Rapport RP

AYED Hatem, ZHANG Zhile

March 2024

1 Partie 1 : Modélisation, instances et résolution par recherche arborescente

1.1

Notons k le nombre de robots et n le nombre de cases. La représentation des états est la suivante :
$\{(x_1, y_1), (x_2, y_2), ..., (x_k, y_k)\}$, $(x_1, y_1), ..., (x_k, y_k) \in [\![1, n]\!]^2$
Chaque paire (x_i, y_i) représente la position du robot $i \in [\![1, k]\!]$ dans la grille.

En supposant que le robot numéro 1 est le robot bleu, et en notant (x_b, y_b) les coordonnées de la cible sélectionnée, l'état final est de cette forme :
$\{(x_b, y_b), (x_2, y_2), ..., (x_k, y_k)\}, (x_i, y_i) \in [\![1, n]\!]^2 \, \forall i \in [\![2, k]\!]$

1.2

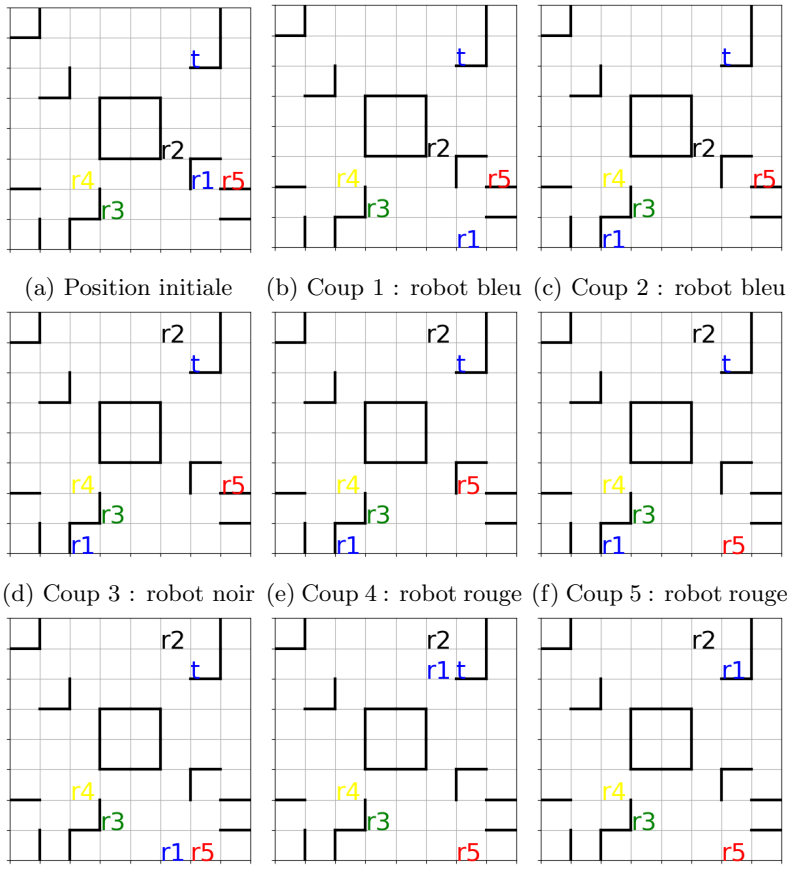
Chaque robot peut avoir accès à au plus n^2 cases.
Etant donné qu'il y a k robots, on en déduit que n^{2k} est une borne supérieure de l'espace d'états.

1.3

A partir d'une case donnée, chaque robot peut avoir accès à au plus 4 cases différentes.
On en déduit ainsi que 4^k est une borne supérieure du nombre de successeurs possibles d'un état.

1.4

Après implémentation d'une procédure par recherche en largeur, voici une illustration graphique de solution optimale pour une instance donnée :



(a) Position initiale (b) Coup 1 : robot bleu (c) Coup 2 : robot bleu

(d) Coup 3 : robot noir (e) Coup 4 : robot rouge (f) Coup 5 : robot rouge

(g) Coup 6 : robot bleu (h) Coup 7 : robot bleu (i) Coup 8 : Victoire

Figure 1: Résolution optimale d'une instance de Ricochet Robots

1.5

Etudions maintenant le temps de résolution de notre recherche par arborescence en fonction des différents paramètres. Nous allons considérer en premier lieu les temps moyens pour $n = 8$ et $k = 5$ dans la figure qui suit :

