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

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) :

$$Minimiser\ f(x)$$
$$Sous\ contraintes\ x \in C$$

Ici :

- x est notre variable de décision, la combinaison des n manoeuvres : $x \in [|0, m|]^n$, avec n le nombre d'avions et m le nombre de manoeuvres possibles.
- $f(x)$ est la fonction objectif, la somme des coûts des n manoeuvres choisies.
- C est l'ensemble des contraintes : il décrit quels x 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*

Figure 1.

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 ... 160^{30} lignes !

En effet, avec cette méthode de résolution, la complexité est exponentielle, en $O(m^n)$. 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.

Nous avons utilisé une plateforme Git en ligne, afin de déposer notre code et de gérer les versions : GitHub.

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

OCamll 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 manœuvres à choisir selon un heuristique : le coût minimal potentiel total de la solution, dans le cas du choix de la manœuvre en question. Il nous permet d'éviter de traiter un grand nombre de solutions.

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

- Si on choisit la manœuvre 1, le coût total sera au minimum 10
- Si on choisit la manœuvre 2, le coût total sera au minimum 20

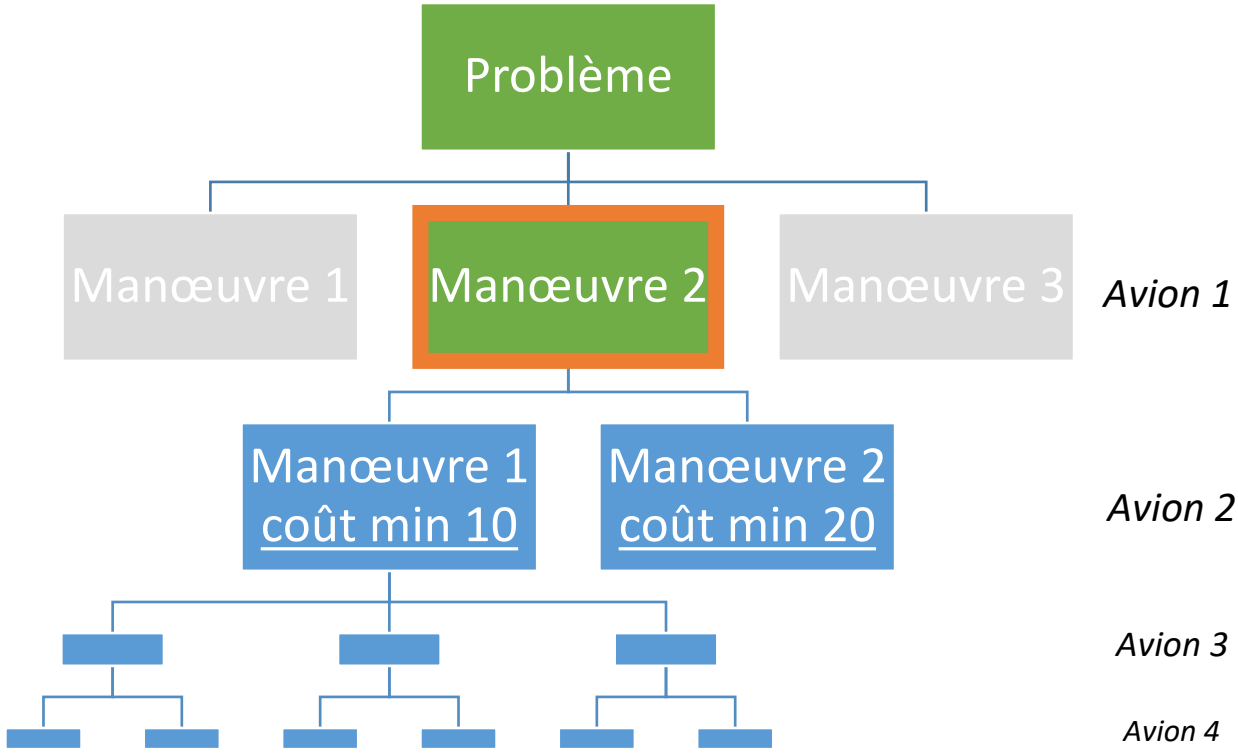

Figure 2.

