Projet Algorithmique pour la Résolution des Problèmes: Accecibilitée en $\mathbb{Z}^2$

Clement COLMERAUER, Maxence DE MONGELAS;

Table des matières

1 Introduction

L'objectif de notre projet est de réaliser et de programmer un algorithme générique, correspondant à un problème de taille exponentielle, puis de l'optimiser. Afin d'explorer un peu plus le problème, nous testerons différentes heuristiques pour répondre à celui-ci.

2 Formalisation du problème

Le but est, étant donné un jeu d'action, savoir si on peut aller du point de départ au point d'arrivé et si possible, trouver le plus court chemin.

— L'ensemble des états S, avec un état étant un point, soit un vecteur a deux dimensions $v = (x, y)$
— L'état de départ $s_0 = (x_0, y_0)$
— L'état final $s_f = (x_f, y_f)$
— Un ensemble d'action A prenant la forme de vecteur a deux dimensions $a = (x', y')$
— Une fonction successeur $\mathtt{suc}(s, a)$ tel que $s' = s + a$
— Un cout tel que $c(s, s') = 1$ car on s'interèsse au plus court chemin

3 Limite dans l'espace de recherche

La première étape est de déterminer la coupe, on coupera évidemment les états deja traités et la frontière. Mais le véritable problème est que $\mathbb{Z}^2$ est infini et il faut trouver comment définir un plan fini. Cette coupe étant a la base de nos heuristique, elles seront détaillées dans les section concernées.

4 Première heuristique

4.1 Fonctionnement

La première heuristique consiste a tracer un cercle de rayon $\mathtt{distance}(s_0, s_f) + \sum_{i=1}^{|A|} |A_i|$ centré sur s_f. Et faire l'algorithme générique dans ce cercle. On privilégie un parcours en largeur afin de garantir qu'il n'y a pas de meilleure solution que celle trouvée.

4.2 Algorithme

Algorithm 1: Algorithme générique d'accessibilité dans $\mathbb{Z}^2$

Entrées: s_0, s_f les points de départ et arrivé, A un ensemble de Vecteur2
Sorties: Un chemin de s_0 à s_f,faux sinon
int limite = arondiSup($|(s_0, s_f)| + \sum_{i=1}^{|A|} |A_i|$);
Liste etats = (s_0);
Liste dejaExplore = $\varnothing$;
Liste aExplorer = (s_0);
tant que */aExplorer/* $\neq$ *0* **faire**
    e = premier(aExplorer);
    si *e* = s_f **alors**
        Liste chemin = $\varnothing$;
        tant que *p[e]* $\neq$ $\varnothing$ **faire**
            chemin.insertTete(e);
            e = p[e];
        fin
        chemin.insertTete(s_0);
        retourner *chemin*
    fin
    pour chaque $a \in A$ **faire**
        s = suc(e,a);
        p[s] = e;
        etat.ajoutFin(s);
        si *s* $\notin$ *{dejaExplorer* $\cup$ *aExplorer}* **alors**
            si *distance entre s et* $s_f \leq$ *limite* **alors**
                aExplorer.ajoutFin(s)
            fin
        fin
    fin
    dejaExplorer.ajout(e);
    aExplorer.enlever(e);
fin
retourner *Faux*

Dans l'optique où la solution ne se trouve pas dans le cercle, on peut faire une dichotomie limitée et élargir le cercle.

Algorithm 2: Algorithme d'accessibilité dichotomique dans $\mathbb{Z}^2$

Entrées: s_0, s_f les points de départ et arrivé, A un ensemble de Vecteur2, n la limite d'itération
Sorties: Un chemin de s_0 à s_f,faux sinon

```
int limite = arondiSup(|(s0, sf)| + Σ_{i=1}^{|A|} |Ai|);
Liste etats = (s0);
Liste dejaExplore = ∅;
Liste aExplorer = (s0);
Liste boucleSuivante = ∅;
pour i = 0 → n faire
    tant que |aExplorer| ≠ 0 faire
        e = premier(aExplorer);
        si e = sf alors
            Liste chemin = ∅;
            tant que p[e] ≠ ∅ faire
                chemin.insertTete(e);
                e = p[e];
            fin
            chemin.insertTete(s0);
            retourner chemin
        fin
        pour chaque a ∈ A faire
            s = suc(e,a);
            p[s] = e;
            etat.ajoutFin(s);
            si s ∉ {dejaExplorer ∪ aExplorer} alors
                si distance entre s et sf ≤ limite alors
                    aExplorer.ajoutFin(s)
                fin
                sinon
                    boucleSuivante.ajoutFin(s);
                fin
            fin
        fin
        dejaExplorer.ajout(e) aExplorer.enlever(e) ;
    fin
    limite = limite * 2;
    aExplorer = boucleSuivante //copie ;
fin
retourner Faux
```