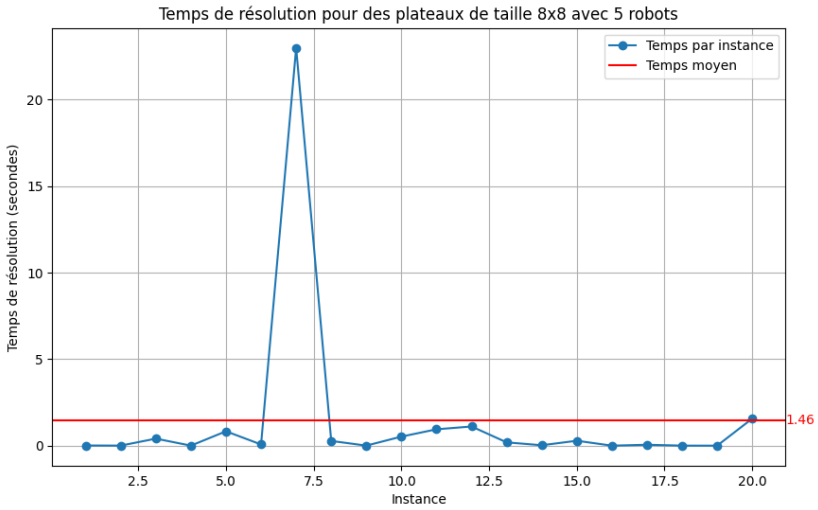


Figure 2: Temps de la méthode sur 20 instances de tailles $n = 8$.

Nous pouvons observer une volatilité assez élevée dans ces temps dûe à la nature stochastique du problème. On observe notamment un pic lors de la 6e instance, où la solution optimale comportait un nombre plus élevé de coups que pour les autres instances. La complexité de notre algorithme étant exponentielle en nombre de coups requis pour résoudre le problème, cela explique la magnitude de ce pic. En effet, pour chaque coup en plus, nous devons aller dans une couche supplémentaire de notre arbre. On relève graphiquement que le temps moyen de résolution de ces 20 instances est de 1.46 seconde. Regardons maintenant comment évolue ce nombre lorsque nous augmentons la taille du plateau.

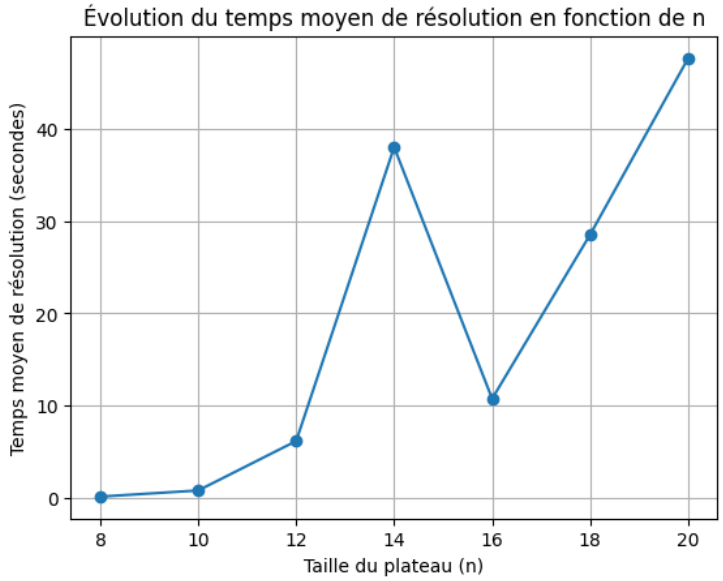


Figure 3: Evolution du temps moyen de réalisation pour n croissant

Essayons d'étudier rapidement la complexité (du moins une approximation) de notre algorithme afin d'interpréter cette courbe. Nous avons utilisé un BFS, dont la complexité temporelle est $O(|V| + |E|)$ avec V et E respectivement le nombre de sommets et d'arêtes de notre graphe d'état. Par 1.3, on en déduit que $O(|E|) = O(4^k|V|)$, d'où finalement une majoration de la complexité temporelle en $O((4n^2)^k)$. On constate comme attendu une nette évolution du temps moyen de résolution à mesure que n augmente, ainsi qu'un outlier pour $n = 14$, encore une fois dû à la nature stochastique du problème. Passé la valeur de $n = 20$, la mémoire RAM de nos ordinateurs a saturé : il suffit d'une instance où l'expoloration complète du graphe est nécessaire pour que le nombre de calculs requis soit démesurément grand. Pour $n = 22$ et $k = 5$, $(4n^2)^k \approx 2.72e16$!!

