus de fusion suivi d'un refroidissement lent du métal. Dans ce processus, quand le métal est en fusion, les atomes se déplacent aléatoirement. Si la baisse de température est suffisamment lente, ces atomes forment une structure cristalline de manière à minimiser l'énergie du système. Le processus opposé est la trempe, au cours de laquelle le métal est refroidi brutalement. Les atomes sont figés dans la position qu'ils avaient quand le métal était en fusion, ce qui forme un verre.

Le recuit a été modélisé et simulé sur ordinateur par Metropolis *et al.* (1953) [Met+53]. Cet article décrit un modèle mathématique du processus de cristallisation. La capacité qu'a un atome de se déplacer librement, c'est-à-dire sa capacité à effectuer une transition vers une position augmentant l'énergie totale du système dépend de la température. La probabilité d'acceptation P_a d'une transition d'un état i à un état j

est :

$$P_a = \begin{cases} 1 & \text{si } E_j < E_i, \\ e^{\left(\frac{E_i - E_j}{k_b T}\right)} & \text{sinon,} \end{cases} \tag{2.1}$$

avec E_i et E_j les niveaux d'énergie des états i et j, T la température et k_b la constante de Boltzmann. Cela veut dire que quand la température est élevée, la probabilité qu'un atome se déplace vers un état d'énergie plus élevée est proche de 1, occasionnant des mouvement aléatoires. Et plus la température diminue, plus cette probabilité faiblit jusqu'à ce que les atomes ne puissent plus aller que vers des états dont l'énergie associée diminue.

Algorithme 1 Processus de recuit simulé

```
1:  function RecuitSimulé(x_0, T_0)
2:      x_i ← x_0
3:      y_i ← f(x_i)
4:      k ← 0
5:      T_k ← T_0
6:      repeat
7:          for l ← 1..N do
8:              x_j ← Voisinnage(x_i)
9:              y_j ← f(x_j)
10:             if AccepteTransition(y_i, y_j, T_k) then
11:                 x_i ← x_j
12:                 y_i ← y_j
13:             end if
14:         end for
15:         k ← k + 1
16:         T_k ← CalculTempérature(T_0, k)
17:     until T_k ≈ 0
18:     return x_i
19: end function

20: function AccepteTransition(y_i, y_j, T_k)
21:     if y_j < y_i then
22:         return Vrai
23:     end if
                                  ▷ Accepte la transition avec une probabilité P_i
24:     r ← RéelAleatoire([0, 1])
25:     if r < e^((y_i − y_j)/T_k) then
26:         return Vrai
27:     end if
28:     return Faux
29: end function
```

À partir de cette simulation du phénomène physique de recuit, Kirkpatrick, Gelatt et Vecchi (1983) [KGV83], et Černỳ (1985) [Čer85] ont indépendamment développé une méta-heuristique permettant de résoudre des problèmes d'optimisation globale. Par analogie au processus de recuit simulé, on simule le déplacement d'un état x_i à l'intérieur de l'espace d'états. Ce processus est décrit dans le pseudo-code 1. On

sélectionne aléatoirement un état initial x_0. On choisit également une température initiale T_0 telle que la probabilité d'acceptation des transitions qui dégradent la solution est proche de 1. La boucle principale (lignes 6– 17) diminue progressivement la température.

Pour chaque palier de température T_k, la boucle imbriquée (lignes 7–14) effectue N itérations. À chaque itération, un état x_j est sélectionné au voisinage de x_i. Sa valeur associée y_j est donnée par la fonction objectif $f(x)$. La probabilité d'acceptation de ce nouvel état est :

$$P_a = \begin{cases} 1 & \text{si } y_j < y_i, \\ e^{\left(\frac{y_i - y_j}{T}\right)} & \text{sinon.} \end{cases} \tag{2.2}$$

L'évolution de la température permet de moduler la manière d'explorer l'espace d'état durant le processus. Au début, quand la température est élevée, toutes les solutions sont acceptées, ce qui permet une exploration aléatoire de l'espace d'état. Cela permet aussi de sortir des minima locaux. Plus la température diminue, moins les transitions dégradant la solution sont acceptées, jusqu'à ce qu'elles soient toutes rejetées quand la température est la plus basse. L'algorithme se comporte alors comme une méthode d'optimisation locale.

2.2.1 Avantages

Le recuit simulé est la seule méta-heuristique pour laquelle une preuve de convergence a été trouvée sous certaines conditions [Ing89]. Celle-ci implique que la décroissance de la température est inférieure à :

$$T_k = \frac{T_0}{\ln k}, \tag{2.3}$$

et que le nombre d'itérations N pour chaque palier de température tend vers l'infini. En pratique, respecter ces contraintes est trop coûteux en temps de calcul, et les implémentations du recuit simulé utilisent plutôt une décroissance plus rapide, comme une suite linéaire ou géométrique, et quelques milliers d'itérations par pallier de température, qui donnent de très bons résultats dans un temps raisonnable.

D'autre part, puisque le recuit simulé ne manipule qu'un seul état, il est plus économe en mémoire vive que les algorithmes basés sur des populations d'états. Cela devient important quand un état consomme plusieurs gigaoctets de mémoire, ce qui rend impossible la présence simultanée en mémoire de plusieurs états.

Enfin, puisque le recuit simulé repose uniquement sur la comparaison de valeurs de la fonction objectif, il n'est pas nécessaire de formuler explicitement la dérivée de cette fonction.

2.2.2 Limites

Par contre, comparé à d'autres méthodes d'optimisation, le recuit simulé comporte certaines limites. En effet, contrairement aux algorithmes évolutionnaires, et plus largement aux algorithmes basés sur des populations de solutions, le recuit résout des problèmes mono-objectif. Par exemple, cela veut dire qu'on ne peut pas optimiser de manière simple à la fois la longueur de la trajectoire des avions *et* la congestion du trafic. On peut contourner cette difficulté en évaluant séparément ces deux valeurs, puis en calculant l'objectif comme la somme pondérée des deux. Cela pose d'autres problèmes, comme le choix des coefficients de pondération.

Et contrairement à d'autres méthodes, comme l'algorithme du simplexe, qui permet de résoudre des problèmes de programmation linéaire, on ne peut pas formuler simplement les contraintes d'un problème. Il faut là aussi intégrer les contraintes en tant que coefficient de pénalisation dans la fonction objectif.

2.2.3 Choix du recuit simulé

Le recuit simulé a été choisi dans le cadre de cette thèse pour deux raisons. Premièrement, cette méthode a été utilisée par d'autres projets du laboratoire ces dernières années [Cha14 ; Gir14 ; Lia+16 ; Ma+16a ; Zho+16 ; VDT17]. Il a donc été possible de réutiliser une base de code existante, et de bénéficier de l'expérience acquise par les autres chercheurs.

Deuxièmement, l'utilisation du recuit simulé permet de réutiliser une grande partie du code source du système multi-agents, notamment les fonctions permettant d'exécuter la simulation, de tracer les trajectoires, et de calculer les composants de la fonction objectif (détection de conflits et nombre de changements de vitesses).

L'utilisation d'une autre méthode d'optimisation aurait été plus complexe. Utiliser un solveur de problèmes de programmation linéaire ou non linéaire aurait nécessité de redévelopper un nouveau modèle pour transcrire le système multi-agents.

Dans une moindre mesure, utiliser d'autres méta-heuristiques aurait également nécessité un plus grand effort de conception et de développement. Par exemple, les algorithmes évolutionnaires sont également utilisés au laboratoire [Del+96 ; Rod+14]. Il existe donc une expertise et du code source réutilisable.

Les algorithmes évolutionnaires se basent sur une population d'« individus » représentant un état. Chaque individu est défini par un gène qui peut être transposé dans l'espace d'état, ce qui permet de calculer la valeur de la fonction objectif (appelée *fitness*). Ces gènes peuvent muter (transformation aléatoire) ou être croisés (échange de portions de gènes entre individus), ce qui modifie la composition de la population. Les étapes de mutation/croisement alternent avec des étapes de sélection basée sur la *fitness* de chaque individu : les meilleurs individus sont conservés, les autres supprimés. Le système converge alors vers la solution optimale.

Les algorithmes évolutionnaires ont quelques avantages par rapport au recuit simulé, notamment la parallélisation des calculs permise par l'utilisation d'une population d'individus (la *fitness* de chaque individu et leurs mutations peuvent être calculés en parallèle), ainsi que la possibilité de faire une optimisation multi-objectifs (comme minimiser à la fois le nombre de conflits et le nombre de changements de vitesse des avions). Mais ils présentent dans le cas de cette thèse l'inconvénient de devoir développer les mécanismes permettant de transposer le modèle existant en gène [1].

Le portage du système multi-agents vers un algorithme du recuit simulé ayant ici pour principale motivation la comparaison des résultats fournis par le SMA avec ceux retournés par une méthode d'optimisation globale basée sur le même modèle afin d'en mesurer la pertinence, le recuit simulé a été préféré à d'autres méthodes d'optimisation globale. En effet, le temps nécessaire à l'implémentation a été privilégié par rapport notamment à la vitesse du calcul de la solution optimale : il n'a fallu que quelques jours pour adapter le code source du système multi-agents, et quelques semaines pour calibrer le recuit simulé, pour obtenir les premiers résultats. Mais le

1. Un autre inconvénient est que la convergence de ces algorithmes n'a pas été prouvée, contrairement à celle du recuit simulé. Mais celle-ci a été démontrée expérimentalement au cours des nombreux travaux les ayant utilisés.

temps de calcul nécessaire à la convergence est bien plus importante que pour le système multi-agents.

2.2.4 Détails de l'implémentation utilisée au cours du doctorat

Le recuit simulé est utilisé dans cette thèse pour comparer les résultats obtenus par le système multi-agents décrit dans le chapitre 4 avec des résultats obtenus par une méthode d'optimisation globale. Il s'agit de l'algorithme 1. La décroissance de la température respecte une suite géométrique :

$$T_k = \alpha^k T_0 \,, \qquad \alpha \in]0,1[\tag{2.4}$$

avec α proche de 1. La valeur choisie est $\alpha = 0{,}95$. Le processus de refroidissement est composé de 200 paliers de température, la température finale est donc $3{,}5 \cdot 10^{-5}$ fois la température initiale. Chaque palier comporte 1 000 itérations.

La température initiale T_0 doit permettre d'accepter un grand nombre de transitions (environs 80 %). Pour cela, on peut appliquer un algorithme de chauffage. C'est l'opposé du processus de refroidissement décrit dans l'algorithme 1 décrit précédemment. On part d'une température basse, puis on l'augmente progressivement. À chaque palier, l'espace d'état est exploré aléatoirement, aucune transition n'étant rejetée. Le processus s'arrête quand 80 % des transitions sont acceptées.

La procédure de chauffage fonctionne correctement quand l'opération de voisinage a une forte probabilité de dégrader la solution. Or, l'opérateur de voisinage du recuit simulé présenté dans le chapitre 4 effectue des transitions qui ne dégradent pas la solution dans au moins 76 % des cas. Il a donc fallu utiliser une autre méthode pour chercher la température initiale.

Algorithme 2 Sélection de la température initiale du recuit simulé

```
1:  function TempératureInitiale(x_0)
2:      x ← x_0
3:      y_min ← f(x)
4:      y_max ← f(x)
5:      for l ← 1..N do
6:          x ← Voisinnage(x)
7:          y_min ← min(y_min, f(x))
8:          y_max ← max(y_max, f(x))
9:      end for
10:     T_0 ← (y_min − y_max) / ln P_min,max
11:     return T_0
12: end function
```

La méthode choisie dans cette thèse pour déterminer la température initiale est différente. Elle est présentée dans l'algorithme 2. La boucle principale (lignes 5–9) explore aléatoirement l'espace d'état, et cherche les valeurs minimale et maximale de la fonction objectif $y_{\min}$ et $y_{\max}$ trouvées durant cette exploration. Cela permet d'avoir une idée de l'amplitude de variation de cette fonction.

On cherche alors la température autorisant la transition de la meilleure solution $y_{\min}$ vers la pire $y_{\max}$ avec une probabilité relativement élevée de 10 %. L'équation (2.2) détermine la probabilité P_{ij} de la transition d'un état i vers un état j. On utilise les états de valeur $y_{\min}$ et $y_{\max}$ trouvés durant l'exploration de l'espace d'état, et on fixe

la probabilité $P_{\text{min,max}}$ à une valeur élevée (par exemple 10 %). On insère les variables y_{min}, y_{max} et $P_{\text{min,max}}$ dans l'équation (2.2) :

$$P_{\text{min,max}} = e^{\left(\frac{y_{\text{min}} - y_{\text{max}}}{T_0}\right)} \tag{2.5}$$

On en déduit la valeur de T_0 :

$$T_0 = \frac{y_{\text{min}} - y_{\text{max}}}{\ln P_{\text{min,max}}} \tag{2.6}$$

2.3 Conclusion

L'algorithme de recuit simulé et le système multi-agents décrits dans ce chapitre servent de base aux algorithmes décrits dans les chapitres 3, 4 et 5.

Les systèmes multi-agents décrits dans les chapitres 3 et 4 permettent la résolution de conflit par auto-régulation de la vitesse des avions dans deux situations différentes. Dans le chapitre 3, le trafic est organisé en Miles-in-Trail.

La méthode proposée dans le chapitre 4 s'applique à du trafic non structuré, et vise comme le projet ERASMUS à diminuer le nombre de conflits pour réduire la charge de travail des contrôleurs aériens.

L'algorithme proposé dans le chapitre 5 permet la construction à la volée de réseaux de routes pour réduire la complexité du trafic Free Flight. Deux approches sont proposées. La première est une version centralisée dans laquelle un agent est inséré dans le SMA décrit dans le chapitre 4. Cet agent contient lui-même un système multi-agents dédié à la construction de routes. La deuxième est une version décentralisée dans laquelle les avions construisent coopérativement le réseau de routes qu'ils empruntent.

Chapitre 3

Résolution de conflits sur un réseau de routes

L'algorithme présenté dans ce chapitre est appelé Plectre v1, pour « Planification coopérative de trajectoires » (Plectre v2 est présenté dans le chapitre 4). Il s'agit d'un système multi-agents dans lequel des avions auto-régulent leur vitesse de manière à résoudre les conflits. Les avions évoluent le long d'un réseau de routes Miles-in-Trail, et échangent leur position grâce à des messages ADS-B.

Ce chapitre présente d'abord le problème, puis l'algorithme développé pour le résoudre, avant de passer aux résultats des tests effectués sur un réseau comportant deux routes sécantes, dans différentes situations de trafic.

3.1 Hypothèses

L'algorithme Plectre v1 a pour but la structuration du trafic en Miles-in-Trail à l'intersection de deux routes. Dans cette simulation, les avions volent le long de deux routes se croisant, comme le montre la figure 1.9. Ils perçoivent la position et la vitesse de leurs voisins et régulent leur vitesse pour résoudre les conflits.

La position et la vitesse des avions sont communiquées par ADS-B, chaque avion étant équipé d'ADS-B In et Out. Dans ce premier algorithme, la distance à laquelle les messages peuvent être échangés n'est pas limitée, chaque avion reçoit donc les positions de tous les autres.

Dans cet algorithme, on considère un avion i cherchant une vitesse v_i lui permettant d'éviter les conflits avec ses voisins j.

La vitesse courante de l'avion i est notée v_i, et sa vitesse optimale, ou vitesse de croisière, est notée $v_{i,\text{opt}}$. Dans ce chapitre, le CI et la vitesse de croisière sont supposés constants pour la durée du vol. Pour s'insérer dans une configuration MIT, les avions peuvent être amenés à choisir une vitesse v_i différente de $v_{i,\text{opt}}$, dans un intervalle de vitesse donné. La borne inférieure $v_{i,\min|v_{i,\text{opt}}} = v_{i,\text{opt}} - 6\,\%$ permet d'insérer l'avion dans un flux sans trop augmenter la consommation de carburant [DP09]. L'intervalle de vitesse $\left[v_{i,\text{opt}} - 6\,\%, v_{i,\text{opt}} + 3\,\%\right]$ est couramment employé par des projets basés sur la régulation de vitesse [Ave+07].

La vitesse optimale des avions générés est choisie aléatoirement dans l'intervalle [447 kt, 487 kt]. 447 kt est la vitesse de croisière d'un Airbus A320 [Air], et 487 kt celle d'un A380. Ces avions sont représentatifs de ceux capables de voler à 36 000 ft et plus, l'A320 faisant partie des plus lents de la catégorie, et l'A380 des plus rapides,

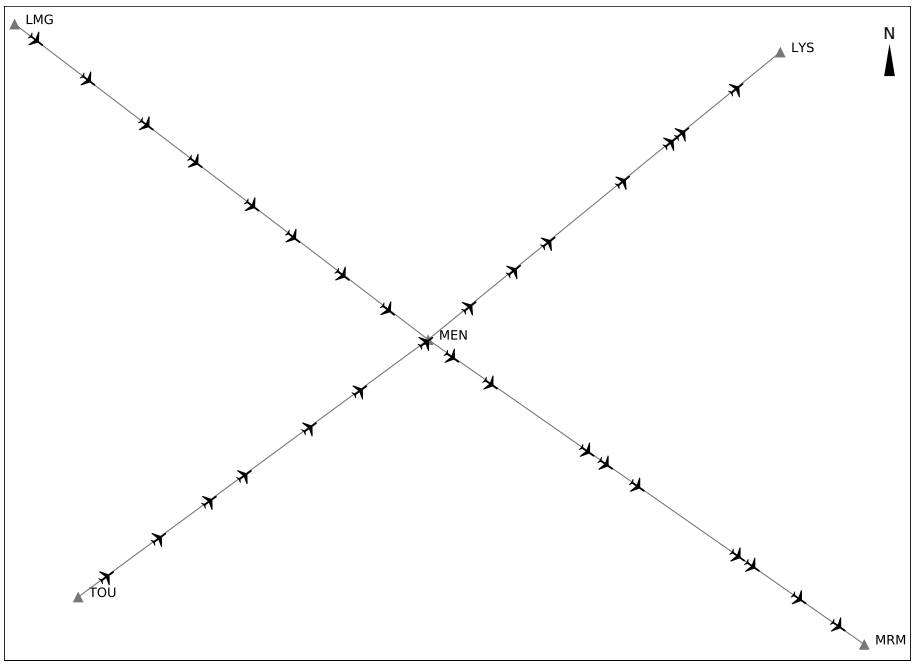


FIGURE 3.1 – Réseau de routes utilisé pour simuler l'intersection de deux flux d'avions.

avec le Boeing 777, par exemple. Tous les avions capables de voler à ces altitudes ont des performances similaires. Pour l'algorithme Plectre v1, cela veut dire que tous les avions générés peuvent voler à la même vitesse, puisque l'intervalle de vitesses [457 kt, 460 kt] est commun à tous les avions.

L'accélération et la décélération des avions sont fixées à ±4 000 NM/h^2, soit ±0,572 m/s^2, valeur fournie par la base de données BADA (Base of Aircraft Data) d'Eurocontrol [Eur11]. Ils respectent également le taux standard de virage [FAA14a, PCG S–6], qui est de 3°/s, ce qui permet d'effectuer un cercle complet en deux minutes.

3.2 Algorithme

Dans ce premier système multi-agents, les avions sont des agents qui diffusent des messages ADS-B. Ils évoluent dans un environnement composé d'un réseau de routes, représenté par un graphe dont les nœuds sont des waypoints identifiés par leur nom. Pour économiser du temps de calcul, les messages ADS-B contiennent, en plus de la position et de la vitesse de l'émetteur, l'arc sur lequel l'avion évolue. Cette information est représentée par le nom des waypoints précédent et suivant. Elle serait néanmoins facilement déductible grâce à la position de l'avion par rapport au réseau de route, à condition que l'avion suive strictement celui-ci.

Au niveau macroscopique, le scénario de Miles-in-Trail décrit dans la figure 3.1 peut être obtenue par une régulation en deux étapes. D'abord, chaque flux d'avions doit être régulé séparément en espaçant les avions et en uniformisant leur vitesse. Ensuite, les flux doivent être synchronisés de manière à faire traverser alternativement

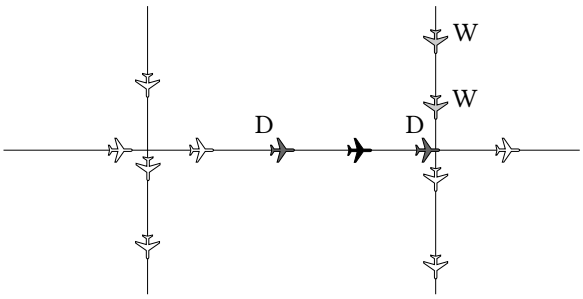


FIGURE 3.2 – Représentation du voisinage d'un avion. L'agent noir reçoit des messages de tous les agents environnants. Il détecte ses voisins directs (D) et ses voisins de waypoint (W).

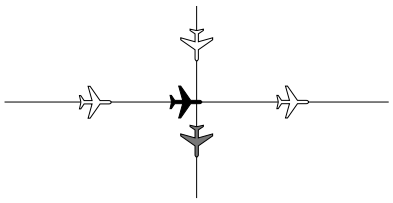


FIGURE 3.3 – Substitution des voisins manquants. L'avion noir n'a ni prédécesseur direct ni prédécesseur au waypoint. Ces rôles sont incarnés par le dernier avion à avoir traversé le croisement, l'avion gris.

le point d'intersection par les avions venant de chaque flux, tout en maintenant les normes de séparation.

Pour obtenir ce résultat, au niveau microscopique, chaque avion sélectionne dans son voisinage deux paires d'avions (figure 3.2) : ses voisins les plus proches appartenant au même flux, c'est-à-dire volant depuis et vers le même waypoint, appelés voisins directs dans cet algorithme ; et les avions entre lesquels il passera l'intersection, en considérant uniquement l'ordre de passage au prochain waypoint, quelque soit leur waypoint d'origine, appelés voisins de waypoint.

Pour éviter les conflits entre voisins directs, l'avion i doit maintenir une vitesse suffisamment basse pour ne pas rattraper son prédécesseur P_i, et suffisamment élevée pour ne pas être rattrapé par son successeur (follower) F_i.

Les voisins de waypoint sont les avions qui appartiennent à des flux différents de celui de l'avions i, et qui se dirigent vers le même waypoint que celui-ci. De la même manière que pour les voisins directs, l'avion i doit maintenir une vitesse suffisamment basse pour éviter d'entrer en conflit avec ses prédécesseurs, et suffisamment élevée pour éviter les conflits avec ses successeurs.

Selon la définition des paragraphes précédents, le premier avion sur une portion de route n'a aucun prédécesseur. Sans cette contrainte, l'algorithme pourrait décider d'accélérer pour éviter son successeur, sans tenir compte d'un éventuel prédécesseur situé sur la portion suivante, et entrer en conflit avec celui-ci. Pour résoudre ce problème, le dernier avion à avoir traversé le croisement est considéré comme le prédécesseur du prochain avion à le traverser, et réciproquement, comme le montre la figure 3.3.

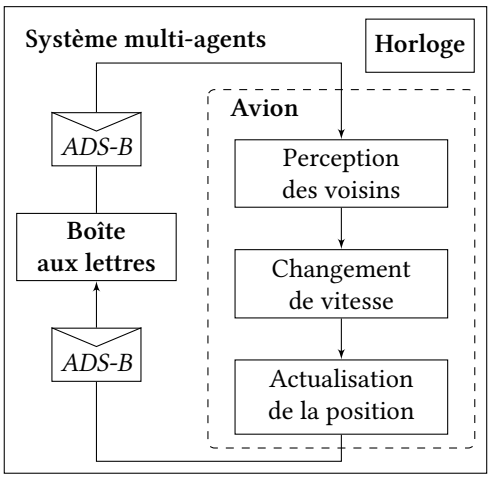


FIGURE 3.4 – Cycle de vie du système multi-agents Plectre v1.

Ces règles de voisinage permettent de former des chaines ininterrompues d'avions, qui sont en interaction de proche en proche. Chaque décision de l'un des maillons de cette chaine influence donc indirectement tous les autres.

Comme évoqué dans la section 2.1.6, et comme le montre la figure 3.4, le processus de décision de chaque agent est une séquence de trois étapes, qui sont exécutées à chaque impulsion de l'horloge. La phase de perception permet aux agents de recevoir les messages ADS-B, ce qui permet de rafraîchir leur représentation interne de l'espace aérien. Durant la phase de décision, les avions mettent à jour leur vitesse sur la base de cette représentation interne. Enfin, la phase d'action permet aux avions de rafraîchir leur position en utilisant la vitesse mise à jour, avant de diffuser un message contenant ces informations.

3.2.1 Perception

Durant la phase de perception, chaque avion commence par stocker les informations relatives à ses voisins, en utilisant les messages ADS-B reçus, qui contiennent l'identifiant du voisin, sa position, sa vitesse, le dernier waypoint qu'il a franchi, et le prochain qu'il atteindra. L'avion déduit la date t_j d'arrivée de chaque voisin j au prochain waypoint grâce à la date courante t, la distance d_j entre le voisin j et ce waypoint, et la vitesse v_j du voisin :

$$t_j = t + \frac{d_j}{v_j} \tag{3.1}$$

3.2.2 Décision

Calculer les vitesses admissibles pour l'avion peut être vu comme la recherche de l'intersection de trois intervalles de vitesse :

1. L'intervalle de vitesses auxquelles l'avion peut physiquement voler, soit l'intervalle $\left[v_{i,\text{opt}} - 6\,\%, v_{i,\text{opt}} + 3\,\%\right]$;
2. L'intervalle de vitesse permettant d'éviter les conflits avec les voisins directs ;

3. L'intervalle de vitesse permettant d'éviter les conflits avec les voisins appartenant aux autres flux.

Pour éviter un conflit, les avions i et j doivent maintenir une distance de séparation S_{ij}. Dans le cas des voisins directs, l'avion i doit être au moins à 5 NM de son prochain waypoint quand son prédécesseur P_i atteint celui-ci, et l'avoir dépassé d'au moins 5 NM quand son successeur F_i atteint cette position ($\S_0 = 5$ NM). Ces deux contraintes définissent un intervalle de dates d'arrivée au point de croisement qui ne provoquent pas de conflit.

À une date t, l'avion i se situe à la distance d_i de son prochain waypoint, et veut calculer sa vitesse maximale $v_{i,\max}$. D'abord, il calcule la vitesse maximale que lui permettent ses performances : $v_{i,\max|v_{i,\text{opt}}} = v_{i,\text{opt}} + 3\,\%$ (vitesse maximale par rapport à la vitesse optimale). En maintenant cette vitesse, il se situera à la distance $d_i = S_0$ avant le waypoint à la date $t_{i,\min|v_{i,\text{opt}}}$ (date minimale par rapport à la vitesse optimale) :

$$t_{i,\min|v_{i,\text{opt}}} = t + \frac{d_i - S_0}{v_{i,\max|v_{i,\text{opt}}}}\,. \tag{3.2}$$

Ensuite, l'avion prend en compte son prédécesseur P_i, de manière à atteindre la position $d_i = S_{iP_i}$ quand P_i atteint le prochain waypoint. Soit v_{P_i} la vitesse de l'avion P_i et d_{P_i} la distance entre P_i et le prochain waypoint de l'avion i. La date d'arrivée t_{P_i} au prochain waypoint est :

$$t_{P_i} = t + \frac{d_{P_i}}{v_{P_i}}\,. \tag{3.3}$$

En combinant (3.2) et (3.3), à cette étape, la date d'arrivée au plus tôt de l'avion au prochain waypoint est :

$$t_{i,\min} = \max(t_{i,\min|v_{i,\text{opt}}}, t_{P_i})\,. \tag{3.4}$$

Sa vitesse maximale est donc :

$$v_{i,\max} = \frac{d_i - S_0}{t_{i,\min} - t}\,. \tag{3.5}$$

De la même manière, la vitesse minimale $v_{i,\min}$ de l'avion est calculée par des équations similaires à (3.5), (3.3), (3.4) et (3.5), dans lesquelles :

- $v_{i,\max|v_{i,\text{opt}}}$ est remplacée par $v_{i,\min|v_{i,\text{opt}}} = v_{i,\text{opt}} - 6\,\%$ (vitesse minimale par rapport à la vitesse optimale) ;
- Le prédécesseur P_i par le successeur F_i ;
- $d_i - S_0$ par $d_i + S_0$;
- Et la fonction max par la fonction min :

$$v_{i,\min} = \frac{d_i + S_0}{\min\left(t + \frac{d_i + S_0}{v_{i,\min}}, t_{F_i}\right) - t}\,. \tag{3.6}$$

Une fois que cet intervalle de vitesse $\left[v_{i,\min}, v_{i,\max}\right]$ est établi, l'avion choisit sa vitesse de manière à être le plus proche possible de sa vitesse optimale $v_{i,\text{opt}}$:

$$v = \begin{cases} v_{i,\max} & \text{si } v_{i,\max} < v_{i,\text{opt}}, \\ v_{i,\min} & \text{si } v_{i,\min} > v_{i,\text{opt}}, \\ v_{i,\text{opt}} & \text{sinon.} \end{cases} \tag{3.7}$$

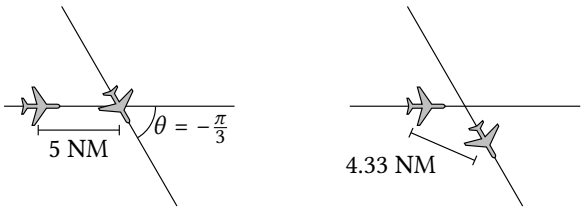


FIGURE 3.5 – Utiliser une distance de séparation $S_0 = 5\,\text{NM}$ pour des voisins au waypoint crée un conflit.

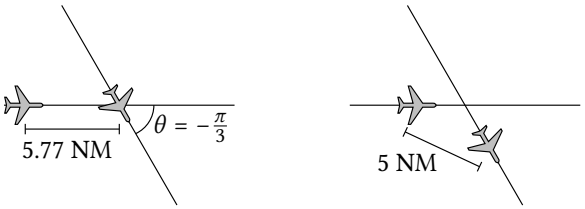


FIGURE 3.6 – Pour éviter les conflits entre voisins au waypoint, la distance de séparation doit être augmentée.

Cette méthode permet à un avion de prendre en compte ses voisins directs (sur le même arc du réseau de routes). Comme tous les avions appliquent la même méthode, chacun finit par trouver une vitesse permettant d'éviter les conflits.

Pour éviter les conflits avec les voisins au waypoint, le même calcul est réalisé une seconde fois, en remplaçant les voisins directs par les voisins au waypoint. Dans cette deuxième étape, l'intervalle de vitesse $\left[v_{i,\min}, v_{i,\max}\right]$ obtenu dans la première étape remplace l'intervalle $\left[v_{i,\min|v_{i,\text{opt}}}, v_{i,\max|v_{i,\text{opt}}}\right]$ (les performances de l'avion), et en utilisant les dates d'arrivée des voisins au waypoint au lieu de celles des voisins directs.

Ce processus en deux étapes permet de calculer l'intersection de trois intervalles de vitesse : l'intervalle permis par les performances de l'avion, l'intervalle permettant d'éviter les conflits avec les voisins directs, et l'intervalle permettant d'éviter les conflits avec les voisins au waypoint. Si à une des deux étapes, les intervalles sont disjoints, l'avion choisit la vitesse moyenne entre la borne supérieure de l'intervalle le plus lent et la borne inférieure de l'intervalle le plus rapide, et l'algorithme s'arrête. Si cette vitesse n'appartient pas à l'intervalle $\left[v_{i,\min|v_{i,\text{opt}}}, v_{i,\max|v_{i,\text{opt}}}\right]$, l'avion sélectionne la vitesse la plus proche à l'intérieur de cet intervalle.

Maintenir une distance de séparation $S_0 = 5\,\text{NM}$ entre deux voisins directs (appartenant au même flux) est suffisant pour garantir une situation sans conflit. Par contre, pour des voisins au waypoint, si la même valeur est utilisée dans le calcul de l'intervalle de vitesse, un conflit peut apparaître, comme le montre la figure 3.5.

Comme indiqué dans [YD13a, Lemma 1], la distance de séparation S_{ij} doit être augmentée en fonction de la vitesse et du cap relatifs des avions :

$$S_{ij} = \frac{S_0\sqrt{\alpha^2 - 2\alpha\cos\theta_{ij} + 1}}{\sin\theta_{ij}}, \tag{3.8}$$

avec $S_0 = 5\,\text{NM}$ la norme de séparation, θ_{ij} l'angle entre les caps des avions, et

$v_i = \alpha v_j$, v_i étant la vitesse de l'avion i et v_j celle de son voisin. En considérant le scénario décrit par la figure 3.6, si les deux avions volent à la même vitesse selon des trajectoires séparées par un angle $\theta_{ij} = \frac{\pi}{3}$, ils doivent être séparés par $S_{ij} \approx 5{,}77$ NM au moment où le premier avion passe le point de croisement pour que leur distance minimale soit de 5 NM ensuite.

L'équation (3.8) est adaptée à la régulation de flux d'avions, mais pas à la gestion d'avions individuellement. À l'échelle d'un flux, en connaissant l'angle entre ceux routes, et en considérant que tous les avions de chaque route vont à la même vitesse, cette équation donne la distance de séparation entre les avions de chaque flux. Les avions doivent alors être déplacés le long de ce flux pour respecter cette position et cette vitesse.

L'équation (3.8) ne peut cependant pas être utilisée directement dans Plectre v1, pour deux raisons. La première est qu'elle considère que la vitesse des deux avions est connue, ce qui permet de calculer le coefficient α. Or, si la vitesse de chaque voisin j de l'avion est considérée constante, la vitesse de l'avion i est inconnue, puisque c'est cette valeur qui est recherchée par l'algorithme de décision. L'équation (3.8) comporte donc deux inconnues, S_{ij} et α.

Le deuxième désavantage de cette équation est qu'elle ne fonctionne pas dans les cas où l'angle θ_{ij} tend vers 0° ou 180° : dans ces cas-là, $\sin \theta_{ij}$ tend vers 0. La deuxième situation ($\theta_{ij} \simeq 180°$) peut être ignorée parce que les avions sont dans une situation de conflit en face-à-face, qui ne peut pas être résolue par régulation de vitesse uniquement. Mais la première ($\theta_{ij} \simeq 0°$) est bien plus courante et et problématique : les avions sur le même arc du réseau de routes peuvent être à quelques dizaines de mètres à côté de la route, par exemple à cause d'un virage. En convergeant vers le même point, leurs caps peut donc différer de quelques degrés, ce qui implique que leur distance de séparation est légèrement supérieure à 5 NM.

Ce deuxième problème peut être contourné de deux manières. La première consiste à considérer que les avions dont le cap diffère de moins de quelques degrés doivent être séparés de 5 NM, ce qui peut provoquer des conflits, puisque par définition les avions ayant des caps différents doivent être séparés de plus de 5 NM pour éviter les conflits au point de convergence. La deuxième manière de prendre en compte les angles θ_{ij} faibles est d'appliquer une marge, en définissant par exemple une valeur valeur plancher de 5° pour θ_{ij}. L'inconvénient est que la distance de séparation S_{ij} résultante est artificiellement haute, ce qui diminue légèrement la capacité du réseau de routes, qui est maximale quand les avions se suivant sur un arc du réseau sont séparés de 5 NM.

Une formulation différente a été implémentée dans Plectre v1. En se basant sur la situation décrite dans la figure 3.7, et en considérant que les avions volent à la même vitesse, un calcul simple permet de déduire la distance S_{ij} en fonction de l'angle θ_{ij} :

$$\cos\left(\frac{\theta_{ij}}{2}\right) = \frac{S_0/2}{S_{ij}/2} \tag{3.9}$$

$$\frac{S_{ij}}{2} = \frac{S_0/2}{\cos\left(\theta_{ij}/2\right)} \tag{3.10}$$

$$S_{ij} = \frac{S_0}{\cos\left(\theta_{ij}/2\right)} \tag{3.11}$$

Même si cette équation ne prend pas en compte les différences de vitesse, elle permet tout de même de prendre en compte les différences de caps faibles. Par manque

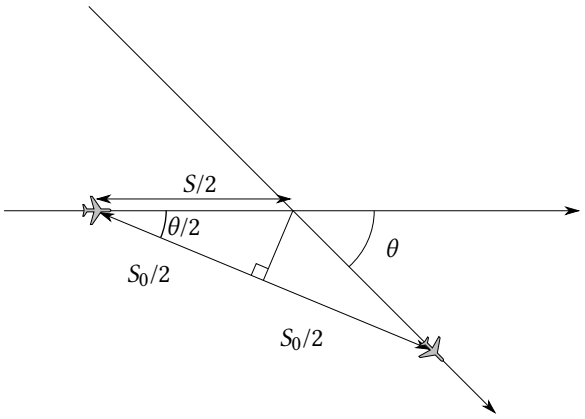


FIGURE 3.7 – La distance S peut être déduite de l'angle θ entre les deux routes.

de temps [1], le problème de la prise en compte des vitesses n'a pas été résolu. Il se peut qu'une partie des conflits non résolus par Plectre v1 soit dû à cette simplification.

3.2.3 Action

Durant la phase d'action, l'avion actualise sa position en fonction de la vitesse calculée durant la phase de décision.

Toutes les 10 secondes, il envoie également un message ADS-B contenant sa position sur le réseau de route et sa vitesse. Il n'est pas nécessaire d'envoyer un message chaque seconde pour que l'algorithme fonctionne : le temps de vol étant suffisamment long, le système peut mettre quelques minutes à converger vers une solution stable dans laquelle les avions ont sélectionné leur vitesse définitive en fonction des choix des voisins.

L'envoi des messages n'est pas effectué au même instant par tous les avions. Dans le cas contraire, des phénomènes d'oscillation pourraient apparaitre : deux avions « résolvent » un conflit par des manœuvres incompatibles et envoient un message au même instant, avant de corriger le conflit généré par ces manœuvres par d'autres manœuvres incompatibles. Ensuite, les avions retournent à leur choix initial, et le cycle recommence.

En décalant le moment où les messages sont envoyés, le premier avion indique quelle est sa trajectoire sans connaitre l'existence du deuxième avion. L'autre avion planifie une trajectoire qui résout les conflits avec le premier. Si le conflit persiste parce que l'action du deuxième avion n'est pas suffisante, le premier avion adapte sa vitesse pour contribuer à la résolution du conflit.

3.3 Résultats

L'algorithme Plectre v1 cherche à reproduire une structuration de trafic aérien conforme au Miles-in-Trail à l'intersection de deux flux d'avions. Dans les scénarios de test mis en place, deux routes définies par 5 waypoints de l'espace aérien Français

1. En effet, à ce moment-là, le développement de Plectre v2 (section 4) avait commencé, et dans ce second algorithme, il n'est pas nécessaire de calculer explicitement S.

ont été utilisées, comme l'indique la figure 3.1. Les waypoints LMG (Limoges) et MEN (Mendes) sont séparés par 116 NM, MEN et MRM (Marseille) par 119 NM, TOU (Toulouse) et MEN par 97 NM, et MEN et LYS (Lyon) par 105 NM. Les avions sont générés aux deux waypoints les plus à l'ouest (TOU et LMG), et les intervalles de temps entre deux départs suivent un processus de Poisson pour chaque flux, ce qui est considéré comme une bonne approximation du trafic réel en route [Sal+12, p. III–C]. La vitesse de croisière $v_{i,\mathrm{opt}}$ de chaque avion est choisie aléatoirement dans l'intervalle [447 kt, 497 kt]

Pour valider l'algorithme, trois versions de deux scénarios sont étudiées, chacune de ces 6 versions étant exécutées 100 fois pour obtenir la moyenne des résultats. Les avions sont générés et volent durant une heure. Un processus Poissonnien prend un unique paramètre λ déterminant le nombre moyen d'avions par heure sur un flux. Les deux scénarios correspondent à deux valeurs différentes du paramètre λ. Dans le premier scénario, les avions sont générés en moyenne toutes les 140 s ($\lambda = 1/140\,\mathrm{s}^{-1}$, soit environ 25,7 avions par heure).

Dans le deuxième scénario, les avions sont générés en moyenne toutes les 110 s ($\lambda = 1/110\,\mathrm{s}^{-1}$, soit environ 32,7 avions par heure). Dans ce scénario, la capacité maximale théorique de cette topologie de routes est atteinte, en considérant que la vitesse de croisière moyenne est de 467 kt et que la distance minimale de séparation entre avions sur deux routes perpendiculaire est de 14,14 NM.

Pour chaque λ, trois versions sont testées. Dans la première, le processus de décision des avions est activé, et tous les avions coopèrent pour résoudre les conflits. Dans la deuxième version de chaque scénario, le processus de décision est désactivé, les conflits ne sont pas résolus. Dans la troisième version, le processus de décision est désactivé pour 10 % des avions, sélectionnés aléatoirement. Cette dernière version permet de vérifier les propriétés de résilience de l'approche proposée, en simulant des situations où certains avions ne peuvent pas ou ne veulent pas coopérer, par exemple à la suite d'une panne.

Durant les simulations, la distance séparant chaque paire d'avions est mesurée en permanence. À chaque fois que deux avions sont séparés par moins de 5 NM, un conflit est enregistré. Un seul conflit est enregistré, quelque soit sa durée. Chaque enregistrement contient la distance minimale mesurée entre les avions durant ce conflit.

Les résultats obtenus pour les différents scénarios sont présentés dans le tableau 3.1, chacune des valeurs indiquées étant la moyenne de 100 simulations. Sans régulation, les distances minimales mesurées entre deux avions durant un conflit sont réparties uniformément dans l'intervalle [0 NM, 5 NM], avec une valeur moyenne de 2,5 NM. Quand le processus de régulation de vitesse est activé, environs deux tiers des conflits sont résolus.

Cependant, la distance minimale mesurée entre chaque paire d'avions est presque toujours supérieure à 4 NM, sauf une fois par simulation en moyenne. La moyenne des distances minimales mesurées est de 4,5 NM. Ces données indiquent que l'algorithme détecte et gère tous les conflits, mais n'est pas capable de maintenir constamment la norme de séparation de 5 NM. La distance mesurée durant les conflits oscille constamment autour de 5 NM ; à chaque fois que la distance passe sous ce seuil, le nombre de conflits est incrémenté. Ce comportement génère un nombre anormalement haut de conflits de faible importance. Pour prendre en compte ce défaut dans la suite de ce chapitre, les conflits dont la distance minimale mesurée est supérieure à 4 NM sont considérés comme des faux positifs, et ne sont pas comptés dans le nombre de conflits. Dans le cas d'une implémentation dans un système opérationnel, il est

TABLE 3.1 – Performances de l'algorithme Plectre v1 (valeurs moyennes pour 100 simulations).

