Rapport de projet OCaml/IA

Résolution de conflits aériens par Branch & Bound

COSTE Dorian

LI Zhen

POUGET Lilian

TOBELEM Sam

Encadrant : Richard Alligier

Table des matières

[Introduction 3](#_Toc532367727)

[Présentation du problème 3](#_Toc532367728)

[Notre organisation 4](#_Toc532367729)

[Première version : séparation primale 5](#_Toc532367730)

[Présentation de Ocaml 5](#_Toc532367731)

[Présentation de l’algorithme *Branch and Bound* 5](#_Toc532367732)

[Benchmarking 6](#_Toc532367733)

[Deuxième version : amélioration avec problème dual et arc-consistence 8](#_Toc532367734)

[Amélioration de l’heuristique par la dualité 8](#_Toc532367735)

[Amélioration avec l’arc-consistance 8](#_Toc532367736)

# Introduction

## Présentation du problème

Nous choisissons le projet « Résolution de conflits aériens par Branch & Bound ». Le projet est mis en place pour nous permettre de confirmer notre maîtrise du langage de programmation fonctionnelle OCaml, et pour complémenter le cours sur les algorithmes d’Intelligence Artificielle *(IA)*.

Le problème que nous devons résoudre est, en soi, simple.

* Nous avons plusieurs avions, indiqués par la lettre i. Nos avions passent dans une zone de contrôle, et doivent tous s’éviter entre eux.
* Nous savons déjà quelles manoeuvres chaque avion peut effectuer (il y a 160 possibilités). Un modèle géométrique a permis de déterminer, pour chaque paire d’avions (i, j), quelles manoeuvres compatibles peuvent effectuer ces avions afin de conserver une séparation suffisante.

Par exemple, on sait que si l’avion 1 effectue la manoeuvre 160 (continuer tout droit), alors l’avion 2 peut effectuer uniquement les manoeuvres 1 à 20 (pas la manoeuvre 160 : il ne peut pas continuer tout droit).

* Nous connaissons également le coût des manoeuvres. La manoeuvre 160 a un coût nul.
* Nous voulons trouver la solution, qui est la meilleure combinaison des manoeuvres de chaque avion. Les manoeuvres choisies pour chaque paire d’avions doivent être compatibles, et la somme des coûts des manoeuvres doit être minimale.

Le problème se formume ainsi comme un problème d’optimisation combinatoire entière (Programmation Linéaire en Nombres Entiers, ou PLNE) :

Ici :

* est notre variable de décision, la combinaison des manoeuvres : , avec le nombre d’avions et le nombre de manoeuvres possibles.
* est la fonction objectif, la somme des coûts des manoeuvres choisies.
* est l’ensemble des contraintes : il décrit quels sont valides au regard de la compatibilité des manoeuvres de chaque paire d’avion.

Lorsqu’il y a peu d’avions et peu de manoeuvres possibles, on résout en testant chaque possibilité.

Prenons un exemple avec 3 avions et 3 manoeuvres possibles :

|  |  |  |  |
| --- | --- | --- | --- |
| **Avion 1 : Manoeuvre** | **Avion 2 : Manoeuvre** | **Avion 3 : Manoeuvre** | **Résultat** |
| 1 | 1 | 1 | Compatible : coût total 30 |
| 1 | 1 | 2 | Incompatible |
| 1 | 1 | 3 | Incompatible |
| 1 | 2 | 1 | Incompatible |
| 1 | 2 | 2 | Compatible : coût total 60 |
| 1 | 2 | 3 | Compatible : coût total 20 |
| 1 | 3 | 1 | Incompatible |
| ... ... ... ... | | | |
| 3 | 3 | 3 | Incompatible |

On trouve alors la solution, qui est le résultat *Compatible* de coût total minimal.

Le problème, c’est que nous voulons traiter de gros problèmes, avec 160 manoeuvres possibles et des dizaines d’avions. Pour 30 avions, notre tableau aurait ... lignes !

En effet, avec cette méthode de résolution, la complexité est exponentielle, en . Nous cherchons donc à optimiser la résolution du problème, pour qu’un ordinateur puisse l’effectuer en un temps raisonnable pour 30 ou 40 avions.

L’algorithme de *Branch and Bound* va nous permettre de supprimer des lignes à notre tableau, d’évacuer des possibilités qu’on sait ne pas être solution : nous n’aurons alors plus à considérer toutes les possibilités.

Au cours du projet, nous évaluons des solutions algorithmiques différentes, dans le but de pouvoir résoudre le problème en un minimum d’opérations possibles. Nous mettons donc en place ces solutions, et les testons pour voir lesquelles sont plus rapides selon les cas.