4.3 Idées d'améliorations

— Ordonner les actions par norme décroissante. Ces actions sont généralement plus à même de faire partie du plus court chemin et on peut eventuellement gagné du temps en les traitant en premier.
— Si toute les actions divèrgent du point final, il ne peut pas y avoir de solution, cela n'est pas couteux à vérifier ($o(n)$).

4.4 Limite de l'heuristique

La compléxité de cet algorithme est exponentiel car cela consiste a parcourir l'ensemble des point du cercle et dans le cas d'une grande distance entre s_0 et s_f avec de petite actions, le temps nécessaire pour trouver une solution est immense, de même que le nombre d'états créés.

Example 4.1.

— Point de départ $s_0 = (1, 0)$
— Point final $s_f = (0, 2^{2n} + 1) = (0, 1025)$

— Actions : $\{(-1,1),(-1,2),(1,-1),(2,1)\}$

5 Deuxième heuristique

5.1 Fonctionnement

Dans l'exemple donnant du mal a notre première heuristique on remarque que l'on pouvait faire des sauts en multipliant certaines actions. On peut redéfinire le problème comme étant une équation tel que :

$$s_0 + \alpha_1 \times A_1 + \alpha_2 \times A_2 + \ldots + \alpha_n \times A_n = s_f$$

On peut donc chercher à atteindre le voisinage de s_f depuis s_0 en faisant des sauts. Pour définir le voisininage de s_f, on peut a nouveau utiliser un cercle. Il y a alors plusieurs options :

— Un cercle de rayon $max(|A_i|)$

— Un cercle de rayon $\sum_{i=1}^{|A|} |A_i|)$

Pour l'instant on utilisera la deuxième qui est plus large. Une fois dans ce cercle, on peut reproduire le schema de la première heuristique et le parcourir. On peut donc définir un algorithme générique.
Malheureusement, le fait de faire des saut, rend le parcours en largeur imparfait et ne nous garanti donc pas forcement la meilleure solution.
Pour calculer le saut nous avons besoin de faire deux étape

1. Notre droite (issu de l'actions) passe-t-elle dans le cercle ?
2. Peut-on s'arreter dans le cercle ?

On procède ainsi :

1. Pour la première étape, on va utiliser les définition géométrique d'un cercle et d'une droite et on va résoudre le système. On commence par trouver l'équation de droite avec notre action (a_1, a_2), comme on sait que la droite $d : ax + by + c = 0$ a pour vecteur directeur $v = (-b, a)$, on construit a une droite comme suit

$$\begin{aligned} a_2x - a_1y + c &= 0 \\ a_1y &= \frac{a_2x - c}{a_1} \\ y &= \frac{a_2}{a_1}x + \frac{c}{a_1} \end{aligned}$$

On pose $m = \frac{a_2}{a_1}$ et $p = \frac{c}{a_1}$. Ensuite nous resolvons l'équation suivants :

$$\begin{aligned} & (x - s_f.x)^2 + (y - s_f.y)^2 = r^2 \\ \Longrightarrow\ & x^2 - 2x \times s_f.x + s_f.x^2 + y^2 - 2x \times s_f.y + s_f.y^2 = r^2 \\ \Longrightarrow\ & x^2 - 2x \times s_f.x + s_f.x^2 + (mx + p)^2 - 2y \times s_f.(mx + p) + s_f.y^2 = r^2 \\ \Longrightarrow\ & x^2 - 2x \times s_f.x + s_f.x^2 + (mx)^2 + 2mxp + p^2 - 2(mx + p) \times s_f.y + s_f.y^2 = r^2 \\ \Longrightarrow\ & x^2 + m^2x^2 - 2xs_f.x + 2mxp - 2s_f.ymx + s_fx^2 + p^2 - 2s_f.y = r^2 \\ \Longrightarrow\ & (m^2 + 1)x^2 + (-2s_f.x + 2mp - 2s_f.ym)x + s_f.x^2 + p^2 - 2s_f.yp + s_f.y^2 - r^2 = 0 \end{aligned}$$

On a une equation du second degrée de la forme $ax^2 + bx + c = 0$ avec :

$$\begin{aligned} a &= m^2 + 1 \\ b &= -2s_f.x + 2mp - 2s_f.ym \\ c &= s_f.x^2 + p^2 - 2s_f.yp + s_f.y^2 - r^2 \end{aligned}$$

En utilisant le discriminant on obtient les deux x qui intersecte avec le point et on obtient les y avec notre équation de droite.