λ (s^{-1})	Processus de décision	Nombre d'avions	Conflits	Conflits < 4 NM
$\frac{1}{140}$	Désactivé	52,3	15,6	12,4
	Activé	52,3	8,4 (−46 %)	0,8 (−94 %)
	Désactivé pour 10 % des avions	52,3	9,3 (−40 %)	1,7 (−86 %)
$\frac{1}{110}$	Désactivé	66,3	27,5	21,4
	Activé	66,3	18,1 (−34 %)	1,5 (−93 %)
	Désactivé pour 10 % des avions	66,5	19,6 (−29 %)	3,7 (−83 %)

λ (s^{-1})	Processus de décision	Distance minimale (NM)	Distance moyenne (NM)	Accélérations	Distance à $v_{i,\text{opt}}$ (%) [a]
$\frac{1}{140}$	Désactivé	0,31	2,54	0	0
	Activé	3,74	4,50	399	1,14
	Désactivé pour 10 % des avions	3,03	4,31	359	1,08
$\frac{1}{110}$	Désactivé	0,07	2,53	0	0
	Activé	3,32	4,49	505	1,40
	Désactivé pour 10 % des avions	2,39	4,31	446	1,26

a. À chaque instant, la distance à la vitesse optimale est calculée ainsi pour chaque avion : $d = 100 \times \frac{|v - v_{i,\text{opt}}|}{v_{i,\text{opt}}}$. La moyenne de toutes les valeurs collectées est reportée dans cette colonne.

néanmoins préférable d'augmenter la norme de séparation à 6 NM de manière à ce que la distance de séparation entre avions ne descende jamais en dessous de 5 NM [2].

Ainsi, si les seuls conflits considérés sont ceux pour lesquels la distance minimale de séparation est inférieure à 4 NM, dans le premier scénario ($\lambda = 1/140\ s^{-1}$), l'algorithme résout 94 % des conflits (0,8 conflit résiduels comparé aux 12,4 conflits initiaux). Dans le deuxième scénario ($\lambda = 1/110\ s^{-1}$), 93 % des conflits sont résolus (1,5 conflits restants par rapport aux 21,4 conflits initiaux). Ces valeurs indiquent que quand tous les avions coopèrent pour résoudre les conflits, les performances de cet algorithmes restent stables et proche de l'optimum jusqu'à ce que la capacité maximale du réseau de routes soit atteinte ($\lambda = 1/110\ s^{-1}$).

D'un autre côté, la vitesse de l'avion est ajustée chaque seconde pour s'adapter au trafic environnant. Cette stratégie amène l'avion à exécuter un grand nombre de petites accélérations. Dans ces scénarios de trafic, le temps de vol moyen est de 28 min (1 680 s), ce qui implique qu'un avion passe jusqu'à un tiers de son temps à accélérer et décélérer (figure 3.8). L'écart moyen entre les vitesses courante et optimale est

2. Il serait encore mieux de modifier le processus de décision pour que cette situation n'apparaisse pas. Mais comme expliqué précédemment, le développement de l'algorithme Plectre v1 a été interrompu au profit de Plectre v2, décrit dans le chapitre 4.

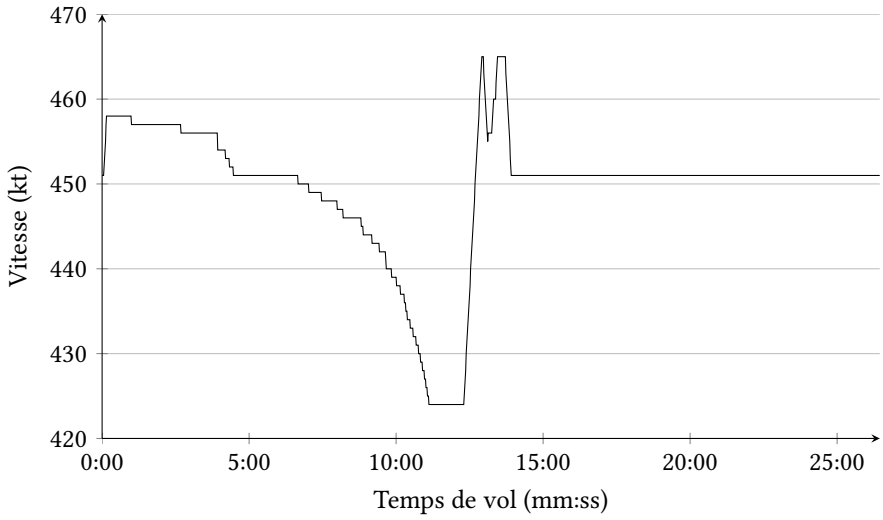


FIGURE 3.8 – Profil de vitesse d'un avion dans l'algorithme Plectre v1. Cet avion ralentit progressivement pour éviter un conflit avec un voisin de waypoint, puis revient à sa vitesse optimale pour la deuxième moitié moitié de son vol (après l'intersection).

moins élevée dans le premier scénario (1,14 %) que dans le deuxième (1,40 %). Cela indique qu'une densité plus faible d'avions impose moins de contraintes à chaque avion : ils ont plus de degrés de liberté et peuvent choisir une vitesse plus proche de la vitesse de croisière. De la même manière, le nombre d'accélérations est plus faible en cas de trafic moins dense.

Comme indiqué dans la section 2.1, la résilience est la capacité d'un système à retrouver un certain niveau de fonctionnement après avoir rencontré un évènement imprévu. Dans une situation nominale, quand tous les avions coopèrent pour résoudre les conflits, ce système multi-agents est capable de résoudre jusqu'à 94 % des conflits. Dans une situation dégradée, quand une partie du système ne fonctionne pas normalement, un système résilient devrait continuer à résoudre les conflits avec une efficacité similaire. La résilience de ce système multi-agents peut donc être mesurée comme la différence entre le nombre de conflits résolus quand le système fonctionne normalement et quand une partie du système ne fonctionne pas. Pour simuler un mode dégradé, il a été choisi de modifier le processus de décision d'un sous-ensemble des avions. La panne de ces agents est simulée en désactivant leur processus de décision. Ils sont capables de diffuser des informations concernant la trajectoire qu'ils prévoient de suivre, mais ils ne cherchent pas à résoudre les conflits et ne changent pas leur trajectoire.

La version des scénarios où le processus de décision de 10 % des avions est désactivé montre la résilience de cet algorithme face aux évènements imprévus, quand certains agents ne coopèrent pas pour résoudre les conflits. Dans le premier scénario ($\lambda = 1/140\,\mathrm{s}^{-1}$), le nombre et la sévérité des conflits est très proche des versions où tous les avions coopèrent : si les seuls conflits considérés sont ceux où la distance de séparation minimale est inférieure à 4 NM, le nombre moyen de conflits augmente de 0,8 à 1,5 (+5 % en comparaison avec les 12,4 conflits dans le scénario non régulé). Cette valeur indique que l'algorithme est résilient aux perturbations locales, (lorsque 10 %

des avions deviennent non-coopératifs), tant que la capacité maximale du réseau de routes n'est pas atteinte. Les avions coopératifs considèrent les avions non-coopératifs comme des contraintes pour leur processus de décision, et sont encore capables de résoudre les conflits seuls.

Cependant, les 10 % d'avions non-coopératifs forcent les avions coopératifs à effectuer plus de manœuvres pour résoudre les conflits. Cela est particulièrement visible pour $\lambda = 1/140\,\mathrm{s}^{-1}$. Dans la version du scénario où 100 % des avions coopèrent, chaque avion vole en moyenne à ±1,14 % de sa vitesse optimale. En effectuant un simple ratio, quand 90 % des avions coopèrent, cette valeur moyenne devrait diminuer à ±1,02 % (1,14 × 0,9). Or, la valeur moyenne obtenue quand 10 % des avions ne coopèrent pas est de 1,08 %. Les avions coopératifs doivent donc fournir un effort plus important pour résoudre les conflits quand 10 % des avions ne coopèrent pas à la résolution de conflits.

Dans le deuxième scénario ($\lambda = 1/110\,\mathrm{s}^{-1}$), le nombre de conflits augmente de 10 % (de 1,5 à 3,7 en comparaison avec les 21,4 conflits initiaux). Dans ce scénario, la capacité maximale du réseau de routes est atteinte, et l'algorithme devient plus sensible aux perturbations. Le trafic est tellement dense que chaque avion non-coopératif a un impact important sur les performances de l'algorithme. Néanmoins, l'algorithme est capable de résoudre 83 % des conflits. Tant que les avions coopératifs connaissent la position et la vitesse de leurs voisins non-coopératifs, ils sont capables d'éviter les conflits, parce que les avions non-coopératifs sont pris en compte comme contraintes dans le processus de décision des avions coopératifs.

3.4 Conclusion

Cette méthode de résolution de conflits fonctionne donc en deux temps. D'abord, les avions déterminent leur ordre de passage au prochain point de croisement en considérant qu'ils volent à leur vitesse optimale $v_{i,\mathrm{opt}}$, sans tenir compte des conflits. Puis ils choisissent la vitesse la plus proche de $v_{i,\mathrm{opt}}$ qui leur permet d'éviter les conflits avec leurs voisins.

La vitesse optimale est ici choisie aléatoirement, mais dépend dans la réalité des performances de l'avion et des préférences de la compagnie aérienne, par le biais du Cost Index (CI, section 1.1.3.3.1), qui permet pour un avion de privilégier le coût du carburant (en volant plus lentement) ou le coût de l'équipage (en cherchant à arriver à l'heure).

Si le processus de décision de Plectre v1 cherchait à optimiser le coût du vol au lieu de l'écart à la vitesse optimale, les résultats obtenus pourraient être différents, notamment dans le cas où les avions veulent arriver en même temps au point de croisement. Par exemple, un avion arrivant légèrement avant un autre privilégie systématiquement dans Plectre v1 d'accélérer pour augmenter la distance de séparation. Si cet avion avait un CI privilégiant les économies de carburant, il serait plus optimal en terme de coût de ralentir fortement pour laisser son voisin passer avant lui.

Mais pour obtenir ce résultat, il faudrait d'une part complexifier le processus de décision pour que la vitesse $v_{i,\mathrm{opt}}$ dépende du CI. Cela implique que $v_{i,\mathrm{opt}}$ varie avec le temps : un avion qui a été contraint de ralentir pour résoudre un conflit pourrait choisir d'accélérer pour rattraper le retard induit.

D'autre part, introduire un mécanisme qui permet aux avions d'échanger leur place dans l'ordre de passage au point de croisement implique l'implémentation de mécanismes de négociation entre avions. Cela nécessiterait probablement l'échange

d'informations plus variées que ce qui est permis par l'ADS-B (tel qu'énoncé dans les hypothèses), comme le coût estimé des différents choix possibles. Mais les résultats obtenus seraient probablement plus pertinents dans l'optique de l'optimisation des coûts du vol.

Il faut noter que l'écart entre la vitesse optimale et la vitesse choisie par les avions dépend de la densité du trafic. Plus le trafic est faible, plus les avions peuvent se rapprocher de la vitesse optimale. Plus le trafic se densifie, plus les vitesses tendent à s'uniformiser, pour que tous les avions du flux maintiennent une vitesse et une distance constante entre eux. Cela implique que les avions dont la vitesse optimale est la plus élevée ou la plus faible doivent choisir la vitesse la plus éloignée de leur vitesse optimale.

Cet algorithme a plusieurs limites. Même si les avions tentent de choisir une vitesse qui reste stable jusqu'au prochain waypoint, en pratque les avions changent constamment de vitesse pour s'adapter au trafic environnant. Il serait plus efficace de planifier uniquement quelques accélérations : cela permettrait de réduire l'oscillation de la vitesse et une meilleure prédictibilité du comportement des avions. De plus, changer le régime d'un réacteur d'avion trop souvent réduit sa longévité.

Les décisions sont basées sur la date d'arrivée au prochain waypoint. Cette stratégie est bien adaptée aux réseaux de routes composés d'arrêtes longues (de l'ordre de quelques dizaines de miles nautiques). Elle fonctionnera donc moins efficacement pour des réseaux plus complexes dont les arrêtes sont plus courtes :

— Premièrement, les avions peuvent ne pas avoir le temps d'ajuster leur vitesse pour éviter les conflits durant le parcours d'une arête. L'algorithme résout donc moins de conflits.

— Deuxièmement, puisqu'ils ne prennent pas en compte le trafic au delà du prochain waypoint, ils sont incapables d'anticiper leurs décisions dans un réseau dense : une manœuvre qui résout un premier conflit *avant* le point de croisement peut en créer un autre *après* ce waypoint ; l'anticipation de l'état du trafic après le point de croisement peut permettre de résoudre les deux conflits en choisissant une manœuvre différente.

En conclusion, l'algorithme Plectre v1 fonctionne pour les réseaux utilisés pour le Miles-in-Trail, mais pas pour les autres réseaux de routes.

Plus généralement, le MIT a quelques désavantages : les avions sont déroutés pour être mis sur un réseau, ce qui rallonge notablement les trajectoires. De plus, les avions appartenant au même flux doivent voler à la même vitesse, qui peut être différente de leur vitesse optimale. Ces deux défauts cause une surconsommation de carburant et donc des pertes financières.

Pour dépasser ces limites, une manière plus générique de représenter le problème de régulation de vitesse pour la résolution de conflits a été développée. Cette approche est basée sur l'estimation et l'échange par messages de trajectoires 4D complètes. En effet, de nos jours, les avions sont capables de communiquer leur future trajectoire 4D, ce qui permet une prédiction de trajectoires plus précise qu'un simple plan de vol basé sur la date de survol de waypoints. L'algorithme décrit dans le chapitre 4 tire avantage de ces informations pour calculer des solutions plus efficacement.

Chapitre 4

Résolution de conflits sans réseau de routes

4.1 Hypothèses

L'algorithme présenté dans ce chapitre, Plectre v2, est une extension de l'algorithme présenté dans le chapitre 3 (Plectre v1). Le problème résolu par Plectre v2 est légèrement différent de celui de Plectre v1. Pour le premier algorithme, la configuration du Miles-in-Trail garantit que tous les conflits peuvent être résolus en régulant uniquement la vitesse des avions, tant que ceux-ci sont suffisamment espacés sur leurs routes. Par contre, dans une situation de trafic plus générale, ou dans les Free Flight Airspace, certains conflits, comme les conflits en face-à-face, ne peuvent pas être résolus par le contrôle de la vitesse. Pour cette raison, le but de Plectre v2 est la simplification du trafic du point de vue des contrôleurs aériens par régulation subliminale. Comme dans le cas du projet ERASMUS (section 1.2.4.2.1), l'algorithme minimise le nombre de conflits, les conflits résiduels étant pris en charge par les contrôleurs.

Pour ERASMUS, la régulation de vitesse a plusieurs avantages sur le changement de cap et d'altitude. Comme évoqué précédemment, les contrôleurs aériens privilégient le changement de cap parce que le conflit est résolu rapidement sans entraîner de surconsommation de carburant. Si l'algorithme prenait ce type de décision, les contrôleurs pourraient voir cela comme une interférence avec leur propre travail.

Le changement d'altitude a également plusieurs désavantages. D'une part, ils nécessitent plus de temps que les changements de cap. D'autre part, les avions situés sur des niveaux de vol différents ne peuvent pas être en conflit, deux niveaux de vol étant séparés de 1 000 ft. Si un avion change d'altitude, le contrôleur peut percevoir une augmentation de la complexité de la situation, puisque la probabilité que cet avion entre en conflit avec un autre augmente.

Les changements de vitesse impactent moins la perception de la complexité par le contrôleur. Par hypothèse, l'écart *maximal* entre la vitesse optimale d'un avion et sa vitesse minimale est de 30 kt (487 kt + 6 %). Or, l'incertitude de vitesse au sol liée au vent suit une loi normale dont l'écart type σ est de 15 kt. Du point de vue du contrôleur aérien, l'« incertitude » liée aux décisions prises par l'algorithme est du même ordre de grandeur (la dizaine de nœuds) que l'incertitude liée au vent.

Du point de vue algorithmique, ce problème résolu par Plectre v2 peut être vu comme une généralisation de celui de Plectre v1, où les avions adaptent leur vitesse

pour minimiser le nombre de conflits, mais sans le haut degré de structuration imposé par le réseau de route du Miles-in-Trail. Les avions peuvent suivre n'importe quelle trajectoire 4D, soit en suivant un réseau de routes, soit en suivant une trajectoire optimale, dans le cadre du Free Flight.

Les avions sont régis par des contraintes similaires à celles de l'algorithme Plectre v1, décrits dans la section 3.1. La vitesse optimale $v_{i,\text{opt}}$ de chaque avion appartient à l'intervalle [447 kt, 487 kt]. Le Cost Index, et par conséquent la vitesse de croisière, sont supposés constants. L'intervalle de vitesses admissibles est $\left[v_{i,\text{opt}} - 6\,\%, v_{i,\text{opt}} + 3\,\%\right]$. Le taux standard de virage de 3°/s est utilisé, permettant aux avions d'effectuer un tour complet en 2 minutes. L'altitude est constante durant le temps de vol, ce qui implique que les avions situés sur des niveaux de vol différents ne peuvent pas être en conflit.

Dans Plectre v1, l'accélération était fixée à ±4 000 NM/h^2 (±1,11 kt/s) pour tous les avions, cette valeur étant définie dans la base de données BADA d'Eurocontrol [Eur11]. Dans Plectre v2, cette valeur définit la borne supérieure de l'accélération. En effet, si dans le premier algorithme la vitesse est mise à jour chaque seconde, dans le deuxième les accélérations sont planifiées avec une granularité de 5 secondes. Dans le modèle qui a été développé, les avions planifient leurs phases d'accélération à une date multiple de 5 secondes (t_0 + 1 845 s, t_0 + 17 590 s, etc.), cette accélération durant elle-même un multiple de 5 secondes.

Si l'accélération était fixée à ±4 000 NM/h^2, les avions ne pourraient pas atteindre leurs vitesses minimales et maximales. Par exemple, un avion dont la vitesse de croisière est de 487 kt pourrait atteindre la vitesse de 498 kt en 10 s, mais pas sa vitesse maximale de 502 kt. De même, une accélération d'une durée de 15 s lui ferait dépasser sa vitesse maximale. Dans Plectre v2, l'accélération de chaque avion est fixée à la valeur la plus proche de 4 000 NM/h^2 qui lui permet d'atteindre sa vitesse maximale (et minimale) en un multiple de 5 secondes. Cela permet de rester proche des hypothèses de Plectre v1, tout en permettant aux avions d'atteindre leurs vitesses extrémales.

La plus grande différence entre les hypothèses de départ de Plectre v1 et v2 est la capacité pour les avions du deuxième algorithme de communiquer leurs trajectoires 4D prévues pour les prochaines minutes. Ces informations sont échangées par un protocole similaire à l'ADS-C (voir la section 1.1.2.4). L'ADS-B utilisé dans Plectre v1 suppose que les avions envoient plusieurs fois par seconde un message contenant la position et la vitesse, sans savoir si un destinataire le recevra. L'ADS-C fonctionne selon un protocole de dialogue entre un avion et une antenne au sol (ou un satellite). L'antenne initie la communication, et demande à l'avion d'envoyer une seule fois ou de manière périodique un message contenant sa position et sa vitesse, mais aussi, optionnellement, la route prévue définie par un ensemble de positions 4D (3D + temps), et des informations météorologiques locales (température, vent).

Actuellement, les avions sont pas équipés de récepteurs ADS-C leur permettant d'échanger directement des informations à propos de leur trajectoire prévue. Un système multi-agents similaire à Plectre v2 ne peut donc pas être implémenté directement. Il nécessite la collecte de ces informations par des équipements au sol. L'algorithme fonctionnerait alors dans un système informatique centralisé, puis les nouvelles trajectoires seraient renvoyées à bord des avions. L'aspect distribué de l'approche multi-agents serait diminué, puisque les avions ne prennent pas de décisions par eux-mêmes, mais les autres avantages liés aux systèmes multi-agents, comme la résilience et la parallélisation des calculs, seraient conservés.

En principe, l'algorithme Plectre v2 ne fait que réduire le nombre de conflits, et délègue la résolution des conflits résiduels aux contrôleurs aériens. Dans l'implémen-

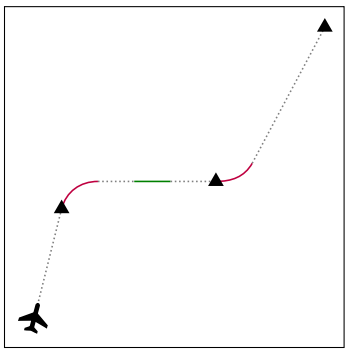


FIGURE 4.1 – Dans l'algorithme Plectre v2, une trajectoire est une courbe composée de segments (vitesse constante en pointillés, accélération en vert) et d'arcs de cercles (en rouge). Les avions volent de waypoint en waypoint (triangles).

tation proposée dans ce chapitre, pour remplacer les contrôleurs dans la résolution des conflits résiduels, Plectre v2 a la possibilité de résoudre les conflits résiduels en retardant l'heure de départ jusqu'à 10 minutes.

4.2 Implémentation d'un système multi-agents

Les trajectoires sont stockées et échangées sous forme de courbes composées d'arcs, qui sont des segments quand les avions volent en ligne droite (sur ces segments, les avions peuvent changer de vitesse), et des arcs de cercle quand les avions changent de cap (à vitesse constante). Ces trajectoires sont différentiables une fois (C1).

Chacune de ces courbes est construite à partir d'un plan de vol défini par un ensemble de waypoints. La trajectoire est construite de cette manière : l'avion doit survoler chaque waypoint, comme le montre la figure 4.1. Quand l'avion atteint un waypoint, il tourne pour se diriger vers le suivant. Les changements de vitesse sont planifiés à certains moments définis, et surviennent uniquement quand l'avion vole en ligne droite.

Comme le montre la figure 4.2, les coordonnées des points d'entrée et de sortie du virage sont calculées à partir du cap de l'avion avant l'entrée dans le virage, de la vitesse de l'avion, qui détermine le rayon du virage, et de la position du waypoint vers lequel se dirige l'avion après le virage. Si le waypoint cible est situé à l'intérieur du virage, le point de sortie est la projection de ce waypoint sur le cercle, c'est-à-dire le point le plus proche du waypoint appartenant au virage.

Connaitre les coordonnées des points d'entrée et de sortie ne suffit pas pour caractériser complètement le virage, il faut également en déterminer le sens. Supposer que les virages ne font jamais plus de 180° suffit dans la plupart des cas : les plans de vol étant conçus pour être les plus courts possibles, ils évitent les virages trop serrés. Cependant, il peut arriver que les plans de vol générés (notamment par les premières versions de l'algorithme décrit dans le chapitre 5) comportent des virages de plus de 180°, comme le montre la figure 4.3. Pour garder l'algorithme Plectre v2 le plus simple possible, la vérification de cette contrainte est déléguée à l'algorithme qui génère le plan de vol.

Dans ce modèle, un avion ne peut pas accélérer et tourner en même temps. Cela permet de représenter tous les virages par des arcs de cercle, et de détecter les conflits

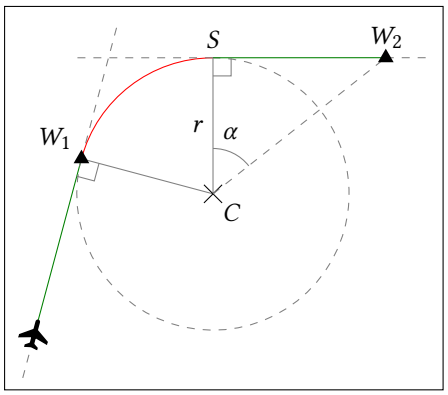


FIGURE 4.2 – Calcul d'un virage dans Plectre v2. Les deux prochains waypoints du plan de vol sont W_1 et W_2. W_1 est le point d'entrée du virage. Le rayon r du virage est déterminé à partir de la vitesse de l'avion (un tour complet en 2 minutes). Les points d'entrée W_1 et de sortie S appartiennent à des droites tangentes au cercle de centre C. Les coordonnées du point de sortie S sont calculées à partir de l'angle α.

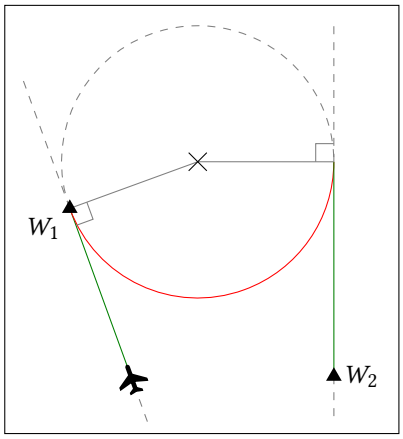


FIGURE 4.3 – Virage incorrectement calculé. Pour déterminer le sens du virage, l'algorithme suppose que le virage fait moins de 180°, ce qui est vrai pour les plans de vol réalistes. Mais dans certains plans de vol, la position des waypoints peut générer ce type virages « à l'envers ». Cette situation doit être prise en compte par les algorithmes de génération de plans de vol.

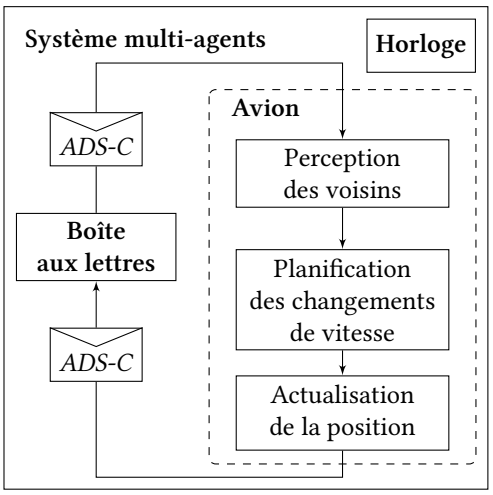


FIGURE 4.4 – Cycle de vie du système multi-agents Plectre v2.

en utilisant une méthode analytique plus simple que si l'avion pouvait accélérer et tourner en même temps. De plus, les plans de vol sont conçus pour minimiser la distance parcourue. Les changements de cap sont généralement petits, et les virages durent rarement plus de quelques dizaines de secondes. Reporter les accélérations après les virages a donc peu d'influence sur le processus de décision.

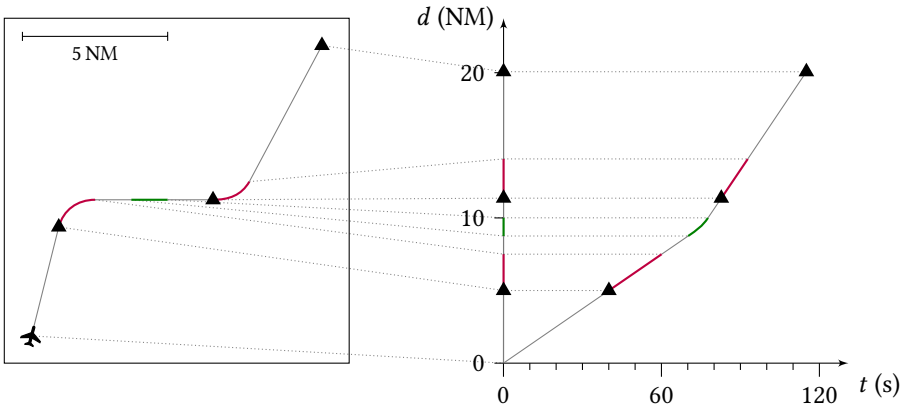
Le cycle de vie du système multi-agents est identique à celui de Plectre v1, comme le montre la figure 4.4. Le système est régulé par un horloge globale. Chaque impulsion d'horloge correspond à une seconde dans la simulation. Tous les agents sont synchronisés : à la fin d'une itération, les agents déposent les messages dans les boîtes aux lettres. Au début de l'itération suivante, ces messages sont délivrés à leurs destinataires. Les seules modifications apportées au système sont les informations échangées dans les messages, qui contiennent des trajectoires 4D, et le processus de décision des agents.

Les avions sont générés 10 minutes avant leur départ prévu. Ils commencent immédiatement à envoyer les messages ADS-C contenant leur trajectoire 4D prévue. Cela permet aux avions de planifier leur changement de vitesse avant d'apparaitre dans la simulation, et aux avions qui évoluent déjà dans la simulation d'anticiper l'apparition de ces nouveaux avions.

Durant le cycle de vie d'un agent, une séquence de trois étapes est répété à chaque itération du système multi-agents, jusqu'à ce que l'agent soit supprimé du système. La phase de perception permet aux agents de recevoir les messages ADS-C et de mettre à jour leur représentation interne de l'espace aérien. Durant la phase de décision, les avions planifient les changements de vitesse en se basant sur cette représentation interne. Durant la phase d'action, les agents rafraichissent leur position en respectant les changements de vitesse planifiés, et diffusent leur message ADS-C. Ces trois étapes sont détaillées dans la suite du chapitre.

4.2.1 Perception : détection de conflits

Plectre v2 régule uniquement la vitesse, c'est-à-dire la dimension temporelle de la trajectoire de l'avion. Les dimensions spatiales et temporelle sont donc gérées



Trajectoire 4D.

Distance sur l'abscisse curviligne en fonction du temps.

FIGURE 4.5 – Représentation interne de la trajectoire d'un avion.

séparément par l'algorithme. Les dimensions spatiales d'une trajectoire sont les dimensions de l'espace géométrique occupées par la trajectoire en 3D. L'altitude d'une trajectoire étant supposée constante, les avions séparés verticalement ne sont pas considérés en conflit. La dimension temporelle est définie par la vitesse de l'avion à chaque instant. L'algorithme régulant uniquement cette dimension, la représentation interne de la trajectoire d'un avion est la distance parcourue par l'avion en fonction du temps, et peut être représentée par une courbe dans un espace 2D, comme le montre la figure 4.5. La dérivée de cette courbe donne la vitesse instantanée de l'avion.

Chaque avion reçoit d'abord les messages de ses voisins, desquels il extrait les trajectoires 4D. Pour détecter les conflits, l'avion échantillonne alors ces trajectoires pour obtenir la position prévue des autres avions toutes les 10 secondes (figure 4.6).

Un conflit survient quand la distance séparant deux avions volant à la même altitude est inférieure à 5 NM horizontalement. Quand un avion suit une route, ses positions futures sont définies par sa position courante et par ses changements de vitesse prévus. Comme les changements de vitesse doivent être choisis durant la phase de décision, tous les conflits potentiels avec les avions du voisinage doivent être détectés, quelque soit la vitesse de l'avion. Donc, pour chaque position prévue des voisins, l'algorithme cherche les intersections entre la route de l'avion et des cercles de 5 NM centré sur les positions des voisins (figure 4.6). Ainsi, quelque soient les changements de vitesse choisis par l'avion, il sera toujours capable de détecter les conflits

Pour calculer ces changements de vitesse, les conflits doivent être représentés dans une forme compréhensible par le processus de décision. Les conflits détectés sont donc projetés dans l'espace 2D contenant la représentation interne de la trajectoire de l'avion (distance parcourue en fonction du temps), comme l'indique la figure 4.6. Chaque intrusion détectée à partir des trajectoires échantillonnées des voisins est projetée à partir de sa position le long de la route de l'avion. La portion de la route en intersection avec le cercle de 5 NM centré sur la position de l'intrus donne un intervalle de positions interdites à l'avion.

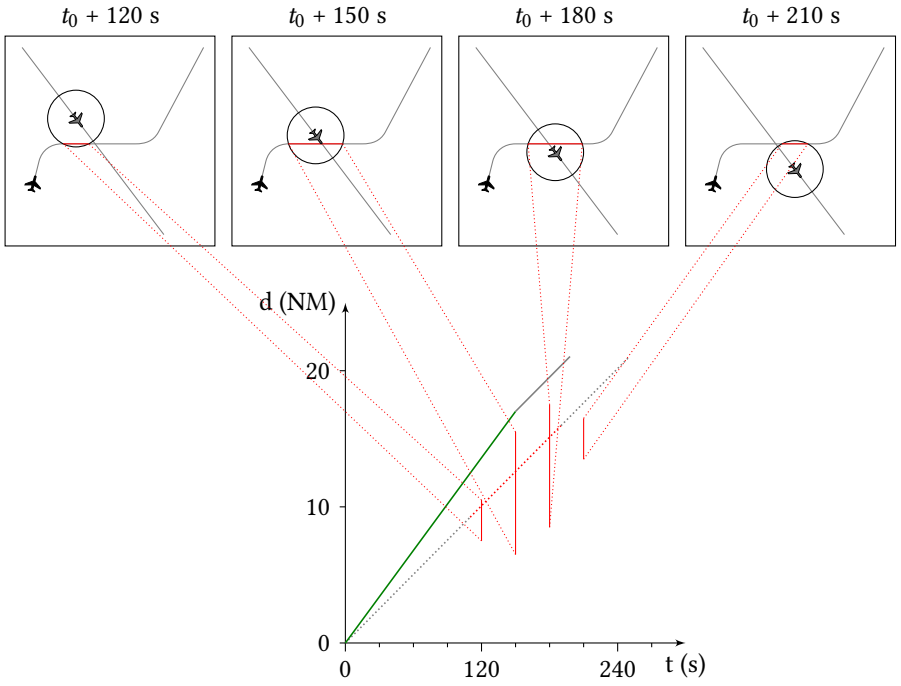


FIGURE 4.6 – Détection et représentation interne des conflits. L'avion noir détecte les conflits potentiels avec l'avion gris pour chaque pas de temps dans les prochaines minutes. Un conflit potentiel est l'intersection (segments rouges) entre un cercle de 5 NM centré sur l'avion gris et le chemin de l'avion noir. Ces segments sont projetés dans la représentation interne de la trajectoire de l'avion noir. L'intersection entre la courbe de la distance parcourue en fonction du temps (en pointillés) et les conflits potentiels indique que si l'avion maintient sa vitesse, un conflit aura lieu. L'avion doit donc accélérer (courbe verte).

Un exemple de ce processus est donné sur la figure 4.6. Les conflits sont symbolisés par les segments rouges représentant l'intersection entre le cercle de 5 NM centré sur la position de l'intrus et les composantes spatiales de la trajectoire de l'avion. Ces segments rouges sont projetés dans la représentation interne 2D de cette trajectoire.

4.2.2 Décision

Les décisions des avions sont mises à jour avec un intervalle de quelques secondes. La convergence du système vers une solution ne nécessitant que quelques itérations, et les conflits ne survenant pas avant plusieurs minutes, il n'est pas nécessaire de rafraichir les décisions plus de 5 ou 6 fois par minutes (ces valeurs sont arbitraires). Pour des raisons de simplicité, le taux de rafraichissement est fixé à 8 secondes : il s'agit du taux de rafraichissement par défaut de la position des avions dans le logiciel externe utilisé pour enregistrer et rejouer le trafic simulé par Plectre v2.

La date de la première décision de chaque avion est sélectionnée aléatoirement parmi les 8 premières secondes de la vie de cet agent. Cela permet d'éviter que les décisions soient prises en même temps par tous les avions, réduisant ainsi le risque

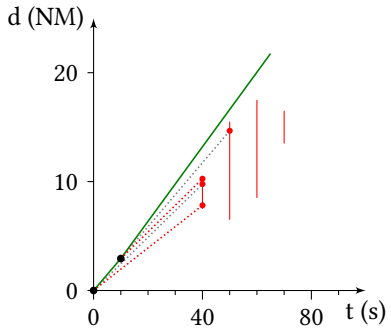


FIGURE 4.7 – Exploration de l'arbre de décisions. Pour chaque pas de temps, trois choix sont testés : accélérer (vert), maintenir sa vitesse (gris), ou décélérer (rouge). Le choix conduisant à la trajectoire le plus longtemps sans conflit est validé. Dans ce cas, l'avion accélère deux fois au début, puis maintient sa vitesse.

d'apparition des phénomènes d'oscillation déjà évoqués. Les oscillations résiduelles sont supprimées en retardant aléatoirement d'une seconde la date de décision, avec une probabilité de 1 %.

La sélection aléatoire de la date de la première décision, et le retardement avec une faible probabilité des décisions suivantes, impliquent aussi qu'en exécutant la simulation plusieurs fois à partir du même scénario, l'ordre dans lequel les avions prennent des décisions soit différent. Comme les décisions prises par le premier avion influent sur les décisions des avions suivants, les conflits peuvent être résolus de manière différente, ou même ne pas être résolus. Les performances de l'algorithme, notamment en terme de nombre de conflits résolus, peuvent donc varier entre deux exécutions de l'algorithme.

Durant la phase de décision, les avions peuvent prendre trois types de décisions pour résoudre les conflits, qui sont, par ordre de préférence, la régulation de la vitesse, le choix d'une route alternative, et le retard de la date de départ. L'objectif de chacune de ces méthodes est le même : maximiser la date du premier conflit qui ne peut pas être résolu.

L'avion tente d'abord de résoudre tous les conflits en régulant sa vitesse. S'il échoue, il essaie de sélectionner la route alternative où le premier conflit insoluble survient le plus tardivement. S'il reste des conflits, l'avion retarde alors son départ afin de retarder au maximum l'apparition du premier conflit dans lequel il est impliqué. L'algorithme s'arrête dès qu'une solution sans conflit est trouvée.

4.2.2.1 Régulation de la vitesse

En utilisant la représentation interne en deux dimensions des conflits décrite dans la figure 4.6, l'avion planifie ses changements de vitesse. Cette représentation interne permet également de représenter la trajectoire de l'avion par une courbe représentant la distance parcourue en fonction du temps. Le but est de résoudre les conflits en régulant la vitesse. L'avion construit donc cette courbe en explorant un arbre de décisions dans lequel, pour chaque pas de temps, l'avion peut maintenir sa vitesse, accélérer ou décélérer au taux maximum, dans la limite de $\pm 4\,000\,\mathrm{NM/h^2}$ (figure 4.7). Pour chaque choix, la date du premier conflit est calculée.

Le problème est résolu par un algorithme glouton qui maximise localement la date du premier conflit : à chaque pas de temps, l'avion évalue les trois choix possibles et sélectionne celui qui retarde au maximum la date du premier conflit.

Un pas de temps de 5 secondes a été choisi. Le taux d'accélération de chaque avion est choisi de manière à passer le plus rapidement possible de la vitesse de croisière $v_{i,\text{opt}}$ à la vitesse maximale $v_{i,\text{max}\,|v_{i,\text{opt}}}$ en un multiple de 5 secondes, sans dépasser l'accélération maximale de 4 000 kt/h (environ 1,11 kt/s). Par exemple, un avion dont la vitesse de croisière est 447 kt sélectionne un taux d'accélération de 0,894 kt/s, qui lui permet de passer de $v_{i,\text{opt}}$ à $v_{i,\text{max}\,|v_{i,\text{opt}}}$ en 15 s, et de $v_{i,\text{opt}}$ à $v_{i,\text{min}\,|v_{i,\text{opt}}}$ en 30 s. L'avion peut donc choisir 9 vitesses différentes.

Algorithme 3 Algorithme de régulation de la vitesse.

```
function OptimisationVitesse(route, retardDépart)
    choix ← {cst, acc, dec}
    D ← {}
    for t ← t_0 + retardDépart to t_fin step Δt do
        meilleurChoix ← cst
        meilleureDateConflit ← t
        for c in choix do
            datePremierConflit ← datePremierConflit(route, D ∪ {c})
            if estValide(D ∪ {c}) then
                if dateConflit > meilleureDateConflit then
                    meilleurChoix ← c
                    meilleureDateConflit ← dateConflit
                end if
            end if
        end for
        D ← D ∪ {meilleurChoix}
    end for
    return D
end function
```

L'algorithme 3 décrit le processus de planification des changements de vitesse. L'ensemble de décisions D est construit itérativement. Pour chaque pas de temps t à partir de la date de départ t_0 + retardDépart, les choix d'accélérer (acc), de décélérer (dec) et de maintenir une vitesse constante (cst) sont testés. La date à laquelle le premier conflit survient est stockée dans la variable datePremierConflit. L'algorithme sélectionne alors la décision retardant au maximum le premier conflit. Si les contraintes de vitesse sont respectées (vérifiées par la fonction estVaide()), cette décision est acceptée.

4.2.2.2 Choix d'une route alternative

Les avions ont également la possibilité de choisir entre plusieurs routes alternatives, comme indiqué dans la figure 4.8. Cette méthode, proposée par Yoo et Devasia [YD13a] et décrite dans la section 1.2.4.3.2, permet de dépasser la capacité maximale théorique d'un croisement entre deux flux d'avions dans une configuration Miles-in-Trail. Pour des routes parallèles, cette capacité est d'un avion tous les 14,14 NM sur chaque route, comme indiqué sur la figure 1.28. La topologie proposée, à trois voies parallèles, permet d'atteindre la capacité d'un avion tous les 5 NM.

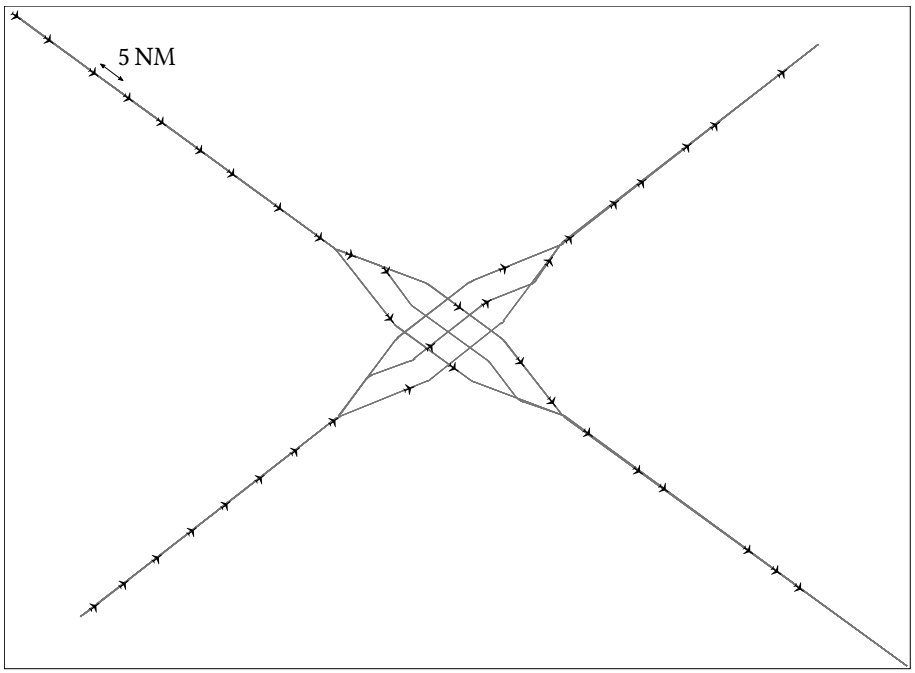


FIGURE 4.8 – Topologie de croisement à trois voies proposée dans [YD11] permettant d'augmenter la capacité du trafic organisé en Miles-in-Trail décrit dans la figure 1.9. Les routes parallèles sont séparés par 5 NM. La capacité maximale théorique de ce réseau permet de placer un avion tous les 5 NM sur chacune des deux routes.

