Algorithmique pour la Résolution des Problèmes
Accessibilité dans $\mathbb{Z}^2$

Eldis YMERAJ
Tolgan SUNER

Decembre 2023

Table des matières

1 Introduction

Le problème d'accessibilité dans $\mathbb{Z}^2$, aussi connu sous le nom de "reachability problem", est un problème classique en informatique théorique et en mathématiques discrètes. Il s'agit de déterminer si, étant donné un état initial et un état cible dans le plan, il est possible d'atteindre l'état cible à partir de l'état initial en utilisant un ensemble de vecteurs $V = \{v_1, v_2, ..., v_n\}$ et les opérations d'addition et de soustraction. Ce problème peut se généraliser dans $\mathbb{Z}^n$, plus nn est grand, plus il est difficile à explorer.
Une configuration est un ensemble état initial, but et set de vecteurs. On peut considérer deux aspects du problème : l'accessibilité du but et le plus court chemin. Ce dernier aspect a d'importantes implications en optimisation et en planification de trajectoires, où la minimisation des coûts ou des distances est souvent cruciale.

1.1 Complexité

La complexité grandit très vite en fonction de nn et du nombre de vecteurs, car cela produit un grand nombre de combinaisons à essayer. Il est classé PSPACE-complexe car il utilise une quantité d'espace mémoire polynomiale par rapport à la taille de l'entrée.
Il faut donc, dans notre programme, optimiser l'exploration afin d'éviter les boucles lorsqu'on retombe sur un état déjà exploré et réfléchir à une façon minimale de stocker l'information.

2 Algorithme générique

L'algorithme générique commence à explorer le plan sous la forme d'un arbre. Nous avons choisi de simuler ce parcours à l'aide d'une file (FIFO), cela nous permet de trouver le plus court chemin car chaque état est parcouru dans l'ordre d'enfilement. Ils sont enfilés au même niveau dans l'arbre enraciné à l'état initial. Nous utiliserons aussi une file de priorité dans le cas de certaines heuristiques afin de privilégier les états les plus intéressants.

3 Accessibilité

Dans de nombreux cas, il n'est pas possible de garantir avec certitude si le but est accessible sans une exploration exhaustive de l'espace des états. Pour les espaces de grande dimension ou avec un grand nombre de vecteurs d'actions, cette exploration peut être pratiquement infaisable en raison de la complexité computationnelle élevée.

Si l'on se concentre seulement sur le problème d'accessibilité, nous avons moins d'informations à stocker puisque nous ne nous intéressons pas au chemin, cela

réduit l'espace mémoire utilisé pour chaque état calculé. Soit $[e, p, s, co]$ un nœud tel que $e \in \mathbb{Z}^n$ est un état de l'arbre possible, $p \in \mathbb{N}$ une profondeur de l'état soit le nombre d'états depuis l'état initial vers l'état actuel, $s \in \mathbb{R}$ un score calculé par différentes heuristiques afin de guider l'exploration de l'arbre afin de trouver, dans certaines conditions, une solution à une configuration donnée, et $co \in \mathbb{N}$ le coût que l'exploration a engendré pour trouver l'état e.

3.1 Estimation

À titre d'information, pour avoir une idée de si le but est accessible, nous calculons un carré centré autour du but de côté $r = \frac{2\lceil \sum_{v \in V} |v| \rceil}{\sqrt{a^2+1}}$,
dans ce carré, on regarde les points accessibles par les actions depuis le but, noté α, que l'on compare avec le nombre de coordonnées entières dans cet espace, noté β. En calculant le rapport entre les deux $\frac{\alpha}{\beta} \times 100$, on obtient le pourcentage de points accessibles depuis le but grâce aux actions. Plus l'espace est dense, plus la probabilité que le point soit accessible est grande, et inversement.