2. Soit $p_1 = (x_1, y_1)$ et $p_2 = (x_2, y_2)$ nos deux points d'intersections. On prend tout les $p_i = (x_i, y_i)$ avec $x_1 \leq x_i \leq x_2$ et $y_1 \leq y_i \leq y_2$. Ce qui nous donne un ensemble de point de $\mathbb{Z}^2$. On va ensuite calculer la distance entre chaque p_i et s_0 et diviser cette distance par la norme du vecteur utilisé. Si le résultat est entier, c'est un α valide et le garde.

Cependant cela ne resoudrait pas notre precédent exemple. Pour cela on peut, en plus de calculer ce saut sur nos actions, le calculer sur des combinaison linéaire de nos actions. Ces combinaisons linéaire sot calculer avec $\alpha_i \in \{0, 1\}$.

Example 5.1. Avec les actions (a_1, a_2, a_3) nous aurons les combinaisons $(a_1, a_2, a_3, a_1 + a_2, a_1 + a_3, a_2 + a_3, a_1 + a_2 + a_3)$.

Nous avons cependant deux cas particuliers à prendre en compte lorsque l'on calcule un saut :

1. $a = (x, 0)$, cela entraînerait une division par 0, il faut alors définir manuellement l'equation de droite de la forme $y = k$
2. $a = (0, y)$, ce cas est plus complexe car on a une equation de droite de la forme $x = k$, on doit donc manuellement affecter cette équation et legèrement altérer le calcule de a, b, c pour la résolution de l'équation du second degrée.

5.2 Algorithme

Algorithm 3: Fonction de saut

Entrées: s_f, d la limite, s notre état actuel et a un vecteur
Sorties: n un ensemble de sauts possible
si $a.y = 0$ **alors**
  $m = 0$;
  $p = s.y$;
  $a = 1$;
  $b = -2 * s_f.x$;
  $c = s_f.x^2 + p^2 - 2 * s_f.y * p + s_f.y^2 - r^2$;
fin
sinon si $a.x = 0$ **alors**
  $m = 0$;
  $p = s.x$;
  $a = 1$;
  $b = -2 * s_f.y$;
  $c = s_f.x^2 + p^2 - 2 * s_f.x * p + s_f.y^2 - r^2$;
fin
sinon
  $m = \frac{a.x}{a.y}$;
  $p = \frac{s.y}{a.y}$;
  $a = m^2 + 1$;
  $b = -2 * s_f.x + (2mp - 2) * (s_f.y * m)$;
  $c = s_f.x^2 + p^2 - 2 * s_f.y * p + s_f.y^2 - r^2$;
fin
$\Delta = b^2 - 4ac$;
si $\Delta < 0$ **alors**
  retourner $\varnothing$;
fin
si $\Delta = 0$ **alors**
  $x = \frac{-b+\sqrt{\Delta}}{2a}$;
  $y = mx + p$;
  si $x, y \in \mathbb{Z}$etdistance$((x, y), s)/|a| \in \mathbb{Z}^+$ **alors**
    retourner distance$((x, y), s)/|a|$
  fin
  sinon
    retourner $\varnothing$
  fin
fin
sinon
  $\alpha = \varnothing$;
  si $a.x \neq 0$ **alors**
    $x_2 = \frac{-b+\sqrt{\Delta}}{2a}$;
    $x1 = \frac{-b-\sqrt{\Delta}}{2a}$;
    $y_1 = mx_1 + p$;
    $y_2 = mx_2 + p$;
    pour chaque $(x, y) \in \mathbb{Z}^2 | x_1 \leq x \leq x_2 et y_1 \leq y \leq u_2$ **faire**
      si distance$((x, y), s)/|a| \in \mathbb{Z}^+$ **alors**
        α.ajout(istance$((x, y), s)/|a|$)
      fin
    fin
    retourner α
  fin
  sinon
    $y_2 = \frac{-b+\sqrt{\Delta}}{2a}$;
    $y_1 = \frac{-b+\sqrt{\Delta}}{2a}$;
    pour chaque $(p, y) \in \mathbb{Z}^2 | x_1 \leq x \leq x_2 et y_1 \leq y \leq u_2$ **faire**
      si distance$((x, y), s)/|a| \in \mathbb{Z}^+$ **alors**
        α.ajout(istance$((x, y), s)/|a|$)
      fin
    fin
    retourner α
  fin
fin