Dans ce cas, nous commençons par séparer et considérer la manœuvre 1. Si, après traitement, on trouve une solution qui coûte 15, on n'aura même pas à considérer la manœuvre 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 manœuvres compatibles les moins chères pour les avions restants (ici : les avions 3 et 4). Trouver une heuristique demande donc un temps constant (au maximum n opérations si les manoeuvres sont déjà triées).

Notons que, dans le calcul de l'heuristique, la compatibilité entre les manœuvres à choisir sur les avions restants (3 et 4) n'est pas prise en compte : on prend en compte uniquement la compatibilité avec la manœuvre de l'avion 2. D'où le fait que l'heuristique n'est qu'une minoration : la solution « prendre toutes les manœuvres 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 manœuvres soient triées en coût croissant. C'est logique : nous voulons considérer en premier les manœuvres 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 manœuvres 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 en une seconde tous les fichiers décrivant des situations à 10 avions.

Notre programme nous donne également accès au nombre de nœuds qu'explore l'algorithme. Un grand nombre de nœuds explorés va de pair avec une inefficacité de l'heuristique : cela signifie que, plusieurs fois, l'heuristique a guidé l'algorithme vers l'exploration d'une branche de l'arbre qui s'est révélée trop coûteuse. L'écart entre l'heuristique et la réalité trompe l'algorithme, qui fait alors plus de recherches inutiles.

Nous cherchons à améliorer nos performances avec les améliorations présentées dans la partie suivante.

Deuxième version : changement de l'heuristique et de la fonction filtre

Amélioration de l'heuristique : choisir les manœuvres par 2

Dans l'exemple précédent, à chaque itération, nous choisissons pour l'avion suivant la manœuvre la moins chère. Cependant, nous pourrions éviter ce choix dans le cas où il est incompatible avec les autres manœuvres peu chères des autres avions.

En effet, si le choix la manœuvre la moins chère implique forcément le choix d'une manœuvre chère pour tous les autres avions, alors c'est un mauvais choix.

Pour tenter d'améliorer les performances, nous implémentons un nouvel heuristique qui, au lieu de sommer les coûts des manœuvres les moins chères pour chaque avion, somme les coûts des couples de manœuvres les moins chers pour chaque paire d'avions.

Ce changement implique plus de calculs pour trouver l'heuristique à chaque itération. En effet : plaçons-nous dans le cas où il reste 14 avions à instancier. L'heuristique naïf additionne les coûts des manœuvres les moins chères : il nécessite 14 recherches dans les tableaux des manœuvres compatibles restantes à chaque avion.

Le nouvel heuristique va dans un premier temps calculer, pour chaque couple d'avions, le couple de manœuvres le moins cher parmi ceux compatibles avec les choix précédents. En supposant qu'il reste 50 manœuvres compatibles à chaque avion sur 160, cela fait $50^2 = 2500$ coûts de couples de manœuvres à calculer. Il y a $\binom{2}{14}$, soit 91 paires d'avions : on arrive déjà à 227 000 opérations.

Mais ce n'est pas terminé : l'heuristique va maintenant comparer les coûts minimaux des 91 paires d'avions afin de sélectionner la paire d'avions la moins chère. On réitère ensuite, sans la paire d'avions trouvée, avec les 12 avions restants, afin de trouver la paire suivante la moins chère … jusqu'à avoir traité tous les avions.

Toutes ces opérations sont le prix à payer pour un heuristique plus rapide. Et les résultats paient.

