Université Paul Sabatier Toulouse III
École nationale de l'aviation civile

Master Recherche Informatique et Télécommunications
parcours Intelligence Artificielle

Allocation de niveau de vol pour la déconfliction de trajectoires 4D

Auteure :
Caroline Becker

Maître de stage :
Nicolas Barnier

Pôle Planification, Optimisation et Modélisation du trafic aérien
Direction des Services de la Navigation Aérienne
Direction de la Technique et de l'Innovation, domaine R & D

Résumé : Pour améliorer l'écoulement des flux de trafic et l'efficacité du contrôle, nous proposons d'optimiser les trajectoires des avions afin d'éviter l'apparition de conflits. Nous nous focaliserons dans ce travail sur l'allocation de niveau de vol pour réaliser cette « déconfliction », de telle manière que les flux qui se croisent dans le plan horizontal utilisent des niveaux différents dans le plan vertical. Ce problème peut être modélisé comme une coloration de graphe et nécessite une détection efficace des croisements entre flux. Il est résolu par programmation par contraintes et par recherche locale en minimisant diverses fonctions de coût, telles que l'écart au niveau de vol demandé ou le nombre d'avions déviés.

Déconfliction - Coloration de graphe - Géométrie algorithmique
Programmation par contraintes - Recherche locale

30 août 2010

Remerciements

Je remercie Jean-Marc Alliot, Nicolas Durand et David Gianazza pour m'avoir à nouveau accueillie à la DTI et m'avoir, lors de mon précédent stage avec eux en 2007, montré qu'il n'y avait rien de mieux pour être entourée de personnes brillantes que de faire de la recherche.

Je remercie toutes les personnes qui ont cru en moi tout au long de ma scolarité. À l'ENAC, citons Pascal Brisset, qui était la gentillesse et le talent incarnés ; Christine Ricci, Hélène Gaspard-Boulinc et Cécile Moura pour m'avoir encadrée et soutenue ; Philippe Brunner, pour m'avoir fait aimer l'économie ; Estelle Malavolti, pour m'avoir toujours soutenue, et évidemment, Nicolas Barnier, pour m'avoir permis de suivre le master IA, m'avoir encadrée tout au long de ce stage et, au-delà, d'être une personne remarquable et une source d'inspiration. À l'IRIT, je remercie Claudette Cayrol pour sa gestion irréprochable du master IA.

Je remercie mes collègues de la DTI, que ce soit Simon Marchal pour le *constraint programming road trip* vers Aussoie, Richard Alligier car quand il y en a un qui fait de la programmation génétique ça va, c'est quand il y en a plusieurs qu'il y a des problèmes ; Charlie Vanaret, pour prouver qu'on peut être à la fois un ninja et un geek ; Mohammad Ghasemi Hamed, pour les discussions philosophiques et (surtout) les gâteaux ; Jean-Baptiste Laborde, grâce à qui nous avons chanté des jours heureux ; Cyril Allignol, pour avoir tout essayé pour promouvoir Emacs ; Nicolas Saporito, parce que « le graphe, c'est la vie » ; enfin, Jean-Baptiste Gotteland pour sa virtuosité à manier les acronymes. Je remercie les personnes des autres labos pour la compagnie au café et aux repas, à savoir l'ensemble du pôle PII (que je remercie aussi pour nous avoir utilisés comme cobayes ou présentés chaque semaine le stripping électronique) ainsi que les labos d'électronique, d'IHM, l'URI drones et le LOTA.

Je remercie Pierre-Selim Huard pour avoir été ma *hotline* sur la syntaxe LaTeX et la grammaire française pendant ce stage et être mon compagnon dans la vie. Je remercie toutes les personnes qui ont contribué à ce que j'en sois là aujourd'hui et que je n'ai pas encore citées, telles que Julien Nuncq pour m'avoir fait découvrir la programmation ou Gabriel Scherer pour m'avoir montré qu'on pouvait encore plus aimer la programmation fonctionnelle que moi.

Table des matières

Table des figures

Introduction

Depuis plus de cinquante ans, les services du contrôle de la navigation aérienne font face à un trafic aérien croissant[1]. Cette croissance, couplée à des exigences sociétales de plus en plus fortes en matière de sûreté, de ponctualité et de réduction des pollutions tant sonores qu'atmosphériques, conduit à des contraintes accrues pesant sur l'ensemble du monde aéronautique et notamment le contrôle aérien. L'entière automatisation de celui-ci relevant encore du domaine de l'utopie et l'amélioration des techniques de communication, navigation et surveillance (CNS) ne permettant pas de satisfaire les aspirations actuelles, de nouveaux champs de recherche doivent être explorés pour faire face à ces nouveaux défis.

Parmi ceux-ci, la déconfliction consiste en la modification en amont des vols, par exemple en modifiant l'heure de départ ou la trajectoire d'un avion, afin que le contrôleur[2] se retrouve devant un trafic « chanceux », c'est-à-dire où le nombre de conflits à résoudre est inhabituellement faible.

En raison de la politique de *hub* des compagnies aériennes, où la majorité des vols se font avec une seule correspondance dans un (très) gros aéroport, tel que Lyon-St Exupéry ou Paris-Charles de Gaulle pour Air France, le retard d'un avion peut perturber une partie importante du trafic de la journée. Retarder des avions au décollage provoque ainsi des surcoûts pour les compagnies aériennes difficilement évaluables. Ici, on s'intéressera plutôt à la modification de l'altitude, ou niveau de vol, d'un avion, afin que deux avions passant à la verticale du même point soient à des altitudes différentes ; le surcoût éventuel est alors lié à une consommation accrue de carburant et donc d'émission de CO_2, donc à des taxes supplémentaires.

Il a déjà été montré qu'il est possible de créer des situations de trafic sans conflits si on considèrait des avions volant « en ligne droite » entre leurs aéroports d'origine et de destination, en se ramenant à un problème de coloration de graphe à résoudre par programmation par contraintes, sous l'hypothèse que l'on sache gérer des « trains d'avions », c'est-à-dire que la séparation entre avions suivant la même trajectoire soit assurée. L'objectif est ici de partir d'une situation plus réaliste, c'est-à-dire où les avions suivent un plan de vol prédéterminé avec passage au-dessus de différents points au sol ou balises, tout en essayant de minimiser l'écart entre le niveau de vol souhaité par les compagnies aériennes, ou Requested Flight Level

[1] Hors éruption de volcan islandais ou attentat de masse.

[2] En France, le contrôle se fait toujours par équipe de deux, mais les nuances entre contrôleur *organique* et *radariste* n'est pas pertinente pour ce sujet. Pour en savoir plus, consulter le rapport du Bureau d'Enquêtes et d'Analyses pour la sécurité de l'aviation civile (BEA) d'avril 2005.

(RFL), et celui réellement utilisé.

Après une présentation du contexte de la gestion du trafic aérien (1), nous parlerons de géométrie algorithmique (2), qui est utilisée pour détecter efficacement les conflits entre flux aériens, puis de programmation par contrainte (3) et de coloration de graphes (4), utilisées pour résoudre et modéliser le problème ; nous parlerons des choix techniques effectués pour la résolution concrète (5) et enfin des résultats obtenus (6).

Première partie

Contexte

1 Gestion du trafic aérien

Le rôle principal du contrôle aérien est d'assurer la séparation, en vol et au sol, des aéronefs. Ceci doit être fait en répondant à de nombreuses exigences : l'espace aérien doit être partagé avec de nombreux autres acteurs que l'aviation civile, tels que l'aviation d'affaire ou l'aviation militaire ; les capacités des secteurs aériens ainsi que celles des aéroports sont limitées ; les compagnies aériennes cherchent à minimiser leurs coûts ; les conditions météorologiques sont changeantes et difficilement prévisibles. Actuellement, la saturation de l'espace aérien est telle qu'il est nécessaire de chercher à optimiser l'écoulement du trafic afin d'éviter des retards trop importants, que ce soit au décollage ou à l'atterrissage.

1.1 Organisation

Pour assurer ce rôle le plus efficacement possible, c'est-à-dire en offrant le maximum de flexibilité aux utilisateurs de l'espace aérien avec la meilleure ponctualité réalisable, il est nécessaire de planifier les différentes opérations permettant l'écoulement du trafic : c'est la gestion du trafic aérien (ATM pour *Air Traffic Management*), structurée en plusieurs phases qui gèrent les problèmes à différents horizons temporels. Celles-ci sont :

- Le *long terme*, plus de six mois avant le vol, phase durant laquelle sont organisées les routes aériennes en fonction des prévisions de trafic. C'est à cette période qu'est négocié le partage de l'espace aérien entre zones civiles et militaires[1]. Les prévisions se font notamment en fonction des plans de vol déjà déposés par les compagnies aériennes, la plupart d'entre elles les déposant de façon systématique tous les six mois.
- La *pré-régulation* consiste en l'organisation d'une journée de trafic donnée par la Central Flow Management Unit (CFMU). L'organisation se fait quelques jours à l'avance en fonction des plans de vol déposés par les compagnies et du contexte du moment (schéma d'ouverture des centres de contrôle, créneaux utilisés, conditions météorologiques, activation des secteurs militaires...). Le contexte peut être visualisé sur la figure 1.1.
- Le *niveau tactique* est dévolu au contrôleur aérien ; son rôle est de détecter les conflits

[1] L'avènement du Flexible Use of Airspace (FUA) a permis de passer d'un partage statique (telle zone est réservée par les militaires et aucun avion civil ne doit y entrer, quel que soit l'usage réel qui en est fait) à une séparation dynamique dépendant de l'activité militaire réelle.

potentiels à 15 minutes et de les éviter ; il est aussi en charge de la coordination entre les secteurs.

– Enfin, le *niveau d'urgence* sert à palier aux dysfonctionnements du système ; à ce niveau, on compte par exemple le Traffic alert and Collision Avoidance System (TCAS), système embarqué résolvant par manœuvre dans le plan vertical les conflits entre deux avions[2].

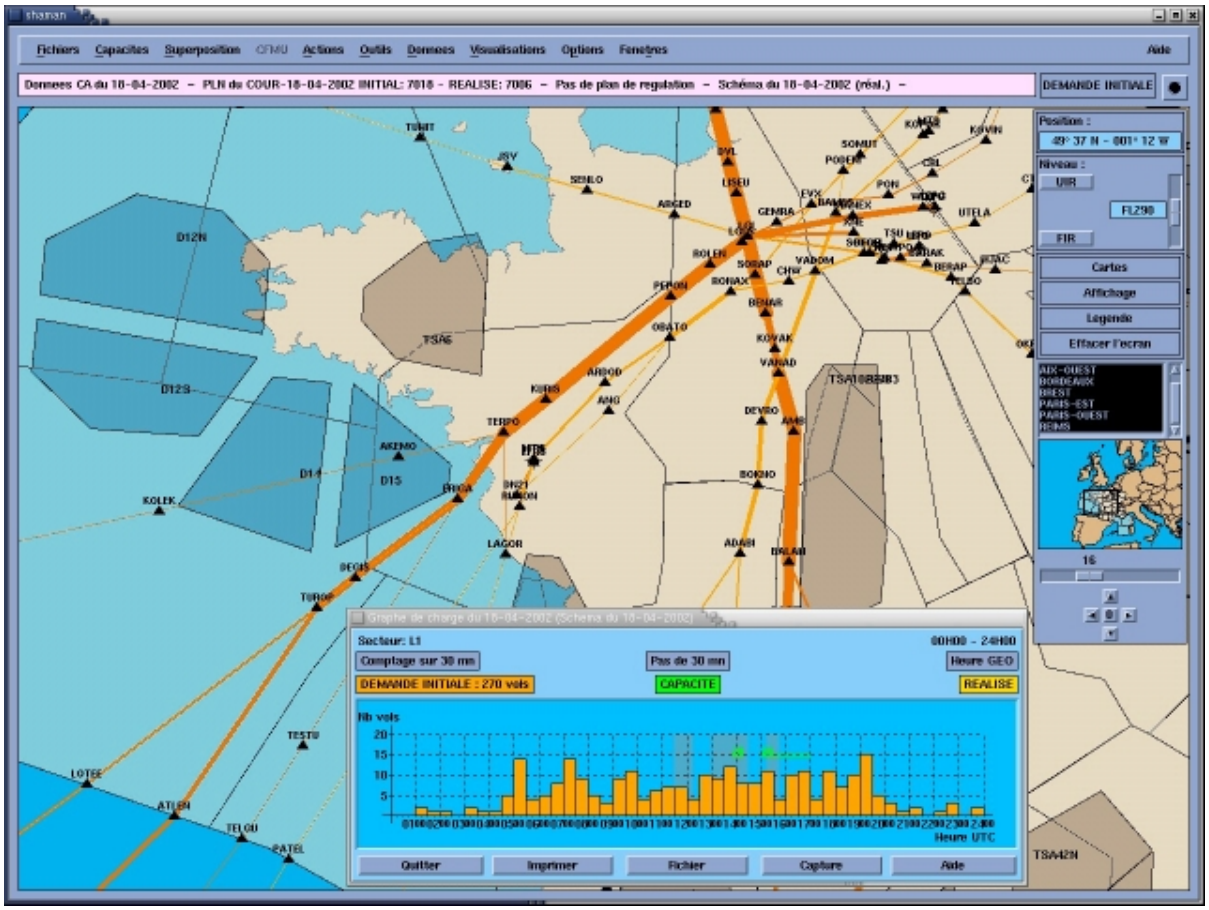

FIG. 1.1 – Visualisation de la charge d'un secteur aérien avec l'outil SHAMAN (System to Help Analysis and Monitoring of Acc resources and Air route Network). Les secteurs réservés aux militaires apparaissent en grisé et les routes aériennes en orange gras, avec les balises survolées représentées par un triangle.

[2]La collision d'Überlingen, en 2002, a été provoquée par des ordres contradictoires entre le TCAS et le contrôleur aérien. Depuis, l'OACI préconise de suivre les ordres du TCAS prioritairement à ceux du contrôle humain.