Dans cette topologie de routes, les trois routes font la même longueur, et si les avions maintiennent la même vitesse entre le point d'entrée et de sortie, la distance de séparation entre avions sera la même avant l'entrée et après la sortie. Autrement dit, si les avions maintiennent leur vitesse, aucun confit n'est généré par une différence de longueur entre les routes que les avions peuvent emprunter.

Dans le scénario de la figure 4.8, les avions ont donc le choix entre trois routes alternatives. Dans Plectre v2, la structure de données représentant les plans de vol a été modifiée : au lieu d'une simple liste de waypoints, les plans de vol sont représentés par un arbre dont la racine est la première portion de route, commune à toutes les alternatives. La racine comporte au moins deux waypoints. Chaque branche de l'arbre représente une des routes alternatives.

Pour déterminer quelle route il emprunte, chaque avion détecte les conflits potentiels et applique l'algorithme de régulation de vitesse (algorithme 3) sur chacune des branches de l'arbre. Il sélectionne alors la branche qui maximise la date du premier conflit. Ce choix peut être modifié jusqu'à ce que l'avion atteigne la branche qu'il a choisie.

Il faut noter qu'il est en théorie possible que cette topologie de routes à plusieurs voies permette à un avion rapide évoluant sur une des routes de dépasser un avion plus lent évoluant sur une autre route. Il faut pour cela que la distance séparant les points d'entrée et de sortie de la topologie soit suffisamment grande, mais aussi que l'écart de vitesse soit suffisant.

Si on considère deux avions volant à la vitesse minimale et maximale permises par hypothèse, soit 420 kt (447 − 6 %) et 502 kt (487 + 3 %). L'écart de vitesse le plus important entre deux avions dans cette simulation est donc de 82 kt.

Pour que l'avion le plus rapide double le plus lent, il faut qu'il atteigne le point d'entrée de la topologie (où les routes divergent) au moment où l'avion lent a parcouru 5 NM depuis le point d'entrée, et qu'il atteigne le point de sortie (où les routes convergent) au moment où l'avion lent doit encore partcourir 5 NM pour atteindre le point de sortie. Autrement, les deux avions seraient en conflit à l'entrée ou à la sortie de la topologie. Il faut donc que, durant l'intervalle de temps nécessaire à l'avion rapide pour parcourir la topologie de routes, l'avion lent ait parcouru 10 NM de moins.

Calculons la longueur minimale d que doivent avoir les trois routes de la topologie pour que l'avion le plus rapide puisse doubler le plus lent entre l'entrée et la sortie de la topologie de routes. L'avion rapide parcourt d à 502 kt pendant que l'avion lent parcourt $d - 10$ NM à 420 kt :

$$\frac{d}{502} = \frac{d-10}{420} \tag{4.1}$$

$$\iff 420d = 502(d-10) \tag{4.2}$$

$$\iff 420d = 502d - 5020 \tag{4.3}$$

$$\iff d = \frac{5020}{502-420} = 61{,}2\,\text{NM}\,. \tag{4.4}$$

Pour que l'avion le plus rapide de la simulation double le plus lent, il faut donc que la topologie ait une longueur de 61,2 NM.

Considérons maintenant deux avions ayant la même vitesse optimale $v_{i,\text{opt}} = 487$ kt. En appliquant le même calcul, on obtient une distance de :

$$d = \frac{5020}{502-458} = 114\,\text{NM}\,, \tag{4.5}$$

et en considérant deux avions ayant la même vitesse de croisière $v_{i,\text{opt}} = 447$ kt :

$$d = \frac{4600}{460-420} = 115\,\text{NM}\,. \tag{4.6}$$

Pour permettre à des avions de se doubler sur cette topologie de route, il faut donc que celle-ci mesure entre 61,2 NM et 115 NM. Or, la topologie utilisée pour les simulations décrites dans les résultats (section 4.2.5) ne mesure que 54,2 NM. Aucun dépassement d'avion n'a donc été observé durant les expérimentations. En théorie, rien ne s'oppose à ce comportement. Il faudrait néanmoins valider ce résultat théorique grâce à des scénarios de test.

4.2.2.3 Régulation de la date de départ

Une action supplémentaire peut être planifiée par les avions : l'application d'un retard avant le départ, qui peut atteindre 10 minutes. Comme le montre l'algorithme 4, les agents appellent les algorithmes de choix de route et de régulation de vitesse avec des retards croissants. Le but de cette fonction est de maximiser la date du premier conflit qui ne peut pas être résolu par la régulation de vitesse ou le choix d'une route alternative. Les avions peuvent ajuster le retard durant le processus de décision jusqu'à leur départ. La date maximale autorisée est de 10 minutes après la date de départ initialement prévue dans le plan de vol.

Algorithme 4 Sélection du retard du départ.

```
function OptimisationRetard
    retardsPossibles = {0, 1, 2, ... , 14, }∪
        {15, 20, ... , 175} ∪ {180, 190, ... , 600}
    meilleurRetard ⟵ 0
    meilleureDatePremierConflit ⟵ 0
    for retard in retardsPossibles do
        datePremierConflit ⟵ OptimisationRouteEtVitesse(retard)
        if datePremierConflit > meilleureDatePremierConflit then
            meilleurRetard ⟵ retard
        end if
        if datePremierConflit = dateMaximalePremierConflit then
            return meilleurRetard
        end if
    end for
    return meilleurRetard
end function
```

Les valeurs possibles du retard vont de 0 à 10 minutes. Pour réduire le nombre d'itérations de l'algorithme 4, la liste retardsPossibles utilise un pas de temps adaptatif, qui évolue de 1 seconde à 10 secondes. De cette manière, l'algorithme itère 90 fois au lieu de 600 si la boucle devait parcourir toutes les valeurs possibles de retard avec un pas de 1 seconde jusqu'à atteindre 10 minutes. Le processus s'interrompt si une solution sans conflit est trouvée avant la dernière itération.

4.2.3 Action

Dans la phase d'action, les décisions prises par les algorithmes de planification de vitesse, de choix de route et d'optimisation du retard sont utilisées pour actualiser la trajectoire prévue. Celle-ci est envoyée aux autres avions par un message ADS-C.

4.2.4 Conclusion

Tous les agents appliquent le même processus de décision de manière itérative. Les simulations montrent que le système converge vers un état stable dans lequel les conflits sont résolus.

Dans une configuration Miles-in-Trail, la structuration temporelle du trafic émerge grâce aux interactions locales des avions du système multi-agents. Quand les avions se suivent sur la même route, ils volent à une vitesse similaire pour éviter les conflits. Quand leurs routes se croisent, la distance entre les avions de chaque route devient homogène et leur permet de traverser alternativement le point de croisement. Quand les routes sont perpendiculaires, les avions trouvent par eux-mêmes la distance de séparation théorique :

$$S = 2 \cdot 5\sqrt{2} = 14{,}14\,\text{NM} \tag{4.7}$$

qui leur permet d'être séparés de 5 NM à l'intersection, comme l'indique la figure 1.28.

Utiliser un algorithme glouton permet d'obtenir de bons résultats après un temps de calcul court. Cependant, puisque l'algorithme glouton n'est qu'une méthode d'optimisation locale, il peut trouver un optimum local et être incapable de trouver une

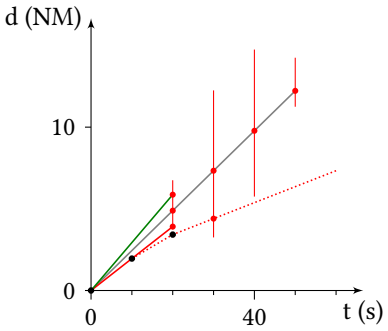


FIGURE 4.9 – Exemple de conflit insoluble par l'algorithme glouton. La trajectoire initiale de l'avion est représentée en gris, et provoque un conflit. Pour le résoudre, il tente de modifier sa vitesse à la date $t = 0$ s. Mais le premier conflit survient à $t = 20$ s, que l'avion accélère (vert) ou décélère (rouge). Il abandonne alors, et conserve sa trajectoire initiale, ce qui provoque un conflit qui dure 40 secondes. Si l'avion utilisait une autre fonction objectif, qui minimise la durée du conflit, il choisirait de décélérer trois fois, ce qui provoquerait un conflit moins grave qui ne durerait que 10 secondes (pointillés rouges).

solution sans conflit, même si elle existe, comme le montre la figure 4.9. Certains conflits ne peuvent donc pas être résolus. Le choix de cette méthode de décision est un compromis entre le temps de calcul et la qualité des résultats.

Les résultats retournés par l'algorithme glouton peuvent cependant être affinés en ajoutant des choix intermédiaires à chaque pas de temps (par exemple des accélérations de ±2 000 NM/h^2 et ±4 000 NM/h^2), mais au prix d'une augmentation du temps de calcul.

Pour valider l'algorithme, différents scénarios de trafic ont été testés. Une description de ces scénarios et des résultats obtenus est présentée dans la section 4.2.5.

4.2.5 Résultats

Plusieurs jeux de scénarios ont été utilisés pour vérifier le fonctionnement de l'algorithme dans différentes configurations de trafic. Le premier jeu reprend les scénarios utilisés pour tester l'algorithme Plectre v1. Il est basé sur un réseau de deux routes dans une configuration Miles-in-Trail. Le deuxième jeu de scénarios est l'implémentation de la topologie de routes parallèles de Yoo et Devasia [YD11] visible dans la figure 4.8, qui permet d'augmenter la capacité de trafic à un croisement de routes. Le troisième jeu de scénarios est basé sur des plans de vol d'avions ayant traversé la France.

Un jeu de scénarios correspond à une topologie de routes. Chaque jeu de scénarios regroupe un ou plusieurs scénarios. Dans les deux premiers jeux de scénarios, chaque scénario correspond à un niveau de charge de trafic. Dans le dernier jeu, chaque scénario correspond à l'activation de différentes fonctionnalités de l'algorithme de régulation (régulation de la vitesse uniquement, du retard appliqué au départ, ou bien les deux activés en même temps).

Enfin, pour chaque scénario, une version correspond à un ratio d'avions ne coopérant pas au processus de résolution des conflits, qui va de 100 % à 0 % d'avions

coopératifs. Chaque scénario est exécuté 10 fois, pour obtenir la valeur moyenne des valeurs mesurées.

4.2.5.1 Miles-in-Trail

Ce premier jeu de scénarios permet de comparer les algorithmes Plectre v1 et Plectre v2. Il reprend donc les scénarios utilisés pour tester Plectre v1 (voir la section 3.3). Dans ce jeu de scénarios, le réseau de routes permet de simuler le croisement de deux flux d'avions structurés en Miles-in-Trail. Comme le montre la figure 3.1, les avions volent le long de deux routes se croisant. Ils reçoivent des messages de leurs voisins contenant leurs trajectoires 4D prévues, et régulent leur vitesse pour résoudre les conflits.

Le réseau de routes utilisé dans cette simulation est composé de deux routes se croisant, définis par 5 waypoints de l'espace aérien Français : la première route est composée des waypoints LMG (Limoges), MEN (Mende) et MRM (Marseille), et la deuxième de TOU (Toulouse), MEN et LYS (Lyon). Les distances entre les waypoints sont : 116 NM pour le segment LMG-MEN, 119 NM pour MEN-MRM, 97 NM pour TOU-MEN, et 105 NM pour MEN-LYS. Les avions sont générés aléatoirement à l'un des deux waypoints les plus à l'ouest (LMG ou MEN). La date de départ des avions le long de chaque route suit une distribution de Poisson, ce qui est considéré comme une approximation valide de flux d'avions [Sal+12, p. III–C].

Trois versions de deux scénarios sont testés. Les avions sont générés sur chaque route selon un processus de Poisson, et volent durant une heure. Chaque scénario correspond à un nombre moyen d'avions générés durant une heure (paramètre λ du processus de Poisson). Dans le premier scénario, les avions sont générés en moyenne toutes les 140 s, ce qui correspond à 25,7 avions par heure. Dans le deuxième scénario, les avions sont générés en moyenne toutes les 110 s, ce qui correspond à 32,7 avions par heure. Cette dernière valeur est la capacité maximale théorique d'un réseau de deux routes perpendiculaires, en considérant une vitesse de croisière moyenne de 467 kt et une distance minimale de séparation entre deux avions d'un même flux de 14,14 NM, cette distance permettant aux avions des deux flux d'être séparés par 5 NM au croisement. La valeur 14,14 NM est donnée par [YD13c, Lemma 1], mais Plectre v2 est capable d'en donner une bonne approximation sans utiliser explicitement ce calcul.

Pour chaque scénario, trois versions sont testés : une première où tous les avions coopèrent à la résolution de conflits (activé), une deuxième où les avions ne prennent aucune décision (désactivé), et une troisième où 10 % des avions ne coopèrent pas (désactivé pour 10 % des avions), leur processus de décision étant désactivé, ce qui permet de valider la résilience de l'algorithme. La vitesse optimale $v_{i,\mathrm{opt}}$ des avions est choisie aléatoirement dans l'intervalle [447 kt, 487 kt], ce qui est représentatif des performances des avions capables de voler à 36 000 ft et plus. Chaque scénario est exécuté 10 fois. Les valeurs moyennes des métriques utilisées pour évaluer l'algorithme (nombre de conflits, distance de séparation minimale mesurée durant un conflit, nombre d'accélérations, etc.) sont reportées dans le tableau 4.1.

Dans le premier scénario, où $\lambda = 1/140\,\mathrm{s}^{-1}$, 86 % des conflits sont résolus quand le processus de décision est activé, passant de 16,8 conflits à 2,3 en moyenne. Ainsi, tous les conflits ne sont pas résolus par le processus de décision des avions. Ce processus est un algorithme glouton qui cherche un optimum local maximisant le temps séparant l'avion de son prochain conflit. Cet algorithme est plus rapide qu'on algorithme

TABLE 4.1 – Performances de l'algorithme Plectre v2 pour la topologie Miles-in-Trail (valeur moyenne après 10 exécutions de la simulation).

λ (s^{-1})	Processus de décision	Nombre d'avions	Conflits	Conflits < 4 NM
$\frac{1}{140}$	Désactivé	51,2	16,8	13,5
	Activé	51,1	2,3 (−86 %)	1,5 (−89 %)
	Désactivé pour 10 % des avions	51,3	2,7 (−84 %)	2 (−85 %)
$\frac{1}{110}$	Désactivé	65	27	21,7
	Activé	65	6,9 (−74 %)	4,2 (−81 %)
	Désactivé pour 10 % des avions	65,8	10,7 (−60 %)	7,4 (−66 %)

λ (s^{-1})	Processus de décision	Distance minimale (NM)	Distance moyenne (NM)	Accélérations	Distance à $v_{i,\text{opt}}$ (%) [a]
$\frac{1}{140}$	Désactivé	0,03	2,31	0,0	0
	Activé	0,04	2,28	11,5	1,4
	Désactivé pour 10 % des avions	0,01	2,08	11,2	1,4
$\frac{1}{110}$	Désactivé	0,0	2,33	0,0	0
	Activé	0,02	2,86	27,8	2,3
	Désactivé pour 10 % des avions	0,0	2,5	23,2	2,3

a. À chaque instant, la distance à la vitesse optimale est calculée ainsi pour chaque avion : $d = 100 \times \frac{|v - v_{i,\text{opt}}|}{v_{i,\text{opt}}}$. La moyenne de toutes les valeurs collectées est reportée dans cette colonne.

d'optimisation globale, mais n'est pas adapté pour trouver un optimum global pour lequel tous les conflits seraient résolus.

Dans le deuxième scénario, dans lequel $\lambda = 1/110\,s^{-1}$, seuls 74 % des conflits sont résolus, ce qui représente 6,9 conflits résiduels sur les 27 initiaux. Cela indique que l'algorithme devient moins performant quand le niveau de trafic approche de la capacité maximale du réseau de routes. Cette perte d'efficacité vient du processus de décision : la valeur λ permet de calculer le temps moyen séparant la génération de deux avions. Les avions sont générés à des distances plus ou moins grandes les uns des autres sur chaque route. Or, quand l'intervalle moyen entre deux générations est égal à la capacité maximale du réseau de routes, il peut arriver que deux avions soient générés trop proches l'un de l'autre, provoquant un confit qui ne peut pas être résolu par l'algorithme.

En comparant les résultats des versions de scénarios de Plectre v2 où tous les avions coopèrent avec les résultats des scénarios équivalents de Plectre v1, dont les résultats sont visibles dans le tableau 3.1 et commentés dans la section 3.3, le nombre de conflits est nettement inférieur en utilisant Plectre v2. Dans le scénario

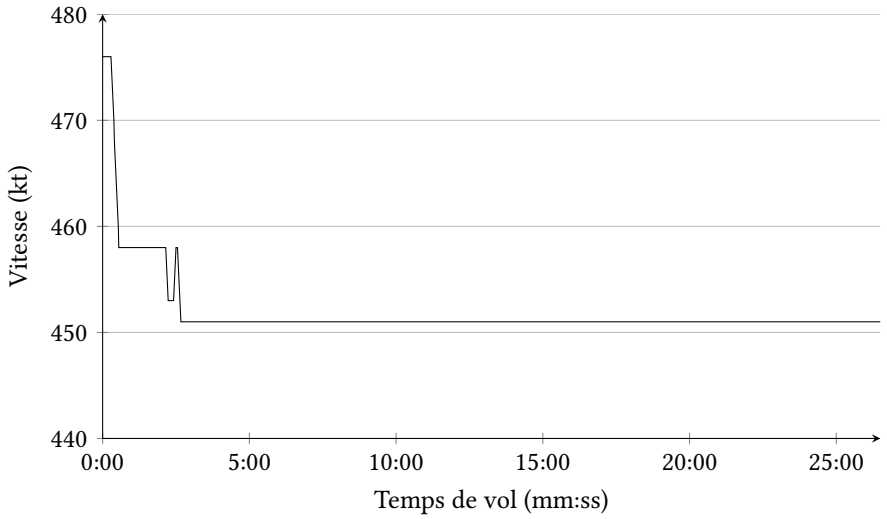


FIGURE 4.10 – Profil de vitesse d'un avion dans le jeu de scénarios Miles-in-Trail de Plectre v2. Cet avion ajuste sa vitesse à chaque fois qu'un voisin apparait dans la simulation, jusqu'à trouver une vitesse permettant d'éviter les conflits.

où $\lambda = 1/140\,\mathrm{s}^{-1}$, le nombre de conflits diminue de 78 %, en passant de 10,3 (v1) à 2,3 (v2). Pour $\lambda = 1/110\,\mathrm{s}^{-1}$, ce nombre diminue de 36 %, en passant de 18,7 (v1) à 6,9 (v2) conflits.

Le temps total durant lequel les avions accélèrent diminue également de 97 %, en passant de 410 s (v1) à 11,5 s (v2) pour $\lambda = 1/140\,\mathrm{s}^{-1}$, et de 94 % en passant de 503 s (v1) à 27,8 s (v2) pour $\lambda = 1/110\,\mathrm{s}^{-1}$. Cela indique que le processus de décision est plus efficace en terme de nombre de changements de vitesse pour Plectre v2, où les avions peuvent anticiper les accélérations de leurs voisins.

Cependant, dans Plectre v2, quand un conflit ne peut pas être évité, les avions ne cherchent pas à minimiser son impact, par exemple en maximisant la distance de séparation minimale, comme le montre la figure 4.9. Cela est dû à l'objectif visé par le processus de décision, qui maximise seulement le temps séparant l'avion de sont premier conflit. Cela provoque des pertes de séparation plus importantes que pour Plectre v1, avec des distances moyennes mesurées de 2,28 NM pour $\lambda = 1/140\,\mathrm{s}^{-1}$ et de 2,86 NM pour $\lambda = 1/110\,\mathrm{s}^{-1}$ (voir le tableau 4.1).

Cela génère également un plus grand nombre de conflits durant lesquels les avions sont séparés par moins de 4 NM (alors que la norme de séparation est de 5 NM). Cette métrique est pertinente pour Plectre v1, dont la fonction objectif maximise la distance de séparation, et où beaucoup de conflits sont générés parce que la distance de séparation de deux avions est proche de 5 NM, et oscille autour de cette valeur. Pour Plectre v2, les distances minimales de séparation sont uniformément réparties entre 0 NM et 5 NM. Ce résultat pourrait néanmoins être amélioré en incluant la durée du conflit ou la distance minimale de séparation entre avions dans la fonction objectif.

Un autre effet de la relative simplicité du processus de décision est que les avions résolvent les conflits, mais n'essaient pas par la suite de se rapprocher de leur vitesse optimale. La figure 4.10 montre le profil de vitesse d'un avion dans Plectre v2. L'avion commence à voler à sa vitesse optimale (476 kt), puis change de vitesse plusieurs

fois pour résoudre les conflits, avant de se stabiliser à 451 kt jusqu'à la fin, ce qui est éloigné de sa valeur optimale. Confrontés à la même situation, les avions de Plectre v1 essaient de revenir à leur vitesse optimale, comme le montre la figure 3.8.

La conséquence de ce comportement est visible dans le tableau 4.1, dont la dernière colonne indique la différence moyenne, en pourcents, entre la vitesse de l'avion à chaque seconde et sa vitesse de croisière $v_{i,\mathrm{opt}}$. Les valeurs de cette colonne sont plus élevées que leurs équivalents pour Plectre v1 (tableau 3.1), passant de 1,2 % à 1,4 % pour $\lambda = 1/140\,\mathrm{s}^{-1}$, et de 1,4 % à 2,3 % pour $\lambda = 1/110\,\mathrm{s}^{-1}$. Pour améliorer ce comportement, l'optimisation de la vitesse pourrait également être incluse dans la fonction objectif, mais le problème résolu par les avions deviendrait un problème multi-objectifs, plus dur à modéliser et à résoudre.

Pour valider la résilience de l'algorithme face aux évènements imprévus, une troisième version a été ajouté à chaque scénario, dans laquelle le processus de décision est désactivé pour 10 % des avions, les rendant non-coopératifs. Les avions non-coopératifs continuent néanmoins à envoyer leur trajectoire prévue, mais ne la modifient pas. Dans le scénario où $\lambda = 1/140\,\mathrm{s}^{-1}$, les résultats sont similaires quand tous les avions coopèrent et quand 10 % d'entre eux ne coopèrent pas, avec une légère augmentation du nombre de conflits non résolus de 2,3 à 2,7. Tant que les avions coopératifs connaissent la trajectoire estimée de leurs voisins non-coopératifs, ils sont capables d'éviter les conflits, ces voisins étant pris en compte comme contraintes de leur processus de décision. Une partie des conflits résiduels est même due à des paires d'avions non-coopératifs en conflit qui ne prenant aucune décision pour les résoudre.

Pour le deuxième scénario, dans lequel l'intervalle moyen entre la génération de deux avions est 110 s, le nombre moyen de conflits non résolus augmente de 6,9 à 10,7. Autrement dit, le nombre de conflits résolus diminue de 74 % à 60 %. Dans ce second scénario, les avions sont générés toutes les 110 s, ce qui est la capacité maximale théorique de ce réseau de routes. Comme évoqué précédemment, le processus de génération des avions peut provoquer des situations où les avions sont en conflit insoluble dès leur génération. Les avions non-coopératifs accentuent le problème. Certains conflits requièrent que tous les avions impliqués coopèrent pour être résolus, et la solution ne peut pas être trouvée si seulement un des avions manœuvre. De plus, certains conflits qui pourraient être résolus ne le sont pas parce qu'aucun des avions impliqués ne coopère.

4.2.5.2 Miles-in-Trail avec routes parallèles

La topologie de routes proposée par Yoo et Devasia [YD14] permet d'augmenter la capacité de trafic d'un croisement de flux organisés en Miles-in-Trail pour atteindre la capacité théorique d'un avion tous les 5 NM sur chaque flux. Comme le montre la figure 4.8, cette topologie sépare d'abord les avions sur trois routes parallèles séparées latéralement par 5 NM. Les avions passent l'intersection, puis les trois sous-flux sont fusionnés en un seul.

Comme indiqué dans la section 4.2.2.2, dans Plectre v2, les avions ont la capacité de choisir parmi plusieurs routes alternatives. Ce comportement rend possible l'implémentation de cette topologie de routes.

Pour valider cette méthode, trois versions de deux nouveaux scénarios ont été testées. Les avions sont générés selon un processus de Poisson sur chaque route. La simulation dure une heure. Dans le premier scénario, l'intervalle de temps moyen séparant la génération de deux avions est fixée à 110 s, comme dans le deuxième ensemble de scénarios de la section 4.2.5.1. Dans le deuxième scénario, l'intervalle

TABLE 4.2 – Performances de l'algorithme Plectre v2 dans le scénario implémentant la topologie de croisement à trois voies parallèles (valeurs moyennes après 10 simulations).

λ (s^{-1})	Processus de décision	Nombre d'avions	Conflits
$\frac{1}{110}$	Désactivé	65,2	26,5
	Activé	65,1	1,5 (−94 %)
	Désactivé pour 10 % des avions	64,6	2,9 (−89 %)
$\frac{1}{80}$	Désactivé	89	57,7
	Activé	89,3	11,4 (−80 %)
	Désactivé pour 10 % des avions	89,5	16,4 (−72 %)

moyen entre deux générations est fixé à 80 s, pour vérifier que cette augmentation du trafic est correctement gérée par cette topologie de routes. Dans ce scénario, les avions sont générés en moyenne tous les 10 NM sur chaque route.

Dans la première version de chaque scénario, le processus de décision des avions est désactivé. Dans la deuxième version, il est activé. Dans la troisième, le processus de décision est désactivé pour 10 % des avions, pour vérifier la résilience de l'algorithme face à des évènements imprévus. Les résultats sont visibles dans le tableau 4.2.

Dans le scénario où $\lambda = 1/110\,s^{-1}$, quand le processus de décision est activé, 94 % des conflits sont résolus en moyenne (1,5 conflits résiduels sur les 26 initiaux). Quand 10 % des avions sont non-coopératifs, le ratio de conflits résolus est réduit à 89 % (2,9 conflits résiduels). Ces résultats sont similaires à ceux obtenus avec la topologie utilisée dans le premier jeu de scénarios (figure 3.1), dans le scénario où un avion est généré toutes les 140 s en moyenne, où respectivement 1,5 et 2 confits ne sont pas résolus (tableau 4.1). Le but de Plectre v2 étant la réduction du nombre de conflits pour diminuer la charge de travail des contrôleurs, augmenter le trafic de 27 % n'augmente pas la part de la charge de travail des contrôleurs liée au nombre de conflits.

Dans le deuxième scénario, où $\lambda = 1/80\,s^{-1}$, l'algorithme résout 80 % des conflits quand tous les avions coopèrent, et 72 % des conflits quand 10 % des avions ne coopèrent pas. Ces résultats sont similaires à ceux obtenus avec la topologie utilisée dans le premier jeu de scénarios (figure 3.1), dans le scénario où un avion est généré toutes les 110 s en moyenne, où respectivement 81 % et 66 % des conflits sont résolus (tableau 4.1). Ces résultats confirment que la capacité d'un réseau de routes Miles-in-Trail peut être augmentée en utilisant un réseau composé de trois routes parallèles à l'intersection. Cette topologie de routes permet à l'algorithme Plectre v2 de gérer une augmentation du trafic de 38 % sans dégradation des performances en terme de ratio de conflits résolus.

4.2.5.3 Trafic réaliste basé sur des plans de vol

Le dernier jeu de scénarios testés est basé sur des plans de vol d'avions sur la France durant 10 heures (entre 4 heures et 14 heures). Ces plans de vol sont visibles sur la figure 4.11. Il a d'abord fallu les filtrer pour obtenir le scénario à étudier :

1. Les waypoints correspondant à la phase de montée et de descente de l'avion sont supprimés, Plectre v2 étant adapté à la phase de croisière ;
2. Seuls les avions volant au FL370 (37 000 ft) sont étudiés, ce niveau de vol étant l'un des plus chargés, et donc un des plus complexes à gérer par Plectre v2. Cela

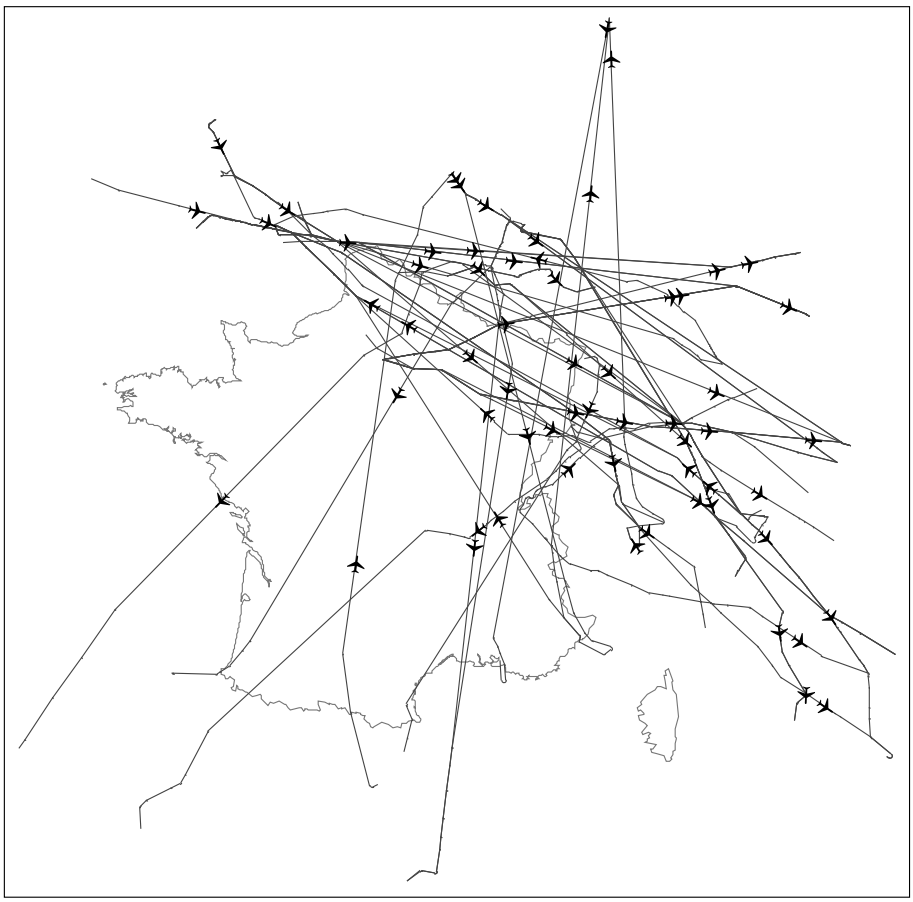


FIGURE 4.11 – Scénario de trafic basé sur de vrais plans de vol, décrit dans la section 4.2.5.3. Cette image montre l'état du trafic à midi.

représente 283 avions dans ce scénario.

Trois scénarios sont testés. Dans le premier scénario, les avions régulent leur vitesse mais pas leur heure de départ. Dans le second scénario, les avions peuvent retarder leur heure de départ jusqu'à 10 minutes, mais pas réguler leur vitesse. Enfin, dans le troisième scénario, les avions régulent à la fois leur heure de départ et leur vitesse.

Pour tester la résilience de l'algorithme face à des évènements imprévus, dans chaque scénario, le processus de décision est désactivé pour un sous-ensemble d'avions. Les ratios d'avions non-coopératifs expérimentés sont 10 %, 20 %, 50 % et 80 %.

Durant chaque simulation, la distance séparant chaque paire d'avions est mesurée chaque seconde. Chaque fois que deux avions sont séparés par moins de 5 NM, un conflit est enregistré (un seul enregistrement pour toute la durée du conflit).

Chaque scénario est exécuté 10 fois. Le nombre de conflits est enregistré (figures 4.12 et 4.13), ainsi que le nombre d'accélérations (figure 4.14) et le retard (figure 4.16) appliqué par chaque avion coopératif.

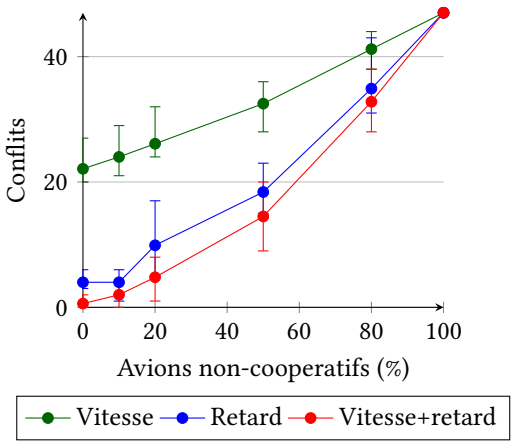


FIGURE 4.12 – Nombre de conflits en fonction du pourcentage d'avions non-coopératifs. Les résultats sont donnés pour chaque scénario : avec régulation de vitesse uniquement, avec régulation de l'heure de départ uniquement, et avec les deux régulations à la fois. Chaque scénario est exécuté 10 fois ; le nombre minimal, moyen et maximum de conflits de chaque scénario est indiqué.

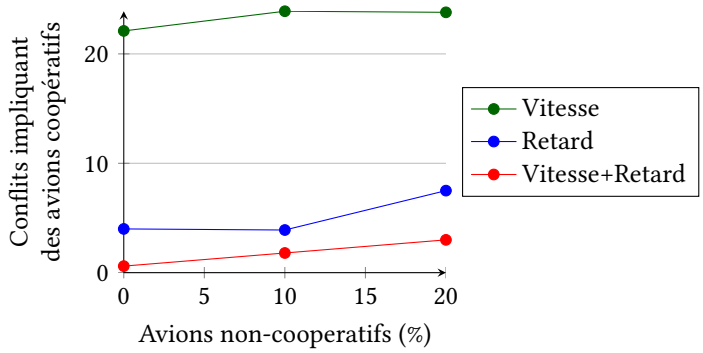


FIGURE 4.13 – Nombre de conflits impliquant des avions coopératifs en fonction du pourcentage d'avions non-coopératifs. Le nombre de conflits est plus faible que dans la figure 4.12 ; les conflits entre avions non-coopératifs ne sont pas comptabilisés dans ce graphique parce qu'ils ne sont pas significatifs dans l'étude des performances de l'algorithme. Les conflits comptés dans cette figure impliquent au moins un avion coopératif. Les résultats sont donnés pour chaque scénario : avec régulation de vitesse uniquement, avec régulation de l'heure de départ uniquement, et avec les deux régulations à la fois.

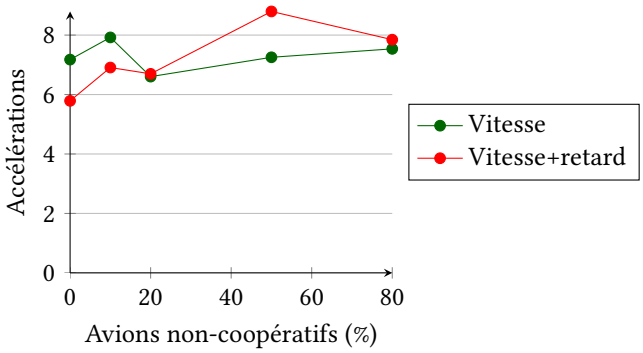


FIGURE 4.14 – Nombre moyen d'accélérations décidées par les avions coopératifs en fonction du pourcentage d'avions non-coopératifs. Les résultats sont donnés pour le scénario où la régulation de l'heure de départ est activée et celui où elle est désactivée.

Les résultats varient entre deux exécutions du même scénario. Dans la figure 4.12, le nombre minimal et maximal de conflits mesurés sont indiqués en même temps que le nombre moyen. Ces variations ont deux origines. Premièrement, les avions rafraichissent leurs décisions toutes les 8 secondes, mais le moment de la première décision est choisie aléatoirement parmi les premières secondes de la durée de vie des agents. Cela implique que d'une exécution à l'autre, les avions vont prendre leurs décisions dans un ordre différent, et les décisions du premier avion influe sur les décisions des suivants. Deuxièmement, dans les version des scénarios impliquant des avions non-coopératifs, ceux-ci sont sélectionnés aléatoirement dans l'ensemble des avions. Un avion peut coopérer au cours d'une exécution et être non-coopératif au cours d'une autre. Et l'influence d'un avion sur le trafic environnant peut varier, par exemple s'il traverse des zones congestionnées ou non.

Une première simulation a été exécutée pour fournir un point de comparaison, dans laquelle le processus de décision de tous les avions est désactivé. Ce scénario génère 47 conflits. La section 4.2.5.3.1 aborde la capacité du système multi-agents à résoudre les conflits quand tous les avions coopèrent. La section 4.2.5.3.2 aborde la résilience de l'algorithme face aux évènements imprévus.

4.2.5.3.1 100 % d'avions coopératifs Dans cette section est abordée la capacité de l'algorithme à résoudre les conflits quand tous les avions coopèrent. Dans le premier scénario, les avions régulent uniquement la vitesse. Comme le montre la courbe verte de la figure 4.12, quand tous les avions coopèrent, 24,9 conflits en moyenne sont résolus sur les 47 initiaux (−53 %). Résoudre les conflits nécessite que les avions accélèrent 7,2 fois en moyenne, ce qui représente une durée d'accélération de 35,9 secondes (figure 4.14).

22,1 conflits ne peuvent pas être résolus par l'algorithme de régulation de vitesse seul. Comme le montre la figure 4.15, les positions de départ de certains avions provoquent des conflits insolubles. La régulation de vitesse ne peut pas non plus résoudre les conflits en face-à-face.

Dans le deuxième scénario, les avions ne régulent pas leur vitesse, mais sont capables de retarder leur départ jusqu'à 10 minutes. Cette stratégie permet au système de résoudre les conflits qui ne peuvent pas être résolus par la régulation de vitesse, comme celui décrit dans la figure 4.15. Le nombre de conflits est alors beaucoup

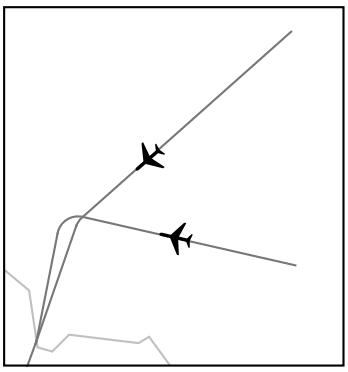


FIGURE 4.15 – Dans les scénarios basés sur des vrais plans de vol, il peut arriver que les avions partent de positions trop proches. Ici, les avions n'ont pas le temps de réguler leur vitesse avant le conflit, leur trajectoire étant trop courte entre leur point de départ et le point de croisement. La régulation de vitesse ne peut pas résoudre ces conflits.

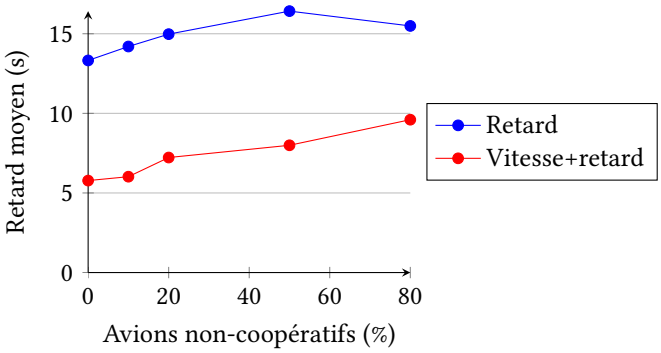


FIGURE 4.16 – Retard moyen décidé par les avions coopératifs en fonction du pourcentage d'avions non-coopératifs. Les résultats sont donnés pour le scénario où la régulation de vitesse est activée et celui où elle est désactivée.

plus faible que dans l'ensemble de scénarios précédent, avec seulement 4 conflits non résolus en moyenne (−91,5 %). Les avions retardent leur départ de 13,3 secondes en moyenne, comme le montre la figure 4.16. Les figures 4.17 et 4.18 montrent que 42 avions retardent leur décollage, et que le retard moyen appliqué par ces avions est de 89 secondes.

Dans le dernier scénario, les avions peuvent réguler leur vitesse et leur heure de départ. Ce scénario est le plus favorable à la résolution de conflits, puisque le degré de liberté du processus de décision est le plus grand. Les avions peuvent résoudre les conflits comme celui de la situation présentée dans la figure 4.15. Comme le montre la figure 4.12 (courbe rouge), quand tous les avions coopèrent, il ne reste que 0 à 2 conflits, avec une moyenne de 0,6 conflits non résolus.

Les avions changent de vitesse 5,8 fois en moyenne, pour une durée d'accélération cumulée de 28,9 secondes. En comparant ce scénario avec celui où les avions régulent uniquement leur vitesse, permettre aux avions de retarder leur départ les aide à réduire

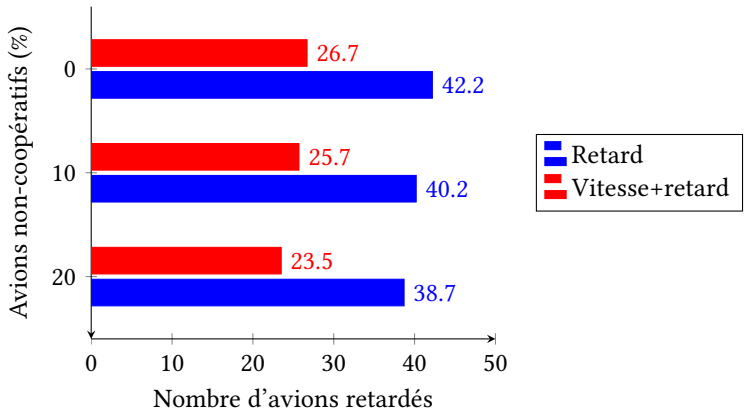


FIGURE 4.17 – Nombre moyen d'avions appliquant un retard en fonction du pourcentage d'avions non-coopératifs. Les résultats sont donnés pour le scénario où la régulation de vitesse est activée et celui où elle est désactivée.