Dans le carré 7 par 7 issu de la configuration ci-dessous, nous dénombrons $\alpha = 3 \times 7 = 21$ et $\beta = 7 \times 7 = 49$, alors $\frac{\alpha}{\beta} 100 = \frac{21}{49} 100 \approx 42.86\%$

```
etat_inital= (0,0)
but= (10,10)
Action= {(2,0),(0,1)}
```

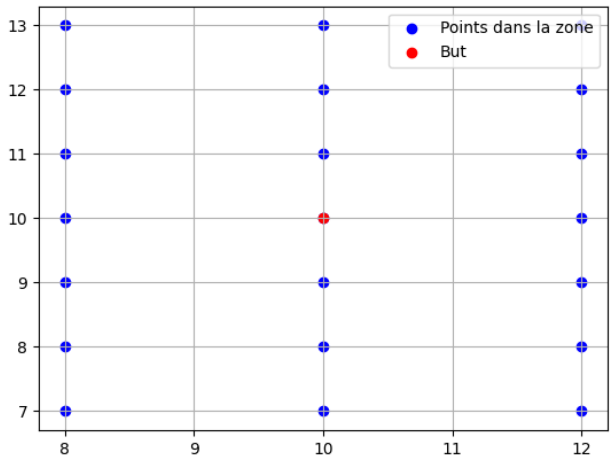

FIGURE 1 – Densité faible.

Implication de l'approche par estimation :

— Estimation Heuristique : Cette méthode fournit une estimation heuristique de l'accessibilité. Elle ne garantit pas l'accessibilité absolue, mais offre une indication préliminaire de la faisabilité d'atteindre le but.
— Adaptabilité : L'approche peut être adaptée à différents espaces et ensembles d'actions, offrant une flexibilité dans son application.
— Limitations : Bien que cette méthode puisse donner une bonne indication de l'accessibilité, elle a ses limites, surtout dans des cas où les chemins vers le but nécessitent des séquences d'actions complexes.

3.2 Arrêt

Un des principaux problèmes est de savoir quand l'état but n'est pas accessible. Dans une configuration donnée, il n'est pas toujours évident de borner la profondeur d'un état. Dans un cas simple, si l'on prend la configuration suivante :

```
etat_inital= (0,0)
but= (10,5)
Action= {(1,0),(0,1)}
```

On peut dénombrer le nombre d'états pour atteindre le but assez facilement. Ici, on a besoin de $10 \times (1,0) + 5 \times (0,1)$, soit 15 étapes que l'on nommera profondeur, alors que pour la configuration :

```
etat_inital: (0,0)
but: (8,6)
Action: {(1,-2),(-1,1)}
```

Il est difficile de savoir combien de fois on aura besoin de chaque vecteur. La profondeur est difficile à déterminer, mais si l'état est bien accessible, le parcours en largeur examinera toutes les possibilités par niveau et s'arrêtera donc avec le résultat et on sait grâce à ce parcours qu'il est minimum.
Et pour :

```
etat_inital= (0,0)
but= (11,5)
Action= {(2,1),(0,1)}
```

On peut facilement remarquer que le but n'est pas atteignable dans cette configuration (11 est impair et les actions et l'état initial sont pairs), alors dans ce cas-là, le programme cherchera indéfiniment.
Dans le cas des heuristiques, cela va dépendre de l'heuristique, mais il est probable que si on restreint l'espace des possibilités pour accélérer les recherches, on coupe des états en dehors de cet espace qui sont nécessaires pour atteindre le but.

4 Plus court chemin

Il est souvent tout aussi important de savoir comment arriver à un état but en un minimum de mouvements. Cela demande davantage d'espace mémoire par

état car nous souhaitons savoir pour tout état de l'arbre comment l'atteindre.

Soit $[e, p, s, ch, co]$ un noeud tel que $e \in \mathbb{Z}^n$ soit un état de l'arbre possible, $p \in \mathbb{N}$ la profondeur associée à l'état e, $s \in \mathbb{R}$ un indice dépendant de l'heuristique utilisée, Soit $ch = (e_0, e_1, \ldots, e_k)$ une séquence d'états dans Z^n où e_0 est l'état initial. Pour tout $i \in \{1, \ldots, k\}$, existe a_i tel que $e_i = \text{succ}(e_{i-1}, a_i)$, $s \in \mathbb{R}$un score calculé par différentes heuristiques afin de guider l'exploration de l'arbre pour trouver, dans certaines conditions, une solution à une configuration donnée, et $co \in \mathbb{N}$ le coût que l'exploration a engendré pour trouver l'état e.

