Résolution de conflit aérien à l'aide d'un algorithme A*

CAILLOUX Jocelin, LAZAR Marouane, SEGNERE-YTER Marine

Février 2018

Master 2 RO

Contents

1 Introduction

Le transport aérien a connu une forte expansion ces dernières années. Il fait aujourd'hui partie intégrante de notre mode de vie. Parallèlement à ce développement, la gestion du trafic aérien est devenue un domaine indispensable à son bon fonctionnement. Dans ce système, les contrôleurs aériens occupent une place centrale et assurent la sécurité des différents vols tout au long de leur trajet. Ces contrôleurs font face quotidiennement à des situations conflictuelles. Dans le présent document, nous nous sommes intéressés qu'aux conflits de trajectoires, c'est-à-dire lorsque deux avions se trouvent trop proches l'un de l'autre. Alors, parmi les problèmes majeurs de l'aéronautique, la détection et la résolution de ces conflits en amont ou dans un temps court est un enjeu majeur.

Dans un premier temps, nous aborderons le problème que nous avons traité, puis sa modélisation. Ensuite, nous présenterons l'algorithme que nous avons mis en place, l'algorithme A*. Enfin, nous discuterons des résultats que nous avons obtenus.

2 Problème

On définit un conflit entre deux avions si leur trajectoires les amènent à se retrouver dans une position ne respectant pas les normes de séparation entre eux. Alors, la résolution de ce conflit consiste à trouver des trajectoires alternatives de manière à respecter ces normes de sécurité et dont la déviation pour les deux avions est de coût minimal par rapport aux trajectoires initiales.
Il s'agit de résoudre un problème d'optimisation sous contraintes, on minimise la somme des coûts de changements de trajectoire pour chaque avion, tout en résolvant l'ensemble des conflits aériens.
Pour cela, nous disposions de 2 types de fichiers différents :

- le premier contenant une liste d'avions et les trajectoires pouvant être prises par ces derniers;
- le second indiquant le coût des manoeuvres et pour chaque couple d'avion et de manoeuvre pris 2 à 2 si un conflit va exister.

3 Modèle

La nature du modèle d'optimisation qu'on traite étant combinatoire, nous proposons de le formuler par un programme linéaire en nombres entiers (PLNE).

3.1 Paramètres du modèle

M : Le nombre de manoeuvres.
N : Le nombre d'avions.
C_m : Coût de la manoeuvre m.

$$NG_{ni,nj,mi,mj} = \begin{cases} 1 \text{ si la manoeuvre mi de l'avion ni crée un conflit avec la manoeuvre mj de l'avion mj.} \\ 0 \text{ sinon} \end{cases}$$

3.2 Les variables de décision

x_{ni} : la manoeuvre assignée à l'avion n_i.

3.3 Le modèle général

$$\min \sum_{ni=1}^{N} C_{x_{ni}} \tag{1}$$

Sous les contraintes:

$$NG_{ni,nj,x_{ni},x_{nj}} = 0 \quad ni, nj \in \{1, ..., N\}^2, ni \neq nj \tag{2}$$

4 Algorithme

Le corps principal de l'algorithme peut être décrit simplement par un petit nombre d'actions :

- Initialisation :
  - Création du premier noeud, racine de l'arbre de recherche, aucun avion fixé.
  - Ajout de la racine dans la file de priorité.
  - Initialisation d'une borne supérieure ub impossible.
- Tant que la borne inférieure du meilleur noeud de la file est plus petit que la borne supérieure :
  - Stocker le noeud n et le retirer de la file de priorité.
  - Sélectionner a, l'avion qui a le plus petit domaine.
  - Pour chaque manoeuvre m réalisable selon l'état de n, calculer une borne inférieure lb en fonction des avions non fixés avec a utilisant la manoeuvre m.
  - Si la borne inférieure lb est plus petite que ub :
    - Estimer une nouvelle borne supérieure. Si elle est plus petite que ub, remplacer ub et stocker la solution associée.
    - Créer le noeud fils et l'ajouter à la file.

– Supprimer n.