Enfin, dans la figure qui suit, nous étudions l'impact du nombre k de robots sur la valeur de la solution optimale. La méthodologie afin d'obtenir ce graphe a été la suivante : nous générons un plateau de taille $8x8$ contenant 5 robots. On compte ensuite le nombre de coups minimum requis pour atteindre la cible. Nous enlevons ensuite le 5ème robot, et nous réitérons. Le processus s'arrête lorsqu'il ne reste plus que le robot bleu. Le graphe figure 3 a été tracé en prenant la moyenne des valeurs pour chaque étape (5,4, ... ou 1 robot) après 100 exécutions de la méthode. On constate que le nombre de coups requis diminue bien à mesure que le nombre de robots augmente, à l'exception du cas $k = 5$. Cela est dû à notre méthodologie : il se peut que dans la configuration initiale du plateau, le 5e robot soit par exemple entre le robot bleu et la cible. Ainsi, la solution optimale comportera ici un coup de plus que pour $k = 4$.

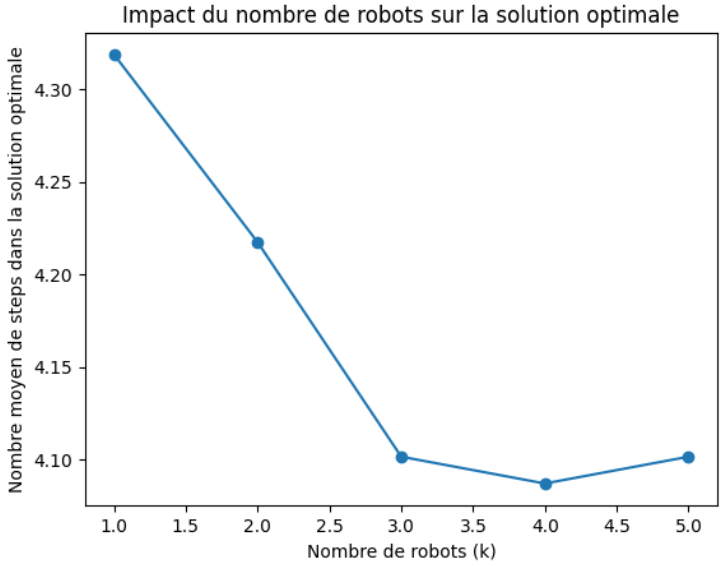


Figure 4: Influence du nombre de robots sur la valeur optimale

2 Partie 2 : Résolution par A*

2.1

L'heuristique h_1 est monotone. Considérons chaque cas envisageable pour montrer cela.

Cas 1 : Nous sommes dans un état i où le robot et la cible se situent sur la même ligne ou la même colonne. $h_1(i) = 1$.
Trois choix possibles s'offrent pour l'état suivant :
- Un état i' où le robot et la cible se situent toujours sur la même ligne ou la même colonne. Ici, $h_1(i') + coût(i, i') = 1 + 1 = 2 \geq h_1(i)$
- Un état j où le robot et la cible se situent sur des lignes et colonnes différentes. $h_1(j) + coût(i, j) = 2 + 1 = 3 \geq h_1(i)$
- Un état k où le robot et la cible sont à la même position et $h_1(k) + coût(i, k) = 0 + 1 = 1 \geq h_1(i)$

Cas 2 : Nous sommes dans un état j où le robot et la cible se situent sur des lignes et colonnes différentes. $h_1(j) = 2$.
Deux successeurs sont possibles cette fois :
- Un état j' où le robot et la cible se situent toujours sur des lignes et colonnes différentes. $h_1(j') + coût(j, j') = 2 + 1 = 3 \geq h_1(j)$
- Un état i où le robot et la cible se situent sur la même ligne ou la même colonne. Ici, $h_1(i) + coût(j, i) = 1 + 1 = 2 \geq h_1(j)$

h_1 est ainsi bien monotone. De plus, une heuristique h est coïncidente si $h(but) = 0$. Ici, pour un état k où le robot et la cible sont à la même position, on a bien $h_1(k) = 0$. Finalement, h_1 étant monotone et coïncidente, on en déduit qu'elle est minorante.

2.2

Nous implémentons A* en considérant l'heuristique $h1$. Comparons l'efficacité de l'algorithme avec le BFS vu précedemment. Pour ce faire, nous avons exécuté pour différentes valeur de n vingt instances du jeu et relevé le temps moyen de résolution par les deux algos. Nous obtenons ainsi le tableau et la courbe qui suivent :

	Temps moyen BFS	Temps moyen A*
$n = 8$	$0.17s$	$0.13s$
$n = 10$	$0.97s$	$0.77s$
$n = 12$	$1.35s$	$0.99s$
$n = 14$	$15.96s$	$11.02s$
$n = 16$	$8.98s$	$6.50s$
$n = 18$	$104.92s$	$58.97s$