1.2 Situation actuelle

Le système actuel fonctionne globalement sans trop de retards attribuables au contrôle aérien[3]. Seulement, il ne pourra pas faire face à une augmentation du trafic, surtout si elle est couplée aux ambitieuses prévisions du projet européen SESAR (Single European Sky ATM Research) qui table en plus sur une augmentation de la sûreté couplée à une diminution des coûts, des retards et des émissions de CO_2.

Historiquement, les gains en capacité se sont faits selon plusieurs axes : le découpage de l'espace aérien en secteurs de plus en plus petits, ainsi que l'amélioration de l'avionique embarquée et des techniques de détection des aéronefs qui ont permis un suivi de trajectoire plus précis.

1.2.1 Normes de séparation

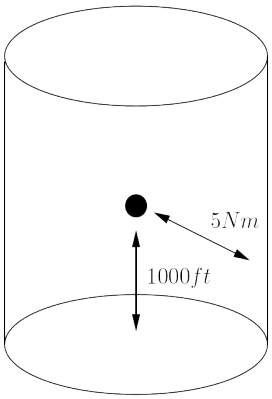

FIG. 1.2 – Normes de séparation en route

Les normes de séparation (représentées figure 1.2) sont liées à l'incertitude concernant la position d'un avion : il faut que deux avions séparés sur l'écran radar (voir figure 1.3), c'est-à-dire séparés dans la représentation mentale du contrôleur, aient des chances très faibles de se trouver en réalité au même endroit. Améliorer la précision des outils déterminant les coordonnées des avions, que ce soit à l'aide de GNSS (Global Navigation Satellite Systems) ou de radars, permet donc une norme de séparation plus fine. Seulement, il existe aussi une autre limite, liée cette fois-ci à la manœuvrabilité des appareils. Ainsi, de la même manière que sur l'autoroute en France, il faille rester à au moins l'équivalent de deux secondes[4] du véhicule qui nous précède, ce qui correspond à la perception-réaction additionnée du temps de réaction du matériel, il est nécessaire que deux aéronefs soient suffisamment éloignés pour laisser une marge de manœuvre en cas de problème. Il faut savoir que les 5 Nm de séparation

[3]Pour mars 2010, retard moyen de 26.1 minutes, avec moins de quatre vols sur dix retardés. Le tiers des retards étaient imputables au contrôle aérien[COfDA10].

[4]Le temps de réaction d'un automobiliste est d'environ 1.5 s et correspond à la durée s'écoulant entre la perception du danger par l'œil et l'action de freiner réellement réalisé par le pied. La demi seconde suivante correspond au temps de réaction du véhicule en lui-même.

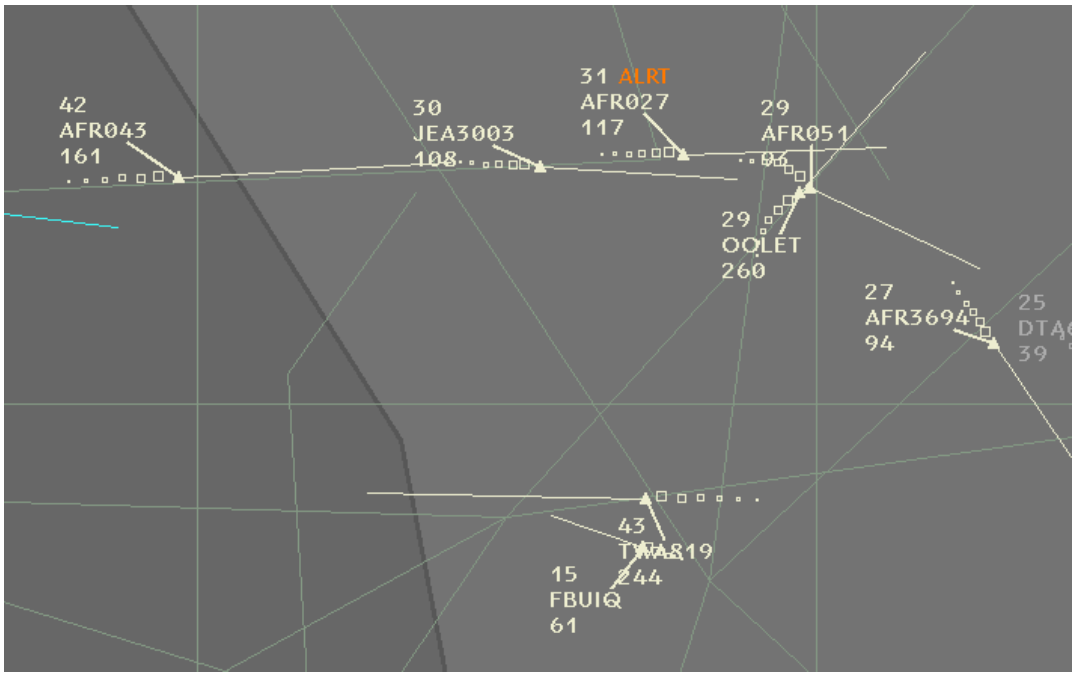

FIG. 1.3 – Capture d'écran de l'outil de visualisation ODS du contrôle aérien.

utilisés actuellement en croisière correspondent à 40 s de vol pour un avion volant à 450 kn, sachant que les avions de transport civile sont peu réactifs car leur stabilité est privilégiée par rapport à leur manœuvrabilité [5].

1.2.2 Taille des secteurs

Comme la charge mentale du contrôleur est corrélée au nombre d'avions qu'il a à traiter[6], il peut être intéressant de délimiter des portions d'espace plus petites dans lesquelles il y aura donc moins d'avions. C'est d'ailleurs le principe du regroupement/dégroupement de secteurs : la même zone aérienne pourra être gérée par une ou plusieurs équipes suivant la densité de trafic, qui varie au cours de la journée. Si de nombreuses recherches portent sur les secteurs, comme on peut le voir à la section 1.3.1, cette technique touche aussi ses limites. En effet, la tâche du contrôleur n'est pas uniquement de gérer la séparation des aéronefs, il doit aussi s'assurer de leur bonne « transmission » au secteur voisin, ce qui nécessite qu'il passe du temps à effectuer la coordination. Des secteurs plus petits signifient donc un temps plus grand consacré à la coordination. De plus, il est nécessaire que les avions restent suffisamment longtemps dans un secteur donné pour que le contrôleur ait le temps de détecter le conflit et

[5] Un avion manœuvrable répondra plus vite et plus intensément à une perturbation, qui peut être volontaire comme un changement de l'angle des gouvernes commandé par le pilote, ou involontaire, comme un changement d'incidence induit par une brusque variation du vent. Dans le cas du transport civile, le confort des passagers est privilégié et les avions sont donc aérodynamiquement stables. Le développement du pilotage électronique permet d'avoir une « illusion de stabilité » et pourrait permettre d'avoir des avions plus manœuvrables.

[6] D'autres paramètres sont à prendre en compte, comme cela sera vu dans les chapitres suivants.

de manœuvrer les avions afin de le résoudre[7].

1.3 Optimisation de la gestion du contrôle aérien

Il est donc nécessaire de trouver d'autres manières d'augmenter la capacité du contrôle aérien. L'automatisation totale n'étant pas envisageable, en raison notamment de la problématique de la gestion des évènements imprévus, l'accent se porte actuellement sur la diminution de la charge mentale des contrôleurs, couplée à une planification à long terme jouant sur l'attribution des créneaux de départ pour éviter la saturation des secteurs en vol.

1.3.1 Optimisation des secteurs

L'optimisation de la taille des secteurs, de leurs formes et des manières dont les dégroupements et regroupements peuvent se faire, ont fait l'objet de nombreuses recherches. La modification de la taille des secteurs n'a pas donné de résultats probants en terme de charge de travail des contrôleurs ([Riv06]). L'optimisation de la taille des secteurs par fusion/fission de graphes, permet de minimiser le temps de coordination, sans pour autant fournir de résultats sur le reste de la charge mentale du contrôleur ([Bic07]). Enfin, l'utilisation de réseaux de neurones permet d'optimiser le dégoupement/regroupement de secteurs en fonction de la charge de travail des contrôleurs ([Gia10]).

1.3.2 De la charge de travail à la charge mentale

DCCXLVIII + CCLII =

```
  748
+ 252
-----
```

FIG. 1.4 – Addition de deux nombres écrits en numération romaine et numération indienne. La charge de travail est la même mais la charge mentale est bien supérieure dans le cas de la numération romaine.

La notion de charge de travail modélise le travail du contrôleur aérien comme une boîte noire : seuls les entrées (nombre d'avions, taille du secteur, propriétés géométriques des flux) et les sorties (nombres de conflits évités) sont regardés, sans prendre en compte la manière dont le travail est effectué et plus précisément l'ensemble des processus mentaux mis en place par le contrôleur pour résoudre sa tâche. Par exemple, la figure 1.4 illustre deux situations où la charge de travail est la même (addition de deux nombres) mais où la charge mentale diffère. En effet, une fois le système de notation positionnelle acquis, la numération indienne en base 10 permet d'effectuer des additions de manière beaucoup plus rapide.

[7] Un avion reste en moyenne 15 minutes dans un secteur.

Pour les contrôleurs aériens, la charge mentale dépend évidemment de la charge de travail, mais aussi de la manière dont le trafic aérien est présenté sur l'écran radar (par exemple s'il y a des informations émergentes).

1.3.3 Déconfliction

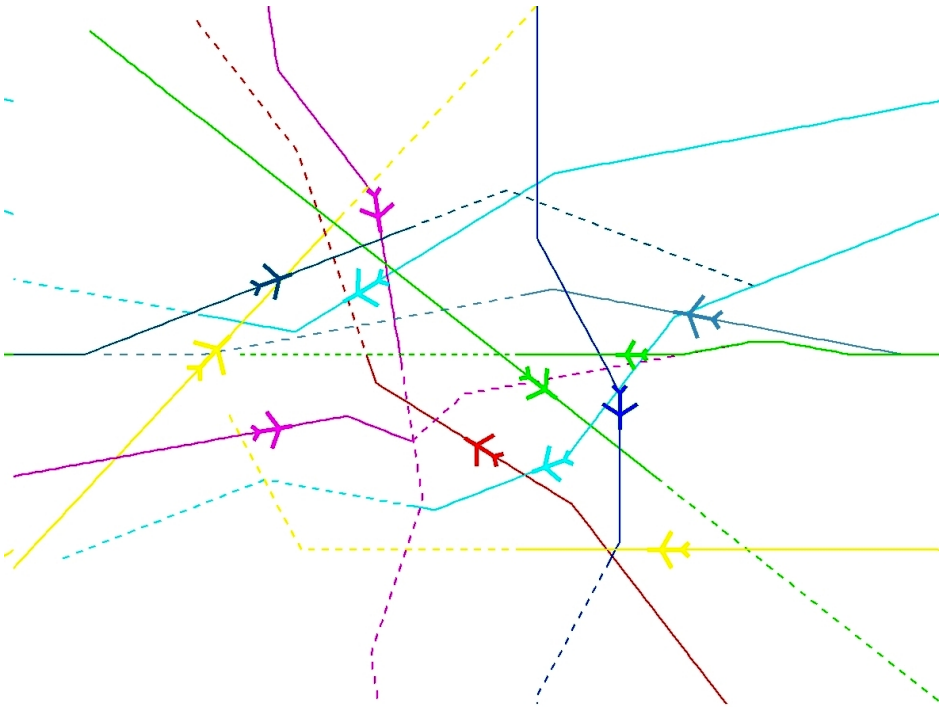

FIG. 1.5 – Illustration d'une situation (non opérationnelle) avec relativement peu d'avions et énormément de conflits, tirée du simulateur CATS. Les trajectoires en pointillés correspondent aux manœuvres calculées automatiquement par le module de résolution de conflits.

Pour évaluer la charge mentale du contrôleur aérien et donc la capacité du secteur qu'il contrôle, il n'est pas possible d'utiliser des critères « simples », tels que le nombre d'avions traversant le secteur pendant une période de temps donnée. En effet, un traffic où tous les avions « se suivent » avec un écart supérieur aux normes de séparation n'a besoin d'aucune intervention du contrôleur humain ; en revanche, si on prend le même nombre d'avions et qu'on les fait converger vers le centre du secteur, comme illustré dans la figure 1.4, cela sera une situation très complexe à gérer car nécessitant de nombreuses manœuvres et une attention permanente et soutenue.

Ainsi, il est nécessaire d'évaluer la *charge mentale* du contrôleur, qui est la ressource véritablement limitante et d'éliminer au maximum les charges « parasites » pour que la plus grande partie possible des ressources cognitives du contrôleur soit effectivement dédiée à la détection et la résolution de conflits.

Esa M. Rantanen et Ashley Nunes ont montrés, dans [RN05], que les contrôleurs utilisent une méthode hiérarchique pour tester si deux avions sont en conflit : ils commencent par

regarder si les avions sont à des altitudes différentes, puis extrapolent leurs trajectoires pour savoir s'ils vont être séparés dans le plan horizontal. La lecture des altitudes se fait très rapidement car elle est clairement indiquée sur l'écran radar, tandis que les extrapolations consomment plus de ressources mentales. Il est donc pertinent de séparer les flux aériens en paramétrant le niveau de vol de chacun, car ainsi, le minimum de ressource mentale est utilisé par le contrôleur pour s'assurer de la bonne séparation des avions.

Deuxième partie

État de l'art

2

Géométrie algorithmique

La géométrie algorithmique est le domaine de résolution de problèmes géométriques à l'aide de l'informatique. La géométrie algorithmique s'est d'abord développée grâce aux recherches en infographie, c'est-à-dire la création et la manipulation d'images grâce à des processus numériques. Depuis, le domaine s'est étendu à des considérations plus mathématiques et moins artistiques, telles que la recherche de l'enveloppe convexe d'un ensemble de points ou la construction de diagrammes de Voronoï. Cette nouvelle extension de la géométrie algorithmique est aussi appelée géométrie algorithmique combinatoire par Franco P. Preparata et Michael Ian Shamos[dBvKOS05], qui comptent parmis les précurseurs de la discipline.