Algorithm 4: Algorithme générique d'accessibilité dans $\mathbb{Z}^2$

Entrées: s_0, s_f les points de départ et arrivé, A un ensemble de Vecteur2,
Sorties: Un chemin de s_0 à s_f,faux sinon

```
int limite = arondiSup(∑_{i=1}^{|A|} |A_i|);
Liste etats = (s_0);
Liste dejaExplore = ∅;
Liste aExplorer = (s_0);
int premiereIter = vrai;
Liste sauts = ∅;
Liste A' = toute combinaison linéaire unique de A ;
tant que /aExplorer/ ≠ 0 faire
    e = premier(aExplorer);
    si e = s_f alors
        Liste chemin = ∅;
        tant que p[e] ≠ ∅ faire
            si e ∈ sauts alors
                chemin.insertTete(ensemble des état sauté avec le saut);
            fin
            sinon
                chemin.insertTete(e);
            fin
            e = p[e];
        fin
        chemin.insertTete(s_0);
        retourner chemin
    fin
    si premiereIter alors
        pour chaque a_i ∈ A' faire
            α = saut(s_f, limite, e, a_i;
            pour chaque α_i ∈ α faire
                s = suc(e,α_i.a_i);
                p[s] = e;
                sauts.ajoutFin(s,α_i, a_i);
                //Si a_i combinaison on le stocke sous sa forme combinatoire aExplorer.insertTete(s);
            fin
        fin
        si /aExplorer/ = 1 alors
            retourner Pas de saut trouvé résolution rapide impossible avec cet algo
        fin
        premiereIter = faux;
    fin
    sinon
        pour chaque a ∈ A faire
            s = suc(e,a);
            p[s] = e;
            etat.ajoutFin(s);
            si s ∉ {dejaExplorer ∪ aExplorer} alors
                si distance entre s et s_f ≤ limite alors
                    aExplorer.ajoutFin(s)
                fin
            fin
        fin
    fin
    dejaExplorer.ajout(e);
    aExplorer.enlever(e);
fin
retourner Faux
```

En théorie, cette heuristique est plus rapide, elle a cependant le défaut de ne pas pouvoir trouver de solution si jamais elle n'atteint pas le cercle en un saut. On peut donc a nouveau effectué une dichotomie limité sur ce cercle. Le calcul de saut sera effectué a chaque élargissement de cercle.

Algorithm 5: Algorithme générique dichotod'accessibilité dans $\mathbb{Z}^2$

Entrées: s_0, s_f les points de départ et arrivé, A un ensemble de Vecteur2, n le nombre max d'itération
Sorties: Un chemin de s_0 à s_f,faux sinon

```
int limite = arondiSup(∑_{i=1}^{|A|} |A_i|);
Liste etats = (s_0);
Liste dejaExplore = ∅;
Liste aExplorer = (s_0);
int premiereIter = vrai;
Liste sauts = ∅;
Liste A' = toute combinaison linéaire unique de A ;
Liste prochaineIteration = ∅ pour i = 0 → n faire
    tant que /aExplorer/ ≠ 0 faire
        e = premier(aExplorer);
        si e = s_f alors
            Liste chemin = ∅;
            tant que p[e] ≠ ∅ faire
                si e ∈ sauts alors
                    chemin.insertTete(ensemble des état sauté avec le saut);
                fin
                sinon
                    chemin.insertTete(e);
                fin
                e = p[e];
            fin
            chemin.insertTete(s_0);
            retourner chemin
        fin
        si premiereIter alors
            pour chaque a_i ∈ A' faire
                α = saut(s_f, limite, e, a_i;
                pour chaque α_i ∈ α faire
                    s = suc(e,α_i.a_i);
                    p[s] = e;
                    si s ∉ aExplorer ∩ dejaExplore alors
                        sauts.ajoutFin(s,α_i, a_i);
                        //Si a_i combinaison on le stocke sous sa forme combinatoire aExplorer.insertTete(s);
                    fin
                fin
            fin
            premiereIter = faux;
        fin
        sinon
            pour chaque a ∈ A faire
                s = suc(e,a);
                p[s] = e;
                etat.ajoutFin(s);
                si s ∉ {dejaExplorer ∪ aExplorer} alors
                    si distance entre s et s_f ≤ limite alors
                        aExplorer.ajoutFin(s);
                    fin
                    sinon
                        prochaineIteration.ajoutFin(s);
                    fin
                fin
            fin
        fin
        dejaExplorer.ajout(e);
        aExplorer.enlever(e);
    fin
    limite = limite * 2;
    aExplorer = boucleSuivante //copie ;
    premierIteration = vrai;
    aExplorer.insertTete(s_0);
fin
retourner Faux
```