5 Heuristique meilleur distance

La première stratégie est de partir du principe que l'on entasse dans une file de priorité l'état le plus intéressant selon notre critère, ici la distance par rapport au but. À chaque étape de la recherche, on prendra l'état qui est le plus proche du but, ce qui nous permet de nous approcher plus rapidement d'une solution.

5.1 Distance

Une fonction distance(e,but)distance(e,but) permet de nous renvoyer la distance entre un état $e = (e_1, ..., e_n)$ et l'état but $b = (b_1, ..., b_n)$ grâce à la formule de distance dans le plan $\delta = \sqrt{\sum_{i=0}^{n-1} (b_i - e_i)^2}$.

6 Heuristique score

On utilise l'heuristique du score dans les deux problèmes d'accessibilité et de recherche du chemin le plus court. Dans les deux cas, cette approche ne retourne pas forcément la profondeur la plus courte.
L'heuristique permet d'orienter la recherche dans l'arbre des possibilités et de déterminer un score à chaque étape. On associe à l'état calculé le rapport de la distance avec le but et la profondeur de l'état actuel, $\frac{\delta}{p}$. Plus le score est petit, plus on se rapproche du but, car à chaque étape on veut que l'état e soit au plus près du but, donc δ le plus petit possible, et de même pour la profondeur p.
Cela permet de trouver une solution qui pourrait être longue à calculer de manière classique.

7 Espace de recherche borné

Pour savoir si le but est accessible nous avons choisi de borne l'espace de recherche grace a une bande autour de la droite passant par l'etat initial et le but.

On regardera le cout et le temps d'execution, le temps d'execution dependra de la machine mais cela nous donne un element de comparaison.
Soit
x, y : la coordonnée de l'état à tester,
x_i, y_i : la coordonnée de l'état initial,
x_b, y_b : la coordonnée du but.

On a choisis le meme δ que pour l'estimation soit deux fois la somme des norme des vecteur des action possible.

$$\delta = \frac{2\lceil \sum_{v \in V} |v| \rceil}{\sqrt{a^2 + 1}}$$

Nous avons définis alors $f(x) = ax + b$, tel que $a = (y_b - y_i)/(x_b - x_i)$ et $b = y_i - a \times x_i$.
$g(x) = ax + b + \delta$ et $h(x) = ax + b - \delta$

On a aussi une borne supérieur $b_{sup}(x) = \begin{cases} y_b + nmax & \text{si } y_b \geq 0, \\ y_b - nmax & \text{sinon.} \end{cases}$

Et une borne inférieur $b_{inf}(x) = \begin{cases} -nmax & \text{si } y_b \geq 0, \\ nmax & \text{sinon.} \end{cases}$

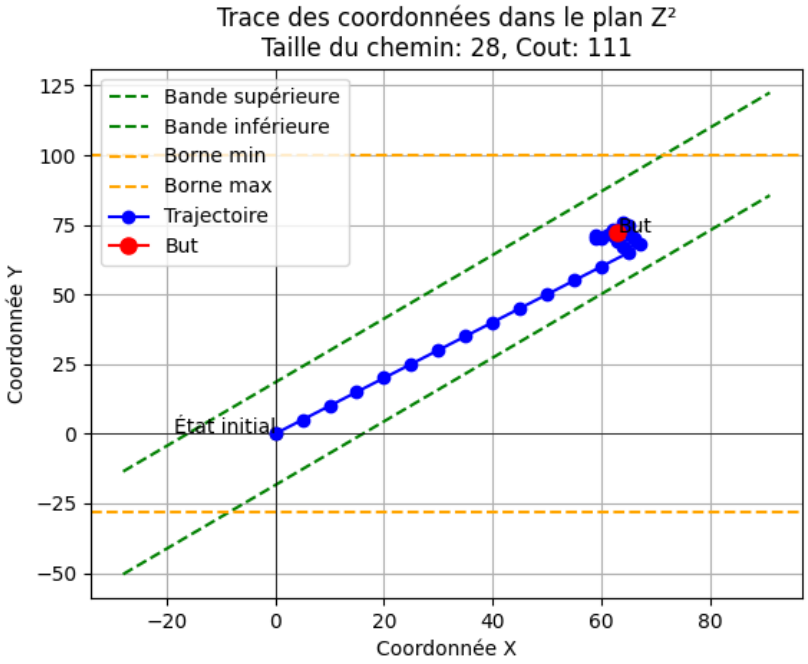

FIGURE 2 – Espace borné.