S'il existe une solution, elle a été trouvée par l'heuristique qui nous permet de calculer une borne supérieure. En effet, s'il ne reste qu'un avion à fixer sur une branche de l'arbre de recherche, la borne supérieure est forcément trouvée si elle existe. Elle est la solution optimale de cette branche.
De plus, l'ordre de priorité dans la file d'attente correspondant aux bornes inférieures associées aux noeuds dans l'ordre croissant, alors nous pouvons arrêter l'algorithme dès que le premier noeud de la file est associé à une borne inférieure égale à la borne supérieure courante.

5 Sous algorithmes

5.1 Calcul de la borne inférieure

La borne inférieure est l'un des éléments les plus importants de l'algorithme. Plus elle est de bonne qualité plus l'algorithme convergera rapidement. De plus, la borne inférieure étant calculée pour chaque noeud, il est important que son calcul soit rapide.

Une borne inférieure de qualité pour ce problème est également très difficile à obtenir.

Nous avons sélectionnée une méthode gloutonne en relâchant les contraintes de conflits entre les avions pas encore fixés. Ainsi, notre borne est obtenue en sélectionnant pour chaque avion non fixé la manoeuvre la moins coûteuse évitant tout conflit avec les avions déjà fixés et présents dans l'état courant du noeud.

Cette borne inférieure est d'une qualité relativement mauvaise mais elle a l'avantage d'être rapide à calculer. Il est possible de détecter ici qu'un noeud ne mènera à aucune solution, bien qu'après expérience cet évènement semble rare dû au grand nombre de manoeuvres.

5.2 Calcul de la borne supérieure

La borne supérieure est un ajout non nécessaire à l'algorithme mais il apporte une aide considérable. Obtenir des bornes supérieures et conserver la plus petite permet d'exclure des noeuds ayant une borne inférieure trop grande et donc de couper des branches de l'arbre de recherche le plus tôt possible. Comme nous

visitons noeuds proposant la meilleure borne inférieure, les noeuds supprimés grâce à la borne supérieure ne seraient jamais visités mais les retirer permet d'économiser la mémoire et d'éviter beaucoup de calculs inutiles à chaque noeud. Elle nous a permis d'obtenir des solutions en un temps raisonnable sur la plupart des instances à 20 avions.

Le calcul de cette borne se fait par un algorithme glouton encore plus rapide que celui de la borne inférieure. Il n'est pas coûteux mais répété à chaque noeud il est non négligeable.

Notre borne supérieure est obtenue en plongeant dans l'arbre de recherche vers la branche la moins coûteuse qui conserve la faisabilité de la solution. Si l'algorithme glouton nous amène à un noeud qui ne propose aucune solution, il est abandonné et nous n'obtenons pas de borne supérieure.

Afin de limiter le nombre de bornes supérieures calculées, nous a stockons la plus grande profondeur dans l'arbre de recherche des noeuds obtenus par la file de priorité. Notre algorithme ne calcule ainsi que les bornes supérieures des noeuds dont la profondeur est supérieure ou égale à la plus grande rencontrée.

6 Améliorations essayées

Nous avons expérimenté en cas d'égalité des bornes inférieures entre deux noeuds de rendre plus prioritaires celui qui a la plus grande profondeur (nous avons également essayé l'inverse). Nous n'avons pas remarqué un grand impact. En effet, nous pensons que comme tous les noeuds ayant une borne inférieure égale seront visités les uns après les autres sans coupure, il n'est pas nécessaire de mettre de priorité entre eux.
Nous pensions pouvoir obtenir plus rapidement des bornes supérieures de qualité grâce à cette astuce.

Lorsque nous obtenons une borne supérieure pour une noeud, peut-être qu'il peut-être intéressant de favoriser ce noeud par rapport aux autres dans la file de priorité.
Soit lb et ub respectivement les bornes inférieures et supérieures obtenues pour le noeud. Alors sa priorité sera de $lb - 1 + 1/(ub + 0.5)$ alors que sans borne supérieure, la priorité d'un noeud est sa borne inférieure.
Ainsi, un noeud ayant obtenu une borne supérieure sera plus prioritaire qu'un noeud ayant obtenu une borne inférieure égale mais n'ayant pas obtenu de borne supérieure. Il sera également moins prioritaire qu'un noeud ayant obtenu une plus petite borne inférieure. Il existe même un rapport entre les noeuds de même borne inférieure pour lesquels nous avons réussi à obtenir une borne supérieure.