La problématique précise étudiée ici est la recherche d'intersections dans un ensemble de segments ; la méthode de résolution, l'utilisation d'une *Sweep line* (ligne de balayage). Si la recherche efficace d'intersections d'un ensemble de segments a pour but initial de superposer rapidemment deux cartes contenant des informations différentes, telles que la pluviométrie et les températures moyennes, elle est ici utilisée pour détecter si deux trajectoires d'avion se croisent.

Effectuée naïvement, la recherche d'intersections entre trajectoires d'avions serait en $\Theta(k^2 \times n^2)$, avec k le nombre moyen de segments par trajectoire et n le nombre d'avions. En effet, il faudrait, pour chacun des $\frac{n\times(n-1)}{2}$ couples de trajectoires, tester si chacun des $k \times k$ couples de segments s'intersectent. Cela, pour une moyenne de 12 segments par trajectoire et d'un millier de trajectoires différentes.

2.1 Boîtes englobantes

Pour éviter d'avoir à rechercher l'existence d'une intersection pour tout couple de segments, il est intéressant de rechercher des conditions nécessaires à cela. Une idée simple est de projeter les segments sur un axe, comme sur la figure 2.1, et de ne tester que les segments dont les projections se superposent.

Il s'agit d'une conditition nécessaire, mais pas suffisante. Sur la figure 2.1, les segments 4 et 5 ont leurs projections qui se superposent, sans pour autant se croiser. Il peut donc être intéressant de projeter à la fois sur l'axe des X et l'axe des Y, afin d'avoir une condition plus restrictive. Mais, comme illustré par la figure 2.2, il ne s'agit toujours pas d'une condition

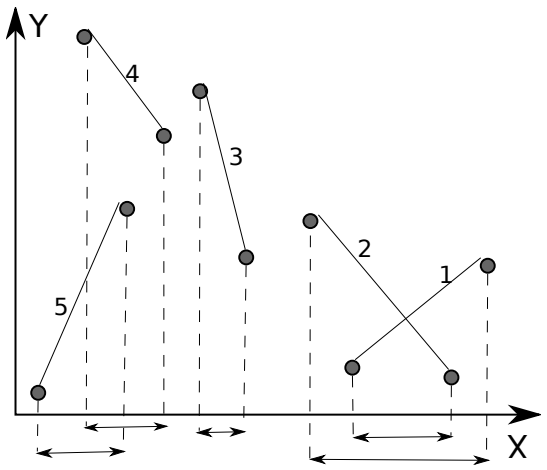

FIG. 2.1 – Projection d'un ensemble de segments sur l'axe X.

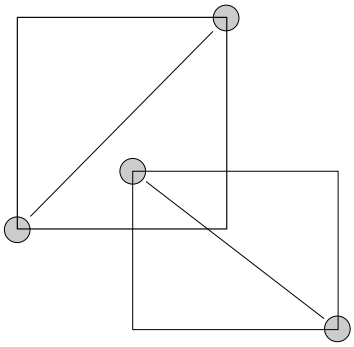

FIG. 2.2 – Deux segments ne se croisant pas et dont les « boîtes englobantes » se superposent.

suffisante. Il est donc nécessaire d'utiliser des propriétés plus fines pour isoler les segments qui se croisent potentiellement.

De plus, l'utilisation de boîtes englobantes n'est pas adaptée aux propriétés des flux aériens, qui ne sont pas des segments mais des lignes brisées, i.e. des suites de segments. En effet, ou on considère la boîte englobant le flux en entier, comme sur la figure 2.1, ce qui nécessite de calculer cette boîte tout en augmentant le nombre de cas où les boîtes se superposent sans qu'il n'y ait conflit ; ou alors, on regarde les boîtes associées à chacun des segments du flux, et certains conflits sont ignorés, comme montré sur la figure 2.1.

Il est donc nécessaire de trouver un algorithme plus efficace pour détecter les segments qui se croisent potentiellement ; la technique choisie est la *sweep line*, qui repose sur une notion de voisinage plus subtile que l'utilisation de simples les boîtes englobantes.

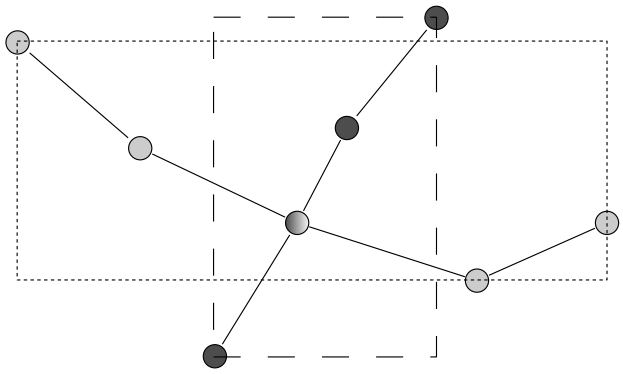

FIG. 2.3 – Deux flux aériens qui se croisent avec les boîtes englobantes globales associées.

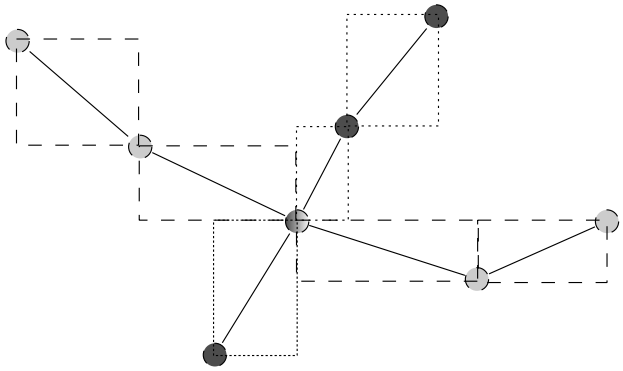

FIG. 2.4 – Deux flux aériens ayant une balise commune avec les boîtes englobantes associées à leurs segments.

2.2 Sweep line

L'idée centrale de l'algorithme de *sweep line*, ou ligne de balayage, est qu'il est inutile de tester si deux segments se croisent s'ils ne sont pas « suffisament proches ». Il faut donc calculer des voisinages de segments, en espérant que le temps consacré à calculer ces voisinages sera largement inférieur au temps gagné à ne pas tester toutes les paires de segments pour y chercher une éventuelle intersection.

Le voisinage utilisé ici est dynamique : pour une droite donnée, deux segments sont dits voisins s'ils sont tous les deux traversés par la droite et qu'aucun autre segment ne se trouve entre eux. Formellement, pour une droite d'équation $y_= y_d$[1], et deux segments s_1 et s_2, s_1 et s_2 sont dit voisins s'il existe un point $(x_1; y_d)$ appartenant à s_1, un point $(x_2; y_d)$ appartenant à s_2 et qu'il n'existe aucun point $x \in [x_1; x_2]$ tel que (x, y_d) appartienne à un segment s différent de s_1 et s_2.

Si deux segments se croisent, il existe forcément une ligne horizontale pour laquelle ceux-ci

[1]Les cas particuliers avec segments horizontaux sont définis plus loin.

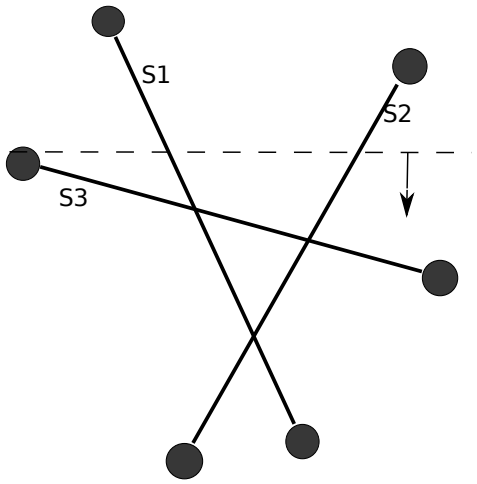

FIG. 2.5 – Illustration de l'algorithme de sweep line lorsque celle-ci, en pointillés sur la figure, est au niveau de l'origine de s_3, les segments s_3 et s_1 sont voisins, ainsi que s_1 et s_2

sont voisins : il suffit de se placer juste au-dessus de leur point d'intersection[2]. Par exemple, la figure 2.5 montre l'étape de l'algorithme juste avant l'intersection de s_1 et s_3. Ainsi, il est inutile de chercher si deux segments qui ne sont jamais voisins se croisent.

Le principe est ensuite de faire glisser cette droite, pour détecter toutes les paires de segments voisins et ainsi tester s'ils se croisent ou non. La ligne balaye ainsi l'ensemble du plan euclidien, d'où le nom de l'algorithme.

2.3 Points évènementiels

Il faut maintenant définir la manière dont la ligne balaye le plan. La faire glisser d'ϵ en ϵ est beaucoup trop long en raison des vérifications redondantes effectuées. En réalité, pour que le voisinage d'un segment s soit différent pour les lignes $y = y_d + \epsilon$ et $y = y_d - \epsilon$, il faut qu'entre ces deux lignes se trouve soit une extrémité d'un segment, soit un croisement entre deux segments. On nomme ces points des *points évènementiels*, ou *event point*. Ainsi, il suffit de faire « sauter » la ligne de point évènementiel en point évènementiel pour avoir l'ensemble des voisinages possibles, sans en oublier.

Les points évènementiels les plus simples à calculer sont les extrémités : il n'y a d'ailleurs rien à faire, elles sont directement accessibles pour un ensemble de segments donnés. Pour détecter les croisements, il suffit de tester si deux segments voisins, pour une position de la sweep line donnée, se croisent. Par exemple, lors de l'étape de la figure 2.5, l'intersection entre s_3 et s_1 est détectée et ajoutée à l'ensemble des points évènementiels. Cette intersection correspond à l'étape suivante, illustrée à la figure 2.6.

[2]On se place dans le cas où il n'y a pas plusieurs segments se croisant au même point pour le moment.

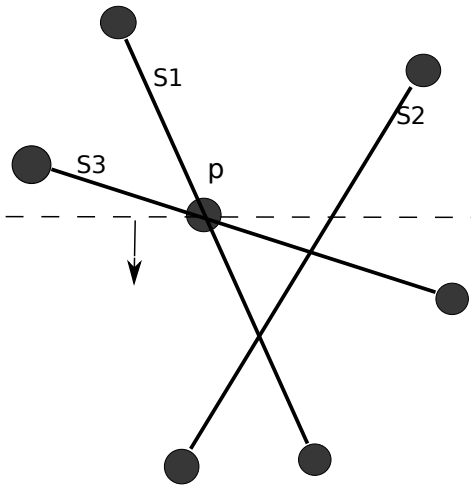

FIG. 2.6 – Étape de l'algorithme de sweep line juste après celle de la figure 2.5. Après le point p, les segments s_2 et s_3 deviennent voisins.

2.4 Algorithme

À l'initation, toutes les extrémités des segments sont rangés dans une file de priorité, de laquelle sont extraits un à un les points évènementiels : l'algorithme 1 détaille ce qui se fait à chaque étape. On note ici $\mathcal{S}$ l'ensemble des segments coupés par la sweep line.

La fonction trouverNouveauPoint est très simple : elle teste si les segments se croisent et, dans le cas où le point d'intersection p_{int} est après p, ajoute p dans la file de priorité. Les éléments de $U(p)$ sont supprimés de $\mathcal{S}$ pour y être ajoutés tout de suite après afin d'inverser leur ordre ; en effet, ils ont été insérés quand la sweep line se trouvait avant p et, après p, leur ordre va donc varier. C'est parce que l'ordre des segments varie au cours de l'exécution qu'il est nécessaire de maintenir un « contexte » associé à la structure de données $\mathcal{S}$.

2.4.1 Complexité

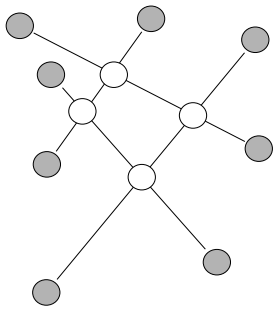

FIG. 2.7 – Graphe planaire associé à un ensemble de segments.

La construction de la file de priorité $\mathcal{Q}$ des points évènementiels se fait en complexité

Algorithme 1 Gestion d'un point évènementiel

Soit $U(p)$ l'ensemble des segments dont l'origine est p
Soit $C(p)$ l'ensemble des segments de $\mathcal{S}$ qui contiennent p
Soit $L(p)$ l'ensemble des segments de $\mathcal{S}$ dont la destination est p
si $L(p) \cup C(p) \cup U(p)$ contient au moins deux segments **alors**
  $\mathcal{I} \leftarrow \mathcal{I} \cup (p, L(p) \cup C(p) \cup U(p))$
  $\mathcal{S} \leftarrow \mathcal{S} \setminus (L(p) \cup U(p))$
  $\mathcal{S} \leftarrow \mathcal{S} \cup U(p) \cup C(p)$
fin si
si $U(p) \cup C(p) \neq \emptyset$ **alors**
  Soit s_d le voisin à droite de p dans $\mathcal{S}$
  Soit s_g le voisin à gauche de p dans $\mathcal{S}$
  trouverNouveauPoint(s_g, s_d, p)
sinon
  Soit s_g le segment le plus à gauche de $U(p) \cup C(p)$
  Soit s_{gg} le voisin à gauche de s_g dans $\mathcal{S}$
  trouverNouveauPoint(s_g, s_{gg}, p)
  Soit s_d le segment le plus à droite de $U(p) \cup C(p)$
  Soit s_{dd} le voisin à droite de s_d dans $\mathcal{S}$
  trouverNouveauPoint(s_d, s_{dd}, p)
fin si

$O(n \log n)$ grâce à l'utilisation d'un arbre binaire de recherche, avec n le nombre de segments. Pour chaque évènement e, l'évènement est supprimé de $\mathcal{Q}$ (en $O(n \log n)$) et au plus deux nouveaux évènements peuvent être ajoutés (en $O(n \log n)$ chacun). Des opérations (insertion, supression, recherche de voisins) sont aussi effectuées sur la structure $\mathcal{S}$, chacune en $O(n \log n)$. Le nombre d'opérations est linéaire avec le nombre de segments associés à l'évènement, c'est-à-dire avec $\text{card}(L(p) \cup C(p) \cup U(p))$, noté $m(e)$. Dans [dBvKOS05], de Berg et Al montrent que $\sum_p m(p) = m = O(n + I)$ avec I le nombre d'intersections dans l'ensemble des segments. Pour ce faire, ils modélisent l'ensemble des segments et de leurs intersections par un graphe planaire, comme illustré sur la figure 2.7 et raisonnent sur les degrés de ses sommets.