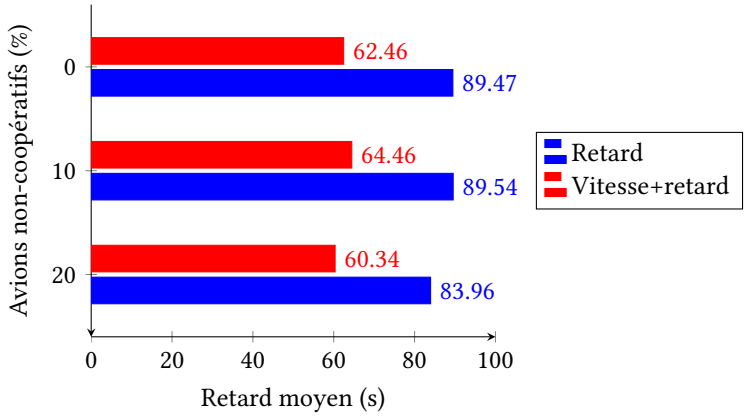


FIGURE 4.18 – Moyenne des retards supérieurs à 0 secondes en fonction du pourcentage d'avions non-coopératifs. Les résultats sont donnés pour le scénario où la régulation de vitesse est activée et celui où elle est désactivée.

légèrement le nombre d'accélérations nécessaires à la résolution des conflits. Le retard moyen appliqué au départ est également réduit de 56,6 % comparé au scénario où les avions régulent uniquement leur heure de départ. Quand tous les avions coopèrent, le retard moyen passe de 13,3 secondes à 5,8 secondes, puisque le nombre d'avions qui retardent leur départ diminue de 42 à 27, et que le retard moyen qu'ils appliquent diminue de 89 secondes à 62 secondes.

La figure 4.19 montre la distribution du nombre d'avions en fonction du retard qu'ils appliquent. Dans le scénario où les avions ne régulent que leur heure de départ, 51 % des avions appliquent un retard de moins d'une minute, 77 % appliquent un retard de moins de 2 minutes, et 94 % de moins de 4 minutes.

Dans le scénario où les avions régulent leur heure de départ et leur vitesse, 71 % des avions appliquent un retard de moins d'une minute, 88 % appliquent un retard de moins de 2 minutes, et 95 % de moins de 4 minutes.

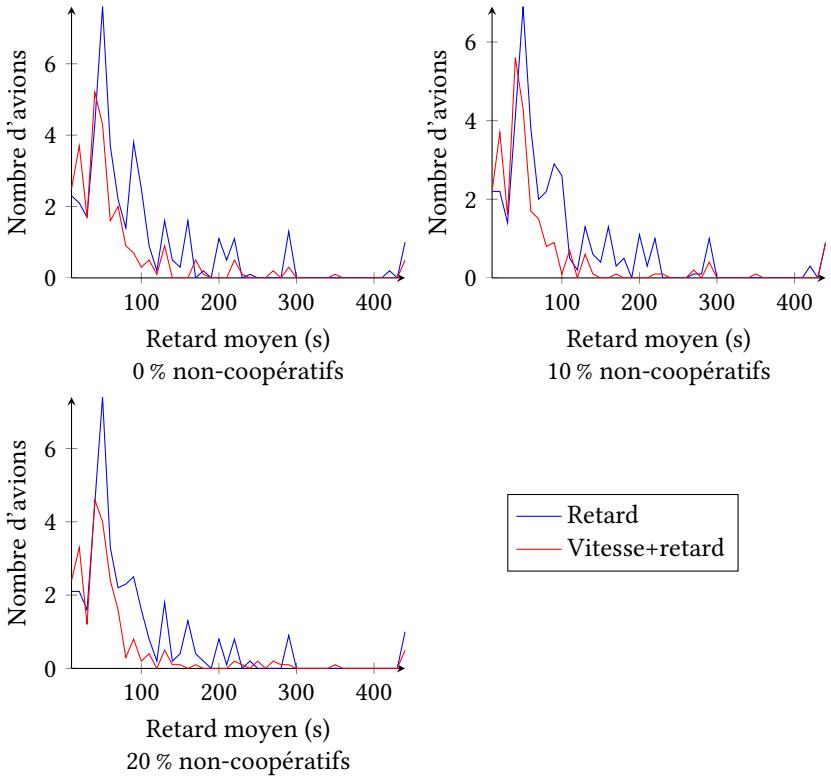


FIGURE 4.19 – Distribution des avions en fonction du retard appliqué. Le retard, en secondes, est arrondi à la dizaine supérieure. Les graphes montrent le résultat pour les scénarios où 0 %, 10 % et 20 % des avions ne coopèrent pas au processus de décision.

Le retard moyen appliqué est situé entre 60 et 90 secondes suivant le scénario étudié. Ces valeurs sont comparables au temps minimum séparant deux décollages dans les aéroports où le trafic est le plus dense, qui est de 90 secondes en heure de pointe.

4.2.5.3.2 Résilience Pour valider la résilience de l'algorithme, pour chacun des trois scénarios (régulation de vitesse uniquement, régulation de l'heure de départ uniquement, régulation de la vitesse et de l'heure de départ), quatre versions additionnels ont été testés, correspondant à différents pourcentages d'avions non-coopératifs. Dans ces versions, la capacité à prendre des décisions est inhibée pour 10, 20, 50 et 80 % des agents. Les avions coopératifs considèrent les avions non-coopératifs comme des contraintes pour leur processus de décision, et résolvent les conflits si le retard et les vitesses autorisés le leur permet. Dans ces simulations, la résilience est analysée à trois niveaux. Au niveau de l'ensemble des agents, la résilience est liée à la charge de travail des contrôleurs aériens. Au niveau de l'ensemble des agents coopératifs, la résilience est liée aux performances générales de l'algorithme. Au niveau d'un seul agent, la résilience est liée à la qualité des décisions de l'avion.

Résilience au niveau de l'ensemble des agents La charge de travail des contrôleurs aériens est partiellement liée à la complexité d'une situation de trafic. Une part de cette complexité provient du nombre de conflits que les contrôleurs doivent résoudre. La figure 4.12 montre le nombre total de conflits qui ne peuvent pas être résolus par l'algorithme en fonction du nombre d'avions non-coopératifs. Ces conflits résiduels doivent être résolus par les contrôleurs, et influent sur la complexité du trafic. Ainsi, les valeurs indiquées par la figure 4.12 donnent une tendance de la complexité perçue en fonction du ratio d'avions non-coopératifs.

Quand 20 % des avions ne coopèrent pas, le nombre de conflits augmente d'environs 10 % par rapport à la version du scénario où tous les avions coopèrent. Dans le premier scénario (régulation de vitesse uniquement), on observe une augmentation de 8,5 % du nombre de conflits, qui passe de 22,1 à 26,1 conflits sur les 47 conflits initiaux. Dans le deuxième scénario (régulation de l'heure de départ uniquement), le nombre de conflits augmente de 12,6 %, en passant de 4 à 9,9 conflits par rapport au nombre de conflits initiaux. Dans le dernier scénario, le nombre de conflits augmente de 8,9 %, en passante de 0,6 à 4,8 conflits, par rapport aux 47 conflits initiaux.

Désactiver la capacité à prendre des décisions pour 20 % des avions augmente le nombre de conflits de seulement 10 % en moyenne par rapport aux scénarios où tous les avions coopèrent. Cette valeur indique que l'augmentation de la charge de travail des contrôleurs causée par le nombre de conflits est limitée si une partie des avions ne coopèrent pas.

Résilience au niveau des agents coopératifs Le nombre de conflits augmente avec le nombre d'avions non-coopératifs. Ces conflits additionnels sont catégorisés en deux groupes : les conflits entre deux avions non-coopératifs, et les conflits impliquant au moins un avion coopératif. Le premier groupe doit être pris en compte pour mesurer les performances générales du système, mais fournit peu d'informations concernant les performances du processus de décision. Ces conflits ne peuvent pas être résolus, et leur nombre tend vers 47 quand le pourcentage d'avions coopératifs tend vers 0.

Le nombre de conflits impliquant au moins un avion coopératif fournit plus d'informations sur les performances de l'algorithme, puisque ceux-ci peuvent être résolus par le processus de décision. Comme le montre la figure 4.13, le nombre de conflits appartenant à cette catégorie augmente légèrement avec le nombre d'avions non-coopératifs, de 5,4 % en moyenne quand 20 % des avions ne coopèrent pas. Dans le scénario où seule la vitesse est régulée, le nombre de conflits augmente de 3,6 %, de 22,1 à 23,8 conflits sur les 47 initiaux. Dans le scénario où seule l'heure de départ est régulée, le nombre de conflits augmente de 7,4 %, de 4 à 7,5 conflits. Quand la vitesse et le retard sont régulés en même temps, le nombre de conflits augmente de 5,1 %, de 0,6 à 3 conflits.

Autrement dit, la moitié des conflits causés par la non-coopération de 20 % des avions implique uniquement des avions non-coopératifs. Quand un cinquième des avions ne participe pas à la résolution des conflits, le nombre de conflits demeure stable, avec une augmentation de 5,4 % des conflits par rapport à la version du scénario où tous les avions coopèrent. Cette valeur indique que l'algorithme est résilient aux perturbations, comme un avion étant ou devenant non-coopératif.

Résilience au niveau d'un seul agent Au niveau des agents individuels, le nombre de changements de vitesse et le retard moyen du départ choisis par chaque avion sont également mesurés pour chaque scénario. Les valeurs moyennes en fonc-

tion du pourcentage d'avions non-coopératifs sont visibles dans les figures 4.14 et 4.16. Comme le montre la figure 4.14, le nombre moyen d'accélérations est stable quand le nombre d'avions non-coopératifs augmente, et se situe entre 5,8 et 8,8 changements de vitesse pour tous les scénarios.

Le retard moyen augmente avec le nombre d'avions non-coopératifs. La figure 4.16 montre que l'augmentation maximale du retard moyen est de 3,8 secondes quand le nombre d'avions non-coopératifs augmente de 0 à 80 %, augmentant de 5,7 s à 9,6 s quand les avions régulent leur retard et leur vitesse.

Les figures 4.17, 4.18 et 4.19 montrent une légère diminution du nombre d'avions qui retardent leur heure de départ, liée à la diminution du nombre d'avions coopératifs, et une stabilité du retard appliqué moyen (variation inférieure à 6 secondes), ainsi que de la répartition des retards.

Le nombre moyen de changements de vitesse et le retard appliqué au départ sont stables quand le nombre d'avions non-coopératifs augmente. Cela indique que le processus de décision des avions est résilient à un nombre important d'avions non-coopératifs.

En conclusion, ces résultats montrent que le système multi-agents Plectre v2 est résilient à trois échelles différentes. Au niveau du système, l'augmentation limitée du nombre de conflits implique une augmentation limitée de la charge de travail des contrôleurs aériens si quelques avions ne participent pas à la résolution des conflits. En ignorant les conflits impliquant uniquement des avions non-coopératifs, l'augmentation du nombre de conflits est limitée, indiquant la résilience de l'algorithme. Enfin, le nombre d'accélérations et le retard moyen est stable, indiquant la résilience au niveau des agents individuels.

4.2.6 Conclusion

L'algorithme Plectre v2 possède plusieurs avantages liés à l'utilisation des systèmes multi-agents. Il est capable de résoudre jusqu'à 100 % des conflits dans les scénarios de test, il est résilient aux perturbations telles que les avions non-coopératifs, et pourrait être implémenté comme système embarqué dans les avions, supprimant le besoin d'équipements au sol.

La section suivante propose l'implémentation d'une méthode d'optimisation globale pour résoudre le problème présenté dans ce chapitre. Cette implémentation permet de mesurer la différence en terme de temps de calcul et d'optimalité des résultats entre le système multi-agents et une méthode d'optimisation globale, souvent utilisées pour résoudre des problèmes similaires.

4.3 Implémentation d'un système centralisé : recuit simulé

Les systèmes multi-agents permettent de faire émerger une solution globale à l'échelle du système à partir d'interactions locales au niveau des agents. Pour déterminer les différences entre le système multi-agents et un algorithme d'optimisation globale, notamment en terme de qualité du résultat, le modèle de Plectre v2 a été reformulé pour être résolu par un algorithme de recuit simulé.

4.3.1 Modèle

4.3.1.1 Fonction objectif

Le modèle développé pour l'optimisation globale est le plus proche possible du modèle utilisé pour le système multi-agents. L'algorithme glouton qui planifie les changements de vitesse des avions du SMA cherche à maximiser l'instant du premier conflit, et s'arrête dès qu'une solution sans conflit est trouvée, ce qui minimise le nombre de changements de vitesse. Le retard du départ est planifié par un processus qui englobe l'algorithme glouton. Celui-ci fixe des valeurs de retard croissantes, et lance l'algorithme glouton en utilisant chacune de ces valeurs, pour chercher le retard qui maximise le temps avant le premier conflit non résolu.

En transposant ce processus en modèle utilisable par un processus d'optimisation globale, on obtient une fonction objectif qui doit, par ordre de priorité :

1. Maximiser la date du premier conflit $t_C \in \mathbb{N}$, en secondes ;
2. Minimiser le nombre total de changements de vitesse $\sum_{i=1}^{N} v_i$, $v_i \in \mathbb{N}$, avec N le nombre d'avions ;
3. Minimiser le retard au départ cumulé de tous les avions $\sum_{i=1}^{N} d_i$, $d_i \in \mathbb{N}$, en secondes.

La fonction objectif est la somme pondérée de ces trois variables. Le coefficient appliqué à chaque variable permet de les hiérarchiser :

— La date du premier conflit t_C est la plus importante, son coefficient est 1 ;

— Le nombre de changements de vitesses $\sum_{i=1}^{N} v_i$ est divisé par le nombre maximal de changements de vitesse $\sum_{i=1}^{N} V_i$, c'est-à-dire la somme des durées des trajectoires divisées par 5, les changements de vitesse étant planifiés avec un pas de 5 secondes, comme dans le SMA. Cette partie de la fonction objectif est donc un réel compris entre 0 et 1 ;

— Le délai total $\sum_{i=1}^{N} d_i$ est divisé par le délai total maximal. Comme $d_i \in [0, D]$, le délai total est inférieur à $D \times N$, N étant le nombre d'avions. Le délai total est donc divisé par cette borne maximale. Le résultat est à son tour divisé par $\sum_{i=1}^{N} V_i$, de manière à ce que cette valeur soit une fraction de la valeur de l'objectif précédent.

La fonction objectif est donc :

$$\max_{y} y = t_C - \frac{\sum_{i=1}^{N} v_i}{\sum_{i=1}^{N} V_i} - \frac{\sum_{i=1}^{N} d_i}{D \times N} \cdot \frac{1}{\sum_{i=1}^{N} V_i} \tag{4.8}$$

Dans ce modèle, un point de l'espace d'états regroupe l'ensemble des changements de vitesse de chaque trajectoire, planifiées avec un pas de temps de 5 secondes, ainsi que le retard à appliquer à chacune d'entre elles.

4.3.1.2 Opérateur de voisinage

Le recuit simulé maintient une liste des avions à réguler. Cette liste est construite tout au long du processus de recuit, par ajout successif d'avions. À chaque fois qu'au cours de l'évaluation de la fonction objectif, un nouveau conflit est détecté, les avions impliqués sont rajoutés à la liste. Aucun avion n'est enlevé de cette liste durant le processus de recuit simulé.

Ainsi, si l'évaluation de la fonction objectif ne détecte pas de conflit dans la solution initiale, le processus est arrêté, puisqu'il n'y a rien à optimiser : soit il s'agit des trajectoires d'origine, sans changement de vitesse ni délai (ce qui est la solution optimale), soit les trajectoires ont déjà été optimisées durant une exécution précédente du recuit simulé, et elles sont considérées comme déjà optimales.

Si un conflit est détecté, les deux avions impliqués sont rajoutés à la liste des avions à réguler, et leurs trajectoires sera optimisées. Si en tentant de résoudre ce premier conflit, un autre conflit impliquant un troisième avion est détecté, celui-ci est rajouté à la liste, et les trois trajectoires seront optimisées par le recuit simulé.

Un point de l'espace d'états contient le retard ainsi que la liste des changements de vitesses à appliquer à chaque trajectoire. Pour passer d'un état à son voisin, l'opérateur de voisinage sélectionne aléatoirement un des avions de la liste des avions à réguler. Il modifie aléatoirement soit son délai, soit un de ses changements de vitesse :

1. Le délai est incrémenté ou décrémenté, si possible (l'avion n'est pas encore parti), avec une probabilité de 5 % ;
2. Si le délai n'a pas été modifié :
   - Un changement de vitesse est inséré avec une probabilité de 10 %, ou 100 % si aucun changement de vitesse n'a été planifié auparavant. La vitesse est choisie aléatoirement parmi la liste des vitesses admissibles ;
   - Un changement de vitesse choisi aléatoirement est supprimé avec une probabilité de 10 % ;
   - Un changement de vitesse choisi aléatoirement est avancé ou retardé de 5 s avec une probabilité de 40 % ;
   - Un changement de vitesse choisi aléatoirement est amplifié ou diminué avec une probabilité de 40 %.

La liste des vitesses admissibles est composé de l'ensemble des vitesses que peut atteindre l'avion en un multiple de 5 secondes en partant de sa vitesse optimale. L'accélération est fixée comme indiqué dans la section 4.1, comme étant la valeur maximale inférieure à 4 000 $\mathrm{NM/h^2}$ qui permet d'atteindre la vitesse maximale à partir de la vitesse de croisière $v_{i,\mathrm{opt}}$ en un multiple de 5 secondes.

4.3.2 Implémentation

4.3.2.1 Intégration dans le système multi-agents

Afin de réutiliser un maximum de code déjà existant, le recuit simulé a été intégré au système multi-agents Plectre v2. Comme indiqué dans la figure 4.20, un nouvel agent est inséré dans le SMA, GlobalSpeedRegulationAgent (agent de régulation de vitesses global). Celui-ci collecte les messages ADS-C envoyés par les agents, lance périodiquement le recuit simulé, et renvoie les trajectoires optimisées aux avions, qui les exécutent. Quand le recuit simulé est activé, le processus de décisions des avions est inhibé.

Comme les avions sont générés 10 minutes avant leur départ, le recuit simulé est exécuté toutes les 9 minutes, de manière à optimiser le retard des avions au moins une fois avant que ceux-ci ne partent.

4.3.2.2 Détails de l'implémentation

4.3.2.2.1 Détection des conflits Parmi les trois variables à optimiser de la fonction objectif, le nombre de changements de vitesse et le délai sont directement issues de

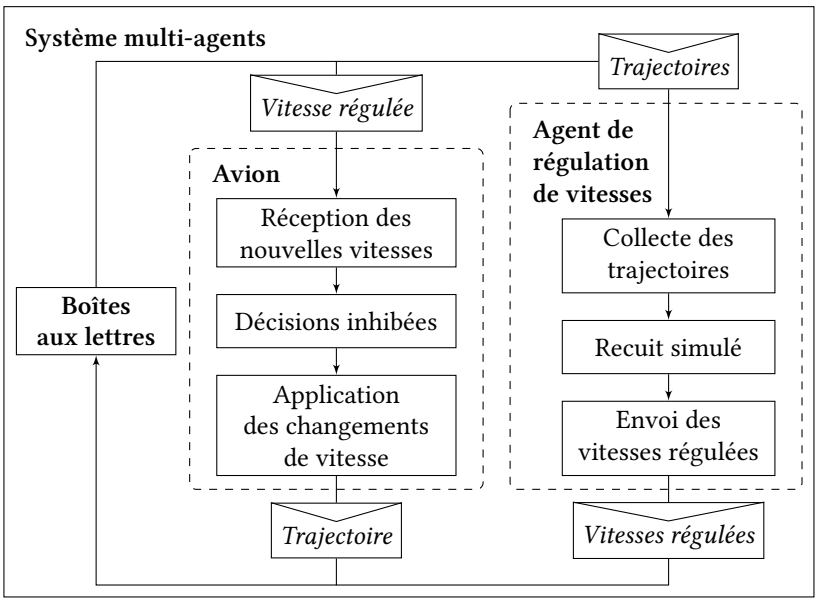


FIGURE 4.20 – Processus d'optimisation global de la trajectoire des avions dans Plectre v2. Un nouvel agent est intégré au système multi-agents décrit dans la section 4.2. Cet agent collecte les trajectoires envoyées par les avions dans les messages ADS-C. Il optimise les changements de vitesse et le retard du départ, et renvoie ces directives aux avions. Ceux-ci appliquent alors les ordres, leur processus de décision étant inhibé si l'agent de régulation de vitesses est activé.

l'état, et sont rapides à calculer. Par contre, le calcul de la date du premier conflit demande plus temps.

Il est en théorie possible de calculer la date du premier conflit de manière analytique, à partir de la trajectoire 4D continue. Les trajectoires sont des courbes composées de sections élémentaires : segments de droite quand l'avion vole à vitesse constante, arcs de cercles quand il tourne, parabole quand il accélère, l'accélération étant constante. On peut parcourir la liste des portions de trajectoires de chaque paire d'avion et déterminer analytiquement à quel instant t les avions sont séparés par exactement 5 NM.

Par manque de temps, il a été décidé d'utiliser une méthode plus simple pour détecter la date du premier conflit. Elle consiste à échantillonner les trajectoires avec un pas de temps déterminé (une seconde), puis à calculer à chaque pas de temps la distance séparant chaque paire d'avions. Théoriquement, cette méthode procède à N^2T mesures de distance, pour N avions et T pas de temps.

En pratique, Plectre v2 n'effectue que $N \times T$ mesures, en indexant les positions des avions dans une HashMap (voir l'annexe A). À chaque pas de temps (T opérations), on stocke l'ensemble des vecteurs position dans la hashmap, puis on parcourt itérativement les positions (N opérations). Pour chaque position, on vérifie si une autre position est située à moins de 5 NM (1 opération, temps d'accès constant garanti par la HashMap).

La méthode basée sur l'échantillonnage des trajectoires a l'avantage d'être simple et rapide à implémenter, contrairement à la méthode analytique, qui nécessite plus

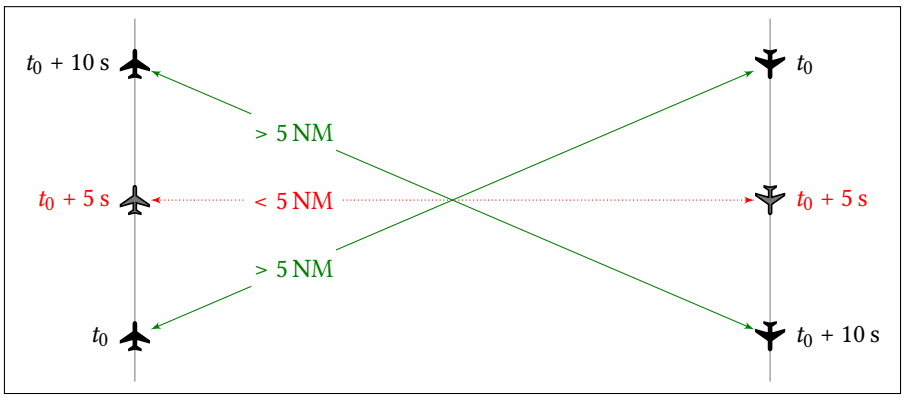


FIGURE 4.21 – Conflit non détecté par le recuit simulé (faux négatifs). Les trajectoires sont échantillonnées toutes les 10 secondes, ici à t_0 et t_0 + 10 s. À ces deux instants, les avions sont séparés par plus de 5 NM, aucun conflit n'est détecté. Or, à t_0 + 5 s, les avions sont à moins de 5 NM, donc en conflit.

de temps de conception. Mais la méthode analytique offre probablement un temps de calcul plus faible, puisque le nombre de portions de chaque trajectoire est peu élevé, entre quelques dizaines et quelques centaines suivant le temps de vol, qu'il faut comparer aux 3 600 échantillons qu'il faut traiter par heure de vol et par trajectoire.

Les deux sections suivantes présentent des améliorations apportées à la méthode basée sur l'échantillonnage des trajectoires, qui est utilisée dans l'évaluation de la fonction objectif du recuit simulé.

Augmentation du pas de temps Pour réduire fortement le temps de calcul, l'algorithme échantillonne les trajectoires avec un pas de 10 secondes. Si un conflit est trouvé à la date t_{10}, celui-ci a probablement commencé quelques secondes avant. Pour trouver la seconde exacte t_1 à laquelle le conflit commence, l'algorithme échantillonne les 10 secondes précédant l'instant t_{10} avec un pas d'une seconde.

L'avantage de cette méthode est de réduire le temps de calcul dédié à cette tâche d'un facteur 10. C'est d'autant plus visible quand l'algorithme a trouvé une solution sans conflit et optimise les deux autres variables : quand un conflit est présent, la recherche de conflits s'arrête dès que celui-ci est trouvé. Sinon, l'algorithme doit vérifier qu'aucun conflit n'est présent sur la totalité de la longueur des trajectoires.

L'inconvénient est que les conflits durant moins de 10 secondes se déroulant entre deux pas de temps peuvent ne pas être détectés, comme le montre la figure 4.21. Ces conflits non détectés sont similaires à celui présenté sur cette figure : ils sont peu sévères, les avions étant en permanence séparés par plus de 4,5 NM durant le conflit. Ce sont néanmoins des faux négatifs dans le calcul de l'instant du premier conflit.

Ajout d'une marge spatiale à la détection de conflits Pour réduire le nombre de conflits non détectés par le recuit simulé tout en gardant un pas de temps de 10 secondes, une approche consiste à ajouter une marge à la distance de séparation requise durant cette première étape de la détection. Ainsi, les conflits qui se produisent

entre deux pas de temps sont quand même détectés, au prix de l'apparition de faux positifs.

Algorithme 5 Détection des conflits avec marge spatiale

```
1:  function DétectionConflits(avions)
2:      for t ← 5..t_max step 10 do
3:          d ← distanceMinimale + marge
4:          cflPotentiel ← DistanceNonRespectée(t, d, avions)
5:          if cflPotentiel ≠ ∅ then
6:              for t′ ← t − 5..t + 4 do
7:                  cfl ← DistanceNonRespectée(t, distance, cflPotentiel)
8:                  if cfl ≠ ∅ then
9:                      return t′
10:                 end if
11:             end for
12:         end if
13:     end for
14:     return ∅
15: end function
```

Il faut alors détecter les conflits les dix secondes précédentes avec un pas de temps d'une seconde, pour vérifier si un conflit a lieu, ou si c'est un faux positif. Comme le montre l'algorithme 5, la détection de conflits repose sur deux processus itératifs imbriqués. Le premier détecte les conflits toutes les 10 secondes avec une marge spatiale. Si un conflit potentiel est détecté, la boucle imbriquée itère sur les 5 secondes précédentes et les 5 secondes suivantes avec un pas d'une seconde pour détecter les conflits.

Ainsi, en théorie, cette procédure allie la rapidité de la détection de conflits avec un pas de temps de 10 secondes avec la précision de la détection de conflits avec un pas d'une seconde. Dans le meilleur des cas, quand les avions sont éloignés la plupart du temps, peu de faux positifs sont détectés, et l'algorithme est aussi rapide que sans l'utilisation de la marge spatiale.

Par contre, dans le pire cas, quand deux avions se suivent un long moment sur la même route avec une distance d'exactement 5 NM, un faux positif est détecté à chaque itération, ce qui conduit l'algorithme à évaluer les conflits toutes les secondes. Cependant, la deuxième étape ne détecte les conflits qu'entre les avions impliqués dans le conflit potentiel détecté durant la première étape, ce qui est plus rapide que si l'algorithme devait détecter les conflits toutes les secondes le long de l'ensemble des trajectoires des avions présents dans la simulation.

La marge spatiale m à appliquer doit être suffisamment grande pour détecter tous les conflits (pas de faux négatifs), tout en étant suffisamment petite pour minimiser le nombre de faux positifs. La borne supérieure $m_{\max}$ de la marge est donnée par la situation décrite dans la figure 4.22, où la vitesse relative des deux avions est de $2 \times (487\,\text{kt} + 3\,\%)$. La marge utilisée dans le recuit simulé est la valeur $m_{\max}$ pondérée par la vitesse relative de chaque paire d'avion. Si deux avions se suivent, leur vitesse relative est nulle, la marge est de 0 NM. Si le cap de deux avions volant l'un vers l'autre à 501 kt est opposé (par exemple, l'un allant vers le nord et l'autre vers le sud), la marge est fixée à sa valeur maximale.

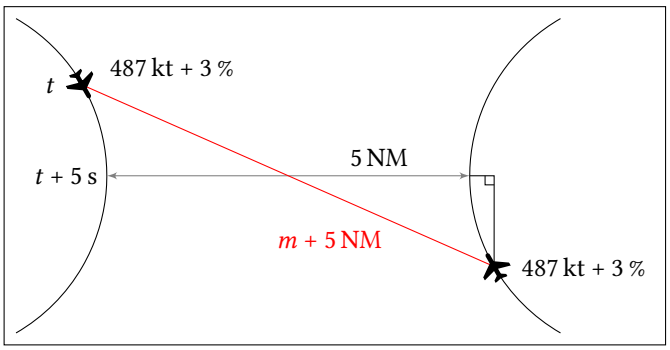


FIGURE 4.22 – Valeur maximale de la marge spatiale m utilisée pour détecter les conflits. Les avions volant à la vitesse plus grande vitesse possible effectuent un virage, et sont au plus près à l'instant t + 5 s. La marge est calculée à partir de leur position à l'instant t.

4.3.2.2.2 Fenêtre temporelle glissante La méthode la plus simple pour optimiser des trajectoires grâce à un recuit simulé (ou tout autre algorithme d'optimisation globale) consiste à collecter l'ensemble des trajectoires du scénario considéré, puis à optimiser en une seule fois la valeur de la fonction objectif liée à la totalité des trajectoires. Cette méthode est la plus simple, mais elle est relativement lente, puisque la combinatoire du problème est élevée.

Pour réduire la combinatoire, il faut découper le problème en problèmes plus petits. Une première approche, utilisée dans [BDG09], consiste à optimiser uniquement les trajectoires d'avions en conflit. Un premier algorithme détecte les avions en conflit, puis le processus d'optimisation globale (dans cet article, un algorithme génétique) modifie les trajectoires. L'algorithme de détection des conflits vérifie alors qu'aucun conflit n'a été provoqué par l'optimisation. Dans le cas contraire, l'optimisation est relancée en incluant les nouveaux conflits.

Cette approche est utilisée à un certain degré dans Plectre v2. Les conflits sont détectés entre tous les avions à chaque évaluation de la fonction objectif. Mais l'opérateur de voisinage ne modifie que les trajectoires des avions en conflit.

Un autre moyen de réduire la combinatoire consiste à utiliser une fenêtre temporelle glissante, ce qui permet de découper temporellement le problème. Ce procédé a notamment été utilisé pour certains problèmes d'optimisation de trajectoires à l'approche des aéroports [Ma+16b ; Ma+16a ; Lia+16].

On optimise uniquement les trajectoires présentes durant une période donnée, qui dure de quelques minutes à quelques heures selon le problème. Les trajectoires optimisées sont celles qui commencent durant cette période, ou celles qui ont déjà commencé mais ne sont pas terminées. Pour ces dernières, on n'optimise que la portion de trajectoire postérieure au début de la période. Le processus d'optimisation ignore les trajectoires terminées avant le début de la période, et celles qui commencent après la fin de celle-ci.

Une fois que les trajectoires de la période sont optimisées, on décale la fenêtre à une date ultérieure, et on recommence le processus. L'utilisation de fenêtres glissantes permet de réduire la taille du problème. Le temps nécessaire à l'optimisation est donc réduit [Ma+16b].

Dans Plectre v2, c'est l'interaction entre l'agent de régulation de vitesses et le

système multi-agent qui fournit ce mécanisme. L'agent de régulation collecte les trajectoires durant 9 minutes, puis lance le processus de recuit simulé pour les optimiser. L'agent ne connait que les trajectoires des avions ayant émis un message durant les 9 dernières minutes :

- Les avions qui ont atteint leur destination ne sont pas pris en compte, parce qu'ils n'envoient plus de message, l'agent de régulation ne les collecte donc pas ;
- L'agent de régulation ne modifie que les portions de trajectoires postérieures au lancement du processus, les portions antérieures des trajectoires des avions en vol ne sont pas modifiées ;
- L'agent peut modifier l'heure de départ et la vitesse des avions qui partent moins de dix minutes après le lancement du processus ;
- Les avions qui ne sont pas encore apparus dans la simulation n'émettent pas encore de messages, leur trajectoire n'est donc pas connu par l'agent de régulation.

4.3.2.3 Limite de l'implémentation : gradient insuffisant

Comme indiqué précédemment, l'opérateur de voisinage impose que si la date du premier conflit n'est pas modifiée, au moins 88 % des transitions ne dégradent pas la solution. La valeur de la fonction objectif n'est ni dégradée ni améliorée quand un changement de vitesse est avancé, retardé, augmenté ou diminué, ce qui arrive avec une probabilité de 76 % (= 95 % × (40 % + 40 %)), sauf si la date du premier conflit est changée par ce changement de vitesse.

Le processus n'a pas de gradient clair à suivre, la plupart des transitions sont donc acceptées par le recuit simulé. C'est pourquoi la procédure de chauffage présentée dans l'algorithme 2 de la section 2.2 est utilisée. Au lieu de partir d'une température initiale basse, puis de l'augmenter progressivement jusqu'à obtenir un taux d'acceptation d'environ 80 %, la méthode utilisée ici fait une exploration aléatoire de l'espace d'états (10 000 itérations), et utilise les valeurs du critère la plus basse et la plus haute pour en déduire la température initiale : celle-ci doit permettre d'accepter une transition de la meilleure solution à la pire avec une probabilité relativement élevée (ici 10 %).

4.3.3 Résultats

Le recuit simulé a été testé sur le même scénario de trafic basé sur des plans de vol que le système multi-agents Plectre v2, décrit dans la section 4.2.5.3. Dans ce scénario, 283 avions volent à 37 000 ft en France entre 4 heures et 14 heures.

Le tableau 4.3 présente les résultats. Les deux premières lignes reprennent les résultats obtenus quand aucun processus de régulation n'est activé, et quand le système multi-agents est utilisé avec 100 % d'avions coopératifs.

Les deux lignes suivantes du tableau présentent les résultats obtenus par le recuit simulé, sans utiliser, puis en activant la détection de conflits avec marge spatiale décrite dans la section 4.3.2.2.1.

Premièrement, on peut remarquer que la méthode de détection de conflits avec utilisation d'une marge spatiale permet bien de diminuer le nombre de conflits non détectés par le recuit simulé (faux négatifs). Ainsi, ce nombre de 2,2 à 1,2 en moyenne, mais au prix d'un temps de calcul presque multiplié par 4, puisqu'il passe de 30 minutes à 1 heure et 50 minutes. Le temps de calcul est plus important à cause de l'apparition

TABLE 4.3 – Performances du recuit simulé intégré à Plectre v2 dans le scénario de trafic réaliste (valeur moyenne après 10 exécutions de la simulation).

Processus de décision	Nombre d'avions	Conflits incluant les faux négatifs	Conflits excluant les faux négatifs	Temps de calcul
Désactivé	283	47,0		1 min 35 s
SMA	283	0,6 (−98,7 %)		3 min 27 s
Recuit simulé sans marge	283	2,6 (−94,5 %)	0,4 (−99,1 %)	30 min 17 s
Recuit simulé avec marge	283	1,7 (−96,4 %)	0,5 (−98,9 %)	1 h 50 min 27 s

Processus de décision	Distance minimale (NM)	Distance moyenne (NM)	Accélérations	Retard (s)
Désactivé	0,00	1,69	0,0	0,0
SMA	1,95	3,81	5,8	5,8
Recuit simulé sans marge	0,00	3,85	17,6	4,3
Recuit simulé avec marge	0,00	4,18	22,9	3,9

de faux positifs, c'est-à-dire d'avions qui ne sont pas en conflit mais dont la distance de séparation est inférieure à la marge, chaque faux positif obligeant l'algorithme à effectuer une passe de détection de conflits.

Par contre, si le nombre de faux négatifs diminue, le nombre de conflits détectés mais non résolus n'évolue pas (il passe de 0,4 à 0,5). Comme le montre la figure 4.23, sur les 10 exécutions de chaque algorithme sur le même scénario, au moins 6 aboutissent à la résolution de tous les conflits (8 pour le recuit simulé avec marge). Dans les autres exécutions, on compte un ou deux conflits résiduels, sauf pour le recuit simulé avec marge, où une exécution laisse quatre conflits.

Les fluctuations du nombre de conflits résolus sont dues à la nature heuristique de ces algorithmes. Dans le système multi-agents, le processus de décision est déterministe, mais l'ordre dans lequel les avions prennent leurs décisions varie, ce qui peut entrainer des différences de résultats. Dans le recuit simulé, les décisions sont prises à heures fixes (multiples de 9 minutes), mais le processus lui-même est aléatoire.

Dans le pire cas rapporté par la figure 4.23, l'algorithme de recuit simulé avec marge est tout de même capable de résoudre 91 % des 47 conflits initiaux. Les heuristiques ne garantissent pas que la solution optimale soit atteinte, mais dans la plupart des situations ces algorithmes donnent de bons résultats.

Deuxièmement, en comparant les deux variantes du recuit simulé avec le système muiti-agents, on observe que ce dernier possède plusieurs avantages. Le nombre de conflits résiduels total est plus faible pour le système multi-agents que pour le recuit simulé (0,6 conflits contre 2,6 et 1,7). Les résultats obtenus par le processus de décision des agents avions comportent moins de faux négatifs que les résultats des deux variantes de détection de conflit utilisées par le recuit simulé.

Le nombre moyen de conflits détectés mais non résolus est quant à lui très similaire pour les trois algorithmes. On observe entre 0,4 et 0,6 conflits résiduels en moyenne.

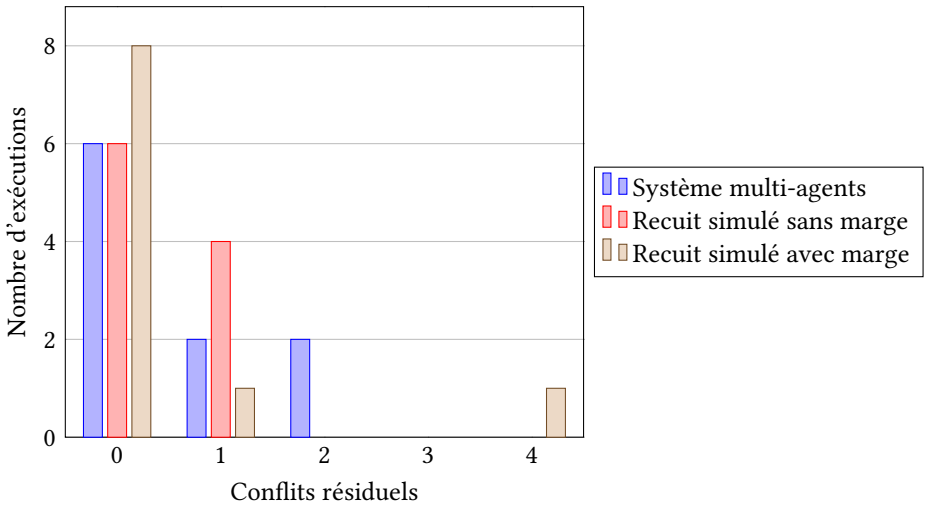


FIGURE 4.23 – Répartition du nombre de conflits résiduels sur les 10 exécutions de chaque algorithme.

Mais pour parvenir à ce résultat, le système multi-agents propose une solution plus proche de l'optimum en beaucoup moins de temps : le SMA s'exécute en moyenne en 3,5 minutes, contre 30 et 110 minutes pour les recuits simulés.

Le SMA planifie également moins de manœuvres : 5,8 accélérations et 5,8 secondes de retard en moyenne, contre 17,6 accélérations et 4,3 secondes pour la première variante du recuit simulé, et 22,9 accélérations et 3,9 secondes pour la seconde. Le fait que le retard moyen soit plus faible pour le recuit simulé que pour le SMA peut paraître étonnant, mais indique que le recuit n'est pas suffisamment proche de la valeur optimale. En effet, la fonction objectif optimise, par ordre de priorité, le nombre de conflits, le nombre d'accélérations, et le retard. Un résultat plus proche de l'optimal contiendrait moins d'accélérations et un retard plus important. En permettant au recuit simulé de converger plus lentement (plus de paliers de température et plus d'itérations par palier), celui-ci serait capable de plus s'approcher de l'optimum, mais au prix d'un temps de calcul plus élevé.

Dans la version actuelle des algorithmes, le système multi-agents fournit donc une meilleure solution, en terme de nombre de conflits et d'accélérations, que les deux versions du recuit simulé, en 10 à 30 fois moins de temps.

Cette lenteur est causée principalement par le coût de l'évaluation de la fonction objectif, et plus précisément par le calcul de la date du premier conflit. En effet, celle-ci doit évaluer après chaque modification des trajectoires par le processus qu'aucun nouveau conflit n'apparaît. Si un conflit est détecté, le processus s'arrête rapidement. Mais si aucun conflit n'a lieu, le processus doit vérifier l'ensemble des trajectoire jusqu'à leur fin.

Le processus utilise une hashMap pour accélérer la détection : à chaque pas de temps, une HashMap est créée, dans laquelle est indexée la position de tous les avions à cet instant. Grâce à cela, le temps de calcul croît linéairement avec le nombre d'avions. Mais il est également linéairement proportionnel avec la durée des trajectoires.

Pour améliorer le processus, il faudrait créer une HashMap qui indexerait les posi-

tions des avions dans les dimensions spatiales et temporelle. À la première exécution du processus, celui-ci échantillonnerait les trajectoires dans leur totalité. Ensuite, à chaque fois qu'une trajectoire serait modifiée, les anciennes positions de la trajectoire seraient supprimées de la HashMap, et les nouvelles insérées à la place. Ainsi, le temps de calcul ne serait plus proportionnel au nombre d'avions, mais uniquement à la durée de la trajectoire modifiée.

Par contre, la manière de détecter les conflits serait différente. En effet, dans la version actuelle du programme, le processus est relativement simple. Celui-ci parcourt les trajectoires jusqu'à trouver un conflit, puis s'arrête. Or, il se peut que la modification d'une trajectoire n'impacte pas la date du premier conflit, si l'avion est impliqué dans des conflits ultérieurs. Dans cette proposition d'amélioration, il ne suffit donc pas de vérifier l'apparition de nouveaux conflits le long de la trajectoire modifiée uniquement, mais il faut maintenir une liste de tous les conflits. Ainsi, après avoir modifié une trajectoire, cette liste de conflits serait rafraichie, en supprimant les conflits impliquant l'avion dont la trajectoire a été modifiée, puis en rajoutant les nouveaux conflits impliquant cet avion.

Dans la version de la détection de conflits avec marge spatiale, la lenteur de l'algorithme provient du grand nombre de faux positifs trouvés lors de la détection de conflits. Ceux-ci donnent en effet lieu à une détection fine des conflits durant les 5 secondes précédant et suivant la date du conflit potentiel détecté. Or, ceci est particulièrement inefficace quand deux avions se suivent à 5 NM de distance durant une longue période : à un instant donné, l'algorithme détecte un conflit potentiel, puis vérifie chacune des 10 secondes entourant cet instant, avant de se rendre compte que c'est un faux positif. Il recommence alors l'opération toutes les 10 secondes, jusqu'à ce que les deux avions divergent. Cela explique que le temps de calcul dans ce cas soit presque quatre fois plus long que sans utilisation de marge spatiale.