Amélioration de la fonction filtre

Notre tuteur, Richard Alligier, nous propose d'implémenter un nouveau filtre utilisant l'algorithme AC3, un algorithme de recherche de connexité. Le filtre est notre fonction qui, à chaque itération, élague les listes des manœuvres compatibles restantes pour chaque avion. Dans notre première version, notre principe était le suivant : lorsque l'on sélectionne une manœuvre m_i pour un avion i,

on enlève pour tout avion j≠i les manœuvres incompatibles avec m_i de la liste des manœuvres pour l'avion j.

En utilisant la connexité et l'algorithme AC3, nous n'allons non pas chercher des manœuvres incompatibles, mais nous assurer que la manœuvre m_i choisie pour l'avion i est compatible avec les manœuvres des domaines des avions j≠i. De plus, lorsque l'on modifie un domaine, on teste également qu'une connexité existe avec celui-ci vers le domaine de chaque avion.

Implémentation d'un filtre initial

Nous implémentons un filtre initial, qui enlève les manœuvres qui n'ont aucune compatibilité avant même le début de l'exploration de l'arbre. Nous espérons réduire le temps d'exécution en réduisant d'emblée la taille des listes à parcourir.

Benchmarking

Nous lançons un script de benchmarking : celui-ci permet de lancer notre algorithme, avec ou sans les nouvelles fonctionnalités, sur quelques fichiers à la suite. Ainsi, on obtient des données statistiques fiables, grâce au grand nombre de situations couvertes.

Une partie des résultats est présentée en annexe.

Analyse des résultats

-Les deux dernières fonctionnalités portant sur le filtre n'améliorent pas les performances. On obtient les mêmes résultats avec et sans ac3 et/ou le filtre initial. En fait, ces fonctionnalités ne réduisent pas le nombre de nœuds explorés, et on remarque que leur incidence sur les temps de calcul est négligeable. L'algorithme AC3 est peut-être mal codé et/ou mal compris, ou simplement inutile : on peut imaginer que chercher à garder des compatibilités ou à éliminer des conflits revient à faire la même chose dans notre cas.

-Par contre, les résultats sont au rendez-vous pour l'heuristique amélioré.

Amélioré	Naïf	Amélioré/naïf
Moyenne nb_noeud/nb_avion	**Moyenne nb_noeud/nb_avion**	**Moyenne nb_noeud/nb_avion**
1,94010582	714,9745503	368,523481
Écart type	**Écart type**	**Écart type**
3,053764054	2382,635702	780,2291402
Médiane	**Médiane**	**Médiane**
1,2	4,8	4

Figure 3.

On trouve ci-contre les résultats en utilisant l'heuristique naïf et l'heuristique amélioré (*mij*) sur les mêmes instances. On remarque que l'heuristique amélioré permet d'explorer 368 fois moins de nœuds que l'heuristique naïf. De plus, l'écart type est amélioré d'un facteur 780, ce qui confirme le fait que cette heuristique est plus efficace et surtout plus constant.

Le temps de calcul pour l'heuristique amélioré est largement compensé par l'économie qu'il apporte à l'algorithme de recherche dans l'arbre. Le nouvel algorithme est plus rapide et parvient donc à traiter plus de situations complexes (c'est pour cela qu'il y a plus de situations répertoriées dans les benchmarks avec l'heuristique amélioré).

Conclusions

En optimisation, l'utilisation de certaines possibilités algorithmiques se vérifient par des tests. Même si certaines données ne seront pas traitées de manière optimale par un nouvel algorithme, si ce dernier est meilleur dans 90% des cas, on le garde. D'où les études statistiques pour déterminer si on garde, ou non, un bout de code.

Nous avons donc, dans une démarche de recherche, essayé d'améliorer notre évaluation, ainsi que notre filtre, les deux principaux éléments de notre algorithme, sans savoir au préalable l'influence qu'aurait cette modification. Nous attendions beaucoup du filtre utilisant AC3, mais après avoir lancer nos tests, on s'est aperçu que c'était en fait la borne qui nous permettant d'optimiser énormément notre programme, là où le filtre AC3 ralentissait juste notre algorithme. Cette démarche de recherche et de test a été très formatrice car elle nous a rappeler que les solutions les plus complexe ne sont pas forcément les meilleures.