2.4.2 Structure de données

L'ensemble des points évènementiels à venir est stocké sous forme de file de priorité, de laquelle est extrait le plus petit élément à chaque mise à jour de la *sweep line* et dans laquelle sont placés les points évènementiels calculés au fur et à mesure. La relation d'ordre totale définie sur les points est anti-lexicographique en y et lexicographique en x, c'est-à-dire qu'un point $p_1 = (x_1; y_1)$ est « avant » un point $p_2 = (x_2; y_2)$ si et seulement si $y_1 > y_2$ (p_1 est « au-dessus » de p_2) ou $y_1 = y_2$, $x_1 < x_2$ (p_1 est sur la même ligne horizontale et « à gauche » de p_2).

En raison des fonctions particulières qui sont appliquées sur S, l'ensemble dans lequel sont rangés les segments coupés par la *sweep line* à un instant donné, il est nécessaire d'utiliser une structure de donnée spéciale, inspirée des arbres binaires de recherche équilibrés et détaillée dans la section 5.5.1.

3 Programmation par contraintes

La programmation par contraintes (PPC, ou *CP* en anglais pour *Constraint Programming*) est un paradigme de programmation apparu à la fin des années 80 [**?**] permettant de résoudre des problèmes combinatoires de grande taille tels que les problèmes de planification et d'ordonnancement. La programmation par contraintes sépare la partie modélisation à l'aide de problème de satisfaction de contraintes, de la partie résolution dont la particularité réside dans l'utilisation active des contraintes du problème pour réduire la taille de l'espace des solutions à parcourir : on parle alors de propagation de contraintes.

On se limite ici aux problèmes de satisfactions de contraintes définis sur des variables à domaine fini, mais la programmation par contraintes sur domaines continus existe.

3.1 Définitions

Formellement, une *variable* x peut prendre un ensemble fini de valeurs, appelé son domaine et noté D_x. Une *contrainte* $c = (R, x_1, x_2, \ldots, x_i)$ est définie comme un sous-ensemble R de $D_{x_1} \times D_{x_2} \times \cdots \times D_{x_i}$. Une contrainte est dite *binaire* lorsqu'elle ne concerne que deux variables. On se limitera dans la suite des définitions aux contraintes binaires, mais toutes sont généralisables aux contraintes n-aires. Un *problème de satisfaction de contraintes*, ou CSP pour *Constraint Satisfaction Problem*, est un ensemble de variables sur lequel est défini un ensemble de contraintes. Par exemple,

$$x \in [|1;3|], y \in [|1;5|], z \in [|3;18|], z = 2 * x, y < z \tag{3.1}$$

est un CSP à trois variables et deux contraintes.

Une *instanciation* d'une variable x correspond à l'attribution d'une valeur $v \in D_x$ à x. Une *instanciation partielle* $\mathcal{I}$ d'un CSP correspond à l'instanciation d'une partie seulement de ses variables ; $x = 1; y = 4$ est donc une instanciation partielle de 3.1. Une *instanciation totale* correspond à l'instanciation de l'ensemble des variables d'un CSP.

En notant $\mathcal{I}(x)$ la valeur de la variable x dans l'instanciation $\mathcal{I}$, on dit que $\mathcal{I}$ *satisfait* une containte $c = (R, x, y)$ lorque $(\mathcal{I}(x), \mathcal{I}(y)) \in R$ (cela suppose que la variable y soit aussi instanciée). Ceci est noté $\mathcal{I} \models c$. Dans le cas inverse, on dit que l'instanciation *viole*

la contrainte. Une instanciation est dite *cohérente* ou *consistante* si elle satisfait toutes les contraintes.

Une valeur x_i du domaine D_x de la variable x a *un support* dans le domaine D_y de la variable y si, pour toute contrainte c portant à la fois sur x et y, il existe une valeur $y_c \in D_y$ telle que l'instanciation $x = x_i, y = y_c$ satisfasse c.

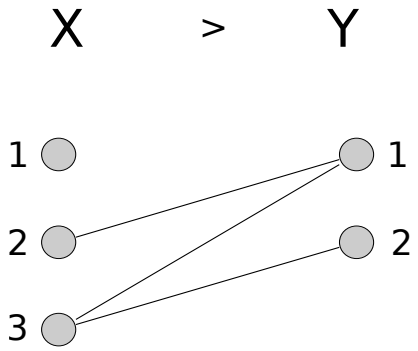

FIG. 3.1 – Graphe associé à la contrainte binaire (exprimée en extension) $x \in [|1; 2; 3|], y \in [|1; 2|], x > y$.

Dans le *graphe associé à une contrainte binaire* exprimée en extension $c = (x, y, \mathcal{R})$, chaque nœud est une valeur de l'une des variables x ou y et il existe un arc entre $v_x \in D_x$ et $v_y \in D_y$ si et seulement si $(v_x, v_y) \in R$. Le graphe est donc biparti, puisqu'il n'existe des arcs qu'entre des valeurs de x et des valeurs de y. Si toute valeur de x a un support dans D_y pour cette contrainte et réciproquement, alors la contrainte c est dite *arc-consistante*. Par exemple, la contrainte de la figure 3.1 n'est pas arc-consistante car la valeur $x = 1$ est sans support dans D_y. Ce graphe, qui représente la micro-structure des contraintes du CSP, peut s'étendre à des contraintes n-aires.

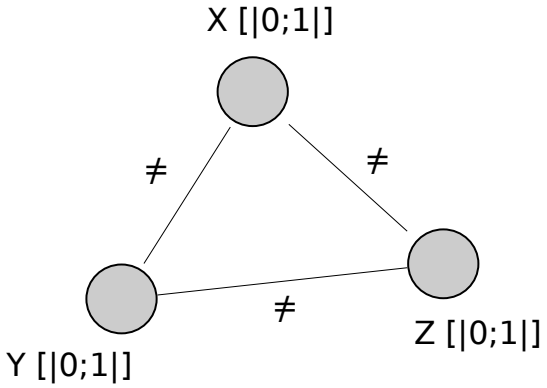

FIG. 3.2 – Graphe associé au CSP $x \in (0; 1), y \in (0; 1), z \in (0; 1), x \neq y, x \neq z, y \neq z$.

Dans le *graphe associé à un CSP à contraintes binaires*, un nœud correspond à une variable du CSP enrichie de son domaine et un arc entre x et y à une contrainte entre ces variables. La figure 3.2 représente ainsi le CSP $x \in [|0; 1|], y \in [|0; 1|], z \in [|0; 1|], x \neq y, x \neq z, y \neq z$.

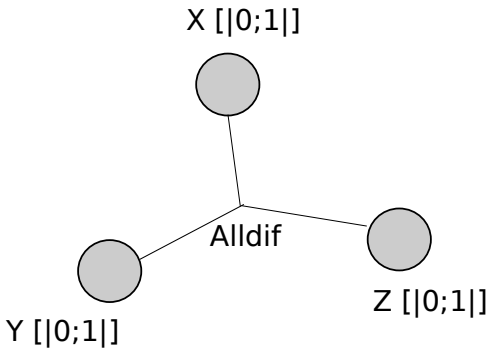

FIG. 3.3 – Hypegraphe associé au CSP $x \in (0;1), y \in (0;1), z \in (0;1)$, Alldiff(x,y,z).

La généralisation de cette représentation aux CSP avec des contraintes non-binaires, se fait naturellement en utilisant non pas un graphe mais un hypergraphe, comme dans la figure 3.3.

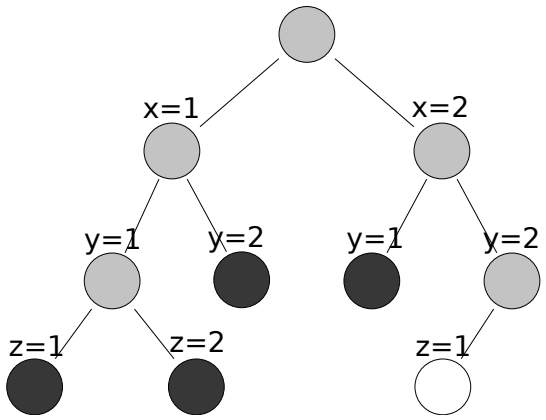

FIG. 3.4 – Arbre de recherche du CSP $x \in [|1;2|], y \in [|1;2|], z \in [|1;2|], x = y, z < y$ pour la méthode de retour-arrière. Un rond noir correspond à un échec (une contrainte se retrouve violée), un point gris à un choix et un point blanc à la découverte d'une solution.

Les techniques de résolution des CSP sont séparées en deux grandes familles : les algorithmes approchés relèvent en général de la recherche locale, comme l'algorithme `MinConflict` de [MPJL93], tandis que les méthodes complètes, détaillées dans la section 3.2, structurent l'exploration de l'espace de recherche en arbre, le plus souvent parcouru en profondeur d'abord[1].

Dans un tel *arbre de recherche*, un nœud correspond à un CSP dont l'ensemble des solutions est égal à l'union des solutions de chacun de ses fils. Le fils d'un nœud peut correspondre à une extension d'une instanciation partielle, comme pour celui de la figure 3.4, ou à l'ajout d'une nouvelle contrainte[2]. La racine de l'arbre de recherche d'un CSP est le CSP complet.

[1] C'est la stratégie classique des systèmes Prolog dont la PCC représente une extension.

[2] On peut par exemple explorer de manière dichotomique le domaine $[|x_1; x_2|]$ d'une variable x en ajoutant la contrainte $x < \frac{x_1+x_2}{2}$ dans le premier fils et $x \geq \frac{x_1+x_2}{2}$ dans le second.

3.2 Résolution

La technique la plus naïve pour résoudre un CSP est de générer toutes les instanciations totales possibles et, pour chacune d'entre elles, déterminer si elle est consistante. Cette méthode est inapplicable pour des CSP de grande taille. Une idée est donc d'instancier les variables non pas toutes en même temps, mais une par une, et de vérifier à chaque nouvelle instanciation si aucune contrainte n'est violée. On crée ainsi un *arbre de recherche*, où un nœud correspond au choix d'une valeur d'instanciation, comme dans la figure 3.4. Si une contrainte est violée ou que tous les fils d'un nœud aboutissent à des échecs, on effectue un retour arrière pour remettre en cause la décision précédente : c'est l'algorithme de *Backtrack*. Deux axes d'amélioration sont explorés pour réduire la taille de l'arbre de recherche et donc le temps de calcul avant de trouver une solution : la propagation de contraintes, et le choix de l'ordre d'instanciation des variables. Il existe également des techniques pour améliorer l'efficacité des retours arrière [**?**], voire pour apprendre de nouvelle contrainte durant la recherche (*nogoods*) mais elles dépassent le champ de cette étude.

3.3 Ordre d'instanciation

L'ordre d'instanciation des variables peut être crucial pour limiter la taille d'un arbre de recherche. Prenons le CSP :

$$(x_1, ..., x_{1000}) \in [|0; 1|]^{1000}, x_1 = x_{1000}, \forall i \in [|2; 999|], x_i < x_{1000} \tag{3.2}$$

Si les variables x_i sont instanciées dans l'ordre des i croissant, tout le sous-arbre correspondant à $x_1 = 0$ va être exploré, soit 2^{998} nœuds, alors que si x_{1000} est instanciée en premier, l'unique solution apparaît sans aucun retour arrière. Il est donc crucial d'avoir un ordre d'instanciation des variables adapté. Différentes heuristiques sont utilisées pour déterminer quelle est la prochaine variable à instancier, lors d'une instanciation partielle donnée. Généralement, on préfère échouer le plus tôt possible (*First Fail Principle*) afin de couper au plus tôt l'arbre de recherche : ce sont donc les variables les plus contraintes qui sont instanciées les premières ([FD95]), ce qui correspond à celles qui ont le domaine le plus petit et/ou celles qui apparaissent le plus souvent dans les contraintes ([Smi99]).

Des méthodes plus subtiles existent, qui « apprennent » dynamiquement quelles sont les variables les plus contraintes, en comptant le nombre de fois où l'instanciation d'une variable aboutit à un échec ([BHLS]).

3.4 Propagation

La propagation de contraintes, pour un nœud de l'arbre de recherche donné, correspond au retrait de valeurs du domaine de variables avant qu'elles soient instanciées ; les valeurs retirées n'auraient pas pu aboutir à une instanciation totale cohérente. Le but est d'éviter l'exploration de parties de l'arbre de recherche qui ne mèneront pas à une solution, par exemple d'éviter les deux premiers sous-arbre de la figure 3.4. Si la propagation des contraintes aboutit à vider entièrement le domaine d'une variable, un échec est découvert et un retour arrière sera alors

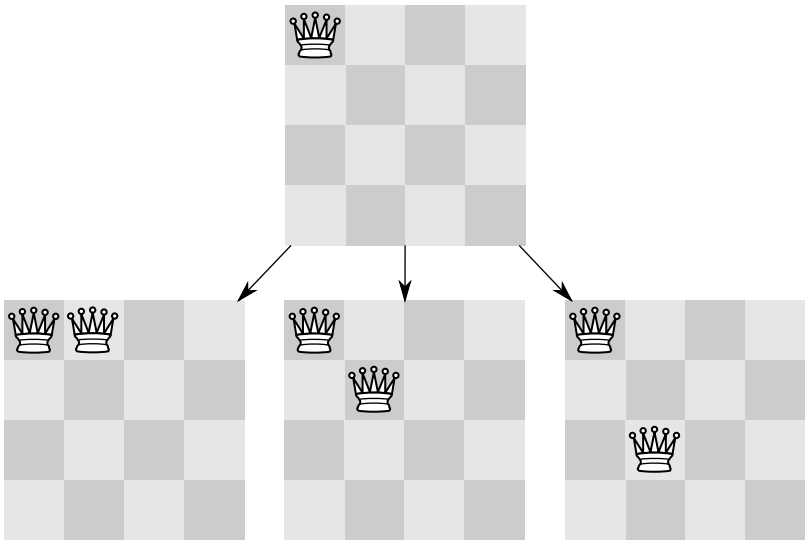

FIG. 3.5 – Partie de l'arbre de recherche du problème des quatre dames (il faut placer quatre dames sur un échiquier 4×4 de manière à ce qu'elles ne se menacent pas, c'est-à-dire qu'il y en ait au plus une sur chaque ligne, chaque colonne et chaque diagonale)

effectué. Certaines contraintes, telles que *Alldiff* , peuvent également détecter directement des échecs. Ce filtrage des domaines est donc une étape d'inférence déterministe, contrairement aux décisions non-déterministes prises à chaque nœud de l'arbre de recherche.