## Notre organisation

Dorian a tout de suite pris le projet en main. Lui et Lilian ont rédigé les grandes lignes de l’algorithme. Zhen a écrit des fonctions annexes (comme des fonctions de filtrage de solutions compatibles). Sam a aidé Zhen, et s’est concentré sur la rédaction du présent rapport.

# Première version : séparation primale

La première version de notre algorithme est notre *Mimimum Viable Product* : il doit … fonctionner.

Nous nous concentrons sur la mise en place de l’algorithme tout simple, car nous voulons rapidement avoir un algorithme qui fonctionne et qu’on pourra améliorer dans un deuxième temps.

## Présentation de Ocaml

OCaml est un langage de programmation de niveau industriel supportant les styles fonctionnel, impératif et orienté-objet (ocaml.org)

Nous profitons surtout des fonctionnalités qu’offre le langage Ocaml pour coder en programmation fonctionnelle (i.e. sans effet de bord).

Ce langage est adapté pour coder des algorithmes d’optimisation. L’avantage est que le code est facile à débugger car, à la compilation, toutes les erreurs de liaison entre les fonctions sont relevées : on peut donc manipuler plus facilement des types complexes (comme des *int list array*).

## Présentation de l’algorithme *Branch and Bound*

Un algorithme par séparation et évaluation, ou Branch and Bound en anglais, est une méthode générique de résolution de problèmes d'optimisation combinatoire. (Wikipédia)

L’algorithme de Branch and Bound ne se contente pas d’énumérer des solutions (*Branch*, ou séparation récursive du problème en plus petits sous-problèmes) : il borne (*Bound*) la solution.

On classe les manoeuvres à choisir selon un heuristique : le coût minimal potentiel total de la solution, dans le cas du choix de la manoeuvre en question. Il nous permet d’éviter de traiter un grand nombre de solutions.

Exemple : supposons que la manoeuvre 2 est sélectionnée pour le premier avion, et que pour le deuxième avion, on sache :

* Si on choisit la manoeuvre 1, le coût total sera au minimum 10
* Si on choisit la manoeuvre 2, le coût total sera au minimum 20

*Avion 4*

*Avion 3*

*Avion 2*

*Avion 1*

Dans ce cas, nous commençons par séparer et considérer la manoeuvre 1. Si, après traitement, on trouve une solution qui coûte 15, on n’aura même pas à considérer la manoeuvre 2, car on sait qu’un tel choix nous donnera de toute manière une solution moins bonne que celle trouvée.

Le calcul de l’heuristique nécessite peu d’opérations : on somme les coûts des manoeuvres compatibles les moins chères pour les avions restants (ici : les avions 3 et 4). Trouver une heuristique demande donc un temps constant (au maximum opérations si les manoeuvres sont déjà triées).

Notons que, dans le calcul de l’heuristique, la compatibilité entre les manoeuvres à choisir sur les avions restants (3 et 4) n’est pas prise en compte : on prend en compte uniquement la compatibilité avec la manoeuvre de l’avion 2. D’où le fait que l’heuristique n’est qu’une minoration : la solution « prendre toutes les manoeuvres les moins chères sur les avions restants » n’est sûrement pas valide !

Pour rendre tous ces calculs plus rapides, nous avons fait en sorte que les manoeuvres soient triées en coût croissant. C’est logique : nous voulons considérer en premier les manoeuvres les moins coûteuses.

## Benchmarking

Le contenu fourni par Richard Alligier comporte plusieurs fichiers de benchmarking. Chaque fichier représente une situation, un problème, et contient des informations sur les trajectoires des avions : entre autres, la matrice de compatibilité des manoeuvres entre chaque avion.

On teste donc notre algorithme sur différents fichiers, pour tester le temps qu’il met à trouver la solution. La valeur qui nous intéresse est la taille maximale d’un problème solvable en un temps raisonnable (i.e. quelques minutes).

Pour cette première version, on arrive à traiter tous les fichiers décrivant des situations à 20 avions, sauf un fichier qui pose problème. Nous cherchons à booster cette performance avec les améliorations présentées dans la partie suivante.

# Deuxième version : amélioration avec problème dual et arc-consistence

## Amélioration de l’heuristique par la dualité

Soit un problème de PLNE :

On distingue la fonction objectif et les contraintes.

Nous pouvons transformer ce problème en son dual, un problème de maximisation qui a la même solution.

Notre problème s’écrit :

La contrainte représente le fait que les manoeuvres doivent être compatibles entre elles

Le principe est de considérer la contrainte comme un coût : on veut minimiser l’écart à la contrainte.

## Amélioration avec l’arc-consistance

Bibliographie

<https://fr.wikipedia.org/wiki/S%C3%A9paration_et_%C3%A9valuation>

<https://ocaml.org/index.fr.html>