Cette version du recuit simulé bénéficierait également de l'amélioration de la détection des conflits grâce à la HashMap spatiotemporelle. En effet, le phénomène décrit ci-dessus, impliquant des avions proches mais pas en conflit durant de longues périodes (faux positifs), ne déclencherait ces longs calculs que quand la trajectoire d'un des avions impliqués est modifiée. Actuellement, un phénomène de ce type nécessite la détection des conflits à chaque modification de n'importe quelle trajectoire.

Une autre manière d'améliorer le temps de calcul consisterait à utiliser une approche optimiste, similaire à celle développée pour l'algorithme génétique utilisé par le projet ERASMUS [BDG09]. Dans cette variante de Plectre v2, les conflits seraient détectés avant de lancer le recuit simulé. Seules les trajectoires des avions en conflits seraient optimisées. Si le résultat de l'optimisation provoque un nouveau conflit, la trajectoire de l'avion impliqué serait alors rajoutée à la liste des trajectoires à optimiser, et le recuit simulé serait relancé avec cette nouvelle liste de trajectoires. Ainsi, la détection de conflits effectuée durant l'évaluation de la fonction objectif n'impliquerait qu'un sous-ensemble restreint des trajectoires.

Il est néanmoins vraisemblable que les gains de performances soient équivalents à ceux obtenus par l'utilisation d'une HashMap spatiotemporelle, et les gains ne se cumuleraient peut-être pas. Dans la variante de la détection de conflits sans marge spatiale, l'amélioration de la HashMap et l'optimisation des trajectoires en conflit uniquement ont le même effet : les conflits ne sont plus détectés qu'entre deux ou trois avions, au lieu d'être détectés entre la totalité des avions.

Dans la variante de la détection de conflits avec marge spatiale, dans les deux améliorations proposées (HashMap ou optimisation d'un sous-ensemble de trajectoires), les effets sont également similaires : le nombre d'occasions où l'algorithme doit filtrer

les vrais et les faux positifs diminue. Soit la trajectoire d'un des avions provoquant un grand nombre de faux positifs est modifiée ou est impliquée dans un conflit, et le processus doit filtrer les vrais et les faux positifs, ce qui ralentit l'évaluation de la fonction objectif. Soit cette trajectoire n'est ni modifiée, ni impliquée dans un conflit, et ce filtrage n'est pas fait, ce qui améliore le temps de calcul.

4.4 Conclusion

Les résultats de ce chapitre montrent que le système multi-agents Plectre v2 permet au trafic de s'auto-structurer de manière à réduire le nombre de conflits. Il est capable de résoudre 98,7 % des conflits quand tous les avions coopèrent, et il est résilient aux perturbations telles que la non-coopération de certains agents.

La comparaison entre ce SMA et un processus d'optimisation globale basé sur un algorithme de recuit simulé montre que le SMA est capable d'obtenir des résultats similaires en terme de nombre de conflits et de changements de vitesse que le recuit simulé. Il nécessite pour cela d'un temps de calcul beaucoup plus faible. La lenteur du recuit simulé est principalement causée par l'évaluation de la date du premier conflit. Plusieurs pistes ont été évoquées pour améliorer ce point, mais n'ont pas pû être explorées.

L'algorithme Plectre v2 permet de résoudre une grande partie des conflits. Comme la complexité du trafic est liée au nombre de conflits aériens, Plectre v2 permet de réduire la complexité moyenne du trafic.

Mais la régulation de vitesse seule ne permet pas de tous les résoudre. Plectre v2 est conçu pour aider les contrôleurs aériens, et pas pour les remplacer. Pour résoudre tous les conflits, les avions doivent exécuter d'autres types de manœuvres que la régulation de vitesse, comme le changement de cap ou de niveau de vol, ou le retard du départ. Permettre aux avions de retarder leur départ permet à Plectre v2 de résoudre tous les conflits résiduels. Actuellement, ces décisions sont prises par les contrôleurs, et il n'est pas prévu de les déléguer à des algorithmes dans les espaces aériens contrôlés par les contrôleurs aériens.

L'impact de l'algorithme sur la complexité perçue par les contrôleurs est positif dans la plupart des cas :

- Soit Plectre v2 arrive a résoudre les conflits en adaptant la vitesse des avions : le nombre de conflits diminue, ainsi que la complexité du trafic ;
- Soit l'algorithme ne peut pas résoudre un conflit : la vitesse n'est pas modifiée, et la complexité perçue par le contrôleur ne varie pas ;
- Soit l'algorithme résout un conflit, mais en génère un second plus tardif, qu'il n'est pas capable de résoudre : la complexité augmente localement, puisque le contrôleur a un conflit de plus à résoudre. Cette dernière possibilité est néanmoins la moins fréquente dans les scénarios étudiés ; le nombre total de conflits est tout de même réduit.

Les décisions des contrôleurs peuvent également impacter l'algorithme : celles-ci peuvent être considérées comme une perturbation imposée par l'environnement aux avions. Ceux-ci peuvent être amenés à modifier les décisions qu'ils ont planifiées. Actuellement, les avions remettent déjà en cause leurs décisions, par exemple à chaque fois qu'un nouvel avion est introduit dans la simulation. Le système multi-agents s'adapte alors à cette perturbation et continue à remplir son rôle de solveur de conflits.

Il serait intéressant de permettre aux avions évoluant dans la simulation de faire évoluer leur altitude au cours du temps. En effet, l'altitude de croisière des avions évolue au cours du temps. Une des raisons est qu'en consommant du carburant, l'avion s'allège, et son altitude optimale augmente. Un avion change donc généralement de niveau de vol une ou plusieurs fois au cours de sa phase de croisière. Ces changements d'altitude peuvent être implémentés de deux manière :

1. Soit le profil vertical de l'avion est défini dans le plan de vol. La trajectoire peut toujours être définie par un ensemble de sections où l'accélération est constante. Si on considère que l'avion n'accélère pas longitudinalement durant sa phase de montée, on peut découper celle-ci en trois sections.

   a) Pour monter, l'avion commence par augmenter son taux de montée (la composante verticale de l'accélération devient positive). Cette première section peut être représentée par une parabole.

   b) Puis l'avion maintient son taux de montée (accélération nulle). Cette section peut être symbolisé par un segment de droite.

   c) Enfin, l'avion arrive à son altitude cible. Il réduit alors son taux de montée, ce qui peut également être représenté par une parabole.

   L'avion peut donc encore détecter les conflits potentiels comme l'intersection entre le cylindre de séparation (5 NM de rayon et 1 000 ft de haut) et les différentes portions de sa trajectoire. Il planifie alors ses changements de vitesse en utilisant la même méthode que décrite précédemment dans ce chapitre.

2. Soit le profil vertical est une des variables à optimiser durant le processus de décision. Plusieurs problèmes se posent alors :

   — Comme évoqué précédemment, un algorithme pouvant faire évoluer l'altitude d'un avion peut être considéré comme intrusif par les contrôleurs aériens. C'est pour cela que le projet ERASMUS se focalise sur les changements de vitesse.

   — Pour permettre aux avions de changer de niveau de vol, il faut modifier la représentation interne de leur trajectoire. Celle-ci doit alors être définie en trois dimensions au lieu de deux : abscisse curviligne (distance parcourue le long de la trajectoire), temps, et *altitude*. L'altitude peut être discrète, avec un palier d'altitude tous les 10 niveaux de vol (1 000 ft), ce qui est suffisant pour y représenter les conflits avec les avions des autres niveaux de vol, et ceux dont l'altitude évolue.

     À chaque pas de temps, le processus de décision peut alors choisir entre quatre manœuvres au lieu de deux : accélérer, ralentir, *monter* de 1 000 ft, *descendre* de 1 000 ft [1].

Grâce à la capacité des SMA d'intégrer des agents non-coopératifs et leur résistance aux évènements imprévus, ils offrent un cadre général pour mélanger du trafic géré par les contrôleurs et du trafic automatisé. À terme, ces algorithmes pourraient collaborer avec les contrôleurs aériens dans le cadre de la gestion du trafic aérien, puisqu'ils

1. On peut même combiner plusieurs manœuvres à la fois, comme monter et accélérer, mais la construction de la trajectoire 4D se complexifie. C'est également en raison de la complexité induite de la représentation de la trajectoire 4D que les avions ne peuvent pas tourner et accélérer en même temps, mais doivent attendre la fin d'un virage pour accélérer.

travaillent à des échelles différentes, sur la dimension temporelle pour l'algorithme et sur les dimensions spatiales pour les contrôleurs aériens.

Le chapitre suivant propose une méthode de structuration spatiale du trafic par la création dynamique de réseaux de routes pour en réduire la complexité. Cet algorithme établit à intervalle régulier une carte de complexité du trafic, afin de détecter les zones dont la complexité dépasse une seuil. Il crée alors un réseau de routes temporaire en agrégeant les trajectoires des avions traversant ces zones. Ce réseau est adapté à chaque configuration de trafic.

Chapitre 5

Structuration sur réseaux temporaires

Le concept de Free Flight Area cherche à donner plus de liberté aux avions en les laissant planifier leur trajectoire sans leur imposer de suivre un réseau de routes. Le FFA permet aux avions de suivre une trajectoire qui minimise la consommation de carburant.

Cependant, l'inconvénient du FFA est une augmentation possible de la complexité du trafic : comme les avions ne sont pas organisés sur un réseau de route, les contrôleurs aériens subissent une charge cognitive plus élevée pour comprendre et réguler le trafic [Hil04, p. 3.1.1]. C'est pourquoi l'implémentation des FFA est envisagé uniquement pour des conditions de trafic peu dense, à haute altitude ou la nuit.

Pour faciliter le contrôle des FFA, la charge de travail peut être réduite en créant des réseaux de routes locaux dans des zones spécifiques, et en laissant les avions suivre des trajectoires personnalisées en dehors de ces zones. L'algorithme Yard (Yet Another Route Designer), présenté dans ce chapitre, remplit deux fonctions.

La première fonction de Yard est la surveillance de la complexité du trafic pour détecter les zones où le trafic provoque une augmentation de la charge de travail des contrôleurs. La complexité est déterminée en utilisant la métrique de convergence décrite dans la section 1.2.3.3.2.

La deuxième fonction de Yard est la création de réseaux de routes temporaires dans les zones précédemment détectées pour réduire localement la complexité du trafic aérien en structurant spatialement les trajectoires. Un réseau de routes local est construit en agrégeant les trajectoires des avions traversant chaque zone complexe détectée durant la première étape. Cette structuration est réalisée par la méthode d'agrégation de trajectoires par minimisation de l'entropie de la densité de trafic décrite dans la section 1.2.3.5.

Ce chapitre présente deux implémentations de cette méthode. La première est une implémentation centralisée. Un agent est ajouté dans le système multi-agents Plectre v2 (chapitre 4). Celui-ci collecte les trajectoires des avions, crée les réseaux de routes, et renvoie les nouveaux plans de vols aux avions.

La deuxième implémentation propose une approche décentralisée, où les avions de Plectre v2, qui collectent déjà les trajectoires de leurs voisins pour résoudre les conflits, modifient leurs trajectoires pour faire émerger un réseau de routes local.

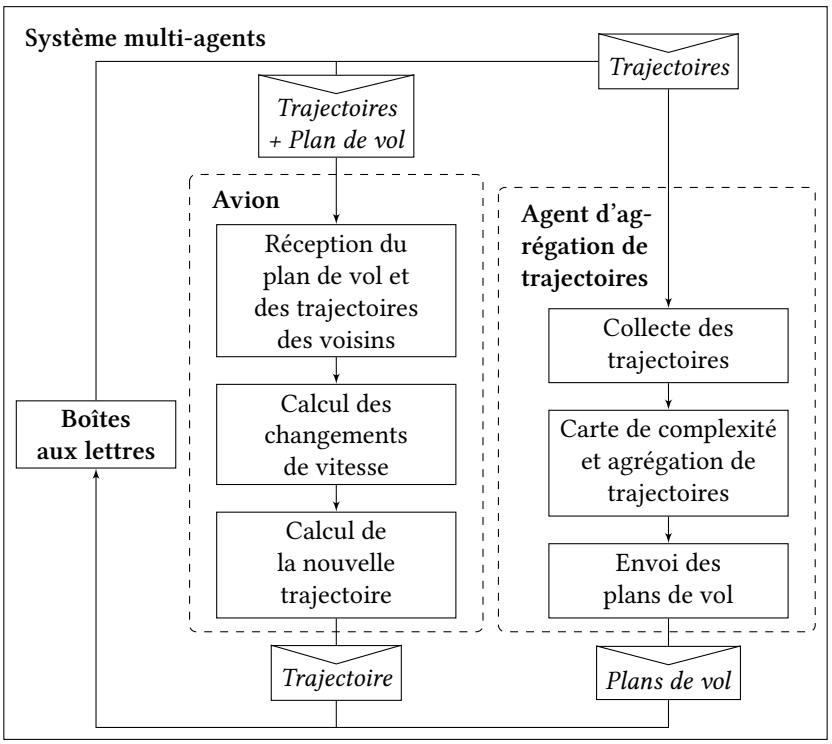


FIGURE 5.1 – Cycle de vie de l'algorithme Yard décrit dans la section 5.1. Un nouvel agent est ajouté au système multi-agents Plectre v2 décrit dans le chapitre 4. Cet agent collecte les trajectoires envoyées par les avions dans les messages ADS-C. Il mesure alors la complexité du trafic, afin de détecter les zones de forte complexité dans lesquelles il crée des réseaux de routes locaux qui réduisent la complexité du trafic. Finalement, ces nouveaux plans de vol sont renvoyées aux avions, qui mettent à jour leurs trajectoires.

5.1 Implémentation centralisée

L'algorithme Yard est construit sur la base du système multi-agents Plectre v2, décrit dans le chapitre 4. Un nouvel agent est inséré dans le SMA, comme le montre la figure 5.1. L'agent d'agrégation de trajectoires commence par collecter les messages ADS-C envoyés par les avions, qui contiennent leurs trajectoires 4D. Il calcule ensuite, toutes les 40 secondes, une carte de complexité grâce à ces trajectoires pour détecter les zones où la complexité est plus élevée que la limite autorisée. Enfin, l'agent groupe les trajectoires dans chacune de ces zones pour créer un nouveau réseau de routes local. Il envoie alors de nouveaux plans de vol aux avions dont les trajectoires doivent être modifiées.

5.1.1 Carte de complexité

La complexité du trafic est mesurée par la métrique de convergence développée par Delahaye et Puechmorel (2000) [DP00, Section 3], décrite dans la section 1.2.3.3.2. Celle-ci est plus précise dans l'estimation de la complexité perçue par les contrôleurs

aériens que la mesure de la densité du trafic (section 1.2.3.3.1), puisqu'elle distingue des situations où les avions convergent, avec un risque de conflit, et celles où ils divergent, sans risque de conflit. La convergence est également beaucoup plus rapide à calculer que la complexité par exposants de Lyapunov (section 1.2.3.3.3), même si cette dernière est plus précise dans l'estimation de la complexité.

La mesure de la convergence utilisée par Yard est une variante robuste de la méthode de Delahaye et Puechmorel [DP00, Section 3], prenant en compte l'incertitude sur la position des avions. En effet, comme évoqué dans la section 1.1.3.3.2, l'avion est capable de suivre sa route avec une précision spatiale de l'ordre de quelques dizaines de mètres, mais la composante longitudinale de sa trajectoire est beaucoup moins précise, l'écart type de l'erreur étant de 15 kt après une heure de vol. Une évaluation robuste de la complexité doit prendre en compte cette incertitude longitudinale. On cherche la valeur maximale de la convergence dans une fenêtre temporelle $\Delta t = \frac{15\,\mathrm{NM}}{v_{i,\min|v_{i,\mathrm{opt}}}}$, où $v_{i,\min|v_{i,\mathrm{opt}}}$ est la vitesse minimale de l'avion i. La vitesse minimale d'un avion dans Plectre étant 420 kt, la plus grande fenêtre est d'environs 128 s.

Pour prendre en compte cette incertitude, les valeurs de la convergence entre les positions de deux avions sont calculées en utilisant une fenêtre temporelle Δt. Pour chaque paire d'avions i et j, et pour deux instants t et $t' \in [t - \Delta t, t + \Delta t]$, on calcule d'abord la position et la vitesse relatives $\vec{p_{ij}}(t, t')$ et $\vec{v_{ij}}(t, t')$:

$$\vec{p_{ij}}(t, t') = \vec{p_j}(t') - \vec{p_i}(t)\,, \tag{5.1}$$

$$\vec{v_{ij}}(t, t') = \vec{v_j}(t') - \vec{v_i}(t)\,, \tag{5.2}$$

On adapte en conséquent l'équation (1.3) permettant de calculer $r_{ij}(t)$, qui est la dérivée de la norme $\|\vec{p_{ij}}(t)\|$, ainsi que l'équation (1.4), donnant la valeur de la convergence C_{ij} :

$$\begin{aligned} r_{ij}(t, t') &= \frac{\vec{p_{ij}}(t, t') \cdot \vec{v_{ij}}(t, t')}{\|\vec{p_{ij}}(t, t')\|} \\ &= \|\vec{v_{ij}}(t, t')\| \cos(\vec{p_{ij}}(t, t'), \vec{v_{ij}}(t, t'))\,, \end{aligned} \tag{5.3}$$

La convergence associée est alors calculée par la formule :

$$C_{ij}(t, t') = \begin{cases} -r_{ij}(t, t') & \text{si } r_{ij}(t, t') < 0 \text{ et } \|\vec{p_{ij}}(t, t')\| < D_{\max}, \\ 0 & \text{sinon.} \end{cases} \tag{5.4}$$

Le processus est illustré par la figure 5.2. La convergence est calculée séparément entre chaque position $\vec{p_i}(t)$ et toutes les positions $p_j(t)$ appartenant à la fenêtre spatiale $D_{\max}$ et temporelle Δt. Ensuite, la valeur robuste de la convergence $\tilde{C_{ij}}(t)$ entre les trajectoires i et j à l'instant t est la valeur maximale de la convergence entre chaque vecteur $\vec{v_i}(t)$ et tous les vecteurs $\vec{v_j}(t')$, avec $t' \in [t - \Delta t, t + \Delta t]$, l'utilisation de la valeur maximale étant l'approche la plus conservatrice :

$$\tilde{C_{ij}}(t) = \max\{C_{ij}(t, t') | t' \in [t - \Delta t, t + \Delta t]\}\,. \tag{5.5}$$

Enfin, la convergence associée à l'avion i, $\tilde{C_i}(t)$, est calculée comme la somme des convergences entre l'avion i et tous ses voisins j à l'instant t :

$$\tilde{C_i}(t) = \sum_{i \neq j} \tilde{C_{ij}}(t)\,. \tag{5.6}$$

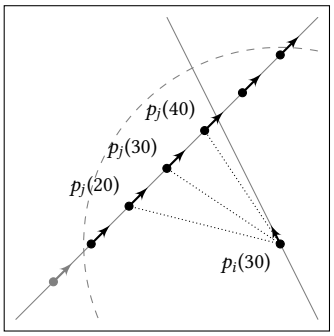


FIGURE 5.2 – Calcul de la convergence $\tilde{C_{ij}}(30)$, dont la position est $\vec{p_i}(30)$. Un premier filtrage permet de sélectionner les $\vec{p_j}(t)$ appartenant à la fenêtre spatiale $D_{\max}$, représentée par le cercle en pointillés, et à la fenêtre temporelle Δt de ±10 s. La convergence est calculée séparément entre $\vec{p_i}(30)$ et chaque $\vec{p_j}(t)$ restant ($p_j(20)$, $p_j(30)$ et $p_j(40)$), pour trouver la valeur maximale.

De cette manière, la convergence créée par trois avions convergeant est le double de la convergence créée par deux avions.

$\tilde{C_i}(t)$ est une valeur de convergence limitée à la position $\vec{p_i}(t)$. Pour calculer une carte de convergence C, toutes ces valeurs isolées doivent être transformées en une grille 2D donc chaque cellule correspond à une zone spatiale de 5 NM sur 5 NM. La valeur aux coordonnées (k, l) est alors calculée comme la valeur maximale parmi les valeurs $\tilde{C_i}(t)$ localisées dans les cellules de coordonnées (m, n) entourant la cellule (k, l) :

$$\begin{aligned} C_{k,l} = \max\{\tilde{C_i}(t) | \vec{p_i}(t) \in (m, n), \\ m \in \{k-1, k, k+1\}, \\ n \in \{l-1, l, l+1\}\} \end{aligned} \tag{5.7}$$

Enfin, pour déterminer les zones où créer les réseaux de routes, l'agent d'agrégation de trajectoires détecte les cases de la grille dont la valeur dépasse le seuil de 500 m/s. Cette valeur correspond au taux de convergence de deux avions de cap opposé allant tous deux à 487 kt. Elle peut être ajustée aux préférences de l'utilisateur. Les cases adjacentes dépassant ce seuil sont agrégé pour former des clusters. Le plus petit cercle entourant ce cluster, calculé par la méthode décrite par Gärtner (1999) [Gär99], définit la zone où le réseau de routes sera créé. Si deux cercles entrent en intersection, les deux clusters sont agrégés pour créer un cercle plus grand. Ce processus se répète tant que des cercles se chevauchent. Ce processus est illustré par la figure 5.3.

Les cartes partielles et la carte globale de convergence sont stockées dans des structures de données appelées HashMaps, dont le fonctionnement est expliqué dans l'annexe A.

L'algorithme de calcul de cartes de complexité est intégré au système multi-agents Plectre v2. L'implémentation centralisée de l'algorithme Yard bénéficie de son intégration dans ce SMA en déléguant les premières étapes du calcul de la carte de convergence aux avions. Grâce aux messages ADS-C, l'avion i connaît les trajectoires de ses voisins. Il peut calculer la convergence $\tilde{C_i}(t)$ le long de sa trajectoire en fonction des autres trajectoires (avions j). Les valeurs $\tilde{C_i}(t)$ lui permettent de calculer une carte partielle de complexité aux environs de sa trajectoire. Cette carte est insérée dans le message « ADS-C » et reçue par l'agent d'agrégation de trajectoires, comme l'indique

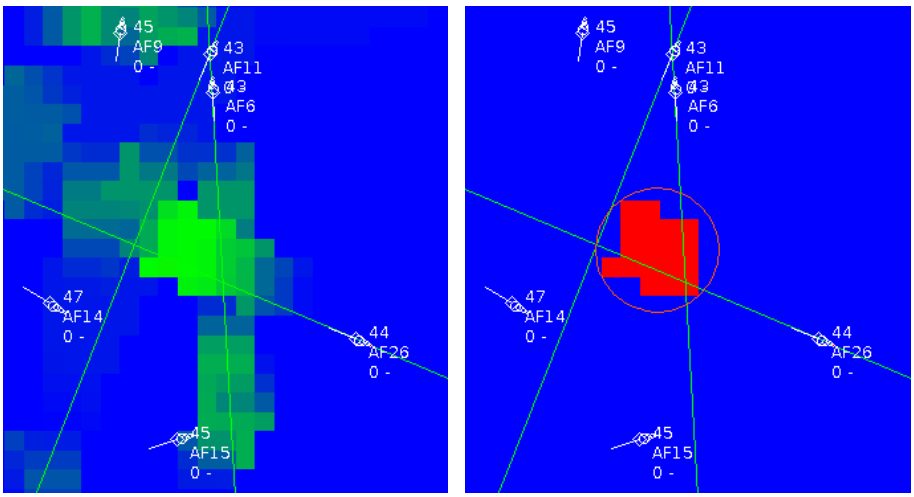


FIGURE 5.3 – À gauche : carte de convergence, les valeurs les plus basses en bleu, les plus élevées en vert. À droite : les cases dont la convergence dépasse un seuil apparaissent en rouge, et la zone où le réseau de routes sera créé est entourée d'un cercle.

la figure 5.1. L'agent agrège alors ces cartes en prenant la valeur maximale à chaque coordonnée pour obtenir la carte de convergence finale.

5.1.2 Création du réseau de routes

Structurer le trafic aérien sur un réseau de routes est un moyen d'en réduire la complexité au prix d'une perte d'efficacité des vols. Pour créer un réseau de routes, une méthode d'agrégation de trajectoires peut être utilisée pour faire émerger des flux de trajectoires. Ici, l'agrégation est utilisée pour créer un réseau de routes temporaires à partir d'un ensemble de trajectoires, de manière à obliger les avions à suivre ce réseau de routes.

Comme décrit dans la section 1.2.3.5, plusieurs méthodes d'agrégation de trajectoires existent, principalement utilisées pour la visualisation de données [GKH11 ; MHF12] ou la classification de trajectoires [PN16]. Les méthodes basées sur la densité du trafic, les plus simples, ne garantissent pas que les trajectoires agrégées soient compatibles avec les performances des avions, puisqu'elles créent des trajectoires avec une courbure excessive. Puechmorel et Nicol [PN15 ; PN16] proposent une méthode d'agrégation de trajectoires basée sur la minimisation de l'entropie de la densité du trafic. Celle-ci produit des trajectoires géométriquement correctes, qui ne possèdent pas de courbure excessive.

Cette méthode est décrite en détail dans la section 1.2.3.5. En résumé, l'entropie est la mesure du désordre d'un système. L'entropie d'une densité de probabilité $d(\vec{x})$, pour $\vec{x}$ appartenant à l'espace Ω, s'écrit :

$$H(\Omega) = -\int_{\Omega} d(\vec{x}) \log(d(\vec{x}))\, \mathrm{d}\vec{x}\,. \tag{5.8}$$

Elle est minimale quand la densité est maximisée localement, c'est-à-dire quand toutes les trajectoires passent au même endroit. L'entropie est maximale quand la densité

est uniforme, c'est-à-dire quand les trajectoires sont réparties uniformément dans l'espace aérien. Minimiser l'entropie d'un ensemble de trajectoires revient à agréger celles-ci.

5.1.2.1 Densité basée sur la position des avions

Les trajectoires sont représentées par les fonctions $\vec{p}_i(t)$ associant à chaque instant $t \in [0, 1]$ un vecteur position. La vitesse, notée $\vec{v}_i(t)$ est la dérivée de cette fonction, et l'accélération $\vec{a}_i(t)$ en est la dérivée seconde.

L'algorithme Yard échantillonne les trajectoires $\vec{p}_i(t)$ avec un pas de temps de 40 s, puis calcule la densité de probabilité grâce à un estimateur par noyau dont la fenêtre h est fixée à 5 NM, ce qui est à peu près la distance parcourue par un avion en 40 s en vitesse de croisière. Cela permet d'obtenir une représentation continue de la trajectoire dans la densité de probabilité (pas d'interruption de la densité entre deux pas de temps). Un exemple de carte de densité est représenté sur la figure 5.4. L'estimation par noyau est pondérée par la vitesse de l'avion à chaque pas de temps, de manière à ne pas sur-représenter les portions de la trajectoire où l'avion est plus lent (approche d'un aéroport), et où il parcourt moins de distance entre chaque pas de temps.

L'agrégation des trajectoires est effectuée par descente du gradient de l'entropie. L'équation (1.31) est la dérivée de l'entropie $\vec{\eta}_i(t)$ à la position $\vec{p}_i(t)$, et définit le mouvement à appliquer en chaque point de la courbe $\vec{p}_i(t)$, à l'exception du premier et du dernier point, qui ne sont pas déplacés. Un coefficient est appliqué pour réduire le déplacement et éviter une divergence causée par un déplacement de trop grande amplitude. La figure 5.4 montre une étape de la descente de gradient.

Les deux dernières étapes, calcul de la carte de densité et déplacement des points, sont exécutées itérativement jusqu'à atteindre une des conditions d'arrêt. La première condition d'arrêt est la constatation de la convergence de l'algorithme, quand le déplacement le plus important d'un waypoint durant l'itération ne dépasse pas le quart du déplacement maximum observé durant le processus complet. La deuxième condition est que le nombre d'itérations atteint son maximum, soit 100 000 itérations.

La figure 5.5 illustre le comportement de l'algorithme. Chaque waypoint est déplacé selon le vecteur normal à la trajectoire. Dans les zones où deux trajectoires sont proches, les waypoints sont attirés vers l'autre trajectoire. Ailleurs, les waypoints se déplacent vers l'intérieur de la courbure, ce qui rend la trajectoire rectiligne.

L'algorithme d'agrégation de trajectoires basée sur la position des avions étant relativement lent, il a été décidé de porter les calculs sur GPU en OpenCL pour les effectuer sur carte graphique. Les détails sont décrits dans l'annexe B.

5.1.2.2 Densité basée sur la position et le cap des avions

Calculer la densité à partir de la position des avions uniquement comporte un inconvénient de taille : l'agrégation de deux flux de trajectoires formant un angle obtus formera un flux où les avions volent en sens opposé. Ce comportement est illustré par la figure 5.9. Ainsi, au lieu de simplifier le trafic en le structurant, l'algorithme peut dans ces conditions le complexifier en créant des conflits en face-à-face.

Comme expliqué dans [NP16], la méthode de groupement de trajectoires par minimisation de l'entropie peut être améliorée pour prendre en compte la direction des flux d'avions. Il suffit de modifier la manière de calculer la carte de densité. Au lieu de calculer la densité en deux dimensions (latitude et longitude), il faut la calculer en

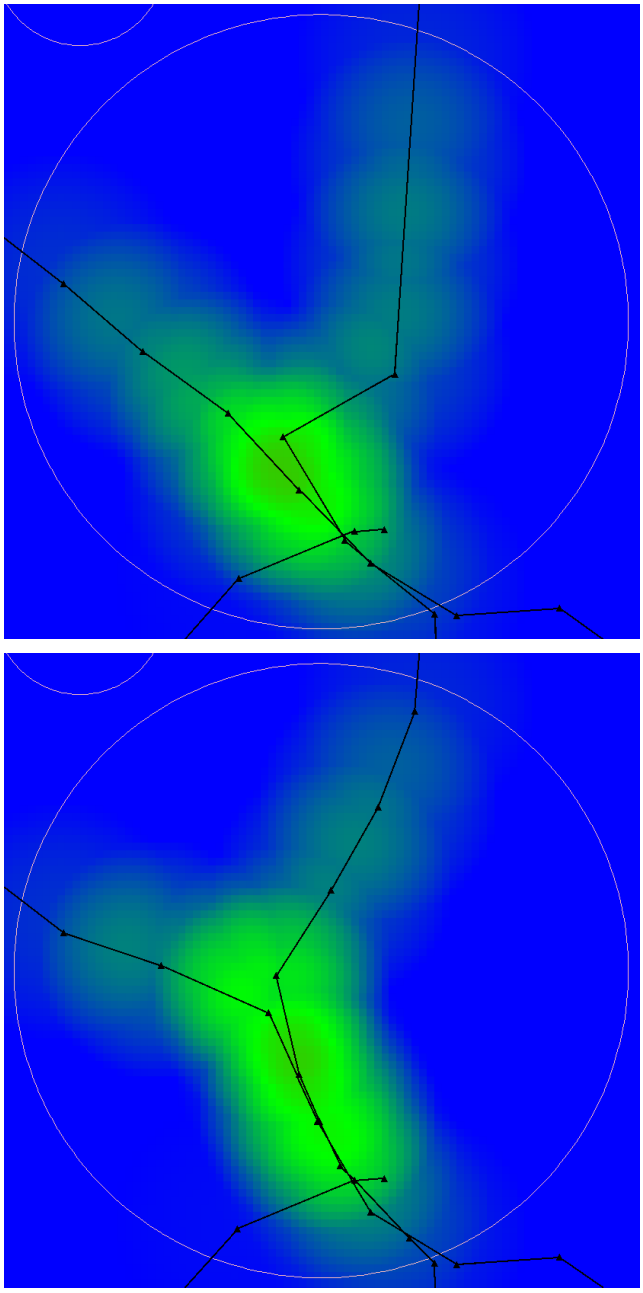


FIGURE 5.4 – Densité calculée au cours du processus d'agrégation de trajectoires. Les deux états sont séparés par 10 itérations, le processus complet pouvant en compter jusqu'à 100 000. Les trajectoires sont en noir, les waypoints à déplacer sont les triangles. Les zones de faible densité sont en bleu, la densité élevée en vert. Avant, la valeur de l'entropie est de 3,515 8. Après, l'entropie est de 3,470 0.

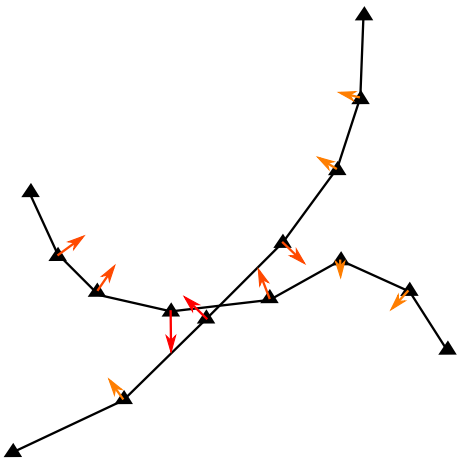


FIGURE 5.5 – L'agrégation de trajectoires par minimisation de l'entropie minimise également la courbure, ce qui a tendance à les rendre rectiligne.

quatre dimensions : latitude, longitude, cap et vélocité. Le cap permet de calculer une densité de probabilité circulaire : les avions volant dans le même sens seront attirés, et ceux de sens contraire repoussés, formant deux groupes de trajectoires.

L'algorithme Yard est basé sur les premiers travaux de Puechmorel et Nicol [PN15], puisque l'implémentation de cet algorithme est antérieure à la publication de [PN16 ; NP16]. À la suite d'une suggestion des auteurs de ces articles datant de 2015, la carte de densité calculée par Yard est enrichie d'une troisième dimension permettant de prendre en compte le cap θ des avions, qui est représenté grâce à une distribution de probabilité normale circulaire (loi de von Mises). Cette représentation est plus simple que celle proposée dans [NP16], puisqu'elle ne prend pas en compte la vélocité des avions. Elle donne néanmoins des résultats valides, comme indiqué dans la suite de ce chapitre (section 5.1.3).

5.1.2.3 Implémentation de l'agrégation de routes

5.1.2.3.1 Implémentation sous forme de système multi-agents L'agrégation de trajectoires est effectuée par un algorithme implémenté sous forme d'un système multi-agents. Cela permet de réutiliser le code de Plectre v2, qui exécute le cycle de vie des agents de manière parallèle, et dont l'expérience montre qu'il ne contient pas de bugs.

Par ailleurs, le concept de plusieurs systèmes multi-agents imbriqués paraît intéressant. En effet, le système multi-agents Plectre v2 contient un agent d'agrégation de trajectoires dont le processus de décision est lui-même un système multi-agents.

Dans le système multi-agents d'agrégation de trajectoires, les agents sont les waypoints composant la trajectoire, issus de son échantillonnage. Ceux-ci évoluent dans un environnement constitué par la carte de densité, et ont accès à celle-ci : « ils sont capables de la percevoir ».

Ils échangent leur position par message avec leurs voisins immédiats le long de la trajectoire à laquelle ils appartiennent. Connaître la position des waypoints voisins est nécessaire pour calculer le vecteur normal à la trajectoire, ainsi que la vitesse de l'avion en ce point.

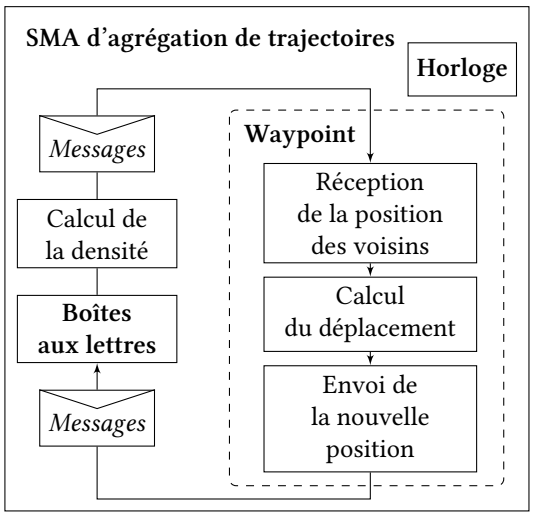


FIGURE 5.6 – Cycle de vie du système multi-agents d'agrégation de trajectoires.

Il s'agit d'agents réactifs, dont les actions obéissent à des règles qui tiennent compte uniquement de la carte de densité dans laquelle ils évoluent et de la position de leurs voisins.

Comme le montre la figure 5.6, le cycle de vie de ce système multi-agents alterne l'actualisation de la carte de densité et l'exécution d'une étape du cycle de vie de l'agent.

5.1.2.3.2 Calcul de la carte de densité Le calcul de l'agrégation de trajectoires peut être fortement parallélisé. Calculer le déplacement d'un waypoint nécessite uniquement de connaître la position courante du waypoint précédent et du suivant, ainsi que la carte de densité (à ne pas confondre avec la carte de convergence utilisée pour *détecter* les zones dans lesquelles agréger les trajectoires). Cette parallélisation est implémentée par le code commun aux systèmes multi-agents.

Le calcul de la carte de densité est également parallélisable. Un noyau de coordonnées $(x, y, \theta) = (0, 0, 0)$ est précalculé. La fenêtre h du noyau d'Epanechnikov est fixée à 30 NM. Le noyau est stocké dans une grille dont la résolution spatiale est de 1,875 NM, le noyau faisant 32 cellules de coté. La résolution angulaire est de $\pi/4$, ce qui est suffisant pour assurer la ségrégation de trajectoires de cap différent. Les résultats montrent que cette résolution spatiale et angulaire donne de bons résultats.

Une fois que ce noyau initial est calculé, une opération de MapReduce permet d'obtenir la carte finale. Le modèle MapReduce a été inventé par Google [DG10] pour simplifier la parallélisation des calculs sur de grandes quantités de données. Ce modèle est très utilisé dans le domaine du big data, mais des implémentations sont proposées par certains langages de programmation, comme Java 8 et son API Stream [Ora15].

Le modèle MapReduce s'appuie sur deux opérations empruntées à la programmation fonctionnelle pour transformer une collection d'éléments. L'opération *map* applique une transformation sur chacun des éléments de la collection. L'opération *reduce* agrège deux à deux les éléments transformés jusqu'à obtenir le résultat final. Pour implémenter une opération s'appuyant sur le modèle MapReduce, le dévelop-

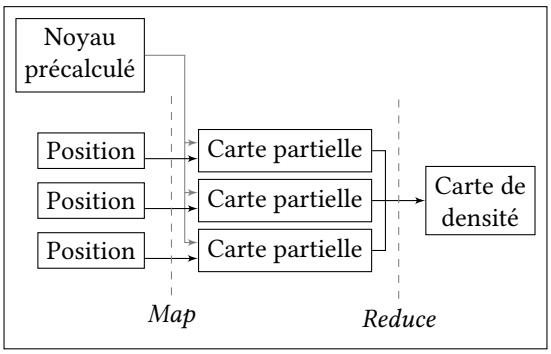


FIGURE 5.7 – Calcul de la carte de densité par MapReduce.

peur doit uniquement fournir ces deux fonctions à la bibliothèque implémentant MapReduce, qui s'occupe de la parallélisation.

Le fonctionnement du calcul de la carte de densité à partir d'une collection de positions est indiqué sur la figure 5.7. L'opération *map* prend en entrée une position et le noyau précalculé, et calcule une carte partielle de densité contenant uniquement le noyau centré sur cette position. La fonction *map* effectue deux opérations :

1. Une translation est appliquée à chaque cellule du noyau précalculé, de manière à la déplacer à sa position spatiale ;
2. La valeur de densité de chaque cellule est pondérée en fonction de la vitesse de l'avion à cette position, conformément à l'équation (1.30).

Puis l'opération *reduce* agrège toutes ces cartes partielles. Si deux cartes partielles contiennent des cellules de coordonnées identiques, leur valeur est additionnée.

Pour économiser de l'espace mémoire et du temps de calcul, les cartes partielles et la carte finale sont stockées sous forme de HashMap, comme expliqué dans l'annexe A.

5.1.2.3.3 Envoi des nouveaux plans de vols aux avions Une fois que le système multi-agents a convergé, les positions résultants des waypoints sont envoyées aux avions, qui intègrent ces routes en tant que nouvelle portion de leur plan de vol.

L'algorithme d'agrégation de trajectoires est exécuté toutes les 40 secondes, dans le cas où une zone complexe a été détectée grâce à la métrique de convergence. Comme le montre la figure 5.1, Yard fonctionne en parallèle de l'algorithme de résolution de conflits par régulation de vitesse décrit dans le chapitre 4.

La résolution de conflits peut être effectuée soit par les avions eux-mêmes, soit par l'agent de régulation de vitesses basé sur la méthode du recuit simulé. Dans le premier cas, les avions intègrent le nouveau plan de vol, puis régulent leur vitesse. Dans le deuxième cas, les avions envoient leurs trajectoires à l'agent de régulation de vitesse et à l'agent d'agrégation de trajectoires. Dans ce cas, à l'itération suivante, l'avion reçoit à la fois son nouveau plan de vol et sa nouvelle planification de vitesse. Il calcule alors sa nouvelle trajectoire, et la renvoie aux deux agents de régulation centralisée. Ceux-ci actualisent leurs décisions si besoin. Le système multi-agents converge finalement vers une solution où les trajectoires sont structurées dans les dimensions spatiales et temporelle.

5.1.3 Résultats

Plusieurs scénarios de test ont été développés pour vérifier le fonctionnement de la méthode. Dans les deux premiers, le cap n'est pas pris en compte dans le calcul de la densité. Dans le troisième, le cap est pris en compte.

Dans le premier scénario, visible dans la figure 5.8, trois routes parallèles de 100 NM de long espacées de 5 NM sont croisées par une quatrième ce qui provoque un accroissement local de la convergence. L'intersection de ces trajectoires est détectée comme une zone à réguler, entourée par un cercle dans la figure 5.8. Les trajectoires sont alors agrégées dans cette zone par l'algorithme de minimisation de l'entropie. L'entropie des trajectoires résultantes est plus faible dans la zone circulaire que dans la situation initiale. La densité d'avions augmente dans cette zone car toutes les trajectoires sont agrégées. Dans le même temps, la courbure est minimisée : les trajectoires sont rectilignes et aucune courbure excessive n'apparaît.

Dans le deuxième scénario, décrit par la figure 5.9, quatre avions convergent vers le point central. Le processus d'agrégation est présenté dans la figure 5.10. Deux avions volent du nord ouest vers le sud est, le troisième vole du nord au sud, et le quatrième vient du sud ouest et s'arrête au centre. Le cap n'est pas pris en compte dans le calcul de la densité. Le résultat est que la quatrième trajectoire est agrégée avec les autres, mais l'avion vole en sens contraire, vers le nord, ce qui provoque des conflits avec les autres.