8 Comparaison

Nous allons comparer différentes configurations pour tenter d'en tirer des conclusions sur leurs propriétés. La recherche d'accessibilité ou de chemin nécessite le même nombre d'étapes car l'exploration est la même, seule la structure du nœud diffère. Ainsi, les résultats sont communs aux deux problématiques. Nous regarderons le coût, le temps d'exécution (cette information est dépendante de la machine qui exécute le programme, mais cela nous donnera une indication sur les différences entre les temps de calcul) et la profondeur.

8.1 Résultat sans heuristique

Cas sans heuristique avec des actions simples et un but "proche" de l'état initial.

```
etat_initial= [0, 0]
but= [5, 5]
action= {(1, 0), (0, 1)}
```

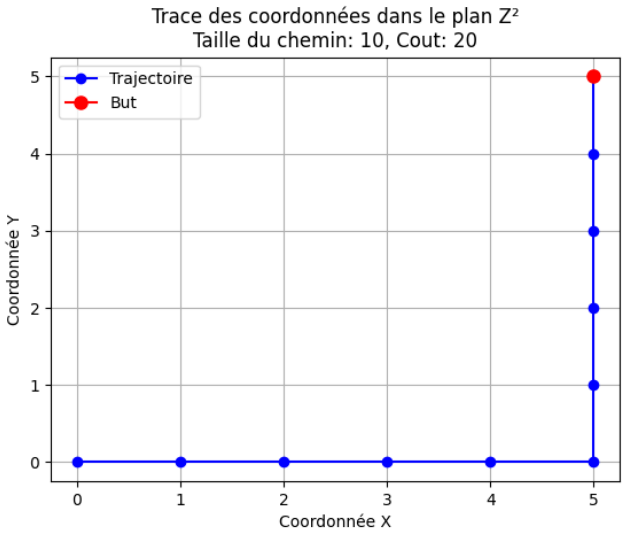

FIGURE 3 – But proche.

Nous pouvons remarquer que l'on trouve bien le chemin le plus court grâce au parcours en largeur. On a un coût de 20 et le temps de recherche d'accessibilité est d'environ 0.07 seconde, alors que le temps de recherche de chemin est d'environ 2.33 secondes.

Cas sans heuristique avec des actions simples et un but "éloigné" de l'état initial.

```
etat_initial= [0, 0]
but= [7, 7]
action= {(1, 0), (0, 1)}
```

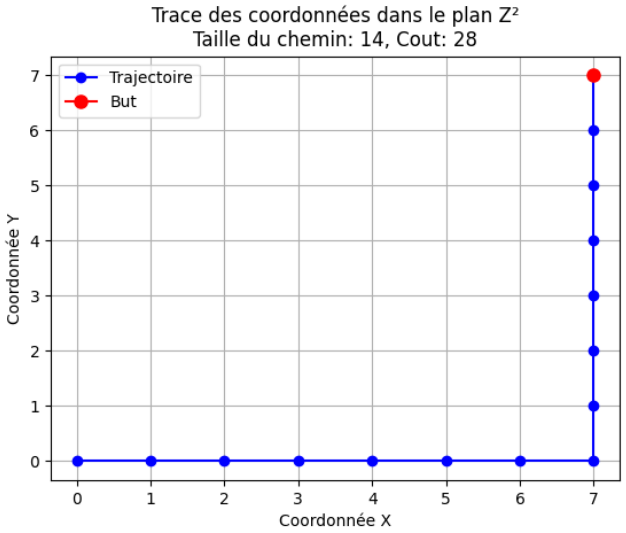

FIGURE 4 – But eloigne.

De même, nous remarquons que l'on trouve bien le chemin le plus court grâce au parcours en largeur. On a un coût de 28 et le temps de recherche d'accessibilité est d'environ 1.85 seconde, alors que le temps de recherche de chemin est d'environ 366.81 secondes.

De plus, nous constatons que pour ces deux configurations, l'écart de temps d'exécution est considérablement plus long alors que les buts ne sont pas encore très éloignés. Nous avons testé avec des valeurs plus éloignées sans que cela ne finisse sur nos machines, ce qui est conforme à sa classification PSPACE.