Afin d'améliorer encore notre projet, on pourrait :

-Affiner notre code pour en améliorer la rapidité et la clarté.

-Tester de nouvelles bornes afin d'en trouver une plus efficace, car on a vu que cela pouvait avoir un grand impact sur la complexité.

-Coder une fonction intelligente qui pourrait prévoir le temps de calcul et choisir entre plusieurs heuristiques en fonction du nombre d'avions. Pour cela on testerait différentes bornes sur un nombre d'avion donné afin de voir si une heuristique se dégage des autres.

Grâce à ce projet, nous avons en tout cas appris concrètement les enjeux de l'optimisation et de l'intelligence artificielle. De plus, cela nous a permis de s'organiser pour travailler en équipe, avec les contraintes que sont les cours et les vacances de noël où nous ne pouvions nous voir. De plus, un étudiant étant en échange, cela nous a permis de prendre en compte les différences de chacun, d'améliorer notre communication afin d'être compréhensible pour tous.

Annexe : données benchmarking

if ac3 mij	nombre d'instance	70	
nb_avion	temps	nœuds	nb_avion/nœuds
5	0.011679	6	1.2
5	0.011640	6	1.2
5	0.019209	6	1.2
5	0.011329	6	1.2
5	0.015475	6	1.2
5	0.011125	6	1.2
5	0.010109	6	1.2
5	0.012477	6	1.2
5	0.011439	6	1.2
5	0.011113	6	1.2
10	0.114699	11	1.1
10	0.098286	11	1.1
10	0.096208	11	1.1
10	0.100071	11	1.1
10	0.092821	11	1.1
10	0.095738	11	1.1
10	0.101471	11	1.1
10	0.094843	11	1.1
10	0.095865	11	1.1
10	0.104250	11	1.1
15	0.327854	16	1.066666667
15	0.315957	16	1.066666667
15	0.352799	17	1.133333333
15	0.337295	16	1.066666667
15	0.371812	18	1.2
15	0.316248	16	1.066666667
15	1.761689	50	3.333333333
15	0.327926	16	1.066666667
15	0.331514	16	1.066666667
15	0.399964	17	1.133333333
15	0.378555	21	1.4
15	0.314080	16	1.066666667
15	0.540619	26	1.733333333
15	0.907943	33	2.2
15	0.358131	18	1.2
15	0.480144	22	1.466666667
15	1.015695	37	2.466666667
15	0.377239	18	1.2
15	0.369204	17	1.133333333
15	0.395006	18	1.2
15	0.794402	53	3.533333333
15	1.227839	40	2.666666667
15	0.359015	18	1.2
15	0.535311	21	1.4
15	0.437819	22	1.466666667
15	0.313517	18	1.2
15	4.098418	105	7
15	0.291280	16	1.066666667
15	0.275400	16	1.066666667
15	0.650612	24	1.6
20	1.008460	25	1.25
20	0.889226	23	1.15
20	423.055723	7903	395.15
20	0.955030	23	1.15
20	1.086202	31	1.55
20	1.695844	47	2.35
20	0.812461	21	1.05
20	0.946325	24	1.2
20	1.230261	26	1.3
20	30.051619	456	22.8
25	425.848971	4398	175.92
25	1.865807	33	1.32
25	24.191343	274	10.96
25	2.221390	33	1.32
25	42.978279	425	17
25	17585.882913	167261	6690.44
25	19.054387	154	6.16
25	2.276680	34	1.36
25	94.603141	1065	42.6
25	10.313150	145	5.8
			7451.696667