Le troisième scénario est identique au deuxième, mais le processus d'agrégation tient compte du cap. Le résultat est visible dans la figure 5.11. Le quatrième avion, dont la nouvelle trajectoire était de sens contraire aux autres, n'est plus agrégée, aucun conflit en face-à-face n'apparaît.

5.1.4 Conclusion

Cette méthode donne des résultats prometteurs pour l'élaboration de réseaux de routes locaux temporaires. Les trajectoires sont correctement agrégées pour former des routes plus simples, en tenant compte de l'orientation des flux d'avions.

Cependant, dans cette implémentation, les décisions sont centralisées au niveau d'un seul agent de régulation intégré au système multi-agents Plectre v2. Un des principaux avantages des SMA est alors perdu : l'algorithme repose sur une seule entité pour prendre toutes les décisions. Décentraliser ce processus permettrait d'améliorer la résilience du système et la parallélisation des calculs.

Le calcul de la carte de convergence est déjà partiellement distribué. Chaque avion calcule une carte partielle le long de sa trajectoire, et envoie celle-ci à l'agent d'agrégation de trajectoires. Celui-ci agrège ces cartes partielles pour obtenir la carte de convergence globale, afin de déterminer les zones où créer les réseaux de routes.

La suite de ce chapitre propose une manière de décentraliser l'algorithme Yard pour que les avions agrègent eux-mêmes leurs trajectoires pour former le réseau de routes.

5.2 Implémentation décentralisée au niveau des avions

5.2.1 Méthode

Comme dans l'implémentation précédente, chaque avion calcule une carte partielle de convergence le long de sa trajectoire à partir des trajectoires des voisins reçues

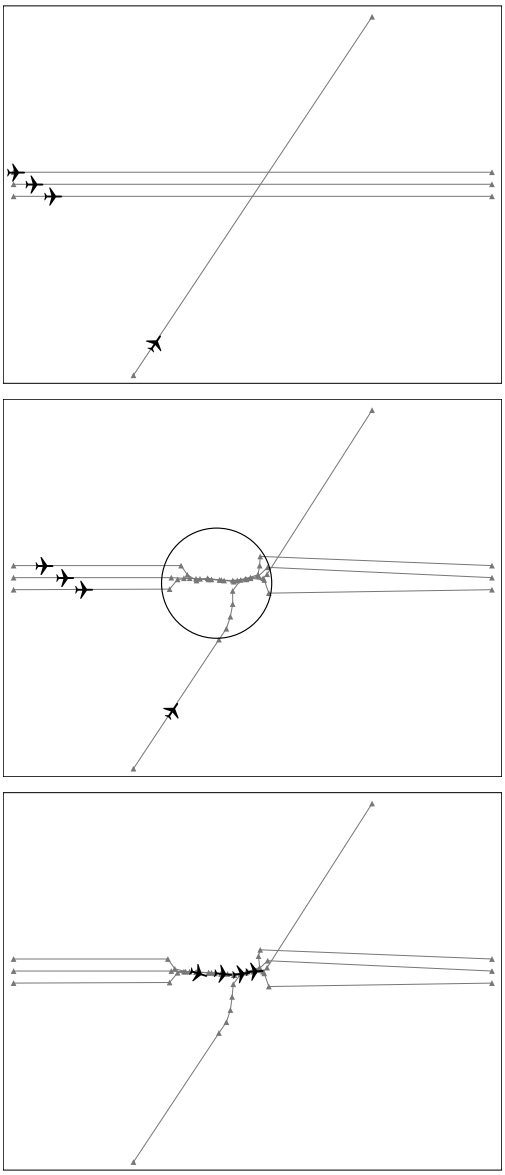


FIGURE 5.8 – Résultat de l'algorithme Yard appliqué au premier scénario. En haut : dans le scénario initial, les trajectoires parallèles longues de 100 NM et espacées de 5 NM sont croisées par une quatrième. Au centre : les plans de vol sont agrégés à l'intérieur de la zone circulaire. En bas : les avions suivent leurs nouveaux plans de vol.

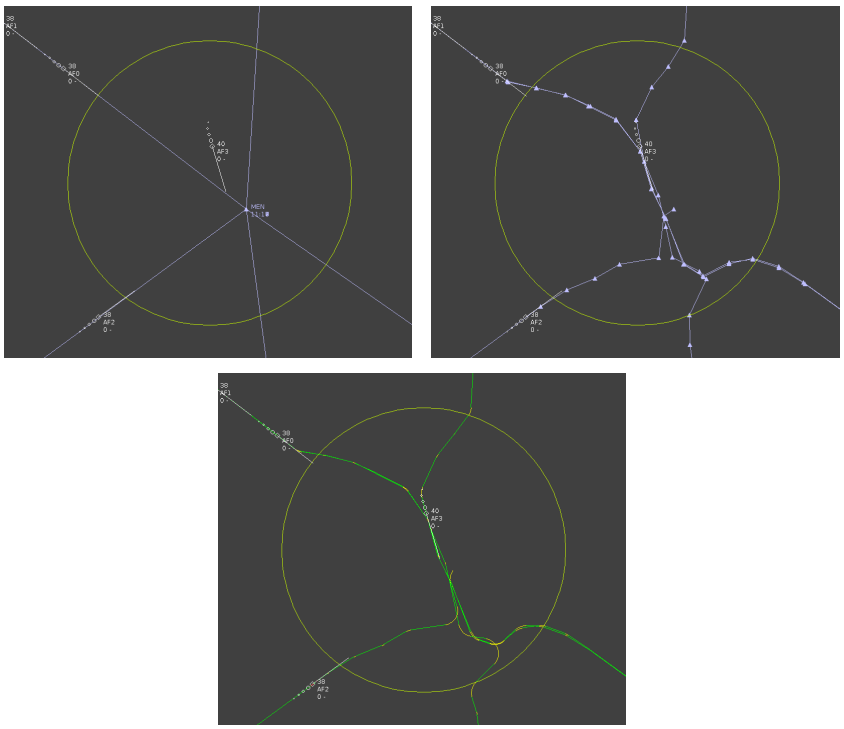


FIGURE 5.9 – Minimisation de l'entropie sans prise en compte du cap. En haut à gauche, situation de départ. En haut à droite, plans de vol après agrégation : les avions se retrouve en conflit en face-à-face. En bas les trajectoires issues des plans de vol.

par ADS-C.

Les messages ADS-C contiennent également les cartes de convergence partielles calculées par les voisins. L'avion agrège alors sa propre carte partielle avec celles-ci pour obtenir la carte globale de convergence.

À partir de cette carte de convergence, une carte de zones à réguler est calculée. Il s'agit d'une grille de même résolution spatiale, dont chaque cellule contient une valeur binaire spécifiant si la convergence dépasse un seuil prédéterminé. Ce seuil est identique à celui utilisé dans la version centralisée de l'algorithme, soit 500 m/s. L'avion applique alors l'algorithme de minimisation de densité sur les points de sa propre trajectoire situés à l'intérieur de ces cellules.

Pour calculer les cartes de densité, l'avion collecte les positions le long de la trajectoire des voisins proches d'une de ses propres positions, c'est-à-dire dont la distance spatiale est inférieure à $D_{\max}$ et la distance temporelle inférieure à Δt. Ces deux constantes sont utilisées dans le calcul centralisé de la carte de convergence.

L'ensemble de ces points permet de calculer la carte de densité utilisée par la méthode d'agrégation de trajectoires. Dans la version centralisée de Yard, tous les waypoints se déplacent en même temps. Dans la version décentralisée, pour reproduire ce comportement, les avions doivent fréquemment s'échanger leurs trajectoires mises à jour. L'avion effectue donc dix itérations de cet algorithme avant d'envoyer sa nouvelle trajectoire aux autres avions.

Si au cours des itérations de l'algorithme, un des waypoints de la trajectoire sort

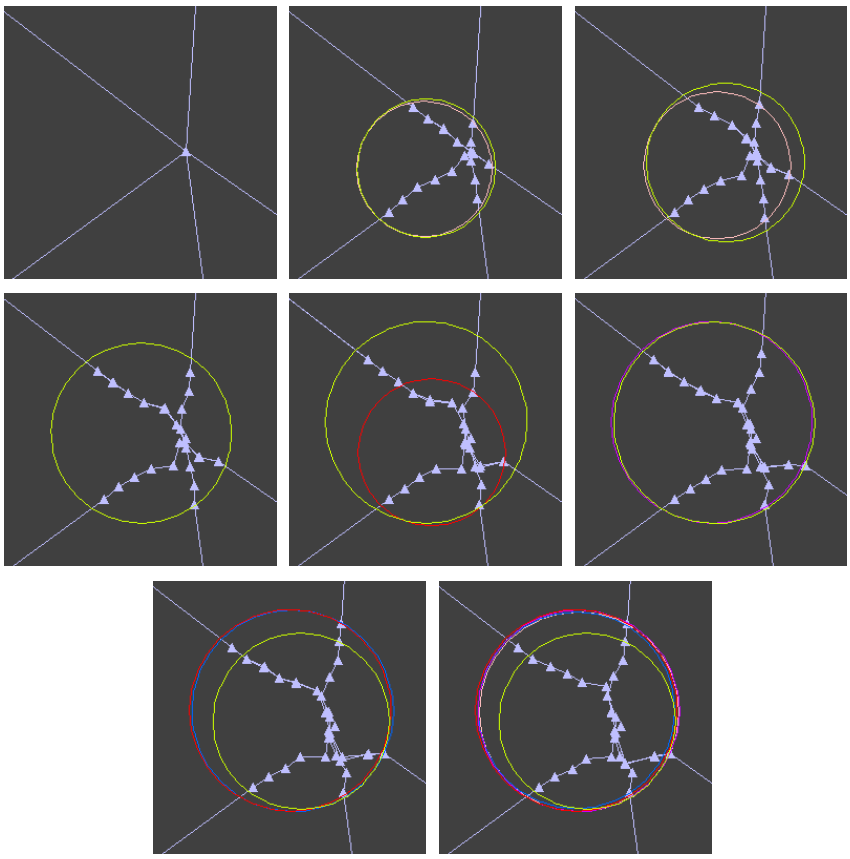


FIGURE 5.10 – Agrégation progressive des trajectoires pour former une route.

des cellules de la carte binaire des zones à réguler, la cellule contenant la nouvelle position du waypoint est à son tour marquée comme étant à réguler. Aucune cellule n'est jamais marquée comme n'étant plus à réguler.

L'algorithme d'agrégation des trajectoires est exécuté en même temps que l'algorithme de résolution de conflits, c'est-à-dire toutes les 8 secondes. La carte binaire spécifiant les zones à réguler est préservée par l'avion entre deux exécutions de l'algorithme. Cela permet de continuer à agréger les trajectoires après que celles-ci se soient suffisamment rapprochées pour que la convergence passe en dessous du seul de déclenchement de l'algorithme.

5.2.2 Limites

Cette méthode comporte une limite qui l'empêche d'obtenir d'aussi bons résultats que la version centralisée. L'algorithme d'agrégation de trajectoires est exécuté toutes les 8 secondes, et effectue 10 itérations, ce qui fait 50 itérations toutes les 40 secondes. La version centralisée, elle, est exécutée toutes les 40 secondes, et effectue 100 000 itérations au maximum. Cela veut dire que la version décentralisée effectue jusqu'à 2 000 fois moins d'itérations dans le même intervalle de temps, et converge 2 000 fois moins vite.

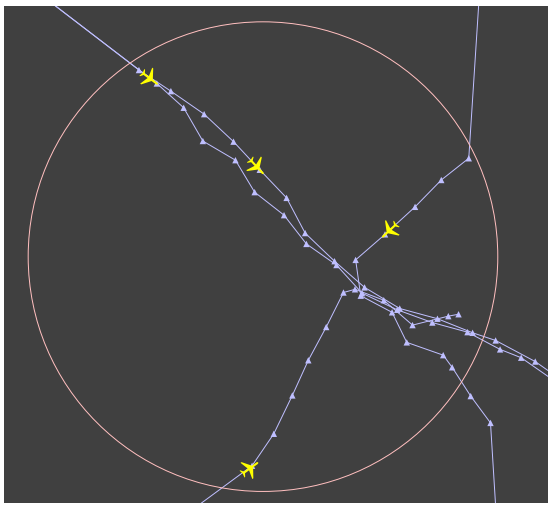


FIGURE 5.11 – Minimisation de l'entropie du scénario initial de la figure 5.9 avec prise en compte du cap. La trajectoire de l'avion venant du sud ouest est correctement agrégée avec les autres, les trajectoires formant un flux unique après le point de croisement : pas de conflit en face-à-face.

En pratique, la version centralisée de l'algorithme effectue entre 600 et 1 000 itérations en moyenne. L'algorithme décentralisé fait donc 20 fois moins d'itérations que l'algorithme centralisé.

Comme dans Plectre v2, l'ordre dans lequel les avions prennent les décisions de déformation des trajectoires varie entre les exécutions de la simulation. Cela n'aurait pas d'importance si l'algorithme décentralisé pouvait itérer autant que la version centralisée : l'agrégation étant effectuée par une descente de gradient, les trajectoires finales seraient rectilignes et correctement agrégées. Mais puisque le nombre d'itérations est faible, il arrive parfois que l'algorithme n'ait pas le temps de converger, et que les trajectoires ne soient pas agrégées correctement.

L'ordre dans lequel les avions prennent leurs décisions peut également influer sur le résultat final. Même en supposant que le nombre d'itérations soit suffisant dans tous les cas pour que l'algorithme converge, le résultat obtenu par la descente de gradient est un optimum local. Le réseau de routes obtenu par deux exécutions de l'algorithme peuvent donc différer, les trajectoires pouvant ne pas être groupées de la même manière.

5.2.3 Résultats

Malgré le taux de convergence plus lent que la version centralisée de l'algorithme d'agrégation de trajectoires, la figure 5.12 montre que la version décentralisée permet d'obtenir des résultats similaires à ceux visible sur la figure 5.11. On peut cependant observer que dans la version décentralisée, les plans de vols résultants sont moins rectilignes, puisque le nombre moins important d'itérations ne permet pas de lisser les trajectoires autant que dans la version centralisée. Ce phénomène est visible pour le plan de vol de l'avion suivant une trajectoire nord-sud.

Cette non-linéarité pourrait être corrigée en lissant les trajectoires. Plusieurs

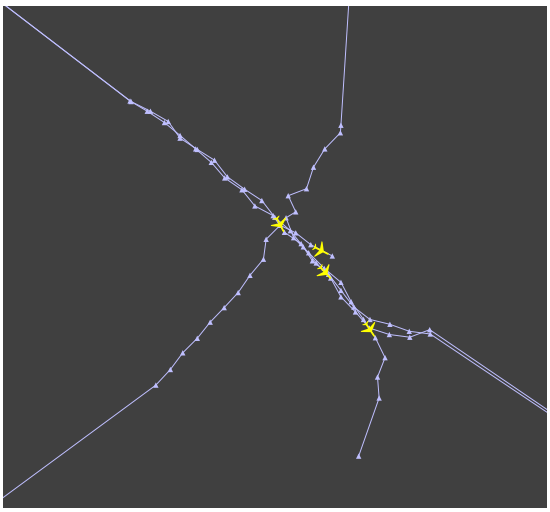


FIGURE 5.12 – Minimisation de l'entropie décentralisée, à partir du scénario initial de la figure 5.9. Les avions agrègent eux-mêmes leurs trajectoires pour minimiser l'entropie, en tenant compte de la direction des flux d'avions.

possibilités existent. Le plus simple est de rajouter une contrainte sur la courbure des plans de vol dans le processus de minimisation de l'entropie, durant l'étape où les points de la trajectoire sont déplacés. On peut également rajouter une étape de post-traitement pour lisser les plans de vol résultant de l'algorithme.

Il faut noter que le résultat de cet algorithme est un plan de vol, qui permet de calculer la trajectoire de l'avion. Comme le montre la figure 5.8, les trajectoires sont plus lisses que les plans de vol qui permettent de les calculer. En effet, l'avion suit son plan de vol du mieux possible, mais il est contraint par son taux de virage maximal. Ce taux, de 3°/s, définit la courbure maximale de la trajectoire.

Comme évoqué dans les chapitres précédents, implémenter un algorithme sous forme de système multi-agents permet de rendre celui-ci résilient aux perturbations. Pour vérifier que l'algorithme Yard décentralisé respecte cette propriété, le scénario présenté dans la figure 5.12 a été exécuté une nouvelle fois, mais en désactivant le processus de décision d'un avion. Le résultat est visible sur la figure 5.13. L'avion rouge ne participe pas à l'agrégation de trajectoires. Les autres avions prennent cela en compte et déforment leur propre trajectoire pour former un flux superposé à la trajectoire de l'avion rouge. Ainsi, le processus de minimisation de l'entropie fonctionne même si un des avions ne coopère pas.

5.2.4 Conclusion

L'algorithme Yard permet de faire émerger un réseau de routes à partir d'un ensemble de trajectoires. Les routes résultantes tiennent compte de la direction des flux d'avions, de manière à ne pas générer de conflits en face-à-face.

Dans ce chapitre, deux versions de cette méthode ont été proposés. La première version est un algorithme centralisé, qui repose sur un agent unique chargé de réguler l'ensemble du trafic. Cette architecture donne des résultats conformes aux attentes,

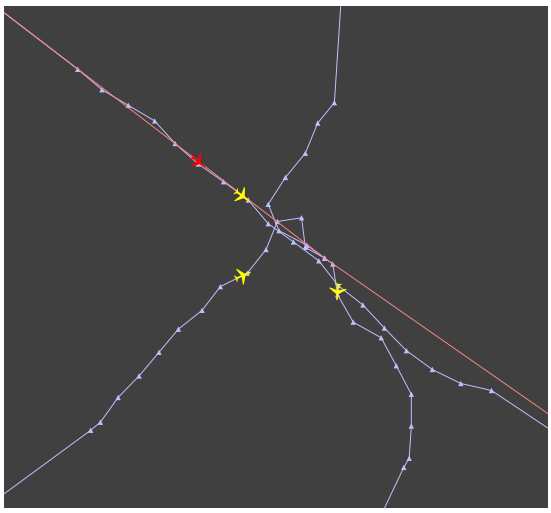


FIGURE 5.13 – Résilience du système pour la minimisation de l'entropie décentralisée. L'avion rouge ne coopère pas à l'agrégation des trajectoires. Les autres avions agrègent la leur sur celle de l'avion non-coopératif, de manière à réduire l'entropie.

puisqu'on observe l'émergence de routes concentrant les flux d'avions. Mais les décisions ne sont pas effectuées de manière collaborative par les avions.

La deuxième version de l'algorithme Yard, décrite dans la section 5.2, propose une manière de décentraliser les décisions au niveau des avions. De cette manière, aucun agent de régulation centralisé n'intervient dans la prise de décisions. Les calculs sont répartis entre les avions impliqués dans les pics locaux de complexité. Le système décentralisé est plus résilient que l'algorithme centralisé à des évènements tels que la non-coopération d'un avion au processus de décision. Si un avion ne coopère pas, les autres déforment leurs propres trajectoires en conséquent.

L'émergence du réseau de routes est lié à une diminution de l'entropie de la densité du trafic. L'entropie étant une mesure du désordre d'un système, et donc de sa complexité, la minimisation de l'entropie permet de diminuer la complexité du trafic aérien. La figure 5.4 montre que l'entropie mesurée diminue bien au cours des itérations.

Il est cependant plus difficile d'interpréter les résultats de Yard que ceux des algorithmes Plectre v1 et v2. Mesurer les performances de ces derniers peut être réalisé en comptant le nombre de conflits et le nombre d'accélérations des avions. Les résultats de Yard sont les plans de vol, ainsi que la valeur brute de l'entropie, qui est un réel positif difficile à interpréter.

Il est donc nécessaire de mesurer la complexité du trafic selon d'autres méthodes, pour étudier de manière plus fine le trafic régulé par Yard. Cette mesure pourrait être réalisée à l'aide de certains algorithmes présentés dans l'état de l'art, comme par exemple la mesure de complexité par exposant de Lyapunov décrite dans la section 1.2.3.3.3.

Les scénarios présentés dans ce chapitre font intervenir un faible nombre d'avions. L'algorithme doit encore être affiné pour fonctionner avec des scénarios plus complexes, impliquant plus d'avions. Cependant, l'application de l'algorithme décentralisé à ces premiers exemples montre que si les avions coopèrent au sein d'un système

multi-agent, le trafic aérien peut s'auto-structurer spatialement. L'algorithme décentralisé est résilient aux perturbations, et produit des résultats comparables à ceux de l'algorithme centralisé.

Conclusion et perspectives

Le but de cette thèse est de montrer que des algorithmes décentralisés au niveau des avions permettent au trafic aérien de s'auto-structurer dans les dimensions spatiales et temporelle.

Dans le système multi-agents Plectre v1, présenté dans le chapitre 3, les avions évoluent sur un réseau de routes et régulent leur vitesse de manière à résoudre les conflits aérien. Dans cet algorithme, le trafic est régulé dans la dimension temporelle.

Dans l'algorithme Plectre v2 du chapitre 4, les avions structurent également leurs trajectoires dans la dimension temporelle, mais en suivant des trajectoires quelconques, sans nécessité d'emprunter un réseau de routes prédéfini. Pour résoudre les conflits, les avions régulent leur vitesse et leur heure de départ.

L'algorithme Yard proposé dans le chapitre 5 est un enrichissement apporté au système multi-agents Plectre v2. Dans Yard, les avions structurent leurs trajectoires dans les dimensions spatiales. Ils font émerger un réseau de routes en agrégeant leurs trajectoires.

Les résultats retournés par les algorithmes Plectre v2 et Yard sont comparables à leur équivalents centralisés. Dans le cas de Plectre v2, le système multi-agents et le recuit simulé sont tous les deux capables de résoudre jusqu'à 100 % des conflits. Dans le cas de Yard, la version décentralisée crée des réseaux de routes similaires à ceux proposés par la version centralisée.

De plus, les systèmes multi-agents sont résilients aux perturbations telles que la non-coopération d'une partie des avions au processus de décision. Dans Plectre v1 et v2, les avions coopératifs prennent les autres comme contraintes dans leur processus de décision, et résolvent une grande partie des conflits. Dans Yard, les avions coopératifs agrègent leurs trajectoires de manière à ce que les trajectoires des avions non-coopératifs soient intégrées au réseau de route émergeant.

Ces travaux ont donné lieu à plusieurs publications nationales et internationales. Ils ont été présentés en France aux conférences ROADEF 2014 [Bre+14], 2015 [Bre+15] et 2016 [Bre+16b]. Deux posters ont été présentés durant les SESAR Innovation Days, en 2014 [BL14] et 2015 [BL15].

Ces travaux ont également été présentés dans deux conférences internationales en 2016. Les articles publiés à cette occasion ont tous les deux été primés. L'article publié à l'occasion de la conférence ICRAT [Bre+16a] a obtenu le *Best Paper Award for the Automation track*. L'article publié lors de la conférence DASC a obtenu le *Best of Session (SE/TFM) Award*, ainsi que le *Best Student Paper Award*.

Les algorithmes proposés dans cette thèse pourront faire l'objet d'améliorations. Il serait intéressant d'appliquer l'algorithme Plectre v2 à des situations de trafic plus variées, comme les vols transatlantiques [Rod+14], ou les Free Route Airspaces. Plectre v2 pourrait également être amélioré de plusieurs manières :

— Actuellement, l'altitude des avions est supposée constante. Or, dans le trafic réel, les avions changent de niveau de vol durant la phase de croisière. Leur altitude optimale dépend en effet de leur masse, qui diminue avec la consommation de carburant. Il serait intéressant que Plectre v2 prenne en compte l'altitude dans le calcul des trajectoires et la détection des conflits ;

— De la même manière, dans le trafic aérien réel, la vitesse de croisière d'un avion évolue au cours du vol. Celle-ci dépend du Cost Index, qui prend en compte le temps de vol et la consommation de carburant. Or, le temps de vol change en fonction, par exemple, des conditions météorologiques, comme la direction du vent, ou bien des manœuvres imposées aux avions pour résoudre les conflits, qui rallongent leur trajectoire. Dans Plectre v2, la vitesse de croisière est supposée constante. Il serait intéressant de prendre en compte des variations de la vitesse de croisière au cours du vol. Par exemple, si l'avion privilégie l'économie de carburant, il peut préférer ralentir au lieu d'accélérer pour résoudre un conflit. Et si l'arrivée de l'avion est retardée par la résolution d'un conflit, il peut choisir d'accélérer pour arriver à l'heure, ou bien ne rien faire pour économiser du carburant ;

— L'intervalle de vitesses admissibles dans Plectre v2 est fixé à [−6 %, +3 %] de la vitesse optimale. Or, cet intervalle n'est qu'un sous-ensemble communément admis de l'intervalle de vitesses défini par les performances de l'avion, qui dépend entre autre du modèle de l'avion et de son altitude. Cet intervalle doit également prendre en compte les préférences des compagnies aériennes. Plectre v2 pourrait également intégrer cette hypothèse ;

— Le modèle de trajectoires utilisé par Plectre v2 est relativement simple. Il considère que les variations de l'accélération sont instantanées (la dérivée de la vitesse est discontinue), et que l'avion ne change pas de vitesse dans les virages. Ainsi, la trajectoire résultante n'est dérivable qu'une seule fois en tout point. Ce modèle pourrait être remplacé par un modèle plus réaliste.

De son côté, l'algorithme Yard peut être amélioré de différentes manières :

— Il n'a actuellement été testé que sur des scénarios simples impliquant quelques avions. Il faudrait étendre les expérimentations à des scénarios plus complexes et réalistes ;

— Les trajectoires ne sont modifiées qu'en fonction de la complexité du trafic. Elles pourraient l'être en plus selon d'autres critères. Par exemple, l'algorithme pourrait prendre en compte des obstacles fixes, comme les zones militaires, ou mobiles, comme les orages ;

— Dans l'éventualité d'une extension 3D de la méthode, la question de la manière de gérer la troisième dimension reste ouverte. Faut-il gérer séparément les différents niveaux de vol, et créer des réseaux de routes différents selon l'altitude ? Ou bien faut-il créer un réseau de route commun à tous les niveaux de vol, alors que les avions séparés en altitude n'interagissent pas ?

Ces propositions d'enrichissement des algorithmes Plectre et Yard sont inclues dans des sujets de thèse de doctorat en cours d'élaboration au laboratoire de l'École Nationale de l'Aviation Civile (ÉNAC).

Pour conclure, ces algorithmes ont été conçus pour fonctionner en collaboration avec les contrôleurs aériens. Plectre v2 s'appuie sur les concepts du projet ERASMUS. Il diminue le nombre de conflits pour réduire la charge de travail des contrôleurs. ERASMUS a bénéficié de nombreuses études sur les facteurs humains. Celles-ci ont

validé le principe de la collaboration entre les contrôleurs et l'algorithme. Il faudrait vérifier que ces études s'appliquent également à Plectre v2, dont le fonctionnement, bien que similaire, diffère tout de même de l'algorithme utilisé par ERASMUS.

Dans le cas de l'algorithme Yard, le concept de Free Flight Area dans lequel il s'inscrit est moins bien défini. Selon la définition employée, les FFA sont gérés par les contrôleurs aériens, ou ne le sont pas. Yard a été conçu dans l'optique de réduire la complexité du trafic aérien au bénéfice des contrôleurs aériens, en structurant spatialement les trajectoires des avions.

Si les secteurs ne sont pas contrôlés par des moyens humains, l'utilité de Yard apparaît moins évidente. Par contre, si les FFA sont gérés par des contrôleurs aériens, il reste encore à mener des études pour vérifier que les routes produites par l'algorithme Yard sont en adéquation avec les besoins des contrôleurs aériens.

Cette thèse présente un système multi-agents permettant de déléguer à des avions coopératifs la résolution de conflits et la création de réseaux de routes locaux. Au delà de ces deux exemples, il est envisageable d'utiliser les SMA pour résoudre une grande variété de problèmes. Dans le domaine de l'ATM, les décisions tendent à être déléguées à des systèmes informatiques : équilibrage de charge entre secteurs ou routes aériennes (Network Manager), élaboration de plans de vol (outils propres à chaque compagnie aérienne), détection de conflits aériens (ERATO), détection de risque de collision (TCAS), etc. Actuellement, ces systèmes d'aide à la décision ont pour tâche d'assister les contrôleurs et les pilotes. Dans un futur plus lointain, ces systèmes devront prendre une partie, voire la totalité, des décisions permettant d'assurer la sécurité et la fluidité du trafic.

Ces systèmes fonctionnent actuellement de manière isolée : par exemple, le Network Manager ne coopère pas avec l'avion, la compagnie aérienne ou le centre de contrôle pour prendre une décision. Les systèmes multi-agents sont un cadre qui permettraient de faire communiquer et collaborer l'ensemble de ces systèmes isolés. Des algorithmes ont déjà été décrits dans la littérature scientifique, impliquant des agents avions, centres de contrôle, ou compagnies aériennes.

Certaines compagnies aériennes acceptent désormais de collaborer avec les centres de contrôle pour ajuster leurs plans de vol en fonction du trafic du jour [Hur+16]. On peut imaginer un système multi-agents qui serait utilisé conjointement par les compagnies aériennes et les autorités de contrôle du trafic pour élaborer les plans de vol, qui équilibrerait la demande en fonction de la capacité des secteurs aériens et du réseau de routes.

L'algorithme Yard montre qu'il est possible de créer des réseaux de routes. De manière plus large, on peut envisager un SMA permettant de structurer l'espace aérien, en incluant les secteurs et le réseau de routes. Des agents *secteurs* collaboreraient avec des agents *routes* (ou *waypoints*, tels qu'ils existent dans Yard) et avec des agents *avions*. Ce SMA permettrait d'adapter le découpage des secteurs et le tracé du réseau de routes à la demande du trafic aérien.

Plus généralement, il serait possible de faire communiquer l'ensemble des systèmes informatiques impliqués dans le trafic aérien (aéroports, centres de contrôle, compagnies aériennes, avions) au sein d'un système multi-agents, ou bien au sein d'un système de systèmes : plusieurs systèmes multi-agents coopérant au sein d'un SMA plus grand, à la manière du SMA Yard inclus dans le SMA Plectre.

Tout système complexe peut être étudié en le divisant en éléments plus simples, qui peuvent à leur tour découpés jusqu'à obtenir des éléments dont le comportement peut être décrit simplement (principe de l'analyse). Il est également nécessaire d'étudier les interactions entre les éléments du système (principe de la systémique).

Le cadre de travail proposé par les systèmes multi-agents permet de concevoir un algorithme permettant de contrôler un système complexe, comme le trafic aérien, en élaborant les règles de comportement et d'interaction des éléments le constituant. Le comportement de chaque agent peut être ajusté sans devoir modifier la totalité du système, ce qui rend le développement d'un SMA plus flexible.

Dans un système centralisé, basé sur un algorithme d'optimisation globale, la difficulté de conception réside dans la prise en compte de la combinatoire, c'est-à-dire de l'ensemble des configurations possibles du système.

Dans un SMA, les décisions sont déléguées aux agents. Les algorithmes à concevoir sont donc plus simples de ce point de vue, puisque chacun régit le comportement d'une entité unique. La combinatoire de chaque algorithme est moins élevée, puisque l'ensemble des configurations possibles d'un agent est plus faible que l'ensemble des configurations possibles du système. Contrairement aux systèmes centralisés, la difficulté du développement d'un SMA réside dans la gestion des interactions entre les agents.

Pour schématiser, dans un système centralisé, les décisions étant prises par une entité unique, celle-ci doit disposer d'une capacité de calcul dont l'importance dépend de la complexité du système géré. Pour un système complexe, il faut disposer d'un serveur puissant, voire d'un datacenter. Dans un SMA, les décisions étant prises par chaque entité du système, la capacité de calcul nécessaire est nettement moins élevée. Par exemple, certains travaux de recherche visent à créer des flottes de drones coopérant pour atteindre des objectifs [Qua+08]. Or, la puissance de calcul d'un drone est relativement faible (de l'ordre de celle d'un smartphone). C'est la coopération des drones qui leur permet de remplir des missions complexes.

L'utilisation de systèmes multi-agents, pour la gestion du trafic aérien comme pour d'autres problèmes, comporte donc certains avantages, comme la complexité réduite des règles de comportement, une puissance de calcul nécessaire plus faible puisque localisée au niveau des agents, et une plus grande flexibilité dans le développement des algorithmes. De plus, il est possible de faire interagir les agents du système avec des humains, qui peuvent contrôler et influer sur le comportement du système.

Ce doctorat a été financé par la société Capgemini Technology Services, tout comme ceux de Clément Peyronne (2012) [Pey12], Laureline Guys (2014) [Guy14] et Brunilde Girardet (2014) [Gir14]. L'intégration des travaux de Clément Peyronne dans un outil d'aide à la décision pour les contrôleurs aériens, couplé au système ERATO, est en cours d'étude. L'intégration des algorithmes Plectre et Yard décrits dans cette thèse est également envisagée.

Annexe A

Stockage et manipulation des grilles

Différentes parties des algorithmes présentés dans cette thèse manipulent des matrices creuses. C'est le cas, par exemple, de l'algorithme Yard du chapitre 5 : dans les cartes de convergence et de densité, les valeurs non-nulles se trouvent le long des trajectoires des avions, laissant de vastes portions de l'espace aérien vides.

S'il peut être avantageux de stocker des matrices denses dans des tableaux multidimensionnels, il est préférable d'utiliser des structures de données plus adaptées pour stocker des matrices creuses. La structure de données choisie pour cette thèse est la classe `HashMap<K, V>` [Ora16] proposée par le langage Java. Cette technique a notamment été utilisée par Chaimatanan (2014) [Cha14].

A.1 Fonctionnement de la classe HashMap

La HashMap, ou table de hachage, est un tableau associatif qui stocke des paires d'éléments (clé, valeur). Les éléments sont indexés par leur clé. Cela implique qu'il n'existe pas de doublons dans l'ensemble des clés stockées par la table de hachage. Le type des clés (`K`) et des valeurs (`V`) est défini à l'initialisation de la HashMap.

Les HashMaps reposent sur un tableau [Ora14] pour stocker les paires (clé, valeur). Elle se servent d'une fonction de hachage, qui calcule une empreinte (aussi appelée condensat ou *hash*) à partir de chaque clé. Cette empreinte sert à calculer la position à laquelle stocker la paire dans le tableau interne. La fonction de hachage doit autant que possible réduire le nombre de collisions (clés différentes retournant la même empreinte).

Voici le fonctionnement d'une HashMap :

1. Les objets utilisés comme clé implémentent la méthode `hashCode()`, qui retourne un entier contenant l'empreinte. Les classes représentant les types primitifs (`Integer`, `Double`, etc.) en proposent tous un. Il est préférable d'en implémenter un adapté pour les autres classes, de manière à réduire la probabilité de collision ;
2. La HashMap applique une deuxième fonction de hachage pour réduire encore le nombre de collisions. Dans Java 8, il s'agit d'un décalage à droite (opérateur $\gg$) de 16 bits du condensat h, suivi d'un *ou exclusif* bit-à-bit (opérateur $\oplus$) avec le même condensat de départ h ($h' = h \oplus (h \gg 16)$), qui permet de déplacer les bits de poids fort vers les bits de poids faible ;

3. La table de hachage calcule le reste de la division euclidienne du condensat par la taille du tableau interne (modulo) pour déterminer l'index où stocker la paire (clé, valeur) dans celui-ci.

Il arrive que plusieurs clés retournent le même condensat, ou bien que deux condensats correspondent au même index dans le tableau interne de la table de hachage. Les éléments devant être stockés au même index sont rassemblés dans une liste chainée, ce qui ralentit légèrement l'accès à un de ces éléments (il faut parcourir la liste). Les éléments dont l'index est identique sont différenciés grâce à la méthode `equals()` de la clé.

Dans l'hypothèse où le nombre de collisions est nul, le temps de lecture ou d'écriture d'un élément dans la HashMap est constant [Ora16]. Pour accéder à un élément, il suffit d'effectuer les trois opérations listées précédemment (hachage de la clé, deuxième hachage, puis calcul de l'index dans le tableau interne). Pour d'autres structures de données, le nombre d'opérations pour accéder à un élément croît avec le nombre d'éléments stockés. Par exemple, les TreeMaps nécessitent $\log_2 N$ opérations pour accéder à un des N éléments stockés.

Pour maintenir un faible taux de collisions dans une HashMap, son tableau interne ne doit pas être rempli à plus de 75 %. Quand ce seuil est dépassé, la taille du tableau est doublée, et les indexes de tous les éléments sont recalculés pour cette nouvelle taille.

A.2 Stockage de matrices creuses

A.2.1 Avantage des HashMaps

Les HashMaps ont trois avantages comparées aux tableaux conventionnels pour stocker des matrices creuses :

— Ils permettent d'économiser de l'espace par rapport à un stockage dans un tableau, dès que le nombre de valeurs non-nulles de la matrice est inférieur à la moitié du nombre total de valeurs : le taux moyen de remplissage du tableau interne est d'environs 50 %, mais chaque élément est une paire (clé, valeur), ce qui prend deux fois plus de place que le stockage des valeurs uniquement ;

— D'autres structures de données nécessitent encore moins d'espace de stockage. C'est le cas des structures basées sur des arbres binaires, comme les TreeMap. Ils utilisent exactement la place dont ils ont besoin pour stocker toutes les paires (clé, valeur). Mais ils ont un temps d'accès proportionnel au logarithme du nombre d'éléments stockés. En comparaison, les HashMap ont un temps d'accès constant ;

— La clé utilisée dans une table de hachage peut être de n'importe quel type, du moment que sa classe implémente les méthodes `hashCode()` et `equals()`. Cela donne beaucoup de liberté dans la manière de définir une clé. En particulier, l'algorithme Yard manipule des cartes géographiques. Les clés utilisées sont donc des coordonnées spatiales, qui peuvent être négatives. Si ces cartes avaient été stockées sous forme de tableau, il aurait fallu connaitre les bornes spatiales de la carte, et effectuer une mise à l'échelle et une translation pour transformer les coordonnées spatiales en coordonnées dans la matrice.

A.2.2 Format de stockage d'une matrice

La manière la plus simple pour stocker une matrice à deux dimensions d'indexes i et j dans une table de hachage consiste à créer une HashMap de HashMaps, dont la signature serait :

```
HashMap<Integer, HashMap<Integer, Double>>
```

La HashMap principale contient les lignes indexées par leur position j. Une ligne est elle-même stockée sous forme de HashMap dans laquelle les valeurs sont indexées par leur position i. L'avantage est qu'on n'a pas à définir de classe à utiliser comme clé, la classe `Integer` possédant une méthode `hashCode()`. L'inconvénient est que pour accéder à un élément, on doit manipuler deux HashMaps au lieu d'une seule. On perd également de la place, puisque le taux de remplissage moyen des tableaux internes d'une HashMap de HashMaps est d'environs 25 % (50 % dans chaque dimension).

Pour résoudre ces deux problèmes, une autre approche consiste à utiliser une seule table de hachage, et à définir une fonction permettant de transformer des coordonnées (i, j) en un scalaire k, avec $i_{\min} \leq i < i_{\max}$ et $j_{\min} \leq j < j_{\max}$. Il s'agit généralement de :

$$k = (j - j_{\min}) \cdot (i_{\max} - i_{\min}) + (i - i_{\min}) \tag{A.1}$$

L'inconvénient de cette approche est qu'il faut connaitre à l'avance les bornes de i et j.

A.3 Implémentation dans cette thèse

A.3.1 Format de stockage des matrices

La méthode utilisée dans les algorithmes décrits dans cette thèse combine les avantages des deux approches précédentes. Une classe `GridCoordinates2D` a été introduite dans ces algorithmes pour servir de clé dans les tables de hachage contenant des matrices. Son but est de stocker des coordonnées (i, j), et de fournir une fonction de hachage. Ainsi, une seule HashMap est utilisée pour stocker une carte, et il n'est pas nécessaire de connaitre les bornes des coordonnées (i, j). La fonction de hachage est celle-ci :

$$h(i, j) = i + 127j \tag{A.2}$$

Le coefficient 127 a été choisi arbitrairement, mais donne de bons résultats, puisque dans les algorithmes, la valeur de i est en général comprise entre −128 et 128. Par exemple, la carte de convergence est stockée dans une grille dont les cellules font 5 NM de côté (soit 9,26 km). Or, les scénarios utilisés pour valider les algorithmes Plectre et Yard impliquent du trafic aérien Français. Les dimensions de la France étant d'environ 1 000 km du nord au sud et d'est en ouest, cette grille fait environ 100 cases sur 100.

D'autres classes ont été définies à partir de `GridCoordinates2D` pour stocker des coordonnées incluant le temps :

$$h(i, j, t) = i + 127j + 31^2 t\,, \tag{A.3}$$

ou un angle :

$$h(i, j, \theta) = 8(i + 127j) + \frac{\theta}{8}\,. \tag{A.4}$$

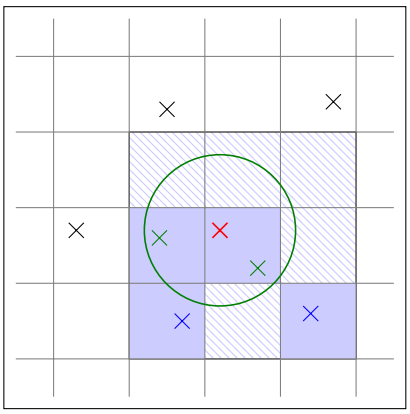


FIGURE A.1 – HashMap utilisée pour faire une recherche de points au voisinage d'une position donnée (croix rouge). Le voisinage est représenté par le cercle vert.

A.3.2 Recherche spatiale basée sur les HashMaps

Les HashMaps peuvent également être utilisées pour indexer et rechercher des éléments positionnés dans l'espace. Ici, la HashMap stocke des listes de points dans chacune des cases non-vides. La signature est similaire à :

```
HashMap<Coordinate, List<Point2D>>
```

Comme le montre la figure A.1, la recherche spatiale se déroule en deux étapes. On cherche tous les points situés à moins d'une certaine distance d'une position donnée, symbolisée par la croix rouge. La première étape consiste à collecter l'ensemble des points situés dans les cases voisines de celle contenant les coordonnées de départ (carré hachuré en bleu). Comme la HashMap ne stocke que les cellules non-vides, les seules cellules effectivement explorées sont quatre cases colorées en bleu foncé.

Dans un deuxième temps, la distance entre le point rouge et chaque point découvert durant la première étape (croix bleues) est calculée. Seuls les points situés à l'intérieur du cercle vert sont conservés (croix vertes).