3.4.1 Cohérence locale

Pour propager les contraintes, on introduit une notion de cohérence locale qui permet de détecter quelles sont les instanciations partielles qui n'aboutiront pas à une solution. Cette propriété, moins forte que la cohérence globale d'un CSP (problème NP-complet), qui correspond à l'existence d'une solution, pourra être établie par des algorithmes de complexité polynomiale à chaque nœud de l'arbre de recherche lors de la résolution du problème.

Cohérence d'arc

La plus connue est celle d'*arc-cohérence*[3] : une contrainte orientée $c = (x, y, \mathcal{R})$ est arc-cohérente si toutes les valeurs du domaine de x ont un support dans le domaine de y. Ainsi, dans la figure 3.1, la contrainte $c = (x, y, \mathcal{R})$ n'est pas arc-cohérente car $x = 1$ n'a pas de support alors que la contrainte $c = (y, x, \mathcal{R})$ l'est car $y = 1$ et $y = 2$ ont toutes deux un support.

Un CSP est arc-cohérent si toutes ses contraintes sont arc-cohérentes. Toutefois, cela ne garantie pas que le CSP soit cohérent. Par exemple, le CSP de 3.2 est arc-cohérent mais n'a pas de solution.

[3] Aussi appelée *arc-consistance*.

Cohérence de bornes

La cohérence d'arc peut être coûteuse à maintenir, surtout pour des variables avec de grands domaines. Pour des contraintes linéaires exprimées sur des variables dont le domaine est un intervalle (ou une union d'intervalles) d'entiers, on peut définir la notion de cohérence de borne, c'est-à-dire vérifier que les *extrema* des intervalles ont tous un support.

3.4.2 Techniques de propagation

Différentes techniques existent pour garantir qu'un certain niveau de cohérence locale est maintenue au fur et à mesure des instanciations. Le choix entre ces techniques dépend des caractéristiques du problème, i.e. s'il est plus intéressant d'effectuer des vérifications peu coûteuses ou d'élaguer au maximum l'arbre de recherche. On présente ici une des plus basiques, *Forward Checking*, qui ne garantit pas l'établissement de l'arc-cohérence d'un CSP, et la plus complète nommée *Maintening Arc Consistency*.

Forward Checking

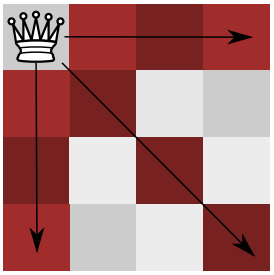

FIG. 3.6 – Illustration de la technique de *Forward Checking* avec le problème des quatre dames.

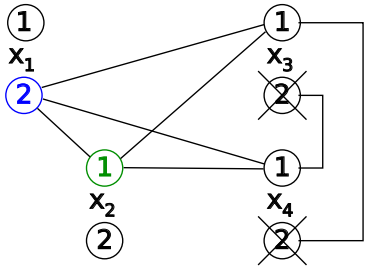

FIG. 3.7 – Illustration de la technique de *Forward Checking* avec un CSP représenté par le graphe de sa micro-structure (chaque valeur possible de chacune des variables est représentée par un nœud), lors de l'instanciation $x_2 = 1$. x_1 a déjà été instanciée à 2 et les variables x_3 et x_4 ne l'ont pas encore été.

La technique de *Forward Checking* (FC) consiste, lors de l'instanciation $x = x_v$ d'une variable, pour chaque contrainte $c = (x, y, \mathcal{R})$ portant sur cette variable, à éliminer du domaine de la variable y les valeurs incohérentes avec $x = x_v$. Cela est illustré sur la figure 3.7 et sur la figure 3.6 où, à l'instanciation de la variable codant la position de la première dame[4] , toutes les positions en conflit, c'est-à-dire sur la même ligne ou diagonale, sont retirées des domaines des autres dames.

Maintening Arc Consistency

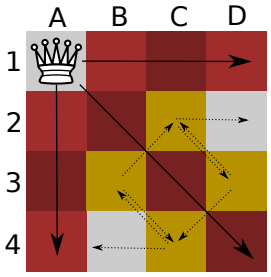

FIG. 3.8 – Illustration de la technique de *Maintening Arc Consistency* avec le problème des quatre dames.

Comme son nom l'indique, *Maintening Arc Constistency* (MAC) signifie qu'à chaque nœud de l'arbre de recherche, l'arc-cohérence est maintenue. Ainsi quand le domaine d'une variable x est modifié, on établit l'arc-cohérence non seulement pour les contraintes $c = (x, y, \mathcal{R})$ dans lesquelles x est impliquée, mais également pour les autres contraintes du CSP dont une des variables a été modifiées. MAC réitère le procédé jusqu'à ce que toutes les contraintes soient arc-cohérentes et une même contrainte est donc susceptible d'être considérée plusieurs fois. Par exemple, sur la figure 3.8, après suppression des valeurs incohérentes dans le domaine des colonnes B, C et D, la valeur 3 est retirée du domaine de B et de D car sans support dans C, puis les valeurs 2 et 4 de C car sans support respectivement dans D et B. La propagation s'arrête là car C se retrouve avec un domaine vide et un échec est donc détecté.

3.5 Optimisation

On peut chercher, lors de la résolution d'un *CSP*, à obtenir une solution quelconque, l'ensemble entier des solutions ou encore une preuve de l'absence de solution. Parmi toutes les solutions d'un CSP, on s'intéresse souvent à obtenir celle qui est de meilleure qualité d'après un certain critère : il s'agit alors d'*optimisation.*

La résolution d'un problème d'optimisation consiste à rechercher l'existence de solution avec une variable supplémentaire c dépendant des variables de décision du CSP, par exemple leur somme ou leur max, et qui représente le coût à minimiser[5]. On peut transformer l'algo-

[4]La i^{e}dame sera dans la colonne i, puisque de toutes manières les contraintes du problème imposent une dame par colonne. La position est donc uniquement codée par le numéro de ligne.

[5]Ou le gain à maximiser.

rithme de Backtrack en algorithme d'optimisation en ajoutant une contrainte « dynamique » sur c, mise à jour à chaque fois qu'une solution de coût $c = c_i$ est trouvée. La recherche est ainsi poursuivie (ou relancée) en ajoutant au CSP la contrainte $c < c_i$. Ainsi, lorsque la n$^{\text{e}}$ recherche échouera avec $c < c_{n-1}$ (preuve d'incohérence du CSP), cela signifiera que la solution précédente de coût c_{n-1}, était optimale. Il s'agit de l'algorithme de *Branch and Bound*.

4

Coloration de graphe

Le problème de la coloration de graphe est né au début du XX^{e}siècle : Francis Guthrie, un mathématicien Sud-Africain, cherchait le nombre minimum de couleurs nécessaires pour que, sur une carte représentant les comtés d'Angleterre, deux comtés ayant une frontière commune ne soient pas de la même couleur. Chaque comté est représenté par un nœud du graphe et la propriété « les comtés A et B ont une frontière commune » est traduite en « les nœuds A et B du graphe sont reliés par une arête ».

Ce problème NP-difficile pour les graphes généraux est utilisé pour modéliser de nombreux sous-problèmes issus d'applications réelles telles que l'allocation de fréquence radio, l'allocation des registres d'un micro-processeur par un compilateur ou encore la résolution d'une grille de Sudoku (en associant une couleur à chaque entier). La séparation de flux aériens, dans sa version la plus simple (sans tenir compte de l'écart au niveau de vol requis), peut également se formuler comme un problème de coloration de graphe. Nous présentons dans ce chapitre quelques techniques de résolution approchée de ce problème.

4.1 Nombre minimal de couleurs

L'un des premiers critères d'optimisation utilisés est le nombre de couleurs. En effet, trouver une coloration acceptable est facile, il suffit de donner une couleur différente à chaque nœud. Il est par contre plus difficile de trouver une coloration ayant un nombre arbitrairement limité de couleurs, voire à déterminer le nombre minimum de couleurs nécessaires à la coloration, aussi appelé nombre chromatique. Ces problèmes sont respectivement NP-complet et NP-difficile pour des graphes généraux, mais peuvent être de classe P pour certaines familles de graphe.

Ici, trouver un minimum de couleurs n'a pas de sens opérationnellement : il n'y a pas de surcoût lié à l'utilisation d'un niveau de vol supplémentaire. En revanche, il peut être intéressant d'avoir une borne inférieure du nombre de couleurs nécessaires, afin de voir si la solution proposée est réalisable opérationnellement[1].

[1]L'altitude maximale de vol des avions de ligne est limitée pour des raisons de résistance à la différence de pression air extérieur/air intérieur (la plupart des avions de ligne volent au-dessous du FL400). La norme de séparation verticale, quant à elle, ne peut descendre en-dessous d'un certain seuil en raison des performances radar et des temps de réaction (comme mentionné dans la section **??**). Le nombre de niveaux de vols possibles

En revanche, des niveaux de vol différents induisent des consommations de carburant et donc des émissions de CO_2 différents. Comme il est difficile de connaître avec précision la masse au décollage d'un avion, il est supposé que le coût associé à un changement de niveau de vol dépend uniquement de l'écart entre le niveau demandé et celui attribué.

4.2 Recherche de clique maximale

Une clique est un sous-graphe complet, c'est-à-dire un sous-ensemble de sommets du graphe tel que chaque sommet est relié à tous les autres sommets de la clique. Clairement, le nombre chromatique d'un graphe est supérieur à la taille de sa clique maximale, puisque si ce n'était pas le cas, deux sommets de la clique seraient de la même couleur alors qu'ils sont, par définition, reliés. La recherche de cliques dans le graphe permet aussi une diminution de la taille de l'arbre et recherche et donc du temps de calcul, puisqu'elle permet de poser des contraintes globales « toutes différentes » (AllDiff) sur les couleurs des n sommets d'une clique donnée, ce qui est plus efficace que $\frac{n\times(n-1)}{2}$ contraintes de différence sur chacune des arêtes. En théorie, la recherche de clique maximale est un problème NP-complet : c'est pourquoi on ne cherche pas exactement la clique maximale mais plutôt un ensemble de cliques, que l'on trouve par recherche gloutonne.

4.3 Algorithmes de recherche approchée

La recherche approchée est une méthode de résolution de problèmes par exploration de l'espace des solutions (admissibles ou non) par voisinages successifs à partir d'une solution ou d'une population de solutions générée aléatoirement ou par un autre type d'algorithme. Elle est particulièrement adaptée à la résolution de problèmes de grande taille lorsqu'ils sont trop difficiles pour des algorithmes complets ou lors que l'on veut une solution dans une limite de temps. Seulement, il n'y a aucune garantie que la solution est optimale : pas de propriété de complétude.

4.3.1 Recherche gloutonne

La recherche gloutonne de solution d'une coloration de graphe est très simple : on prend successivement chaque nœud, à qui on applique la plus petite couleur non encore attribuée à l'un de ses voisins. Si cette méthode trouve une coloration acceptable en un temps relativement court, elle peut être gourmande en nombre de couleurs utilisées et son efficacité dépend de l'ordre dans lequel les nœuds sont colorés. Commencer par les nœuds de plus grand degré (c'est-à-dire les plus « contraints ») permet d'améliorer l'efficacité de cette recherche.

---

est donc limité, en général à une vingtaine de FL différents.

4.3.2 Recherche locale

La recherche locale est une famille de métaheuristiques fondée sur la notion de voisinage. Pour chaque point de l'espace de recherche trouvé, de coût [2] donné, on sélectionne l'un de ses voisins pour aboutir à un point de coût suffisament bas. Ce qui définit une méthode de recherche locale est la manière dont est défini le voisinage d'une solution donnée ainsi que la méthode de sélection dans ce voisinage.

La sophistication des méthodes de recherche locale peut être très grande et s'inspirer du monde physique et biologique : citons par exemple le recuit simulé, inspiré de la métallurgie, les algorithmes génétiques ou les techniques de colonie de fourmis. Ici, nous nous focaliserons sur une méthode de recherche locale particulière adaptée à la coloration de graphe et facile à mettre en œuvre, la recherche taboue ([GL97]).

4.3.3 Recherche taboue

Comme son nom l'indique, le principe de base de la recherche taboue est d'éviter de repasser par des solutions déjà explorées en les rendant « taboues » pendant une partie de l'exécution de l'algorithme. Le but est évidemment d'éviter que l'algorithme « boucle » autour de bonnes solutions mais surtout d'explorer en profondeur l'espace de recherche en choisissant à un instant donné des solutions qui peuvent être plus coûteuses mais qui permettent de se rapprocher d'une autre encore meilleure que les précédentes.

Dans une recherche taboue, il est nécessaire de configurer la manière dont l'ensemble des solutions taboues est implémenté, le temps durant lequel une solution est taboue et le voisinage d'une solution. Il est aussi possible, lorsque le tabou porte non pas sur un point de l'espace de recherche mais une famille de points comme ici, de définir des critères d'aspirations, c'est-à-dire des conditions pour lesquelles il est possible de prendre quand même un voisinage tabou.