8.2 Résultat avec heuristique

Cas avec heuristique de distance et un but "éloigné" de l'état initial.

```
etat_initial= [0, 0]
but= [63, 72]
action= {(2, 3), (3, 1), (5,5)}
```

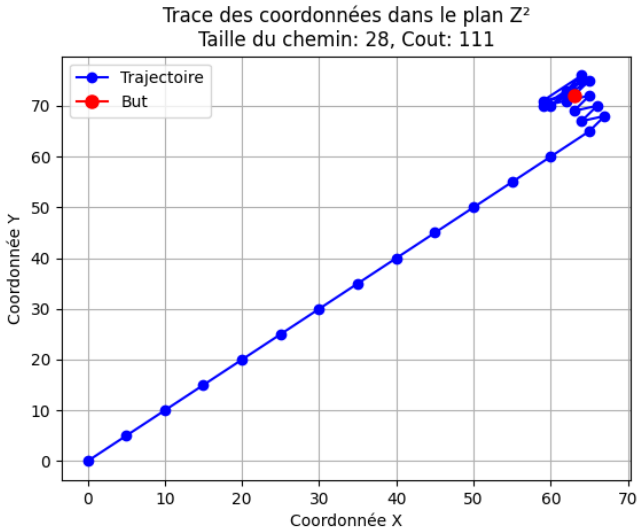

FIGURE 5 – Heuristique distance.

On peut remarquer que l'on trouve bien un chemin, pas le plus court, grâce à la file de priorité. On a un coût de 32 et le temps de recherche d'accessibilité est d'environ 0.0 seconde, alors que le temps de recherche de chemin est d'environ 0.006 seconde.

Cas avec heuristique de score et un but "éloigné" de l'état initial.

```
etat_initial = [0, 0]
but = [63, 72]
action = ((2, 3), (3, 1), (5,5)}
```

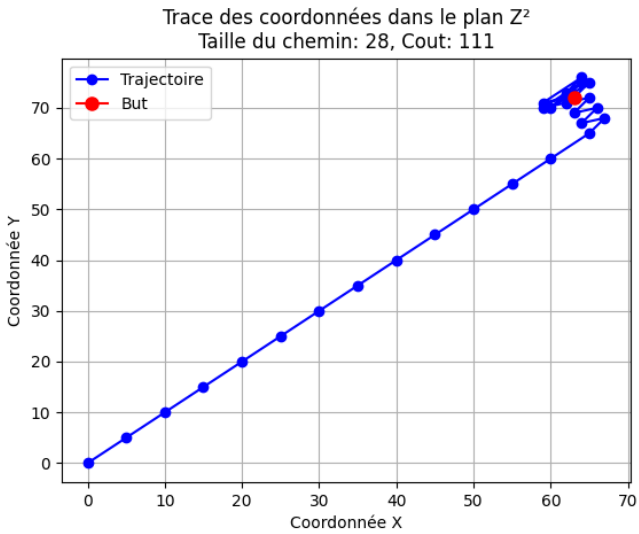

FIGURE 6 – Heuristique score.

On peut remarquer que l'on trouve bien un chemin, pas le plus court, grâce à la file de priorité. On a un coût de 32 et le temps de recherche d'accessibilité est d'environ 0.0009 seconde, alors que le temps de recherche de chemin est d'environ 0.009 seconde.

Remarque : L'heuristique de distance et de score semble donner les mêmes résultats. Il ne nous a pas été possible de trouver des configurations où ils étaient différents, mais nous n'avons pas non plus réussi à prouver qu'ils étaient équivalents.
Nous remarquons que les heuristiques sont très efficaces pour réduire le temps d'exécution afin de trouver un chemin, mais il est difficile de conclure sur la minimalité du chemin trouvé.

9 Conclusion

En pratique, le problème d'accessibilité trouve des applications variées. En informatique, il est crucial pour la composition de services, où il faut orchestrer divers services pour obtenir un résultat spécifique. En robotique, ce problème est essentiel pour déterminer la faisabilité des mouvements des bras robotiques, en tenant compte de leurs contraintes. Ces exemples soulignent l'importance du problème dans l'analyse et la conception de systèmes complexes.

Table des figures