Moyenne nb_noeud/nb_avion
106.5042
Ecart type
799.9823
Mediane
1.2

if ac3 naive	nombre d'instance	63	
nb_avion	temps	nœuds	nb_avion/nœuds
5	0.000038	6	1.2
5	0.000051	6	1.2
5	0.000981	6	1.2
5	0.000902	6	1.2
5	0.000051	6	1.2
5	0.000043	6	1.2
5	0.000052	6	1.2
5	0.000038	6	1.2
5	0.000037	6	1.2
5	0.000047	6	1.2
10	0.000837	11	1.1
10	0.000501	11	1.1
10	0.000646	11	1.1
10	0.001287	11	1.1
10	0.000827	11	1.1
10	0.005268	31	3.1
10	0.007162	48	4.8
10	0.000652	11	1.1
10	0.001081	11	1.1
10	0.001213	11	1.1
15	1.306555	5499	366.6
15	0.031456	207	13.8
15	0.013656	70	4.666666667
15	0.005753	28	1.866666667
15	0.014813	80	5.333333333
15	0.020684	82	5.466666667
15	0.475165	1963	130.8666667
15	0.001676	16	1.066666667
15	0.002140	16	1.066666667
15	0.001820	17	1.133333333
15	9.167667	35978	2398.533333
15	0.111968	467	31.13333333
15	0.328001	1487	99.13333333
15	0.011995	55	3.666666667
15	0.015202	80	5.333333333
15	0.158184	592	39.46666667
15	1.621820	6392	426.1333333
15	0.028143	96	6.4
15	0.011029	45	3
15	0.014849	50	3.333333333
15	182.021968	239408	15960.53333
15	1.305163	4233	282.2
15	11.864076	49345	3289.666667
15	0.026728	107	7.133333333
15	0.056348	239	15.93333333
15	0.373479	1361	90.73333333
15	6.775808	22010	1467.333333
15	0.039010	141	9.4
15	0.010260	42	2.8
15	0.111586	260	17.33333333
20	2.454993	5694	284.7
20	0.027353	51	2.55
20	0.035801	69	3.45
20	0.097313	281	14.05
20	40.432592	113682	5684.1
20	0.007079	21	1.05
20	0.500812	938	46.9
20	0.978196	2213	110.65
20	8.386395	27714	1385.7
25	139.203667	85047	3401.88
25	20.736243	28780	1151.2
25	849.891253	198369	7934.76
25	5.041066	7566	302.64

Moyenne nb_noeud/nb_avion
714.97455
Ecart type
2382.6357
Mediane
4.8

Ci-dessus : résultats avec filtre initial, avec AC3, comparaison des 2 heuristiques.

Ci-dessous : résultats avec filtre initial, sans AC3, comparaison des 2 heuristiques.

Heuristique amélioré à gauche, naïf à droite.