4.4 Application à la coloration de graphe

Philippe Galinier et Alain Hertz proposent, dans [GH06], une utilisation efficace de la recherche taboue appliquée à la coloration de graphe. On se focalisera ici sur l'exploration de colorations totales. Un voisinage d'une coloration correspond à une coloration qui diffère d'au plus un nœud. Ainsi, pour une k-coloration d'un graphe à n nœuds, chaque solution a un voisinage de taille $n \times (k-1)$, puisque chaque nœud peut prendre $k-1$ autres couleurs. Cela signifie aussi que seuls $n \times k$ mouvements sont possibles pour une coloration donnée.

Ainsi, il peut être ingénieux de stocker l'ensemble des mouvements tabous non pas sous forme de liste de mouvements, mais de matrice $\mathcal{M}_{n,k}$ où $m(j,c) = i$ signifie que le mouvement correspondant à l'attribution de la couleur c au nœud j est interdit avant l'itération i. De cette manière, on sait en temps constant si un mouvement est tabou ou pas, au prix de maintenir une structure de taille $n \times k$.

[2] Dans le contexte de la recherche locale, le coût est une agrégation du nombre de contraintes violées et du coût tel que défini dans la partie programmation par contrainte.

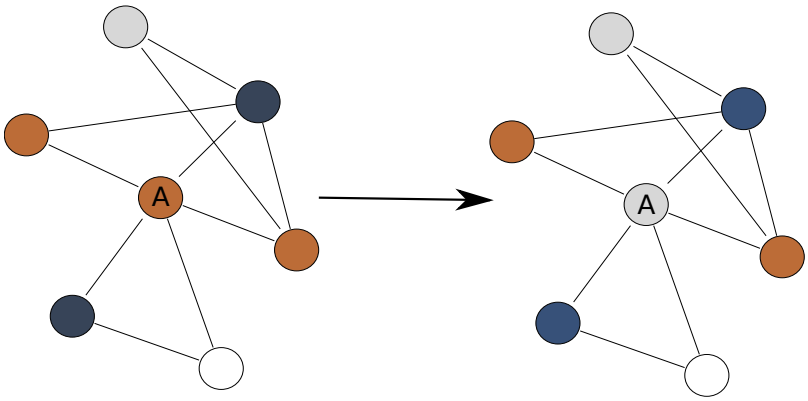

FIG. 4.1 – Une coloration d'un graphe et une coloration voisine du même graphe par changement de la couleur du nœud A.

Comme le but est de trouver une coloration du graphe admissible, le coût associé à une solution correspond au nombre de conflits dans le graphe, c'est-à-dire à la somme, pour chaque nœud, du nombre de voisins ayant la même couleur qu'eux. Ainsi, le coût d'un mouvement, c'est-à-dire l'attribution de la couleur c au nœud j, correspond au nombre de conflits créés par ce mouvement, *i.e.* le nombre de voisins de j ayant la couleur c, moins le gain lié au changement de couleur, c'est-à-dire le nombre de voisins de j ayant l'ancienne couleur. À nouveau, il est intéressant de stocker dans la matrice $\mathcal{M}_{n,k}$ le coût de chacun des mouvements et de mettre à jour ces différents coûts lors de chaque mouvement. Ainsi, quand la couleur c est attribuée au noeud j qui avait auparavant la couleur b, tous les mouvements attribuant la couleur b à un voisin de j voient leur coût augmenter de 1 alors que ceux attribuant la couleur c à ces mêmes voisins voient leur coût diminuer de 1.

Pour un voisinage donné, seule la meilleure solution, c'est-à-dire celle de gain maximal, est retenue. La trouver nécessite toutefois de parcourir l'ensemble de la matrice de voisinage.

L'algorithme **??** présente le fonctionnement général de cette recherche locale, où Λ désigne une constante (dont la valeur conseillée est 5), et f une fonction évaluant le nombre de conflits au sein de la coloration.

Algorithme 2 Tabucol

Construire une solution aléatoire c_{rand}
$c = c_{rand}$, $i = 0$
La liste tabou est vide
tant que $f(c) \neq 0$ ou $i < max_{iter}$ **faire**
  incrémenter i
  Choisir un mouvement (n, col) de coût minimal
  Rendre (n, col) tabou pendant $L + \Lambda \times f(c)$ itérations
  si le mouvement est de coût négatif **alors**
    Effectuer le mouvement (n, col)
  fin si
fin tant que

Troisième partie

Mise en œuvre et résultats

5

Réalisation

L'ensemble des techniques vu précédemment va être appliquée à notre problématique précise : la séparation de flux aériens.

5.1 Formalisation

Pour prendre en compte le fait que de nombreux vols suivent la même route aérienne et pour réduire la taille du problème, les vols d'une journée de trafic donnée seront agrégés en *flux* selon certains critères détaillés dans la section 5.4. Il est alors possible de représenter l'ensemble du problème par un graphe $G = (V, E)$, où chaque flux est associé à un nœud V et où deux sommets sont reliés par une arête E si et seulement si les deux flux se croisent dans le plan horizontal. Trouver une allocation de niveaux de vol où deux flux qui se croisent ne sont pas au même niveau correspond à trouver une coloration du graphe $col : V \to C$, en notant C l'ensemble des « couleurs » ou niveaux de vol disponible ([BB02]).

On cherche en outre à se rapprocher le plus possible des *Requested Flight Level* (RFL), c'est-à-dire des niveaux de vol demandés par les compagnies aériennes et proches de l'altitude optimale, *i.e.* celle où la consommation de carburant est la plus faible. On associe alors à chaque couple (flux, niveau attribué) un coût $cost : V, C \to \mathbb{R}$, pouvant dépendre de plusieurs facteurs tels que le nombre d'avions du flux ou la longueur de la trajectoire associée. Le problème revient donc à trouver c de façon à optimiser

$$f(cost(v_1, c(v_1)), ..., cost(v_n, c(v_n)))$$

f peut être la minimisation de la somme ou la minimisation du coût maximal, aussi appelée « minimax ». Toutefois, à notre connaissance, la littérature associée à la coloration de graphe se limite uniquement à des coûts fixes pour une couleur donnée, c'est-à-dire où le coût d'un couple (nœud, couleur) ne dépend que de la couleur.

5.2 Architecture

Afin de pouvoir tester des algorithmes et représentations de données différentes sans que la réécriture du code soit nécessaire, le problème est découpé en petits modules indépendants

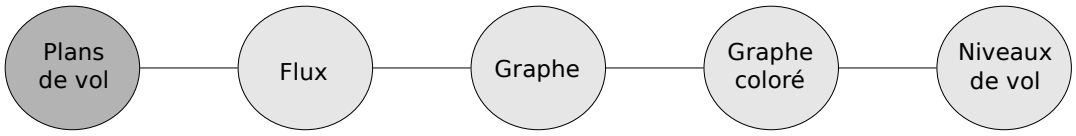

FIG. 5.1 – Différentes étapes de résolution.

(cef. section 5.1). Le rôle de chaque module est de convertir les données manipulées dans un format différent. Chaque représentation met en lumière des informations différentes, qui sont calculées à chaque étape : on passe ainsi des plans de vols individuels à une agrégation en flux d'avions ; les conflits entre ces flux sont ensuite détectés, puis mis en évidence à l'aide d'un graphe de contraintes ; une solution à ces contraintes est présentée sous forme de coloration du graphe ; enfin, l'attribution d'un niveau de vol à chaque flux, c'est-à-dire l'appariement entre une couleur du graphe et un niveau de vol, permet de résoudre le problème réel.

On remarquera que si cette architecture permet de tester différentes méthodes de manière indépendante, cela ne veut pas forcément dire que les problèmatiques soient totalement découplées. Par exemple, suivant l'optimisation à faire sur les niveaux de vol, il peut être intéressant de fusionner les étapes de coloration et d'attribution du niveau de vol : l'effectuer une permutation circulaire sur les couleurs / niveaux attribués ne change rien au problème de coloration de graphe mais change drastiquement l'écart cumulé au RFL.

5.3 Lecture des données

La première partie du traitement consiste en la lecture de fichiers générés par CATS (CAML All purpose Traffic Simulator), un simulateur de vol développé par la DSNA/DTI. Chaque fichier contient, pour une journée, les plans de vols déposés ainsi que les coordonnées des balises en place. Seulement, il arrive que des incohérences apparaissent entre les balises présentent dans la partie contenant les plans de vol et celle contenant les positions géographiques des balises.

Comme les balises peuvent évoluer avec le temps, avec par exemple la suppression de balises devenues obsolète ou la création de nouvelles, notamment virtuelles[1], comme l'incohérence des données est un problème connu de CATS et comme assez peu de balises sont concernées, il a été décidé d'ignorer les balises dont les coordonnées n'étaient pas disponibles, ce qui revient soit à ignorer le début ou la fin du vol si la balise est à une extrémité du plan de vol, soit à considérer que l'avion va en ligne droite entre la balise précédant et la balise suivant la balise ignorée.

Une autre catégorie de balises ignorées sont celles ne correspondant pas au niveau de vol demandé sur le plan de vol de l'avion ; elles correspondent donc à des phases de manœuvre de l'avion, typiquement le décollage ou l'atterissage. Hors, séparer des flux d'avions à l'aide de leur altitude n'a de sens qu'en phase de croisière, c'est-à-dire suffisament loin des aéroports[2].

[1] Une balise virtuelle est une cordonnée non associée à un équipement au sol tel qu'une antenne VOR (VHF Omnidirectional Range, antenne de diffusion omnidirectionnelle à très haute fréquence).

[2] Sur un aéroport, tous les avions sont à la même altitude.

5.4 Construction de flux

Il arrive que deux avions, décollant du même aéroport et ayant même destination, n'empruntent pas exactement le même plan de vol (figure 5.2), par exemple pour éviter de surcharger certains secteurs de contrôle. Symétriquement, des avions ayant des départs et destinations différentes peuvent se retrouver à un moment sur le même « couloir », par exemple parce que la route directe de l'un passerait par une zone militaire.

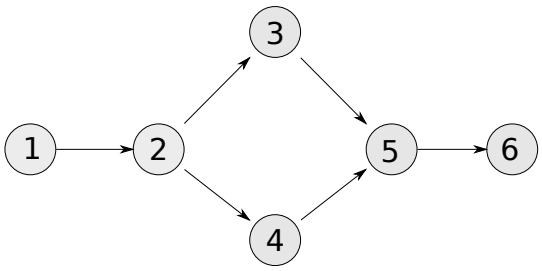

FIG. 5.2 – Les deux flux n'en forment qu'un sur les portions 1-2 et 5-6 et sont séparés sur les autres.

Plusieurs solutions peuvent être envisagées dans ces cas :
- Considérer qu'il n'y a en fait qu'un seul flux.
- Considérer qu'il y a deux flux différents et qu'ils doivent être à des niveaux de vol différents.
- Considérer qu'il y a deux flux, sans contrainte de niveau de vol différent.
- En multicoloration uniquement, considérer qu'il y a trois flux, le « commun » et les deux « séparés » et dire que les couleurs d'un séparé doivent être aussi les couleurs du commun. On peut choisir s'il est nécessaire que les « séparés » aient des couleurs différentes ou non.

On commence par considérer qu'il y a deux flux sans contrainte de niveau de vol différent, ce qui est la solution la moins contrainte. Afin d'éviter d'avoir à reconstruire les flux à chaque fois, la dernière solution est écartée ; celle retenue est de considérer que deux flux sont identiques s'ils passent par exactement les mêmes balises. Ensuite, les conflits entre flux sont marqués différemment s'il s'agit d'un croisement pur ou alors d'une superposition : un croisement sera toujours interprété comme une exigence de niveaux de vol différents alors qu'une superposition peut s'interprêter de différentes manières (flux inclus dans un autre ou flux en conflits).

Temporalité Pour simplifier la détection des conflits et pour rendre la déconfliction par paramètrage du niveau de vol robuste aux imprévus, tels que retard au décollage ou conditions météorologiques perturbées, on ne prend pas en compte la dimension temporelle de la trajectoire des avions ; ainsi, deux avions passant par les mêmes points de report, mais un le matin, l'autre l'après-midi, seront sur le même flux.

De plus, l'objectif à terme est de coupler la déconfliction par allocation de niveaux de vol à la déconfliction par retard au décollage : il n'y a donc pas de sens à considérer que deux avions ne sont pas en conflits car ils passent à la verticale de la même balise à 20 minutes

d'intervalle si l'un d'entre eux va ensuite être retardé de 30 minutes au décollage.

5.5 Construction du graphe de contraintes

Une fois les différents flux construits, il est nécessaire de rechercher les conflits entre eux, qui peuvent être, comme dit précédemment, des croisements au-niveau d'une balise, des croisements en-dehors des balises et des superpositions des flux sur au moins un segment.

Si la recherche de superpositions et croisements au niveau d'une balise entre deux flux est suffisament rapide lors qu'effectuée avec un algorithme naïf, en revanche, la recherche de croisements entre deux balises au milieu de segments de trajectoires est beaucoup trop longue pour pouvoir être effectuée entre tous les couples de flux[3]. Il est donc nécessaire d'utiliser un algorithme plus efficace permettant de limiter le nombre de calculs d'intersection.

5.5.1 Adaptation de l'algorithme de sweep line

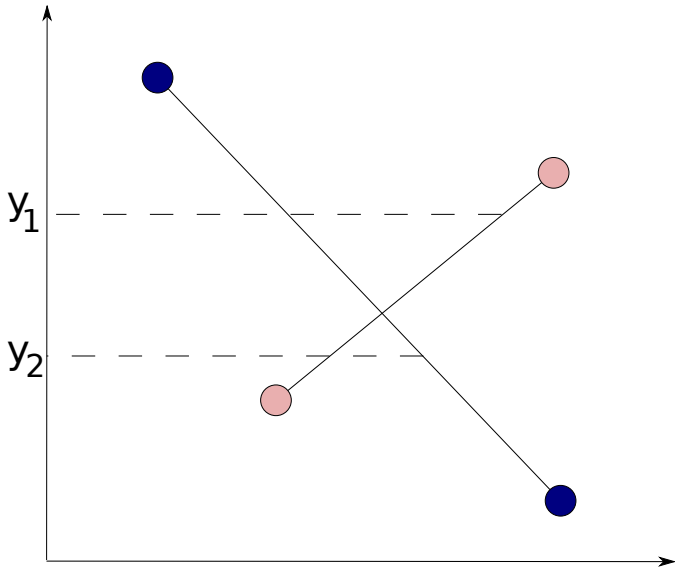

FIG. 5.3 – Deux segments et leurs ordres respectifs suivant la position de la sweep line : si celle-ci est $y = y_1$, le segment foncé est avant, et si elle est $y = y_2$, c'est le segment clair.