Il faut noter que les meilleurs résultats sont obtenus quand la taille des cellules de la grille spatiale est égale au rayon du cercle, de manière à minimiser le nombre de cellules à explorer durant la première étape. Cette méthode donne donc les meilleurs résultats pour les problèmes dont la taille du voisinage est fixe et connu à l'avance. On peut par exemple s'en servir dans la détection de conflits, puisque la distance de séparation entre avions est fixée à 5 NM. Si la dimension de la fenêtre n'est pas connue ou constante, d'autres algorithmes d'indexation spatiale pourraient mieux fonctionner, comme les k-d trees ou les R-trees.

Sans cette indexation spatiale, une recherche des points situés à l'intérieur du cercle nécessitent de mesurer la distance entre le point rouge et *tous* les autres points, ce qui est extrêmement coûteux en temps de calcul.

Annexe B

Implémentation de Yard sur carte graphique

Les calculs de l'algorithme Yard sont relativement lents. La partie de l'algorithme la plus consommatrice en temps de calcul est la version centralisée de l'agrégation de trajectoires décrite dans la section 5.1.2. Par exemple, le scénario de la figure 5.11 agrège 4 trajectoires, ce qui nécessite 8 minutes de temps de calcul.

Même si Yard était utilisé dans la phase tactique, ce délai de 8 minutes serait compatible avec les contraintes opérationnelles, puisque l'horizon temporel habituellement considéré pour cette phase est de 20 minutes. Par contre, ce temps de calcul rend d'autant plus long l'ajustement précis des paramètres de l'algorithme durant son développement, puisque ce processus nécessite un grand nombre d'essais successifs.

Pour tenter de réduire ce délai, il a été décidé de porter l'algorithme d'agrégation de trajectoires sans prise en compte du cap (la version avec prise en compte du cap a été développée par la suite) dans le langage OpenCL, pour profiter des performances des cartes graphiques sur ce type de calcul.

B.1 Calcul sur carte graphique

Le General-purpose processing on graphics processing units (GPGPU) est une famille de technologies permettant d'effectuer des calculs sur cartes graphiques, pour profiter de leur capacité à effectuer des calculs massivement parallèles.

Les CPU appartiennent à la classe d'architecture matérielle appelée Single instruction on single data (SISD). Leur principe de fonctionnement est d'exécuter séquentiellement un ensemble d'instructions sur une donnée à la fois. Ils sont pourvus d'un petit nombre de cœurs de calcul optimisés pour exécuter cette séquence d'opérations le plus rapidement possible.

Contrairement aux CPU, les GPU appartiennent à la classe Single instruction multiple data (SIMD). Ils sont prévus pour exécuter le même ensemble d'opérations sur un ensemble de données *en parallèle*. Ils sont pourvus d'un grand nombre de cœurs de calcul (plusieurs centaines), tous exécutant la même instruction en même temps [1] sur un élément différent de l'ensemble de données.

1. En pratique, le fonctionnement d'un GPU est plus complexe : il est capable d'exécuter plusieurs fonctions différentes en même temps, et alloue dynamiquement les calculs sur chaque cœur pour accélérer les calculs.

B.1.1 Historique des méthodes de calcul sur GPU

Jusqu'au début des années 2000, il n'était pas possible de faire du calcul général sur un GPU [Fra+10]. Celui-ci prenait en entrée un modèle 3D (des polyèdres définis par un ensemble de triangles) et des textures à y appliquer (des grilles 2D de pixels). La carte graphique effectuait alors un ensemble figé d'opérations qui permettait de calculer l'image à afficher à l'écran (une grille 2D de pixels).

Puis il est devenu possible d'interagir avec ce processus au moyens de *shaders*, c'est-à-dire de programmes permettant d'effectuer des traitements entre deux étapes du rendu d'une scène 3D. Notamment, les *vertex shaders* permettent de modifier les sommets du modèle 3D, et les *pixel shaders* permettent de modifier les pixels des textures ou ceux de l'image finale.

Les shaders peuvent être détournés de leur but initial pour effectuer des calculs généraux. Par exemple, on peut écrire une matrice 2D sous forme de texture, puis appliquer un shader sur cette « texture », qui effectue en fait un calcul sur la matrice, et récupérer la texture transformée par le shader pour lire le résultat.

En 2007, Nvidia publie le langage CUDA (Compute Unified Device Architecture), qui permet d'écrire un calcul généraliste dans un langage proche du C, sans devoir écrire son programme sous forme de shader. Le CUDA est utilisable uniquement sur les cartes graphiques Nvidia.

En 2009, le Khronos Group, déjà à l'origine d'OpenGL, propose un langage similaire à CUDA, l'OpenCL. À la différence de son prédécesseur, l'OpenCL peut être exécuté sur une grande variété de matériel, dont les CPU et les GPU, et pas uniquement sur les GPU Nvidia.

B.1.2 Principe de fonctionnement

Algorithme 6 Addition de deux vecteurs sur CPU.

```
function AdditionVecteurs(double[] a, double[] b, taille)
    c ← double[taille]
    for i ← 0..taille do
        c[i] ← a[i] + b[i]
    end for
    return c
end function
```

Les algorithmes 6 et 7 montrent le fonctionnement général de l'addition de deux vecteurs respectivement sur CPU et GPU. Pour additionner deux vecteurs sur CPU (algorithme 6), le programme itère sur chaque élément des vecteurs a et b, et calcule la somme de a et b à l'index i, avant de stocker le résultat. Toutes les opérations sont effectuées séquentiellement.

L'adaptation de ce calcul sur GPU est légèrement plus complexe, comme le montre l'algorithme 7. Un GPU possède sa propre mémoire vive. Il faut commencer par transférer les vecteurs a et b sur la mémoire du GPU, et y allouer la mémoire nécessaire à stocker le résultat.

Le corps de la boucle de l'algorithme sur CPU est transféré dans une procédure exécutée sur le GPU. Dans la fonction `additionVecteurs()`, l'appel à la fonction `exécuteSurGPU()` demande au GPU d'exécuter la fonction `additionVecteursGPU()` en parallèle pour chaque index $i \in [0, \text{taille} - 1]$. Chacun de ces

Algorithme 7 Addition de deux vecteurs sur GPU.

```
function AdditionVecteurs(double[] a, double[] b, taille)
    a_G ←TransfèreSurGPU(a)
    b_G ←TransfèreSurGPU(b)
    c_G ←AlloueSurGPU(double[taille])
    ExécuteSurGPU(AdditionVecteursGPU, a_G, b_G, c_G, taille)
    c ←TransfèreDepuisGPU(c_G)
    return c
end function

procedure AdditionVecteursGPU(double[] a_G, double[] b_G, double[] c_G, i)
    c[i] ← a[i] + b[i]
end procedure
```

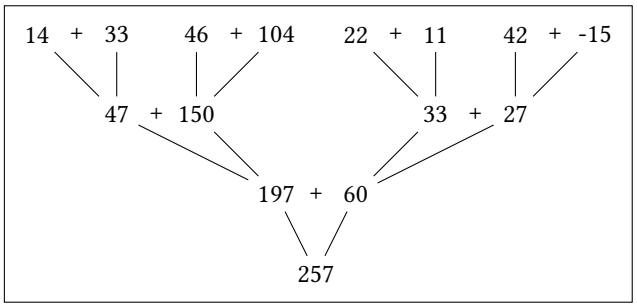


FIGURE B.1 – Addition de 8 nombres par réduction successive. Les nombres sont additionnés deux à deux. Toutes les additions sont effectuées en parallèle à chaque étape : la somme est calculée en 3 itérations au lieu de 7 dans le cas d'un calcul par accumulation.

appels parallèles est appelé *thread*. Ici, un thread correspond à l'appel de la fonction `additionVecteursGPU()` pour un index i particulier.

Ainsi, pour adapter un calcul pour le GPU, il faut identifier les portions de calcul pouvant être effectuées en parallèle (ici l'intérieur de la boucle `for`). Il faut alors déplacer cette portion des calculs dans une procédure exécutée par le GPU. Le pilotage de l'appel à cette fonction est effectué par le code exécuté par le CPU.

Certains calculs nécessitent une transformation plus importante pour tirer pleinement parti du GPU. Par exemple, additionner tous les éléments d'un tableau, ou en chercher la valeur minimale ou maximale peut être effectuée en faisant une réduction progressive de l'ensemble de valeurs pour en extraire une seule.

Ce processus [Cat10] est décrit par la figure B.1. À partir de la liste de valeurs de la première ligne, quatre additions sont effectuées en parallèle. Les quatre résultats intermédiaires sont à leur tour additionnés deux à deux en parallèle. Le processus itère jusqu'à l'obtention du résultat final. Pour n éléments de départ, le processus nécessite $\log_2 n$ itérations, à comparer aux $n-1$ itérations nécessaires pour une implémentation sur CPU, qui additionne les éléments deux par deux.

B.1.3 Spécificités du GPGPU

Il est nécessaire de prendre en compte les spécificités de l'architecture d'un GPU dans le développement d'un programme GPGPU. Dans le modèle OpenCL, la carte graphique dispose d'une mémoire globale dédiée. Elle est relativement lente, mais peut stocker une grande quantité de données. Le programme hôte (sur le CPU) peut lire et écrire sur la mémoire globale.

Les cœurs de calcul sont rassemblés en groupes, chaque groupe disposant d'une mémoire locale plus réduite, mais plus rapide. celle-ci n'est accessible que depuis les threads, qui doivent se charger du transfert de données entre la mémoire globale et locale.

Enfin, chaque cœur possède une mémoire privée, qui permet de stocker les variables d'un thread. C'est la plus rapide, mais aussi la plus petite.

Pour donner un exemple, la Nvidia GeForce GT 730M utilisée durant le doctorat est équipée d'environ un gigaoctet de mémoire globale. Les cœurs sont rassemblés en deux groupes de 192 cœurs (soit 384 cœurs au total), chacun se partageant une mémoire locale de 48 kio [2].

D'autre part, la fréquence du GPU est faible comparée à celle d'un CPU. Par exemple, la carte graphique utilisée est cadencée à 758 MHz, alors que le CPU (un Intel Core i7-4900MQ) est équipé de 8 cœurs cadencés à 2,8 GHz. Cette lenteur relative du GPU est compensée par le nombre bien plus élevé de cœurs. Ainsi, le GPU est plus performant que le CPU quand il doit exécuter un traitement sur au moins plusieurs centaines d'éléments, alors que le CPU est bien meilleur sur le traitement de quelques dizaines d'éléments.

Deux goulots d'étranglement peuvent ralentir l'exécution d'un programme sur un GPU. Le premier est le débit de transfert de données entre la mémoire RAM et la mémoire globale du GPU. Pour diminuer ce temps de latence, il faut éviter au maximum de transférer des données vers le GPU, par exemple en évitant de transférer des résultats intermédiaires entre le CPU et le GPU, et en calculant le plus de résultats intermédiaires possibles sur le GPU

Le deuxième goulot d'étranglement est le temps de latence avant le lancement d'un thread sur le GPU. Il est donc préférable de lancer une seule fonction complexe qui effectue la totalité des calculs plutôt que plusieurs fonctions élémentaires qui n'en font qu'une partie.

B.2 Implémentation de l'agrégateur de trajectoires sur GPU

La méthode d'agrégation de trajectoires est la partie de l'algorithme la plus adaptée à un portage sur GPU. Par exemple, la carte de densité de la figure 5.4 fait 167 lignes sur 192 colonnes, soit 32 064 éléments. Un noyau utilisé par l'estimateur par noyaux fait $32 \times 32 = 1024$ éléments. Ces deux grandeurs donnent une idée du potentiel de portabilité des calculs sur GPU.

B.2.1 Étapes du calcul

Pour agréger les trajectoires, chaque waypoint $\vec{p}_i(t)$ composant celles-ci est déplacé selon le résultat de l'équation (B.1), qui retourne le gradient de l'entropie à la position

2. **Kibi**octet, pour **ki**lo **bi**naire. Un kibioctet vaut 1 024 octets (2^{10}), par opposition au kilooctet, qui vaut 1 000 octets (10^3).

$\vec{p}_i(t)$:

$$\vec{\eta}_i(t) = \int_\Omega \left(\frac{\vec{p}_i(t) - \vec{x}}{\|\vec{p}_i(t) - \vec{x}\|} \right)_{\mathcal{N}} K_h'(\|\vec{p}_i(t) - \vec{x}\|) \log d(\vec{x}) \, \mathrm{d}\vec{x} \, \|\vec{v}_i(t)\| \tag{B.1a}$$

$$- \left(\int_\Omega K_h(\|\vec{p}_i(t) - \vec{x}\|) \log d(\vec{x}) \, \mathrm{d}\vec{x} \right) \left(\frac{\vec{a}_i(t)}{\|\vec{v}_i(t)\|} \right)_{\mathcal{N}} \tag{B.1b}$$

$$+ \left(\int_\Omega d(\vec{x}) \log d(\vec{x}) \, \mathrm{d}\vec{x} \right) \left(\frac{\vec{a}_i(t)}{\|\vec{v}_i(t)\|} \right)_{\mathcal{N}} \tag{B.1c}$$

Or, cette équation est composée d'éléments plus simples qui peuvent être calculés en avance et réutilisés pour calculer le déplacement des waypoints. Ces composants sont mis en évidence par différentes couleurs dans l'équation B.1.

B.2.1.1 Calcul du noyau et de sa dérivée

L'estimateur par noyau utilise un noyau d'Epanechnikov, dont la formule est donnée par l'équation (1.29). Il est borné spatialement par la taille de sa fenêtre. Il est donc possible de le pré-calculer dans une grille dont la taille des mailles est identique à la grille contenant la carte de densité. Le noyau $K_h(\|\vec{x}\|)$, en rouge dans l'équation (B.1), est calculé une fois pour toutes dès le début de l'exécution de l'algorithme. Il sera réutilisé par la suite en effectuant une translation vers la position $\vec{p}_i(t)$ de chaque waypoint.

De la même manière, la dérivée du noyau, également en rouge dans l'équation (B.1), est pré-calculée et stocké dans une grille contenant les valeurs retournées par l'équation :

$$\frac{\vec{p}_i(t) - \vec{x}}{\|\vec{p}_i(t) - \vec{x}\|} \cdot K_h'(\|\vec{x}\|) \,. \tag{B.2}$$

B.2.1.2 Transfert des waypoints sur la carte graphique

Les waypoints sont ensuite transférés dans la mémoire globale du GPU. Comme le nombre de waypoints de chaque trajectoire est différent, ceux-ci sont stockés sous la forme d'une liste monodimensionnelle de vecteurs position $\vec{p}_i(t)$. Les trajectoires sont séparées par un vecteur initialisée à la valeur spéciale NaN (Not a Number) que peuvent prendre les nombre à virgule flottante. Une valeur NaN est également insérée à la première et à la dernière position de la liste. Pour deux trajectoires p_1 et p_2, la liste est de la forme :

$$\left\{ \vec{\mathrm{NaN}}, \vec{p}_1(t_0), \vec{p}_1(t_1), \ldots, \vec{p}_1(t_n), \vec{\mathrm{NaN}}, \vec{p}_2(t_0'), \vec{p}_2(t_1'), \ldots, \vec{p}_2(t_n'), \vec{\mathrm{NaN}} \right\}$$

Ainsi, pour calculer le vecteur vitesse à partir des vecteurs positions, il suffit de connaître l'index du vecteur position dans la liste, ce qui permet d'extraire les positions précédentes et suivantes. Si l'une de ces deux positions est le vecteur $\vec{\mathrm{NaN}}$, cela signifie que la position courante est une des extrémités de la trajectoire. Cette information est alors prise en compte pour calculer la vitesse en cette position.

Les vecteurs vitesse $\vec{v}_i(t)$, en violet dans l'équation (B.1), sont calculés en parallèle, chacun se basant sur les positions précédente et suivante le long de la trajectoire.

B.2.1.3 Dimensions de la carte de densité

À partir de la liste des waypoints, l'algorithme calcule les limites spatiales de la carte de densité. Ce rectangle englobant est utilisé dans deux cas. Premièrement, les dimensions du rectangle permettent de calculer l'espace mémoire à allouer sur la carte graphique pour stocker la grille contenant la carte de densité. Ce rectangle est assez grand pour contenir l'ensemble des waypoints et leur noyau d'Epanechnikov.

Deuxièmement, le rectangle englobant permet de convertir les coordonnées spatiales des waypoints (coordonnées réelles) en coordonnées à l'intérieur de la carte de densité (coordonnées entières d'origine $(0, 0)$).

Le rectangle englobant est calculé par réduction. La première étape de ce calcul consiste à transformer chaque paire de positions :

$$\left\{ \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}, \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} \right\}$$

en un rectangle :

$$\begin{pmatrix} x_{\min} \\ y_{\min} \\ x_{\max} \\ y_{\max} \end{pmatrix} = \begin{pmatrix} \min(x_1, x_2) \\ \min(y_1, y_2) \\ \max(x_1, x_2) \\ \max(y_1, y_2) \end{pmatrix}$$

Ensuite, les rectangles sont agrégés deux à deux, dans un processus analogue à celui décrit par la figure B.1, chaque rectangle intermédiaire englobant les deux rectangles agrégés, jusqu'à obtenir le rectangle englobant final.

B.2.1.4 Carte de densité

La carte de densité est la partie la plus difficile à adapter au calcul sur GPU. La densité de probabilité est la moyenne pondérée des noyaux centrés sur les waypoints. La valeur de chaque cellule de cette grille est donc la moyenne pondérée de valeurs extraites du noyau d'Epanechnikov pré-calculé.

À chaque position, il faut donc faire la somme de plusieurs valeurs. Or, dans ce cas précis, il est difficile de paralléliser cette addition. Il faut éviter les accès concurrents, c'est-à-dire l'écriture d'une valeur dans la même case par deux threads différents, ce qui conduirait au stockage d'une valeur incorrecte. Il faut donc trouver un moyen de synchroniser les différents threads pour que chacun écrive à son tour dans une case de la grille. Pour calculer les valeurs de la grille, plusieurs approches peuvent être utilisées.

B.2.1.4.1 Écriture séquentielle des noyaux On part d'une grille dont les valeurs sont initialisées à 0, puis on ajoute itérativement chaque noyau. Ici, 1 024 threads fonctionnent en parallèle, un par cellule du noyau d'Epanechnikov pré-calculé. Il n'y a pas d'accès concurrent, puisque chaque thread écrit dans une cellule différente de la carte de convergence.

C'est cette méthode qui a été choisie pour sa simplicité. Mais c'est la méthode la plus lente, puisqu'il faut itérer autant de fois qu'il y a de waypoints. D'autre part, il n'y a que 1 024 threads, ce qui est un nombre relativement faible pour exploiter pleinement les possibilités de la carte graphique.

Il est probable que la version de l'algorithme avec prise en compte du cap utilise soit comparativement plus performante. En effet, les noyaux font 1 024 × 8 cellules

(les caps sont échantillonnés avec un pas de $\pi/4$). Il y a donc 8 fois plus de threads exécutés en parallèle, ce qui diminue comparativement le temps perdu à itérer.

En effet, les GPU sont plus performants pour l'exécution massivement parallèle de tâches. Mais, vu leur relative lenteur par rapport aux CPU, ils sont moins efficaces pour les tâches séquentielles. Augmenter le nombre de tâches parallèles diminue le ratio entre le nombre d'itérations et le nombre de tâches parallèles.

B.2.1.4.2 Additions par réduction On part d'une grille en 3 dimensions (largeur × hauteur × nombre de waypoints), et on écrit en parallèle un noyau par couche de la grille. Ensuite, la grille finale est calculée par réduction, en agrégeant deux à deux les cellules de coordonnée spatiale identique, de la même manière que dans la figure B.1.

Cette méthode est relativement inefficace, puisqu'il faut exécuter autant de threads que de cellules de la grille 3D, et que la plupart des threads seront inoccupés la plupart du temps, voire durant la totalité du processus, puisque cette grille est en grande partie remplie de valeurs nulles.

De plus, la consommation d'espace mémoire est bien plus importante, puisqu'il faut multiplier la taille de la carte de convergence par le nombre de waypoints, ce qui peut excéder la mémoire disponible.

B.2.1.4.3 Opérations atomiques Le GPU est capable de synchroniser lui-même certaines opérations arithmétiques simples, comme l'addition de deux nombres entiers. Ces opérations sont exécutées par des circuits matériels dédiés, et sont donc performantes.

Le GPU ne peut pas faire d'addition atomique entre deux nombres flottants, mais une astuce [HH16] permet d'émuler logiciellement ce comportement pour additionner deux flottants simple précision (32 bits). Cependant, cette méthode est moins performante que son équivalent pour les nombres entiers, puisqu'il s'agit d'une méthode logicielle et pas matérielle.

Il faudrait vérifier si cette méthode est plus rapide que l'addition séquentielle de noyaux, décrite dans la section B.2.1.4.1.

Une variante de cette approche consiste à utiliser des nombres entiers, qui supportent les opérations atomiques, comme verrous, ce qui permet aux threads de « capturer » temporairement une cellule de la grille, le temps d'effectuer l'addition. Le verrou est ensuite relâché pour qu'un autre thread puisse effectuer son propre calcul. Ce processus est similaire au mécanisme de verrou proposés par les CPU pour faire du multi-threading.

B.2.1.5 Calcul de l'entropie

Le calcul de l'entropie est facilement parallélisable. Une fois la carte de densité calculée, il suffit de lancer autant de threads que de cellule de la carte de densité. Chacun calcule $d(\vec{x}) \log d(\vec{x})$ (en orange dans la ligne (B.1c)) à l'intérieur de sa cellule, et stocke le résultat intermédiaire dans une grille de même dimension.

Ensuite, l'entropie est calculée comme la somme de toutes ces valeurs intermédiaires, par la méthode de réduction (en jaune dans la ligne (B.1c)).

B.2.1.6 Autres résultats intermédiaires du calcul

Les deux autres intégrales, à la ligne (B.1a) et (B.1b), sont calculées de la même manière. Les parties surlignées en vert sont calculées en parallèle, et les parties bleues

(intégration) sont obtenues par réduction pour obtenir la somme.

Une fois que tous les résultats intermédiaires sont obtenus, il ne reste plus qu'à calculer le résultat de l'équation (B.1) pour chaque waypoint, afin de déplacer les waypoints.

Le processus se répète alors, en commençant par la mise à jour des vecteurs vitesse, décrite dans la section B.2.1.2, jusqu'à ce que le système converge.

B.2.2 Limites de l'implémentation

En l'état actuel, l'implémentation souffre de plusieurs limites. Certaines ont été découvertes tardivement, et n'ont pas été résolues dans le temps imparti à l'implémentation du calcul sur GPU. La version GPU du calcul atteint donc à peine les performances de la version CPU, alors qu'il peut en théorie être bien plus performant. Deux problèmes ont été identifiés, leur résolution pouvant améliorer grandement la vitesse de calcul.

Premièrement, par habitude, tous les nombres flottants manipulés sont en double précision (type double, 64 bits). Ils le sont également dans la version CPU du programme. Or, sur un GPU, les calculs en double précision sont beaucoup plus lents qu'en simple précision. Par exemple, la Nvidia GeForce GTX Titan est 24 fois plus rapide pour des calculs en simple précision [Ang13].

Deuxièmement, comme évoqué précédemment dans cette annexe, le temps de lancement d'une fonction sur un GPU est relativement long. Il est donc plus efficace de lancer une seule fonction qui effectue la totalité des calculs plutôt que lancer plusieurs fonctions séquentiellement qui n'en effectuent qu'une partie.

La limite de cette approche est que l'allocation mémoire ne peut être faite que par le programme hôte, sur le CPU. Il faut donc déterminer l'espace mémoire nécessaire durant la totalité du processus. C'est possible pour la plupart des tableaux, dont la taille ne dépend que du nombre de waypoints. Par contre, c'est plus difficile pour la carte de densité, qui peut changer de taille au cours du processus, quand un waypoint se déplace au delà des limites de la carte. Il faut donc, soit déterminer à l'avance la taille maximale que peut prendre la carte de densité durant le processus, pour allouer l'espace mémoire adéquat, soit partir d'une taille de carte initiale, et être capable d'interrompre le processus en cours de route s'il s'avère que sa taille est insuffisante. La première possibilité est non triviale, et la seconde probablement complexe à implémenter.

Ces problèmes démontent que la technologie GPGPU demande un grand niveau de maitrise, à la fois du langage C, de ses dérivés CUDA et OpenCL, et une bonne connaissance du fonctionnement d'une carte graphique. Une fois que ces compétences sont acquises, les problèmes, s'ils sont adaptés au calcul sur GPU, peuvent bénéficier d'un accroissement de performances.

Nomenclature

t, t'	Nomenclature Instants dans la simulation.
N	Nombre d'avions.
λ	Nombre moyen d'avions générés sur chaque route, en s^{-1} dans les scénarios où le trafic est généré aléatoirement.
i, j	Identifiants d'avions.
$v_{i,\text{opt}}$	Vitesse optimale, ou vitesse de croisière, de l'avion i.
$v_{i,\text{min}\,\vert\,v_{i,\text{opt}}}$, $v_{i,\text{max}\,\vert\,v_{i,\text{opt}}}$	Vitesses minimale et maximale imposées par les performances de l'avion i, déterminées en fonction de $v_{i,\text{opt}}$.
θ_{ij}	Différence entre les caps des avions i et j.
S_0	Norme de séparation entre deux avions ($S_0 = 5\,\text{NM}$).
S_{ij}	Distance de séparation minimale entre les avions i et j pour éviter les conflits, qui peut être plus grande que S_0, notamment en fonction de leurs cap et vitesse relatifs.
$\vec{p_i}(t)$	Vecteur position de l'avion i à l'instant t.
$\vec{v_i}(t)$	Vecteur vitesse de l'avion i à l'instant t.
$\vec{a_i}(t)$	Vecteur accélération de l'avion i à l'instant t.

Nomenclature de l'algorithme du chapitre 3

P_i	Le plus proche voisin précédant l'avion i.
F_i	Le plus proche voisin suivant l'avion i (follower).
j	N'importe quel voisin de l'avion i, y compris P_i et F_i.
t_j	Date d'arrivée du voisin j au prochain waypoint que va atteindre l'avion i.
d_j	Distance séparant le voisin j du prochain waypoint que va atteindre l'avion i.
v_j	Vitesse du voisin j.
$t_{i,\text{min}\,\vert\,v_{i,\text{opt}}}$, $\textbf{Nomenclature}t_{i,\text{max}\,\vert\,v_{i,\text{opt}}}$	Dates d'arrivée minimale et maximale au prochain waypoint imposées par les performances de l'avion, déterminées en fonction de $v_{i,\text{opt}}$.
$v_{i,\text{min}}$, $v_{i,\text{max}}$	Vitesses minimale et maximale trouvées par le processus de décision de l'avion dans Plectre v1.
d_i	Distance séparant l'avion i de son prochain waypoint.
v_i	Vitesse choisie par le processus de décision de l'avion i.

Nomenclature de la carte de convergence

$\vec{p_{ij}}(t)$	Position relative des avions i et j à l'instant t.
$\vec{v_{ij}}(t)$	Vitesse relative des avions i et j à l'instant t.
$r_{ij}(t)$	Dérivée de la distance entre $\vec{p}_i(t)$ et $\vec{p}_j(t)$ par rapport au temps.
$\vec{p_{ij}}(t, t')$	Position relative de l'avion i à l'instant t et de l'avion j à l'instant t', utilisée dans la version robuste de la mesure de convergence.
$\vec{v_{ij}}(t, t')$	Vitesse relative de l'avion i à l'instant t et de l'avion j à l'instant t', utilisée dans la version robuste de la mesure de convergence.
$r_{ij}(t, t')$	Dérivée de la distance entre $\vec{p}_i(t)$ et $\vec{p}_j(t')$ par rapport au temps, utilisée dans la version robuste de la mesure de convergence.
$D_{\max}$	La convergence est calculée entre des avions séparés par moins de cette distance maximale.
Δt	La mesure robuste de la convergence utilise uniquement des positions dans un intervalle de temps $[t - \Delta t, t + \Delta t]$.
$C_{ij}(t)$	Convergence entre $\vec{p}_i(t)$ et $\vec{p}_j(t)$.
$C_{ij}(t, t')$	Convergence entre $\vec{p}_i(t)$ et $\vec{p}_j(t')$, utilisée dans la version robuste de la mesure de convergence.
$\tilde{C_{ij}}(t)$	Convergence robuste calculée entre les positions $\vec{p}_i(t)$ et $\vec{p}_j(t')$, avec $t' \in [t - \Delta t, t + \Delta t]$.
$\tilde{C_i}(t)$	Convergence robuste calculée à la position $\vec{p}_i(t)$, pour tous les avions $j \neq i$.
C	Grille 2D contenant la carte de convergence.
(k, l), (m, n)	Coordonnées dans la grille C.
$C_{k,l}$, $C_{m,n}$	Valeur de la convergence aux coordonnées (k, l) (resp. (m, n)) dans la carte C.

Nomenclature de la minimisation de l'entropie

t	Instant dans la simulation. Pour chaque trajectoire, t est normalisé dans l'intervalle $[0, 1]$.
l_i	Longueur de la trajectoire de l'avion i.
Ω	Espace dans lequel la densité est calculée.
$\vec{x}$	Position dans l'espace Ω.
$d(\vec{x})$	Valeur de la densité à la position x, calculée grâce à un estimateur par noyau.
$H(\Omega)$	Entropie calculée pour la densité de probabilité $d(\vec{x})$.
K_h	Noyau mis à l'échelle, de fenêtre h, utilisé pour calculer la densité de probabilité.
$\vec{\eta_i}(t)$	Déplacement à appliquer à la position $\vec{p}_i(t)$ pour minimiser l'entropie de la densité du trafic.

Glossaire

ACAS Airborne Collision Avoidance System. 43
ACC Area Control Center. 4
ADS-B Automatic Dependant Surveillance – Broadcast. 9, 13, 47, 57, 72
ADS-C Automatic Dependant Surveillance – Contract. 10, 47, 72
ANSP Air Navigation Service Provider. 4, 37
ARTCC Air Route Traffic Control Center. 4, *voir* ACC
ATFM Air Traffic Flow Management. 1
ATM Air Traffic Management. 16, 49, 131

CAUTRA système de Coordination AUtomatique du TRafic Aérien. 15
CCR Centre de contrôle régional. 4, 7, 9, 11, 17, *voir* ACC
CDR Conflict Detection and Resolution. 8
CI Cost Index. 12, 57

DGAC Direction générale de l'Aviation civile. 3, 4, 15
DSNA Direction des services de la Navigation aérienne. 4

ERASMUS En Route Air traffic Soft Management Ultimate System. 36, 44
ERATO En-Route Air Traffic Organizer. 17, 132

FFA Free Flight Airspace. 7, 33, 71
FIR Flight Information Region. 4
FL Flight Level. 5
FMS Flight Management System. 3, 5, 13
FRA Free Route Airspace. 6
ft foot, ou pied. 5

GPGPU General-purpose processing on graphics processing units. 137

ICAO International Civil Aviation Organization. *voir* OACI
IFR Instrument Flight Rules. 3, 6, 9

kt nœud. 13, 57

MINIT Minutes-in-Trail. 15

MIT Miles-in-Trail. 14, 44, 57, 69

NextGen U.S. Next Generation Air Transportation System. 17, 49
NM Nautical Mile. 12, 14, 15

OACI Organisation de l'aviation civile internationale. 1, 16
ODS Operational Display System. 15

SESAR Single European Sky ATM Research. 17, 28, 49
SID Standard Instrument Departure. 35
SIMD Single instruction multiple data. 137
SISD Single instruction on single data. 137
SMA système multi-agents. 45, 131
STAR Standard Terminal Arrival Routes. 35

TCAS Traffic Collision Avoidance System. 43, 44
TMA Terminal Manoeuvring Area. 11, 38
TOC Top of Climb. 10
TOD Top of Descent. 10

VFR Visual Flight Rules. 3, 6

ÉNAC École Nationale de l'Aviation Civile. 130

Bibliographie

[AB91] L. Amodei et M. N. Benbourhim. « A Vector Spline Approximation with Application to Meteorology ». In : *Curves and Surfaces*. Sous la dir. de Pierre-Jean Laurent, Alain Le Méhauté et Larry L. Schumaker. San Diego, CA, USA : Academic Press Professional, Inc., 1991, p. 5–10. isbn : 0-12-438660-1. url : http://dl.acm.org/citation.cfm?id=114172.114174.

[ABM05] Erik P. Anderson, Randal W. Beard et Timothy W. McLain. « Real-time dynamic trajectory smoothing for unmanned air vehicles ». In : *IEEE Transactions on Control Systems Technology* 13.3 (mai 2005), p. 471–477. issn : 1063-6536. doi : 10.1109/TCST.2004.839555.

[ADG98] Jean-marc Alliot, Nicolas Durand et Geraud Granger. « FACES : a Free flight Autonomous and Coordinated Embarked Solver ». In : *2nd USA/EUROPE ATM R&D seminar*. 1998.

[AGD04] Nicolas Archambault, Géraud Granger et Nicolas Durand. « Heuristiques d'ordonnancement pour une résolution embarquée de conflits aériens par une méthode séquentielle ». In : *RIVF 2004, 2ème Conference Internationale Associant Chercheurs Vietnamiens et Francophones en Informatique*. Hanoï, Viet Nam, 2–5 fév. 2004. url : http://hal-enac.archives-ouvertes.fr/hal-00938234.

[Air] Air Canada. *Our Fleet : Airbus A320-200 (320)*. url : http://www.aircanada.com/en/about/fleet/a320-200xm.html (visité le 25/08/2015).

[All11] Cyril Allignol. « Planification de trajectoires pour l'optimisation du trafic aérien ». Thèse de doct. Institut National Polytechnique de Toulouse (INP Toulouse), 13 déc. 2011. 167 p.

[Ang13] Chris Angelini. *Nvidia GeForce GTX Titan 6 GB : GK110 On A Gaming Card*. Tom's Hardware. 19 fév. 2013. url : http://www.tomshardware.com/reviews/geforce-gtx-titan-gk110-review,3438-2.html (visité le 21/03/2017).

[Arc+08] James K. Archibald et al. « A Satisficing Approach to Aircraft Conflict Resolution ». In : *IEEE Transactions on Systems, Man, and Cybernetics, Part C : Applications and Reviews* 38 (4 juil. 2008), p. 510–521. issn : 1094-6977. doi : 10.1109/TSMCC.2008.919162.

[AT12] Adrian K. Agogino et Kagan Tumer. « A multiagent approach to managing air traffic flow ». In : *Autonomous Agents and Multi-Agent Systems* 24 (1 jan. 2012), p. 1–25. doi : 10.1007/s10458-010-9142-5. url : http://dl.acm.org/citation.cfm?id=2124496.2124503.

[Ave+07] Philippe Averty et al. « Could ERASMUS speed adjustments be identifiable by air traffic controllers ? » In : *Proceedings of the VII USA/Europe Air Traffic Management R&D seminar, Barcelona*. Juil. 2007.

[Bal+02] Mark G Ballin et al. « NASA Langley and NLR research of distributed air/ground traffic management ». In : (2002).

[BDG09] Deirdre Bonini, Carole Dupré et Géraud Granger. « How ERASMUS can support an increase in capacity in 2020 ». In : *Proceedings of CCCT : the 7th International Conference on Computing, Communications and Control Technologies*. 2009.

[BL14] Romaric Breil et Laurent Lapasset. *Cooperative Aircraft Trajectories Planning*. 4th SESAR Innovation Days. Poster. Universidad Politécnica de Madrid, nov. 2014. URL : https://hal-enac.archives-ouvertes.fr/hal-01131014.

[BL15] Romaric Breil et Laurent Lepasset. *Cooperative air traffic structuring*. 5th SESAR Innovation days. Poster. Déc. 2015. URL : https://hal-enac.archives-ouvertes.fr/hal-01240308.

[Bon+13] Pierre Bonami et al. « Multiphase Mixed-Integer Optimal Control Approach to Aircraft Trajectory Optimization ». In : *Journal of Guidance, Control, and Dynamics* 36.5 (sept.–oct. 2013), p. 1267–1277. ISSN : 0731-5090. DOI : 10.2514/1.60492. URL : http://arc.aiaa.org/doi/abs/10.2514/1.60492.

[Bor+14] Gianluca Borghini et al. « Analysis of neurophysiological signals for the training and mental workload assessment of ATCos ». In : *SESAR 2014, 4th SESAR Innovation Days*. Madrid, Spain, nov. 2014. URL : https://hal-enac.archives-ouvertes.fr/hal-01087183.

[BP01] Antonio Bicchi et Lucia Pallottino. *Optimal Conflict Resolution for Air Traffic Control*. 2001. URL : http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.32.8390.

[BP08] Flavien Balbo et Suzanne Pinson. « An agent oriented approach to transportation regulation support systems ». In : *Proceedings of the 5th Workshop in Agent in Traffic and Transport*. 2008, p. 225–242.

[BP98] Dimitris Bertsimas et Sarah Stock Patterson. « The Air Traffic Flow Management Problem with Enroute Capacities ». In : *Operations Research* 46.3 (1998), p. 406–422. DOI : 10.1287/opre.46.3.406. eprint : http://pubsonline.informs.org/doi/pdf/10.1287/opre.46.3.406.

[Bre+14] Romaric Breil et al. « Planification coopérative de trajectoires d'avions ». In : *ROADEF 2014, 15ème congrès annuel de la Société française de recherche opérationnelle et d'aide à la décision*. Société française de recherche opérationnelle et d'aide à la décision. Bordeaux, France, fév. 2014. URL : https://hal.archives-ouvertes.fr/hal-00946387.

[Bre+15] Romaric Breil et al. « Planification coopérative de trajectoires d'avions ». In : *ROADEF 2015, 16èmec onférence ROADEF*. Société Française de Recherche Opérationnelle et Aide à la Décision. Marseille, France, fév. 2015. URL : https://hal-enac.archives-ouvertes.fr/hal-01134556.

[Bre+16a] Romaric Breil et al. « Multi-agent Systems for Air Traffic Conflicts Resolution by Local Speed Regulation ». In : *7th International Conference on Research in Air Transportation (ICRAT 2016)*. Best Paper Award for Automation track. Philadelphie, PA, United States, juin 2016. URL : https://hal-enac.archives-ouvertes.fr/hal-01342623.

[Bre+16b] Romaric Breil et al. « Système multi-agents pour la structuration du trafic aérien ». In : *ROADEF 2016 17ème congrès annuel de la société Française de Recherche Opérationnelle et d'Aide à la Décision*. Compiègne, France, fév. 2016. URL : https://hal-enac.archives-ouvertes.fr/hal-01285238.

[Can+12] José Miguel Canino et al. « A Multi-Agent Approach for Designing Next Generation of Air Traffic Systems ». In : *Advances in Air Navigation Services*. Sous la dir. de Tone Magister. InTech, 1er août 2012. Chap. 9. ISBN : 978-953-51-0686-9. DOI : 10.5772/2574. URL : http://www.intechopen.com/books/advances-in-air-navigation-services.

[Cap+03] Davy Capera et al. « Emergence of organisations, emergence of functions ». In : *AISB'03 symposium on Adaptive Agents and Multi-Agent Systems*. 2003, p. 103–108.

[Cat10] Bryan Catanzaro. *OpenCL™ Optimization Case Study : Simple Reductions*. AMD. 24 août 2010. URL : http://developer.amd.com/resources/articles-whitepapers/opencl-optimization-case-study-simple-reductions/ (visité le 21/03/2017).

[CCL10] Georgios Chaloulos, Eva Crück et John Lygeros. « A simulation based study of subliminal control for air traffic management ». In : *Transportation Research Part C : Emerging Technologies* 18 (6 déc. 2010) : *Special issue on Transportation Simulation. Advances in Air Transportation Research*. Sous la dir. de Majid Sarvi et Vu N. Duong, p. 963–974. DOI : 10.1016/j.trc.2010.03.002.

[CCM13] Loïc Cellier, Sonia Cafieri et Frederic Messine. « A decomposition-based optimal control approach for aircraft conflict avoidance performed by velocity regulation ». In : *ATACCS 2013, 3rd International Conference on Application and Theory of Automation in Command and Control Systems*. Naples, Italy : ACM, mai 2013, p. 111–113. ISBN : 978-1-4503-2249-2. DOI : 10.1145/2494493.2494508. URL : https://hal-enac.archives-ouvertes.fr/hal-00909697.

[CD14] Sonia Cafieri et Nicolas Durand. « Aircraft deconfliction with speed regulation : new models from mixed-integer optimization ». In : *Journal of Global Optimization* 58.4 (1er avr. 2014), p. 613–629. DOI : 10.1007/s10898-013-0070-1. URL : http://hal-enac.archives-ouvertes.fr/hal-00935215.

[Čer85] Vladimír Černý. « Thermodynamical approach to the traveling salesman problem : An efficient simulation algorithm ». In : *Journal of optimization theory and applications* 45.1 (1985), p. 41–51.

[CGG04] Davy Capera, Marie-Pierre Gleizes et Pierre Glize. « Self-organizing agents for mechanical design ». In : *Engineering Self-Organising Systems*. Springer, 2004, p. 169–185.

[Cha14] Supatcha Chaimatanan. « Strategic planning of aircraft trajectories ». Thèse de doct. Université Paul Sabatier - Toulouse III, juil. 2014. URL : https://tel.archives-ouvertes.fr/tel-01064452.

[Chi+12] Rahul Chipalkatty et al. « Merging and Spacing of Heterogeneous Aircraft in Support of NextGen ». In : *Journal of Guidance, Control, and Dynamics* 35.5 (sept. 2012), p. 1637–1646.

[CL07] Eva Crück et John Lygeros. « Subliminal air traffic control : Human friendly control of a multi-agent system ». In : *American Control Conference*. ACC '07. (9–13 juil. 2007). 2007, p. 462–467. DOI : 10.1109/ACC.2007.4282641.

[Cla+01] Mario SV Valenti Clari et al. « Cost-benefit study of free flight with airborne separation assurance ». In : *Air Traffic Control Quarterly* 9.4 (2001), p. 287–309.

[Cob+12] Jose Antonio Cobano et al. « Efficient Conflict Resolution Method in Air Traffic Management Based on the Speed Assignment ». In : *Proceedings of the 2Nd International Conference on Application and Theory of Automation in Command and Control Systems*. ATACCS '12. London, United Kingdom : IRIT Press, 2012, p. 54–61. ISBN : 978-2-917490-20-4. URL : http://dl.acm.org/citation.cfm?id=2325676.2325684.

[DA97] Nicolas Durand et Jean-Marc Alliot. « Optimal Resolution of En Route conflicts ». In : *Séminaire Europe/USA*. Saclay, 6 juin 1997.