Étant donné que tous les noeuds créés doivent être visités jusqu'à ce que la borne inférieure des meilleurs noeuds rejoignent la borne supérieure courante. Cette tentative d'amélioration a un gros impact sur certaines instances mais n'influe pas le temps de résolution dans la plupart des cas.

Nous fixons les avions de manière non lexicographique. Lorsque nous récupérons un noeud, nous sélectionnons l'avion possédant le plus petit domaine en fonction des avions déjà fixés dans l'état du noeud. Cette amélioration peut ralentir la résolution de certaines instances mais accélère beaucoup les plus difficiles en ralentissant l'explosion combinatoire des noeuds. En effet, chaque noeud a ainsi le moins de fils possible.

Nous avons eu l'idée d'utiliser la programmation linéaire pour le calcul des bornes inférieures en relâchant des variables binaires en variables continues. La résolution fonctionnait avec cette nouvelle borne inférieure mais le temps de calcul était beaucoup trop long (nous utilision un solveur gratuit, glpk, et nous avions N*M variables et le nombre de contraintes dépendait principalement du nombre de combinaisons créant un conflit entre deux avions).
Nous avons abandonné cette amélioration, avec un peu plus de temps peut-être que nous aurions insisté car la borne inférieure obtenue ne peut pas être pire que celle que nous avons actuellement.

7 Perspectives d'amélioration

Nous pensons que la meilleure perspective d'amélioration du A* pour ce problème est de guider la recherche d'une autre manière qu'avec la borne inférieure. Par exemple, un algorithme de recherche locale pourrait juger du potentiel d'une branche en tenant compte de la taille des domaines restant à fixer. Nous perdrions certaines propriétés de notre A* mais si nous arrivons à guider l'algorithme rapidement vers des solutions proches de l'optimal le nombre de noeuds visités pourrait ainsi être grandement diminué.

Il est possible d'améliorer l'algorithme en utilisant une méthode de résolution inspirée de la programmation par contraintes. En effet, stocker le domaine des variables dans chaque noeud, bien que coûteux en mémoire, est possible grâce à notre borne supérieure qui réduit le nombre de noeud créés. Toutefois, cela implique de recopier à chaque noeud une matrice de booléens de tailles N*M et l'utilisation d'intervalles pour les domaines n'est pas pertinent ici.

8 Résultats

En l'état actuel, notre algorithme permet de résoudre les instances jusqu'à 20 avions. Pour les instances de 20 avions testées, nous obtenons généralement la solution en quelques dizaines de secondes.

Concernant les instances de 25 avions, la résolution est généralement trop longue mais notre algorithme arrive à trouver la solution optimale de certaines de ces instances en un temps raisonnable.

9 Conclusion

L'algorithme A* fonctionne et permet de trouver une solution optimale jusqu'à une vingtaine d'avions. Cet d'algorithme n'est toutefois pas recommandable pour ce type de problème car la forte combinatoire du problème impose

Nos recherches bibliographiques nous ont appris que des algorithmes génétiques étaient préférés pour la résolution de conflits aérien alors que l'algorithme A* est, lui, plutôt utilisé dans des recherches de plus court chemin. En effet, étant une heuristiques et stockant peu d'états différents, un algorithme génétique peut obtenir des résultats satisfaisants pour un grand nombre d'avions en un temps raisonnable sans se soucier de la combinatoire du problème.

References

[1] *http://clusters.recherche.enac.fr/.*
Conflits aériens et recherche opérationnelle

[2] Geraud Granger
Détection et résolution de conflits aériens : modélisations et analyse.
Conflits aériens et algorithme A*, 15-25:108–115, 2002.

[3] *http://maiaa.recherche.enac.fr/fr/applications-atm/resolution-de-conflits-aeriens/.*
Conflits aériens et recherche opérationnelle, 2018.