L'algorithme retenu pour cela est celui présenté dans le chapitre 2. Pour qu'il soit efficace, il est nécessaire d'utiliser des structures de données adaptées pour conserver les points évènementiels à venir ainsi que les segments explorés. Comme la seule opération nécessaire sur l'ensemble des points à venir est de trouver le prochain qui sera balayé, c'est-à-dire déceler le point minimum suivant une relation d'ordre prédéfinie. La structure idéale pour cette opération est le *tas* ou *maximier*, puisqu'elle donne accès à l'élément en temps constant, sa suppresion en $\Theta(log(n))$ et l'insertion d'un nouvel élément dans la structure en $\Theta(log(n))$.

[3]La détection reste inachevée après deux journées de calcul.

Pour les segments, les opérations effectuées sur la structure de donnée sont :
- trouver le segment le plus à gauche (ou à droite) de la structure ;
- pour un segment donné, trouver son voisin immédiatement à gauche (ou à droite) ;
- fusionner deux ensembles ;
- calculer la différence de deux ensembles ;
- pour un point donné, trouver tous les segments qui se finissent par ce point ;
- pour un point donné, trouver tous les segments qui contiennent ce point.

Si de Berg et Al. utilisent des arbres binaires équilibrés, dont les opérations élémentaires (ajout, suppression ou recherche d'un élément) se font en $\Theta(log(n))$, il est tout de même nécessaire de modifier cette structure pour qu'elle puisse s'adapter à une fonction de comparaison évolutive. En effet, si un segment est dit « plus petit » qu'un autre quand il est situé à gauche de celui-ci sur la *sweep line*, cela signifie que l'ordre de deux segments s'inverse au passage de leur intersection. Il est donc nécessaire d'ajouter un « contexte » à la fonction de comparaison de l'arbre. Ce contexte correspond à la sweep line du moment, comme illustré dans la figure 5.3. Lors de la gestion d'un point évènementiel (algorithme 1), les segments dont l'ordre s'inversent, c'est-à-dire ceux qui contiennent le point, sont retirés de la structure dans laquelle ils étaient stockés avec l'ordre précédent le point, puis ajoutés à nouveau, avec le nouvel ordre.

5.5.2 Recherche de cliques pour les contraintes Alldiff

Comme vu précédemment, en programmation par contraintes, l'utilisation de contraintes les plus globales possibles permet non seulement de gagner en lisibilité et compacité, mais aussi et surtout de mieux propager les contraintes

Il peut donc être intéressant de ne pas poser les contraintes de différence de niveau de vol aussitôt détectées, mais essayer de déceler le plus de cliques possibles dans le graphe afin de poser une contrainte Alldiff sur n variables plutôt que $\frac{(n-1)\times n}{2}$ variables de différences. Certaines de ces cliques sont très faciles à trouver car elles correspondent à plusieurs flux différents passant par la même balise, comme sur la figure 5.4. D'autres nécessitent de passer par le graphe des différences afin d'y détecter les cliques, comme sur la figure 5.5.

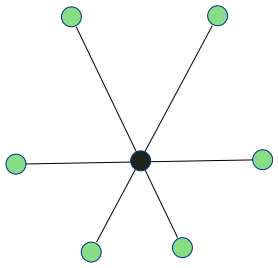

FIG. 5.4 – Lorsque trois flux passent par la même balise, ils sont directement mis dans une contrainte *Alldiff*.

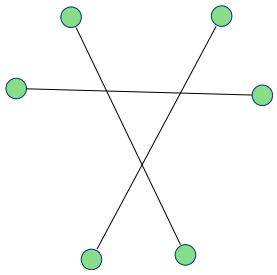

FIG. 5.5 – Ici, les trois flux pourraient être dans la même contrainte Alldiff, mais cela n'est pas détecté directement par l'algorithme de *sweep line*.

5.6 Heuristiques utilisées

En raison de la nature du problème, à savoir la minimisation de l'écart entre le niveau de vol demandé et le niveau de vol attribué, il a semblé naturel de commencer par instancier les variables représentant les niveaux de vol par distance au niveau de vol demandé croissant. Ainsi, la première solution admissible trouvée dans l'arbre de recherche est déjà très bonne et peu de retours arrières seront nécessaires pour trouver une meilleure solution. C'est d'ailleurs ce qu'il s'est passé : pour les dix journées de tests, en utilisant comme fonction objectif la minimisation du maximum d'écart, seules deux n'ont pas eu leur solution optimale trouvée du premier coup.

Comme le modèle utilise de très nombreuses variables, lorsqu'une solution optimale est trouvée le retour-arrière ne se fait pas par dé-instanciation puis ré-instanciation de variable mais par retour direct à la racine de l'arbre. Ce choix permet lui aussi des temps de calculs réduits.

Enfin, pour détecter au plus tôt les instanciations partielles sans solutions, les variables les plus contraintes, c'est-à-dire ayant le plus petit domaine et étant présentes dans le plus grand nombre de contraintes, sont instanciées en premières.

5.7 Agrégation des préférences et justice sociale

Le but du sujet est de minimiser l'écart entre le niveau de vol que les compagnies aériennes souhaitent utiliser pour leurs vols et celui qui leur est réellement attribué. Toutefois, il n'est pas possible de donner à tous les vols de toutes les compagnies leurs niveaux de vol préféré si l'on veut que les conflits soient évités. Il est donc nécessaire de pouvoir comparer plusieurs solutions acceptables (*i.e.* où il n'y a aucun conflit) afin de savoir lesquelles sont préférables.

Il est facile, pour un vol donné, de savoir si une solution s_1 est préférable à une autre solution s_2 : il suffit de regarder si l'écart entre le niveau de vol demandé et celui attribué dans s_1 est plus petit que celui attribué dans s_2. En revanche, que signifie une solution meilleure du point de vue d'une compagnie ? Avec le moins de vols « détournés » possibles ?

Avec l'écart cumulé sur tous ses vols minimum [4] ? Plus complexe encore, qu'est-ce qu'une solution globalement meilleure ?

Une première méthode consiste à minimiser l'écart cumulé, ce qui correspond à un optimum global pour la société (minimisation de la surconsommation de carburant et donc d'émissions de CO_2. Toutefois, ce n'est pas parce qu'une solution est globalement optimale qu'elle est juste et donc qu'elle sera acceptée par l'ensemble des acteurs. Il est donc nécessaire d'avoir une solution qui soit juste, ce qui a été défini comme une solution où les agents les plus « perdants » le sont le moins possible.

[4] En réalité, la réponse est très simple : une solution est meilleure pour une compagnie aérienne si elle est moins chère. Mais cela ne fait que déplacer le problème.

6

Résultats

6.1 Existence de solution et instance surcontrainte

La première partie du travail a consisté à prouver l'existence de solutions où les flux se croisant ne sont pas à la même altitude. Ce problème avait déjà été étudié par Nicolas Barnier et Pascal Brisset [BB02], mais avec des hypothèses beaucoup plus simplificatrices sur la trajectoire des avions, supposés aller de leur aéroport de départ à leur aéroport d'arrivée en ligne droite. Ici, un modèle plus réaliste est utilisé : les avions sont supposés suivre un plan de vol, c'est-à-dire se déplacer en ligne droite de balise en balise. Il s'agit encore d'une simplification ne prenant pas en compte ni la manœuvre réelle de l'avion lors d'un changement de cap (la trajectoire réellement suivie n'est pas une ligne brisée mais une grande courbe), ni les raccourcis éventuellement donnés par le contrôleur aérien, ni l'effet du vent, ni la précision limitée des Flight Management System (FMS), etc.

L'utilisation de la programmation par contrainte nous permet d'affirmer que, pour l'ensemble des journées étudiées, le problème est surcontraint : il n'est pas possible d'attribuer à chaque flux un niveau de vol unique tel que tous les flux se croisant soient à des niveaux différents, si on se limite aux niveaux de vols disponibles opérationnellement.

Pour palier à cela, plusieurs solutions peuvent être envisagées. Dans cette étude, nous avons appliqué la programmation par contraintes à une version simplifiée du problème, où les flux les plus petits, c'est-à-dire composés de moins de trois avions, sont ignorés. L'intérêt d'utiliser la programmation par contrainte est qu'elle offre des garanties d'optimalité. Nous avons aussi utilisé la recherche locale afin de tester une méthode de résolution radicalement différente et plus souple, mais sans garantie d'optimalité ou d'admissibilité des solutions trouvées. De plus il s'agirait alors de mettre sur le même plan une déviation par rapport au niveau de vol espéré, dont le coût est assez simple (consommation carburant et émission de CO_2) et un conflit entre avions, c'est-à-dire la charge mentale du contrôleur, dont le coût va dépendre de la situation (le contrôleur a-t'il des doutes ou est-il sûr que le conflit va se produire ? La situation était-elle déjà chargée ou au contraire calme ?). Déterminer la meilleure manière de combiner ces deux coûts dépasse le cadre de ce travail.

Il aurait été possible aussi d'utiliser des contraintes relâchées ou *soft constraint* pour exprimer le coût d'une autre manière : des contraintes « dures » exprimeraient le fait que deux flux en conflit soient sur des niveaux de vol différent et les contraintes « douces » que

le flux a son niveau de vol souhaité.

6.2 Utilisation de la programmation par contraintes

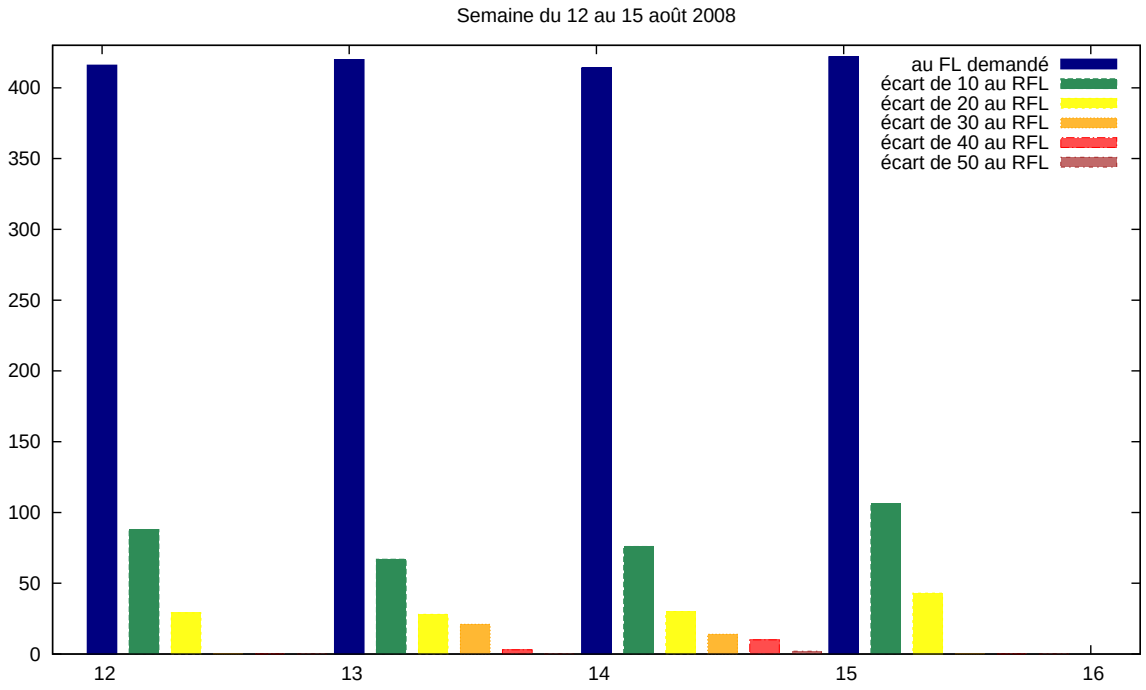

FIG. 6.1 – Nombre de flux à un écart donné par rapport à leur niveau de vol demandé en minimisant l'écart maximal.

De manière prévisible, il n'est pas possible d'obtenir une preuve d'optimalité pour des fonctions de type « Somme », c'est-à-dire où l'on cherche à minimiser le nombre de flux n'ayant par leur niveau de vol demandés. En effet, la propagation de contraintes est beaucoup plus difficile dans ce cas que dans celui où l'on cherche à minimiser l'écart maximal car les domaines des variables sont beaucoup moins réduits.

Comme le montre la figure 6.2, les flux se retrouvent beaucoup plus éloignés de leur niveau de vol optimal avec ««« Somme » plutôt qu'avec « Max ». Cela signifie donc plus d'externalités négatives, c'est-à-dire d'effets supplémentaires non pris en compte dans la modélisation, telles que des émissions de CO_2 supplémentaires ou le coût du carburant, ainsi q'une solution beaucoup moins juste[1].