[DAN96] Nicolas Durand, Jean-marc Alliot et Joseph Noailles. « Automatic aircraft conflict resolution using genetic algorithms ». In : *Proceedings of the Symposium on Applied Computing*. ACM, 1996.

[DBS06] M. Dorigo, M. Birattari et T. Stutzle. « Ant colony optimization ». In : *Computational Intelligence Magazine, IEEE* 1.4 (nov. 2006), p. 28–39. ISSN : 1556-603X. DOI : 10.1109/MCI.2006.329691.

[Del+96] Daniel Delahaye et al. « Genetic Algorithms for Air Traffic Control System ». In : *14th IFORS Triennal Conference*. 8 juil. 1996.

[Dev+11] Santosh Devasia et al. « Decoupled Conflict-Resolution Procedures for Decentralized Air Traffic Control ». In : *IEEE Transactions on Intelligent Transportation Systems* 12.2 (juin 2011), p. 422–437. ISSN : 1524-9050. DOI : 10.1109/TITS.2010.2093574.

[DFB02] David Dugail, Eric Feron et Karl Bilimoria. « Stability of intersecting aircraft flows using heading change maneuvers for conflict avoidance ». In : *Proceedings of the American Control Conference*. Anchorage. AK, 8–10 mai 2002, p. 760–766.

[DG10] J. Dean et S. Ghemawat. *System and method for efficient large-scale data processing*. US Patent 7,650,331. 19 jan. 2010. URL : https://www.google.com/patents/US7650331.

[DGA07] Direction Générale de l'Aviation Civile. *Du morse à la souris : 60 ans de contrôle en route*. Sept. 2007. ISBN : 978-2-11-097012-1. URL : http://www.developpement-durable.gouv.fr/Du-morse-a-la-souris-60-ans-de.html.

[DGD09] Raphael Deau, Jean-Baptiste Gotteland et Nicolas Durand. « Airport surface management and runways scheduling ». In : *ATM 2009, 8th USA/Europe Air Traffic Management Research and Development Seminar*. Napa, United States, juin 2009. URL : https://hal-enac.archives-ouvertes.fr/hal-00940951.

[Dou+12] Nourelhouda Dougui et al. « A Light-Propagation Model for Aircraft Trajectory Planning ». In : *Journal of Global Optimization* (15 mar. 2012). DOI : 10.1007/s10898-012-9896-1. URL : http://recherche.enac.fr/~mongeau/LPA-revise-auteurs.pdf.

[DP00] Daniel Delahaye et Stéphane Puechmorel. « Air Traffic Complexity : towards Intrinsic Metrics ». In : 3rd USA/Europe Air Traffic Management R&D Seminar. Eurocontrol. Napoli, Italy, 13–16 juin 2000. URL : http://atmseminar.eurocontrol.fr/past-seminars/3rd-seminar-napoli-italy-june-2000/papers/paper_012.

[DP09] Luis Delgado et Xavier Prats. « Fuel consumption assessment for speed variation concepts during the cruise phase ». In : *Proceedings of the Conference on Air Traffic Management (ATM) Economics*. 2009.

[DP10] Daniel Delahaye et Stéphane Puechmorel. « Air traffic complexity based on dynamical systems ». In : *49th IEEE Conference on Decision and Control (CDC)*. Atlanta, États-Unis, déc. 2010, p. 2069–2074. DOI : 10.1109/CDC.2010.5718004. URL : http://hal-enac.archives-ouvertes.fr/hal-00938405.

[DSN15] Direction des services de la Navigation aérienne, éd. *ERATO Electronic Environment*. 5 mar. 2015. URL : http://dsnaservices.com/portfolio/erato/ (visité le 21/12/2016).

[Ehr05] Rüdiger Ehrmanntraut. « The Potential of Lateral Offset for the Multi Sector Planner ». In : *The 24th Digital Avionics Systems Conference, 2005. DASC 2005*. T. 1. Oct. 2005, 3.B.1–1. ISBN : 0-7803-9307-4. DOI : 10.1109/DASC.2005.1563341.

[Emi] Emirates. *Emirates A380 Specifications*. URL : http://www.emirates.com/english/flying/our_fleet/emirates_a380/emirates_a380_specifications.aspx (visité le 25/08/2015).

[Eura] Eurocontrol. *Free Flight Airspace*. URL : https://ext.eurocontrol.int/lexicon/index.php/Free_Flight_Airspace (visité le 07/12/2016).

[Eurb] Eurocontrol, éd. *Network Operations*. URL : http://www.eurocontrol.int/network-operations (visité le 10/01/2017).

[Eur08] Eurocontrol. *Mediterranean Free Flight*. 18 jan. 2008. URL : https://www.eurocontrol.int/eec/public/standard_page/proj_MFF.html (visité le 07/12/2016).

[Eur11] Eurocontrol Experimental Centre, éd. *User Manual for the Base of Aircraft Data (BADA) Revision 3.9. EEC Technical/Scientific Report No. 11/03/08-08*. Rapp. tech. Avr. 2011. URL : http://www.eurocontrol.int/eec/gallery/content/public/document/eec/other_document/2011/EEC-Technical-Report-110308-08.pdf (visité le 22/03/2017).

[Eur14] Eurocontrol. *Aircraft Equipage Requirements in the European Commission IRs 1207/2011 and 1028/2014*. 26 sept. 2014. url : http://www.eurocontrol.int/spi-ir (visité le 09/11/2016).

[Eur16] Eurocontrol. *ACAS II overview*. 2016. url : http://www.eurocontrol.int/dossiers/acas-ii (visité le 12/12/2016).

[FAA10] Federal Aviation Administration. *14 CFR Part 91 : Automatic Dependent Surveillance – Broadcast (ADS–B) Out Performance Requirements To Support Air Traffic Control (ATC) Service ; Final Rule*. Mai 2010.

[FAA14a] Federal Aviation Administration. *Aeronautical Information Manual, Official Guide to Basic Flight Information and ATC Procedures*. Avr. 2014. url : http://www.faa.gov/atpubs.

[FAA14b] Federal Aviation Administration. *Air Traffic Control System Command Center (ATCSCC)*. 25 fév. 2014. url : https://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/systemops/system_ops/atcscc/ (visité le 23/02/2017).

[Fei+05] Karen M. Feigh et al. « Analyzing air traffic management systems using agent-based modeling and simulation ». In : *6th USA/Europe Air Traffic Management Research and Development (ATM R&D) Seminar*. Georgia Institute of Technology, 2005.

[Fer99] Jacques Ferber. *Multi-agent systems : an introduction to distributed artificial intelligence*. T. 1. Addison-Wesley Reading, 1999.

[Fra+10] Jean-Sébastien Franco et al. *Programmation sur Carte Graphique. Introduction*. Laboratoire Bordelais de Recherche en Informatique. 2010. url : http://www.labri.fr/perso/granier/Cours/LP/2010-2011/07_GLSL.pdf (visité le 20/03/2017).

[Fra+99] E. Frazzoli et al. *Resolution of conflicts involving many aircraft via semidefinite programming*. Rapp. tech. Massachusetts Institute of Technology, avr. 1999.

[Gär99] Bernd Gärtner. « Fast and robust smallest enclosing balls ». In : *European Symposium on Algorithms*. Springer. 1999, p. 325–338. url : https://people.inf.ethz.ch/gaertner/subdir/texts/own_work/esa99_final.pdf.

[GDA97] Géraud Granger, Nicolas Durand et Jean-Marc Alliot. « Optimal resolution of en route conflicts ». In : *AGARD MSP 1997, Workshop on Air traffic management*. Budapest, Hongrie, 27–29 mai 1997. url : http://hal-enac.archives-ouvertes.fr/hal-00938232.

[Geo+03] J. P. Georgé et al. « Real-time Simulation for Flood Forecast : an Adaptive Multi-Agent System STAFF ». In : *AISB'03 Symposium on Adaptive Agents and Multi-Agent Systems(AAMAS'03)*. Sous la dir. de D. Kazakov, D. Kudenko et E. Alonso. University of Wales, Aberystwyth : SSAISB, avr. 2003, p. 109–114.

[Gir+13] Brunilde Girardet et al. « Generating optimal aircraft trajectories with respect to weather conditions ». In : *ISIATM 2013, 2nd International Conference on Interdisciplinary Science for Innovative Air Traffic Management*. Toulouse, France, juil. 2013. url : https://hal-enac.archives-ouvertes.fr/hal-00867818.

[Gir14] Brunilde Girardet. « Generating optimal and global aircraft trajectories with respect to weather conditions ». Theses. INSA de Toulouse, déc. 2014. URL : https://tel.archives-ouvertes.fr/tel-01127339.

[GKH11] Maxime Gariel, Fabrice Kunzi et R. John Hansman. « An algorithm for conflict detection in dense traffic using ADS-B ». In : *Digital Avionics Systems Conference (DASC), 2011 IEEE/AIAA 30th*. 16–20 oct. 2011. ISBN : 978-1-61284-797-9. DOI : 10.1109/DASC.2011.6095916. URL : http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6095916&isnumber=6095898.

[Gre00] Steven M. Green. « En route Spacing Tool : Efficient Conflict-free Spacing to Flow-Restricted Airspace ». In : *ATM2000 3rd USA/Europe Air Traffic Management R&D Seminar, Napolo*. 2000.

[GSF11] M. Gariel, A.N. Srivastava et E. Feron. « Trajectory Clustering and an Application to Airspace Monitoring ». In : *Intelligent Transportation Systems, IEEE Transactions on* 12.4 (déc. 2011), p. 1511–1524. ISSN : 1524-9050. DOI : 10.1109/TITS.2011.2160628.

[Guy14] Laureline Guys. « Planification de Trajectoires d'Avions sans Conflit : Fonctions Biharmoniques et Fonction de Navigation Harmonique ». Theses. Université Toulouse 3 Paul Sabatier, oct. 2014. URL : https://hal-enac.archives-ouvertes.fr/tel-01084037.

[HH16] Anca Hamuraru et Vincent Hindriksen. *Atomic operations for floats in OpenCL – improved*. Stream Computing. 9 fév. 2016. URL : https://streamcomputing.eu/blog/2016-02-09/atomic-operations-for-floats-in-opencl-improved/ (visité le 21/03/2017).

[Hil+05] J. C. Hill et al. « A multi-agent system architecture for distributed air traffic control ». In : *AIAA Guidance, Navigation and Control Conference*. 2005.

[Hil04] Brian Hilburn. « Cognitive complexity in air traffic control : A literature review ». In : *EEC note* 4.04 (2004).

[HRV01] JM Hoekstra, RCJ Ruigrok et RNHW Van Gent. « Free flight in a crowded airspace ? » In : *Progress in Astronautics and Aeronautics* 193 (2001), p. 533–546.

[Hua+14] Shimeng Huang et al. « Compact Configuration of Aircraft Flows at Intersections ». In : *IEEE Transactions on Intelligent Transportation Systems* 15.2 (avr. 2014), p. 771–783. ISSN : 1524-9050. DOI : 10.1109/TITS.2013.2287205.

[Hur+16] Christophe Hurter et al. « CAP : Collaborative advanced planning, trade-off between airspace management and optimized flight performance : Demonstration of En-Route reduced airspace congestion through collaborative flight planning ». In : *DASC 2016, IEEE/AIAA 35th Digital Avionics Systems Conference*. Digital Avionics Systems Conference (DASC), 2016 IEEE/AIAA 35th. Sacramento, United States : IEEE, sept. 2016. DOI : 10.1109/DASC.2016.7777947. URL : https://hal-enac.archives-ouvertes.fr/hal-01465908.

[Hur14] Christophe Hurter. « Toward image based algorithm to support interactive data exploration ». Accreditation to supervise research. Université Toulouse 3, nov. 2014. URL : https://tel.archives-ouvertes.fr/tel-01132020.

[HVR98] J Hoekstra, RNHW Van Gent et R Ruigrok. « Conceptual design of free flight with airborne separation assurance ». In : *Guidance, Navigation, and Control Conference and Exhibit*. 1998, p. 4239.

[IAT16] International Air Transport Association. *IATA Forecasts Passenger Demand to Double Over 20 Years*. 18 oct. 2016. URL : http://www.iata.org/pressroom/pr/Pages/2016-10-18-02.aspx.

[ICA] International Civil Aviation Organization. *ICAO FIR World*. URL : http://gis.icao.int/FIRWORLD/ (visité le 09/11/2016).

[ICA13a] International Civil Aviation Organization. *Global Air Transport Outlook to 2030 and Trends to 2040*. ICAO circular. International Civil Aviation Organization, 2013. ISBN : 978-92-9249-187-1.

[ICA13b] International Civil Aviation Organization. *Global Operational Data Link Document (GOLD)*. Avr. 2013. URL : http://www.icao.int/APAC/Documents/edocs/GOLD_2Edition.pdf.

[ICA15] International Civil Aviation Organization. *Automatic Dependant Surveillance – Broadcast Seminar and Fourteenth Meeting of Automatic Dependant Surveillance – Broadcast (ADS-B) Study and Implementation Task Force (ADS-B SITF/14)*. 9 avr. 2015. URL : http://www2010.icao.int/APAC/Meetings/2015%20ADSBSITF14/IP11_USA%20AI.4%20-%20Differences%20in%20ADS-B%20requirements_final.pdf (visité le 02/12/2016).

[IJt+15] Martijn IJtsma et al. « Computational assessment of different air-ground function allocations ». In : *11th USA/Europe Air Traffic Management R&D Seminar*. 2015.

[Ind] Indra, éd. *Automation and simulation > Air Traffic Control Automation System*. URL : http://www.indracompany.com/en/air-traffic-control-automation-system (visité le 21/12/2016).

[Ing89] Lester Ingber. « Very fast simulated re-annealing ». In : *Mathematical and computer modelling* 12.8 (1989), p. 967–973.

[ISD14] Doug Isaacson, Alexander Sadovski et Damek Davis. « Tactical Scheduling for Precision Air Traffic Operations : Past Research and Current Problems ». In : *Journal of Aerospace Information Systems* 11.4 (2014), p. 234–257.

[JLM03] Ali Jadbabaie, Jie Lin et Stephen A. Morse. « Coordination of Groups of Mobile Autonomous Agents Using Nearest Neighbor Rules ». In : *Automatic Control, IEEE Transactions on* 48 (6 2003), p. 988–1001. ISSN : 0018-9286. DOI : 10.1109/TAC.2003.812781. URL : http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1205192&isnumber=27134.

[Jor13] Tom Jorquera. « An adaptive multi-agent system for self-organizing continuous optimization ». Thèse de doct. Université de Toulouse, Université Toulouse III-Paul Sabatier, 2013.

[KGV83] S. Kirkpatrick, C. D. Gelatt et M. P. Vecchi. « Optimization by Simulated Annealing ». In : *Science* 220.4598 (1983), p. 671–680. issn : 00368075, 10959203. url : http://www.jstor.org/stable/1690046.

[Kos+97] J. Kosecka et al. « Generation of conflict resolution manoeuvres for air traffic management ». In : *Intelligent Robots and Systems, 1997. IROS '97., Proceedings of the 1997 IEEE/RSJ International Conference on.* T. 3. Sept. 1997, 1598–1603 vol.3. doi : 10.1109/IROS.1997.656571.

[Kos+98] Jana Kosecka et al. « 2 ½ D conflict resolution maneuvers for ATMS ». In : *Proceedings of the 37th IEEE Conference on Decision and Control.* T. 3. Tampa, FL, 16–18 déc. 1998, p. 2650–2655. isbn : 0-7803-4394-8. doi : 10.1109/CDC.1998.757853.

[KY00] James K. Kuchar et Lee C. Yang. « A Review of Conflict Detection and Resolution Modeling Methods ». In : *IEEE Transactions on Intelligent Transportation Systems* 1.4 (déc. 2000), p. 179–189. issn : 1524-9050. doi : 10.1109/6979.898217.

[Lan+15] François Lancelot et al. « Human-in-the-Loop Multi-agent Approach for Airport Taxiing Operations ». In : *Trends in Practical Applications of Agents, Multi-Agent Systems and Sustainability The PAAMS Collection.* Advances in Intelligent Systems and Computing 372. Springer, jan. 2015, p. 235–236. doi : 10.1007/978-3-319-19629-9_29. url : https://hal-enac.archives-ouvertes.fr/hal-01403426.

[Leg03] François Legras. « Organisation dynamique d'équipes d'engins autonomes par écoute flottante ». Thèse de doct. ONERA - Office national d'études et de recherches aérospatiales, 4 déc. 2003. url : http://perso.telecom-bretagne.eu/francoislegras/publications/index.php?idpublication=6852.

[LFM10] Mircea Lupu, Eric Feron et Zhi-Hong Mao. « Traffic complexity of intersecting flows of aircraft under variations of pilot preferences in maneuver choice ». In : *49th IEEE Conference on Decision and Control (CDC).* Atlanta, GA, 15–17 déc. 2010, p. 1189–1194. isbn : 978-1-4244-7745-6. doi : 10.1109/CDC.2010.5717044.

[LFM11] Mircea F. Lupu, Eric Feron et Zhi-Hong Mao. « Influence of Aircraft Maneuver Preference Variability on Airspace Usage ». In : *IEEE Transactions on Intelligent Transportation Systems* 12.4 (déc. 2011), p. 1446–1461. issn : 1524-9050. doi : 10.1109/TITS.2011.2159267.

[Lia+16] Man Liang et al. « Multi-layer Point Merge System for Dynamically Controlling Arrivals on Parallel Runways ». In : *DASC 2016, 35th Digital Avionics Systems Conference.* IEEE. Sacramento, United States, sept. 2016. url : https://hal-enac.archives-ouvertes.fr/hal-01355074.

[LR12] Kiran Lokhande et Hayley J Davison Reynolds. « Cognitive workload and visual attention analyses of the air traffic control tower flight data manager (TFDM) prototype demonstration ». In : *Proceedings of the Human Factors and Ergonomics Society Annual Meeting.* T. 56. 1. Sage Publications. 2012, p. 105–109.

[LZG04] Florin Leon, Mihai Horia Zaharia et Dan Gâlea. « A New Approach in Agent Path-Finding using State Mark Gradients ». In : *Computer Science Journal of Moldova* 12.3 (2004), p. 406–423.

[Ma+16a] Ji Ma et al. « Integrated Optimization of Terminal Manoeuvring Area and Airport ». In : *6th SESAR Innovation Days (2016)* . Sous la dir. d'EUROCONTROL. Proceedings of the SESAR Innovation Days 2016. Delft, Netherlands, nov. 2016. URL : https://hal-enac.archives-ouvertes.fr/hal-01404006.

[Ma+16b] Ji Ma et al. « Merging Flows in Terminal Moneuvering Area using Time Decomposition Approach ». In : *7th International Conference on Research in Air Transportation (ICRAT 2016)*. Philadelphie, PA, United States, juin 2016. URL : https://hal-enac.archives-ouvertes.fr/hal-01343823.

[Maa+16] Jerom Maas et al. « The effect of swarming on a voltage potential-based conflict resolution algorithm ». In : *submitted to the 7th International Conference on Research in Air Transportation*. 2016.

[Mao+05] Zhi-Hong Mao et al. « Stability of intersecting aircraft flows using heading-change maneuvers for conflict avoidance ». In : *IEEE Transactions on Intelligent Transportation Systems* 6.4 (5 déc. 2005), p. 357–369. ISSN : 1524-9050. DOI : 10.1109/TITS.2005.858789.

[MCA11] Caroline Martin, Julien Cegarra et Philippe Averty. « Analysis of Mental Workload during En-route Air Traffic Control Task Execution Based on Eye-Tracking Technique ». In : *Engineering Psychology and Cognitive Ergonomics : 9th International Conference, EPCE 2011, Held as Part of HCI International 2011, Orlando, FL, USA, July 9-14, 2011. Proceedings*. Sous la dir. de Don Harris. Berlin, Heidelberg : Springer Berlin Heidelberg, 2011, p. 592–597. ISBN : 978-3-642-21741-8. DOI : 10.1007/978-3-642-21741-8_63. URL : http://dx.doi.org/10.1007/978-3-642-21741-8_63.

[MDF07] Zhi-Hong Mao, David Dugail et Eric Feron. « Space Partition for Conflict Resolution of Intersecting Flows of Mobile Agents ». In : *IEEE Transactions on Intelligent Transportation Systems* 8.3 (4 sept. 2007), p. 512–527. ISSN : 1524-9050. DOI : 10.1109/TITS.2007.902646.

[Met+53] Nicholas Metropolis et al. « Equation of state calculations by fast computing machines ». In : *The journal of chemical physics* 21.6 (1953), p. 1087–1092.

[MF01] Zhi-Hong Mao et Eric Feron. « Stability and performance of intersecting aircraft flows under sequential conflict resolution ». In : *Proceedings of the 2001 American Control Conference*. T. 2. Arlington, VA : IEEE, 25–27 juin 2001, p. 722–729. ISBN : 0-7803-6495-3. DOI : 10.1109/ACC.2001.945800.

[MFB00] Zhi-Hong Mao, Eric Feron et Karl Bilimoria. « Stability of intersecting aircraft flows under decentralized conflict avoidance rules. Fluid Dynamics and Co-located Conferences ». In : *18th Applied Aerodynamics Conference*. American Institute of Aeronautics et Astronautics, 14 août 2000. DOI : 10.2514/6.2000-4271.

[MFB01] Zhi-Hong Mao, Eric Feron et Karl Bilimoria. « Stability and performance of intersecting aircraft flows under decentralized conflict avoidance rules ». In : *IEEE Transactions on Intelligent Transportation Systems* 2.2 (juin 2001), p. 101–109. ISSN : 1524-9050. DOI : 10.1109/6979.928721.

[MHF12] Aude Marzuoli, Christophe Hurter et Éric Féron. « Data visualization techniques for airspace flow modeling ». In : *CIDU 2012, Conference on Intelligent Data Understanding*. Boulder, United States, oct. 2012, p. 79–86. DOI : 10.1109/CIDU.2012.6382187. URL : https://hal-enac.archives-ouvertes.fr/hal-01022432.

[Min11] Ministère de l'Environnement, de l'Énergie et de la Mer, éd. *Redevances de navigation aérienne*. 26 oct. 2011. URL : http://www.developpement-durable.gouv.fr/Redevances-de-navigation-aerienne.html (visité le 12/01/2017).

[Min15a] Ministère de l'Environnement, de l'Énergie et de la Mer, éd. *4-Flight : prêt pour le déploiement*. 12 juin 2015. URL : http://www.developpement-durable.gouv.fr/4-Flight-pret-pour-le-deploiement.html (visité le 21/12/2016).

[Min15b] Ministère de l'Environnement, de l'Énergie et de la Mer, éd. *Environnement Electronique ERATO*. 17 déc. 2015. URL : http://www.developpement-durable.gouv.fr/Environnement-Electronique-ERATO,38503.html (visité le 21/12/2016).

[MMH02] Lik Mui, Mojdeh Mohtashemi et Ari Halberstadt. « Notions of Reputation in Multi-Agents Systems : A Review ». In : *AAMAS '02 Proceedings of the first international joint conference on Autonomous agents and multiagent systems : part 1*. (15–19 juil. 2002). ACM. Bologna, Italy : International Foundation for Autonomous Agents et Multiagent Systems, 2002, p. 280–287. ISBN : 1-58113-480-0. DOI : 10.1145/544741.544807. URL : http://dl.acm.org/citation.cfm?id=544807.

[NBA] National Business Aviation Association. *Miles-in-Trail (MIT)/Minutes-in-Trail (MINIT)*. URL : https://www.nbaa.org/ops/airspace/tfm/tools/mit.php (visité le 02/12/2016).

[NBD03] Minh Nguyen-Duc, Jean-Pierre Briot et Alexis Drogoul. « An application of multi-agent coordination techniques in air traffic management ». In : *IAT 2003. IEEE/WIC International Conference on Intelligent Agent Technology*. 13–16 oct. 2003, p. 622–625. ISBN : 0-7695-1931-8. DOI : 10.1109/IAT.2003.1241159.

[Nex16] U.S. Next Generation Air Transportation System. *NextGen Works*. 23 fév. 2016. URL : https://www.faa.gov/nextgen/works/ (visité le 12/12/2016).

[Nie95] William P. Niedringhaus. « Stream Option Manager (SOM) : automated integration of aircraft separation, merging, stream management, and other air traffic control functions ». In : *IEEE Transactions on Systems, Man and Cybernetics* 25.9 (sept. 1995), p. 1269–1280. ISSN : 0018-9472. DOI : 10.1109/21.400505.

[NMD15] NMD/NSD Operations Roadmap Team. *European Free Route Airspace Developments*. Version 1.0. Eurocontrol, 16 mar. 2015. URL : http://www.eurocontrol.int/sites/default/files/publication/files/free-route-airspace-developments.pdf.

[NP16] Florence Nicol et Stéphane Puechmorel. « Unsupervised aircraft trajectories clustering : a minimum entropy approach ». In : *ALLDATA 2016*. Lisbonne, Portugal, fév. 2016. URL : https://hal-enac.archives-ouvertes.fr/hal-01367590.

[Olf06] Reza Olfati-Saber. « Flocking for Multi-Agent Dynamic Systems : Algorithms and Theory ». In : *Automatic Control, IEEE Transactions on* 51 (3 2006), p. 401–420. ISSN : 0018-9286. DOI : 10.1109/TAC.2005.864190. URL : http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1605401&isnumber=33736.

[Ora14] Oracle. *Code source de la classe HashMap de Java 8*. 4 mar. 2014. URL : http://hg.openjdk.java.net/jdk8/jdk8/jdk/file/687fd7c7986d/src/share/classes/java/util/HashMap.java (visité le 07/03/2017).

[Ora15] Oracle. *Tutoriel sur l'opération de réduction de l'API Stream de Java 8*. 2015. URL : https://docs.oracle.com/javase/tutorial/collections/streams/reduction.html (visité le 12/03/2017).

[Ora16] Oracle. *Documentation de la classe HashMap de Java 8*. 2016. URL : https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html (visité le 07/03/2017).

[Pec+06] Michal Pechoucek et al. « Autonomous agents for air-traffic deconfliction ». In : *Proceedings AAMAS-06 – Industry Track*. ACM Press, 2006, p. 1498–1505.

[Pey12] Clément Peyronne. « Modélisation mathématique et résolution automatique de conflits par algorithmes génétiques et par optimisation locale continue ». Theses. Université Paul Sabatier - Toulouse III, déc. 2012. URL : https://tel.archives-ouvertes.fr/tel-00855296.

[Pic05] Gauthier Picard. *Cooperative Agent Model Instantiation to Collective Robotics*. IRIT. 2005. URL : http://www.irit.fr/ESAW04/PAPERS/ESAW04_Paper15.pdf.

[Pic14] Gauthier Picard. « Adaptive multi-agent systems ». Habilitation à diriger des recherches. Université Jean Monnet, déc. 2014. URL : https://hal.archives-ouvertes.fr/tel-01122053.

[PN15] Stéphane Puechmorel et Florence Nicol. « Entropy minimizing curves with application to automated flight path design ». Juin 2015. URL : https://hal.archives-ouvertes.fr/hal-01162820.

[PN16] Stéphane Puechmorel et Florence Nicol. « Entropy minimizing curves with application to flight path design and clustering ». working paper or preprint. Juil. 2016. URL : https://hal.archives-ouvertes.fr/hal-01349675.

[Pri+02] A.R. Pritchett et al. « Examining air transportation safety issues through agent-based simulation incorporating human performance models ». In : *Digital Avionics Systems Conference, 2002. Proceedings. The 21st.* T. 2.7A5. Oct. 2002, p. 1–13. DOI : 10.1109/DASC.2002.1052917.

[PS09] Michal Pechoucek et David Sislak. « Agent-Based Approach to Free-Flight Planning, Control, and Simulation ». In : *Intelligent Systems, IEEE* 24 (1 2009), p. 14–17. ISSN : 1541-1672. DOI : 10.1109/MIS.2009.1. URL : http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4763649&isnumber=4763642.

[Qua+08] Markus Quaritsch et al. « Collaborative Microdrones : Applications and Research Challenges ». In : *Proceedings of the 2Nd International Conference on Autonomic Computing and Communication Systems*. Autonomics '08. Turin, Italy : ICST (Institute for Computer Sciences, Social-Informatics et Telecommunications Engineering), 2008, 38 :1–38 :7. ISBN : 978-963-9799-34-9. URL : http://dl.acm.org/citation.cfm?id=1487652.1487690.

[RMB13] C.G. Rieger, K.L. Moore et T.L. Baldwin. « Resilient control systems : A multi-agent dynamic systems perspective ». In : *Electro/Information Technology (EIT), 2013 IEEE International Conference on*. Mai 2013, p. 1–16. DOI : 10.1109/EIT.2013.6632721.

[Rod+14] O. Rodionova et al. « North Atlantic Aircraft Trajectory Optimization ». In : *Intelligent Transportation Systems, IEEE Transactions on* 15.5 (oct. 2014), p. 2202–2212. ISSN : 1524-9050. DOI : 10.1109/TITS.2014.2312315.

[RVH99] RCJ Ruigrok, RNHW Van Gent et JM Hoekstra. *The transition towards free flight : a human factors evaluation of mixed equipage, integrated air-ground, free flight ATM scenarios*. Rapp. tech. SAE Technical Paper, 1999.

[Sal+12] Erwan Salaün et al. « Aircraft Proximity Maps Based on Data-Driven Flow Modeling ». In : *Journal of Guidance, Control, and Dynamics* 37.2 (mar. 2012), p. 563–577. ISSN : 0731-5090. DOI : 10.2514/1.53859.

[Sel14] Selex ES, éd. *Air Traffic Management*. 2014. URL : http://www.us.selex-es.com/capabilities/atm (visité le 21/12/2016).

[Ser+15] Marina Sergeeva et al. « 3D Sectors Design by Genetic Algorithm Towards Automated Sectorisation ». In : *5th SESAR Innovation days*. Bologna, Italy, déc. 2015. URL : https://hal-enac.archives-ouvertes.fr/hal-01240312.

[SES16] Single European Sky ATM Research Joint Undertaking. *Objectives*. 2016. URL : http://www.sesarju.eu/discover-sesar/objectives (visité le 09/11/2016).

[Sis13] David Sislak. « Agent-Based Approach to Air-Traffic Modeling, Simulation and Collision Avoidance ». Thèse de cand. Czech Technical University in Prague, Faculty of Electrical Engineering, jan. 2013. 131 p. URL : https://dspace.cvut.cz/bitstream/handle/10467/14183/Habilitace_Sislak_2013.pdf?sequence=1.

[SSP08] David Sislak, Jiri Samek et Michal Pechoucek. « Decentralized Algorithms for Collision Avoidance in Airspace ». In : *AAMAS '08 Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*. T. 2. International Foundation for Autonomous Agents et Multiagent Systems, 2008, p. 543–550. ISBN : 978-0-9817381-1-6. URL : http://dl.acm.org/citation.cfm?id=1402301.

[Sun+15] Emmanuel Sunil et al. « Metropolis : Relating airspace structure and capacity for extreme traffic densities ». In : *ATM seminar 2015, 11th USA/EUROPE Air Traffic Management R&D Seminar*. 2015.

[SV01] James A Sethian et Alexander Vladimirsky. « Ordered upwind methods for static Hamilton–Jacobi equations ». In : *Proceedings of the National Academy of Sciences* 98.20 (2001), p. 11069–11074.

[SVP09] David Sislak, Premysl Volf et Michal Pechoucek. « Accelerated A* Path Planning ». In : *AAMAS '09 Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*. International Foundation for Autonomous Agents et Multiagent Systems, 2009, p. 1133–1134. isbn : 978-0-9817381-7-8. url : http://dl.acm.org/citation.cfm?id=1558176.

[ŠVP09] David Šišlák, Premysl Volf et Michal Pechoucek. « Flight trajectory path planning ». In : *Proc. of Intl. Scheduling and Planning Applications Workshop*. 2009, p. 76–83.

[ŠVP11] David Šišlák, Premysl Volf et Michal Pechoucek. « Agent-Based Cooperative Decentralized Airplane-Collision Avoidance ». In : *IEEE Transactions on Intelligent Transportation Systems* 12.1 (mar. 2011), p. 36–46. issn : 1524-9050. doi : 10.1109/TITS.2010.2057246.

[TA07] Kagan Tumer et Adrian Agogino. « Distributed agent-based air traffic flow management ». In : *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*. AAMAS '07. Honolulu, Hawaii : ACM, mai 2007. isbn : 978-81-904262-7-5. doi : 10.1145/1329125.1329434.

[TDN14] Tambet Treimuth, Daniel Delahaye et Sandra Ulrich Ngueveu. « Resectorisation d'espace aérien ». In : *ROADEF - 15ème congrès annuel de la Société française de recherche opérationnelle et d'aide à la décision*. Société française de recherche opérationnelle et d'aide à la décision. Bordeaux, France, fév. 2014. url : https://hal.archives-ouvertes.fr/hal-00946367.

[Tha] Thales, éd. *Gestion du trafic aérien*. url : https://www.thalesgroup.com/fr/global/activites/aeronautique/air-traffic-management (visité le 21/12/2016).

[TM08] Kyle Treleaven et Zhi-Hong Mao. « Conflict Resolution and Traffic Complexity of Multiple Intersecting Flows of Aircraft ». In : *IEEE Transactions on Intelligent Transportation Systems* 9.4 (déc. 2008), p. 633–643. issn : 1524-9050. doi : 10.1109/TITS.2008.2006771.

[TPS97] Claire Tomlin, George J. Pappas et Shankar Sastry. « Noncooperative conflict resolution [air traffic management] ». In : *Proceedings of the 36th IEEE Conference on Decision and Control*. T. 2. San Diego, CA, 10–12 déc. 1997, p. 1816–1821. isbn : 0-7803-4187-2. doi : 10.1109/CDC.1997.657827.

[Tre+15] Tambet Treimuth et al. « Parallel complexity computation based on dynamical systems ». In : *DASC, 2015 IEEE/AIAA 34th Digital Avionics Systems Conference*. .1C2. Digital Avionics Systems Conference (DASC), 2015 IEEE/AIAA 34th. Prague, Czech Republic : IEEE, sept. 2015, p. 1–8. isbn : 978-1-4799-8939-3. doi : 10.1109/DASC.2015.7311340. url : https://hal-enac.archives-ouvertes.fr/hal-01223896.

[ÜB94] Cem Ünsal et John S. Bay. « Spatial Self-Organization in Large Populations of Mobile Robots ». In : *Intelligent Control, 1994., Proceedings of the 1994 IEEE International Symposium on*. (16–18 août 1994). 1994, p. 249–254. isbn : 0-7803-1990-7. doi : 10.1109/ISIC.1994.367809. url : http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=367809&isnumber=8414.

[VB01] H. Van Dyke Parunak et Sven Brueckner. « Entropy and Self-Organization in Multi-Agent Systems ». In : *Proceedings of the International Conference on Autonomous Agents (Agents 2001)*. ERIM. 2001, p. 124–130.

[VDT17] Andrija Vidosavljevic, Daniel Delahaye et Vojin Tošić. « Homotopy Route Generation Model for Robust Trajectory Planning ». In : *Air Traffic Management and Systems II*. Sous la dir. d'ENRI. T. 420. Lecture Notes in Electrical Engineering. Springer, fév. 2017, p. 69–88. doi : 10.1007/978-4-431-56423-2_4Âů. url : https://hal-enac.archives-ouvertes.fr/hal-01234103.

[Vil04] Jacques Villiers. *Automatisation du contrôle de la circulation aérienne : "ERASMUS", une voie conviviale pour franchir le mur de la capacité*. T. 58. Études & documents. Institut du Transport Aérien, 2004. 60 p.

[Wel08] Jean-Baptiste Welcomme. « MASCODE : Un système multi-agent adaptatif pour concevoir des produits complexes. Application à la conception préliminaire avion ». Thèse de doct. IRIT – EADS Innovation Works, 5 mai 2008. url : http://tel.archives-ouvertes.fr/tel-00446144/.

[Wol+07] Shawn R. Wolfe et al. « Comparing Route Selection Strategies in Collaborative Traffic Flow Management ». In : *IAT '07, IEEE/WIC/ACM International Conference on Intelligent Agent Technology*. Fremont, CA, nov. 2007, p. 59–62. isbn : 978-0-7695-3027-7. doi : 10.1109/IAT.2007.54.

[Wol+09] Shawn R. Wolfe et al. « A Multi-Agent Simulation of Collaborative Air Traffic Flow Management ». In : *Multi-Agent Systems for Traffic and Transportation Engineering*. Sous la dir. d'Ana Bazzan et Franziska Klügl. 2009. Chap. 18, p. 357–381. doi : 10.4018/978-1-60566-226-8.ch018.

[WSJ08] Shawn R. Wolfe, Maarten Sierhuis et Peter A. Jarvis. « To BDI, or Not to BDI : Design Choices in an Agent-Based Traffic Flow Management Simulation ». In : *Proceedings of the 2008 Spring simulation multiconference*. Avr. 2008, p. 63–70.

[WVI04] Steven Wollkind, John Valasek et Thomas R. Ioerger. « Automated conflict resolution for air traffic management using cooperative multiagent negotiation ». In : *AIAA Guidance, Navigation, and Control Conference*. 2004.

[YD11] Jeff Yoo et Santosh Devasia. « Flow-capacity-maintaining, decentralized, conflict resolution with aircraft turn dynamics ». In : *American Control Conference (ACC)*. San Francisco, CA, juin 2011, p. 2759–2764. ISBN : 978-1-4577-0080-4.

[YD12] Jeff Yoo et Santosh Devasia. « Application of provably-safe conflict resolution for air traffic control ». In : *IEEE 51st Annual Conference on Decision and Control (CDC)*. 10 déc. 2012, p. 478–483. ISBN : 978-1-4673-2065-8. DOI : 10.1109/CDC.2012.6427035.

[YD13a] Jeff D. Yoo et Santosh Devasia. « Decoupled Conflict Resolution Procedures for Non-perpendicular Air Traffic Intersections with Different Speeds ». In : *52nd IEEE Conference on Decision and Control*. Firenze, 10–13 déc. 2013, p. 275–280. ISBN : 978-1-4673-5714-2. DOI : 10.1109/CDC.2013.6759894.

[YD13b] Jeff D. Yoo et Santosh Devasia. « On-demand Conflict Resolution Procedures for Air Traffic intersections ». In : *American Control Conference (ACC)*. Washington, DC, 17–19 juin 2013, p. 6322–6327. ISBN : 978-1-4799-0177-7.

[YD13c] Jeff D. Yoo et Santosh Devasia. « Provably Safe Conflict Resolution With Bounded Turn Rate for Air Traffic Control ». In : *IEEE Transactions on Control Systems Technology* 21.6 (21 nov. 2013), p. 2280–2289. ISSN : 1063-6536. DOI : 10.1109/TCST.2012.2236328.

[YD14] Jeff D. Yoo et Santosh Devasia. « On-Demand Conflict Resolution Procedures for Air-Traffic Intersections ». In : 2014, p. 1–12. DOI : 10.1109/TITS.2014.2301292. À para.

[YK97] Lee C Yang et James K Kuchar. « Prototype conflict alerting system for free flight ». In : *Journal of Guidance, Control, and Dynamics* 20.4 (1997), p. 768–773.

[Zho+16] Jun Zhou et al. « Optimal Design of SIDs/STARs in TMA Using Simulated Annealing ». In : *35th Digital Avionics Systems Conference, Enabling Avionics For UAS/UTM (UAS Traffic Management) (DASC 2016)*. Sacramento, CA, United States, sept. 2016. URL : https://hal-enac.archives-ouvertes.fr/hal-01379998.

Résumé

La gestion des flux de trafic aérien (ATFM) cherche à structurer le trafic de manière à réduire la congestion dans l'espace aérien. La congestion étant causée par les avions volant dans les mêmes portions de l'espace aérien en même temps, l'ATFM organise le trafic dans les dimensions spatiales (ex. le réseau de routes) et dans la dimension temporelle (ex. séquencement et fusion de flux d'avions atterrissant ou décollant aux aéroports).

L'objectif de cette thèse est de développer une méthodologie qui permet au trafic aérien de s'auto-structurer dans les dimensions spatiales et temporelle quand la demande est élevée. Cette structuration disparait quand la demande diminue. Pour remplir cet objectif, un système multi-agents a été développé, dans lequel les avions coopèrent pour structurer le trafic. Les systèmes multi-agents possèdent plusieurs avantages, incluant une bonne résilience aux perturbations, la résilience étant la capacité du système à modifier ses décisions de manière à retrouver un état stable après l'occurrence d'une perturbation dans son environnement.

Dans ce système, trois algorithmes sont implémentés, visant à réduire la complexité du trafic de trois manières différentes. Le premier algorithme permet aux agents avions volant sur un réseau de route de réguler leur vitesse de manière à réduire le nombre de conflits, un conflit se produisant quand deux avions ne respectent pas les normes de séparation. Le deuxième algorithme permet aux avions de résoudre les conflits quand le trafic n'est pas structuré par un réseau de routes. Le troisième algorithme crée des réseaux de routes locaux temporaires pour structurer le trafic.

Les trois algorithmes implémentés dans ce système multi-agents permet de réduire la complexité globale du trafic, qui devient plus simple à gérer pour les contrôleurs aériens. Ces algorithmes sont appliqués à des exemples réalistes et sont capables de structurer le trafic de manière résiliente.

Abstract

Air Traffic Flow Management (ATFM) aims at structuring traffic in order to reduce congestion in airspace. Congestion being linked to aircraft located at the same position at the same time, ATFM organizes traffic in the spatial dimension (e.g. route network) and in the time dimension (e.g. sequencing and merging of aircraft flows taking off or landing at airports).

The objective of this thesis is to develop a methodology that allows the traffic to self-organize in the time and space dimensions when demand is high. This structure disappears when the demand diminishes. In order to reach this goal, a multi-agent system has been developed, in which aircraft cooperate to structure traffic. Multi-agent systems have several advantages, including a good resilience when confronted with disruptive events, resilience being the ability of the system to adapt its decisions in order to get back to a stable state when confronted to a disruption in its environment.

In this system, three algorithms have been implemented, aiming at reducing traffic complexity in three different ways. The first algorithm allows aircraft agents flying on a route network to regulate speed in order to reduce the number of conflicts, a conflict occurring when two aircraft do not respect separation norms. The second algorithm allows aircraft to solve conflicts when the traffic is not structured by a route network. The third algorithm creates temporary local route networks allowing to structure traffic.

The three algorithms implemented in this multi-agent system allow to decrease overall traffic complexity, which becomes easier to manage by air traffic controllers. This algorithm was applied on realistic examples and was able to structure traffic in a resilient way.