Table 1: Tableau des temps moyens pour 20 exécutions de A* et BFS

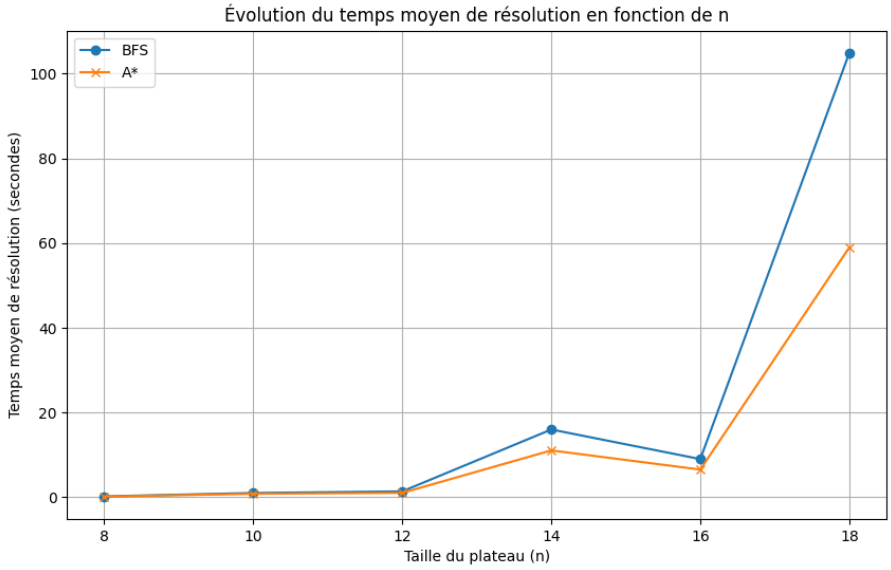


Figure 5: Evolution du temps moyen de réalisation en fonction de n pour A* et BFS

On peut constater que malgré la simplicité de l'heuristique $h1$, A* est plus efficace que notre premier algorithme, allant jusqu'à être presque deux fois plus rapide sur les 20 dernières instances.

2.3

Nous continuons à utiliser l'algorithme A* mais en nous appuyant sur l'heuristique $h2$. De la même manière, nous comparons la tendance de l'évolution du temps de résolution moyen avec la taille du plateau de jeu pour l'algo BFS, l'algo A* avec l'heuristique $h1$, et $h2$. Nous obtenons ainsi le tableau et la courbe qui suivent :

	Temps moyen BFS	Temps moyen A* avec $h1$	Temps moyen A* avec $h2$
$n = 8$	$0.33s$	$0.28s$	$0.14s$
$n = 10$	$3.47s$	$2.15s$	$0.86s$
$n = 12$	$3.13s$	$1.98s$	$1.20s$
$n = 14$	$20.88s$	$15.81s$	$5.54s$
$n = 16$	$41.43s$	$34.16s$	$7.68s$

Table 2: Tableau des temps moyens pour 20 exécutions de A*($h1$, $h2$) et BFS

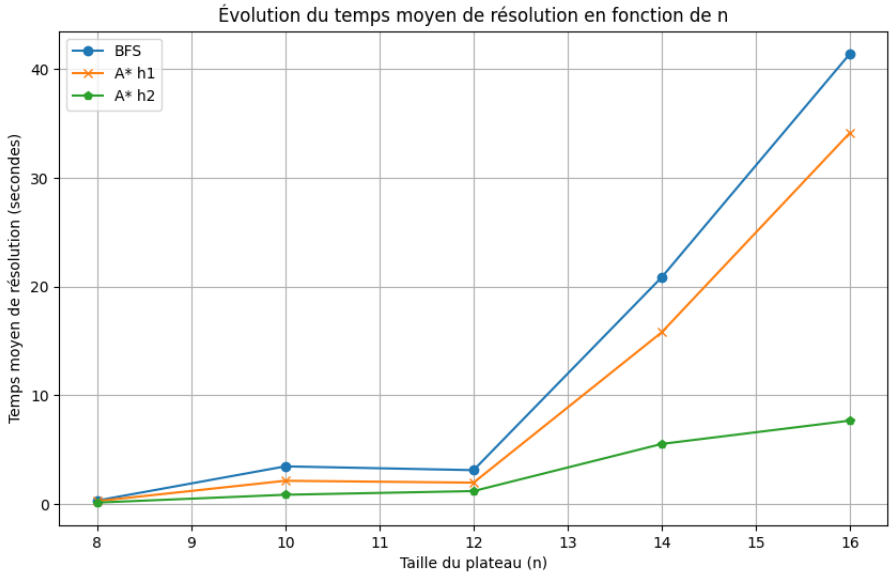


Figure 6: Evolution du temps moyen de réalisation en fonction de n pour A*($h1$, $h2$) et BFS

On peut constater une très nette amélioration du temps de résolution en comparaison de nos deux anciens algorithmes. Cela est dû au fait que l'heuristique

$h2$ est bien plus sophistiquée que $h1$ et nous aiguille avec bien plus de précision vers la solution optimale.