[1]John Rawls ayant montré dans *A Theory of Social Justice* que la solution la plus juste est celle qui minimise le plus fort coût ou maximise le plus grand gain [**?**]. Il est possible de rafiner cette notion de justice, comme développé dans la section 6.2.1

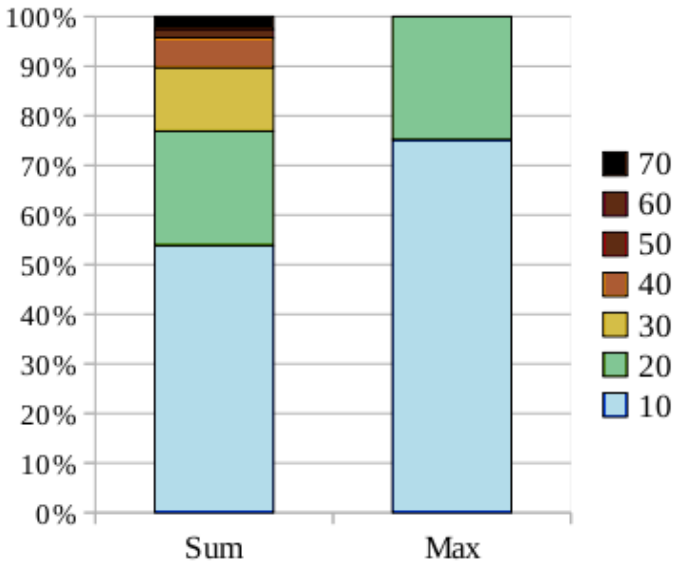

FIG. 6.2 – Flux n'ayant pas obtenu leur niveau de vol souhaité par écart au niveau de vol, pour la journée du 12 aout 2008, en utilisant deux coûts différents.

6.2.1 Leximin

Une fois que l'écart maximum au niveau de vol demandé est minimisé, il est encore possible d'améliorer la solution en mimimisant le nombre de flux étant à cet écart max e, puis le nombre de flux étant à l'écart $e - 1$, etc. Cette technique s'appelle *leximin* [**?**]. En faisant ainsi on obtient des résultats similaires à celui d'un minimax utilisé précédement.

Un autre ordre lexicographique pourrait être définit en donnant des ordres de priorités à certains flux, typiquement ceux qu'il est le plus pénalisant de dévier, c'est-à-dire ceux avec le plus d'avions et/ou de passagers. Cette définition de l'ordre de priorité peut se faire au sein du concept de Collaborative Decision Making de SESAR, où les compagnies aériennes négocieraient les ordres de priorité. Il sera peut-être nécessaire de faire payer le droit d'être prioritaire, pour que les compagnies révèlent leurs préférences réelles (si elles sont prêtes à payer pour un service, c'est qu'il « vaut le coup ») alors que des négociations sans contreparties risqueraient d'induire des biais.

6.3 Utilisation de la recherche taboue

Pour améliorer les solutions trouvées par la programmation par contrainte ou pour trouver une solution « acceptable », c'est-à-dire ne respectant pas les contraintes de différence de niveaux pour tous les flux se croisant mais ne créant toutefois « pas trop » de conflits et avec des niveaux de vol proches de ceux souhaités.

Si la recherche taboue donne de bons résultats lorsque l'on cherche une coloration acceptable du graphe, il est nécessaire de configurer finement l'équilibre entre exploration de

colorations non admissibles (c'est-à-dire avec conflits) et celle de colorations admissibles avec de grands écarts. À cette configuration fastidieuse a été préférée l'utilisation d'heuristiques de programmation par contrainte plus fines, puisque cette méthode donnait de bons résultats avec le problème réellement traité.

6.4 Perspectives

6.4.1 Optimalité

Il est assez frustrant de ne pas avoir de preuve d'optimalité pour les solutions trouvées en cherchant à minimiser le nombre d'avions n'ayant pas leur RFL, mais il serait intéressant de tester des heuristiques complexes ou adaptatives, telles que celles présentées par Simon Marchal dans [Mar10]. Toutefois, nous sommes ici en présence d'un problème opérationnel concret et d'autres points sont à améliorer proritairement à celui-ci : il s'agit de parvenir à modéliser de manière plus réaliste encore le problème ainsi que de discuter de la validité des solutions proposées aux principaux acteurs concernés.

6.4.2 Incertitude sur les trajectoires 4D

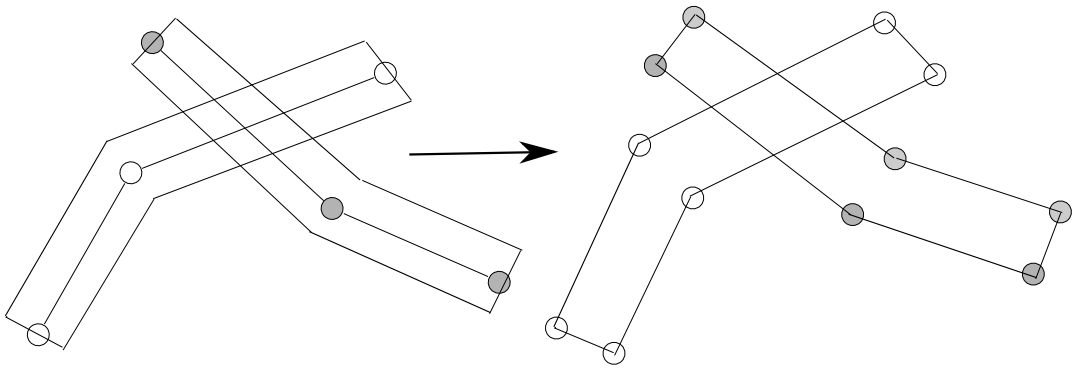

FIG. 6.3 – Représentation de deux portions de flux aériens sous forme de ligne brisée augmentée d'une marge d'incertitude et/ou de norme de séparation, puis sous forme de contour de polygone.

Les trajectoires aériennes utilisée ici sont des lignes brisées, comme illustré sur la première partie de la figure 6.4.2. Hors, il peut-être intéressant d'y ajouter une marge qui représenterait l'incertitude sur la position précise de l'avion et/ou la norme de séparation de 5Nm. On ne chercherait plus a priori alors des intersections de segments, mais des superpositions de polygones.

En réalité, savoir si deux polygones se superposent revient à savoir si leurs contours se croisent, comme illustré sur la deuxième partie de la figure 6.4.2 et la généralisation de l'algorithme de *sweep line* aux contours a déjà été faite par de Berg, van Breveld, Overmars et Schwarzkopf dans [dBvKOS05].

6.4.3 Mise en place opérationnelle

Il est nécessaire, pour poursuivre des recherches dans ce sens, de savoir si les compagnies aériennes sont prêtes à accepter, en contre-partie de moins de retards au décollage qui leurs sont coûteux, des niveaux de vol sous-optimaux, dans un contexte où elles subissent une fiscalité les incitant justement à limiter leurs consommation de carburant.

Conclusion

Cette étude a pour objet d'explorer la possibilité de supprimer les conflits aériens potentiels en utilisant comme degré de liberté le niveau de vol des flux aériens, dans la perspective de combiner cette méthode avec la déconfliction par retard des vols au décollage afin de la rendre plus résistante aux incertitudes.

Elle s'inscrit au sein de l'ambitieux projet SESAR portant sur l'optimisation de la gestion du trafic aérien. Elle se situe au niveau stratégique, c'est-à-dire à long terme, en gardant l'opérateur humain au cœur de la boucle de contrôle et en se focalisant sur sa charge mentale et non plus sa charge de travail.

Un algorithme efficace issu de la géométrie algorithmique, celui de *sweep line*, a été utilisé pour détecter efficacement les conflits entre flux aériens. La séparation des flux a été modélisée comme un problème de coloration de graphes et a ensuite été résolus par programmation par contraintes, méthode qui avait déjà fait ses preuves lorsque le degré de liberté de la déconfliction est le retard au décollage.

Bibliographie

[BB02] Nicolas Barnier and Pascal Brisset. Graph coloring for air traffic flow management. In *CPAIOR'02 : Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems*, pages 133–147, Le Croisic, France, Mars 2002.

[BHLS] Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In *16th European Conference on Articial Intelligence*.

[Bic07] Charles-Edmond Bichot. *Élaboration d'une nouvelle métaheuristique pour le partitionnement de graphe : la méthode de fusion-fission. Application au découpage de l'espace aérien*. PhD thesis, INPT, 2007.

[COfDA10] Eurocontrol Central Office for Delay Analysis. Delays to air transport in europe, Mars 2010.

[dBvKOS05] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfired Schwarzkopf. *Computational Geometry - Algorithms and Applications*. Springer-Verlag, 2005.

[FD95] Daniel Frost and Rina Dechter. Look-ahead value ordering for constraint satisfaction problems. In *IJCAI'95 : Proceedings of the 14th international joint conference on Artificial intelligence*, pages 572–578, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[GH06] Philippe Galinier and Alain Hertz. A survey of local search methods for graph coloring. *Computers & Operations Research*, 33(9) :2547 – 2562, 2006. Part Special Issue : Anniversary Focused Issue of Computers & Operations Research on Tabu Search.

[Gia10] David Gianazza. Forecasting workload and airspace configuration with neural networks and tree search methods. In *Artificial Intelligence*, volume 174, pages 530–549, May 2010.

[GL97] F. Glover and M. Laguna. Tabu search, 1997.

[Mar10] Simon Marchal. Stratégie de recherche adaptative pour la déconfliction de trajectoires 4d, Septembre 2010.

[MPJL93] Steven Minton, Andy Philips, Mark D. Johnston, and Philip Laird. Minimizing conflicts : A heuristic repair method for constraint-satisfaction and scheduling problems. *J. ARTIFICIAL INTELLIGENCE RESEARCH*, 58 :161–205, 1993.

[PEC03] Principles, Examples, and Jens Clausen. Branch and bound algorithms -, 2003.

[Riv06] Thomas Rivière. *Optimisation de graphes sous contraintes géométriques : création d'un réseau de routes aériennes pour un contrôle sector-less.* PhD thesis, 2006.

[RN05] Esa M. Rantanen and Ashley Nunes. Hierarchical conflict detection in air traffic control. In *The International Journal of Aviation Psychology*, volume 15, pages 339 – 362, Octobre 2005.

[Smi99] Barbara M. Smith. The brélaz heuristic and optimal static orderings. In *In Proceedings CP'99*, pages 40541–8, 1999.

Quatrième partie

Annexes

Glossaire

Acronymes

- **ASM** : Air Space Management (gestion de l'espace aérien)
- **ATM** : Air Traffic Management (gestion du trafic aérien)
- **CAML** : Categorical Abstract Machine Language
- **CATS** : CAML Air Traffic Simulator
- **CFMU** : Central Flow Management Unit (unité de gestion et d'optimisation des flux aériens)
- **CNS** : Communication, Navigation, Surveillance
- **CODA** : Central Office for Delay Analysis (bureau d'analyse des retards d'Eurocontrol)
- **CSP** : Constraint Satisfaction Problem (problème de satisfaction de contrainte)
- **CENA** : Centre d'Études de la Navigation Aérienne
- **DSNA** : Direction des Services de la Navigation Aérienne
- **DTI** : Direction de la Technique et de l'Innovation
- **FL** : Flight Level (niveau de vol)
- **FMS** : Flight Management System (système de gestion du vol)
- **FUA** : Flexible Use of Airspace (utilisation flexible de l'espace aérien : système de négotiation de l'espace aérien entre utilisation civile et militaire)
- **GNSS** : Global Navigation Satellite System (Système de positionnement par satellites)
- **IFR** : Instrument Flight Rules (vol aux instruments)
- **OACI** : Organisation de l'Aviation Civile Internationale
- **ODS** : Operational Display System
- **RFL** : Requested Flight Level (niveau de vol demandé)
- **SESAR** : Single European Sky ATM Research
- **SHAMAN** : System to Help Analysis and Monitoring of Acc resources and Air route Network
- **TCAS** : Traffic alert and Collision Avoidance System (système d'alerte de trafic et d'évitement de collision)
- **VFR** : Visual Flight Rules (vol à vue)

Unités

- **kn** : Nœud $= 1,852\ km.h^{-1}$

– **Nm** : Mille marin (Nautical mile) = 1 852 m

Lexique

Altitude, hauteur et niveau de vol : En aéronautique, la *hauteur* désigne la distance entre l'avion et la projection verticale de l'avion sur la surface de la Terre. Cette donnée, difficile à mesurer, n'est pas utilisée par les avions. À la place est mesurée l'altitude-pression, c'est-à-dire la pression atmosphérique dans l'environnement immédiat de l'avion, cette altitude-pression étant décroissante avec la hauteur. Ainsi, deux avions ayant deux altitudes-pressions différentes sont assurés d'être séparés verticalement. L'*altitude*estla hauteur correspondant à la pression mesurée dans le modèle de l'atmosphère standard, exprimé en pieds. Le *niveau de vol* correspond à l'altitude, à une constante multiplicative près : par exemple, un avion au niveau de vol 270 se trouve à 27 000 pieds d'altitude.

Externalité : En économie, effet non pris en compte par le marché. Lorsqu'une industrie chimique s'intalle à un endroit, la pollution de la rivière voisine est une externalité négative, le développement d'entreprises de restauration une externalité positive. L'un des objectifs d'une bonne politique économique est d'intégrer les externalités dans le marché, notamment par incitation fiscale.

Hub : Un aéroport sert de hub à une compagnie aérienne lorsque celle-ci y effectue un nombre conséquent de ses escales. L'activité typique d'un aéroport de hub est une à deux heures d'arrivées quasi-exclusives, quarante minutes à une heure de quasi-absence d'activité et une à deux heures de départs quasi-exclusifs. Ainsi, un très grand nombre de paires Origine/Destination peuvent être vendues aux clients des compagnies aériennes sans que celles-ci aient besoin de multiplier le nombre de vols. Pour n aéroports de départs et m aéroports d'arrivées, passer par un hub signifie assurer $n + m$ vols tandis que des liaisons directes signifient $n \times m$ vols. L'aéroport Lyon St-Exupéry est l'un des hub d'Air France, l'autre étant Paris-Charles de Gaulles, mais dont l'activité de *hub* est moins typique en raison des nombreux vols d'autres compagnies qui y sont assurés.

Crédits

Illustrations

- 1.1 : CENA
- 1.5 : DTI, Pôle Planification, Optimisation et Méthodes
- 3.4, 3.6, 3.8 : Travaux dérivés de Chess qlt45.svg, par Cburnett, licence Creative Commons Attribution-Share Alike 3.0 Unported
  (`http://commons.wikimedia.org/wiki/File:Chess_qlt45.svg`)
- 3.7 : Domaine public
  (`http://commons.wikimedia.org/wiki/File:Forward-arc-1.svg`)

Packages LaTeX

- Algorithmes mis en pages à l'aide du package *algorithms*.