5.3 Idee d'amélioration :

— Dans l'implémentation actuelle, les combinaisons linéaire ne sont pas unique (ie (a_1, a_2) donnera $(a_1, a_2, a_1 + a_2, a_2 + a_1)$)
— Lors du calcul de saut, il pourrait etre intéressant de vérifier si un des points accéssible n'est pas directement l'état final
— Lors de la dichotomie, il pourrait être intéressant de calculer des sauts a partir des cercle $\mathcal{C}_n, n > 1$ pour revenir dans le premier cercle, le cout en complexité (surtout spatial) reste a estimer

5.4 Limite de l'heuristique :

Nous n'avons pas trouvé de contre exemple, infirmant l'éfficacité de notre heuristique. Cependant nous ne pouvons pas prouver que cette heuristique trouve le plus court chemin, ni ne répond directement a la question de l'accessibilité. Pour le premier point cependant, cette heuristique pourrait etre utiliser pour guider une autres consacrée a la recherche de plus court chemin. La principale limite est sur la recherche de saut, en effet, sauf si toute les actions divergent du point d'arivé, il existera une combinaison linéaire permettant d'arriver dans le cercle, nous ne garantissons pas cependant l'arrêt dans ce cercle, d'où l'interêt de la dichotomie.

6 Méthodologie de test :

Pour commencer, nous effectuons un test avec le jeu de données suivant :
— $s_0 = (0, 0)$
— $s_f = (10, 10)$
— $A =$ les 4 vecteurs unitaire
Cela nous permet de confirmer si l'heuristique est fonctionnel.
De la même façon, nous effectuons un test avec un jeu n'ayant pas de solution :
— $s_0 = (0, 0)$
— $s_f = (10, 10)$
— $A = \{(0, 1)\}$
Dans un second temps, nous utilisons le contre exemple de l'heuristique 1, a savoir :
— $s_0 = (1, 0)$
— $s_f = (0, 2^n + 1)$
— $A = \{(-1, 1), (-1, 2), (1, -1), (2, 1)\}$
Avec $n = 2, 4, 5, 8, 9, 10$ Sur la première heuristique nous ne somme pas allé plus loin que $n = 8$, celui-ci prenant déja un temps considérable (intérompu après plus d'une heure d'éxécution) et un grand nombre d'état créé (pour un grand n, il y aura une memory overflow).

7 Pour aller plus loin

— JEROME, leroux. The general vector addition system reachability problem by Presburger inductive invariants. Logical Methods in Computer Science, 2010, vol. 6.
— CZERWIŃSKI, Wojciech, LASOTA, Sławomir, LAZIĆ, Ranko, et al. Reachability in fixed dimension vector addition systems with states. arXiv preprint arXiv :2001.04327, 2020.

8 Conclusion

Bien que simple a formaliser, le problème d'accessibilité dans $\mathbb{Z}^2$ est difficile a résoudre, d'autant plus avec notre faible expérience et notre niveau en mathématique. Cependant, en consultant la littérature (voir section précédente), nous avons vu que nous étions sur la bonne piste avec notre deuxième heuristique en exploitant la semi linéarité des vecteur (et notamment la stabilité par produit externe entier) pour faire des saut, ainsi que la réinterprétation du problème sous la forme d'un systeme d'équation.
Cela nous a permis d'élargir notre corpus de compétence et renforcer nos méthode sur les processus itératif.