if bandb mij	nombre d'instance	70	
nb_avion	temps	nœuds	nb_avion/nœuds
5	0.011813	6	1.2
5	0.013783	6	1.2
5	0.015166	6	1.2
5	0.012861	6	1.2
5	0.010826	6	1.2
5	0.011150	6	1.2
5	0.011318	6	1.2
5	0.012260	6	1.2
5	0.011203	6	1.2
5	0.011632	6	1.2
10	0.095375	11	1.1
10	0.097269	11	1.1
10	0.093667	11	1.1
10	0.095590	11	1.1
10	0.092408	11	1.1
10	0.098103	11	1.1
10	0.092865	11	1.1
10	0.094277	11	1.1
10	0.096518	11	1.1
10	0.094270	11	1.1
15	0.318894	16	1.066666667
15	0.314355	16	1.066666667
15	0.363889	17	1.133333333
15	0.335488	16	1.066666667
15	0.368429	18	1.2
15	0.306331	16	1.066666667
15	1.726739	50	3.333333333
15	0.311383	16	1.066666667
15	0.312143	16	1.066666667
15	0.384957	17	1.133333333
15	0.353814	21	1.4
15	0.293288	16	1.066666667
15	0.503314	26	1.733333333
15	0.871840	33	2.2
15	0.349811	18	1.2
15	0.444512	22	1.466666667
15	0.978041	37	2.466666667
15	0.367583	18	1.2
15	0.367371	17	1.133333333
15	0.388730	18	1.2
15	0.806919	53	3.533333333
15	1.235682	40	2.666666667
15	0.352156	18	1.2
15	0.514752	21	1.4
15	0.416551	22	1.466666667
15	0.284431	18	1.2
15	4.073565	105	7
15	0.281098	16	1.066666667
15	0.271996	16	1.066666667
15	0.643648	24	1.6
20	1.001241	25	1.25
20	0.893560	23	1.15
20	412.307103	7903	395.15
20	0.849048	23	1.15
20	1.012712	31	1.55
20	1.664664	47	2.35
20	0.770550	21	1.05
20	0.838356	24	1.2
20	1.087263	26	1.3
20	27.825514	456	22.8
25	409.392333	4398	175.92
25	1.724341	33	1.32
25	22.639666	274	10.96
25	2.118656	33	1.32
25	40.824041	425	17
25	16492.214138	167261	6690.44
25	18.590674	154	6.16
25	13.317198	34	1.36
25	93.683570	1065	42.6
25	60.002316	145	5.8

Moyenne nb_noeud/nb_avion
106.504238
Ecart type
799.982325
Mediane
1.2

if bandb naiv	nombre d'instance	63	
nb_avion	temps	nœuds	nb_avion/nœuds
5	0.000042	6	1.2
5	0.000040	6	1.2
5	0.000078	6	1.2
5	0.000893	6	1.2
5	0.000048	6	1.2
5	0.000038	6	1.2
5	0.000038	6	1.2
5	0.000038	6	1.2
5	0.000037	6	1.2
5	0.000041	6	1.2
10	0.000381	11	1.1
10	0.000171	11	1.1
10	0.000311	11	1.1
10	0.000437	11	1.1
10	0.000297	11	1.1
10	0.002588	31	3.1
10	0.003826	48	4.8
10	0.000284	11	1.1
10	0.000518	11	1.1
10	0.000510	11	1.1
15	0.827400	5499	366.6
15	0.016863	207	13.8
15	0.006757	70	4.666666667
15	0.001971	28	1.866666667
15	0.007200	80	5.333333333
15	0.007850	82	5.466666667
15	0.283272	1963	130.8666667
15	0.000569	16	1.066666667
15	0.000358	16	1.066666667
15	0.000787	17	1.133333333
15	5.652388	35978	2398.533333
15	0.057513	467	31.13333333
15	0.215491	1487	99.13333333
15	0.005517	55	3.666666667
15	0.009537	80	5.333333333
15	0.087085	592	39.46666667
15	0.781857	6392	426.1333333
15	0.009431	96	6.4
15	0.004632	45	3
15	0.004389	50	3.333333333
15	102.790074	239408	15960.53333
15	0.617025	4233	282.2
15	7.387952	49345	3289.666667
15	0.010598	107	7.133333333
15	0.022743	239	15.93333333
15	0.187882	1361	90.73333333
15	3.029281	22010	1467.333333
15	0.013024	141	9.4
15	0.004922	42	2.8
15	0.033952	260	17.33333333
20	1.268705	5694	284.7
20	0.009246	51	2.55
20	0.014904	69	3.45
20	0.059848	281	14.05
20	26.136482	113682	5684.1
20	0.002136	21	1.05
20	0.192130	938	46.9
20	0.434034	2213	110.65
20	4.688891	27714	1385.7
25	62.696645	85047	3401.88
25	8.735176	28780	1151.2
25	447.561103	198369	7934.76
25	2.165224	7566	302.64

Moyenne nb_noeud/nb_avion
714.97455
Ecart type
2382.6357
Mediane
4.8

Bibliographie

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

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