TP AAIA numéro 2 : IDA* pour le monde des blocs

Les algorithmes A* et IDA* sont utilisés pour résoudre des problèmes de planification modélisés sous la forme de la recherche de plus courts chemins dans des graphes d'états. Dans ce TP, nous allons utiliser IDA* pour résoudre le problème du monde des blocs qui est un problème de planification classique et emblématique. L'objectif est de trouver la plus petite suite d'actions à faire pour changer la configuration d'un ensemble de blocs.

Pour ce TP, nous utiliserons le langage C++11. Pour compiler le programme `src.cpp` vous pourrez utiliser la commande

```
g++ -std=c++11 -g src.cpp -o nomExec
```

en phase de développement, et la commande

```
g++ -std=c++11 -O3 src.cpp -o nomExec
```

pour améliorer la performance (l'option `-O3` demande au compilateur d'optimiser le code).

Attention : Vous devez répondre à un questionnaire disponible sur Moodle.

1 A* et IDA*

Pour commencer, nous vous proposons de répondre à quelques questions sur les algorithmes A* et IDA*.

Questions (réponses à donner sur Moodle) :

Q1 : Quand `A*` ne parvient pas à résoudre un problème, est-ce d'abord par manque de temps ou de mémoire ?

Q2 : La quantité de mémoire utilisée par `A*` est-elle linéairement proportionnelle à la taille d'un chemin solution, au nombre de solutions partielles de coût inférieur à la solution optimale, ou à la taille de l'espace d'états ?

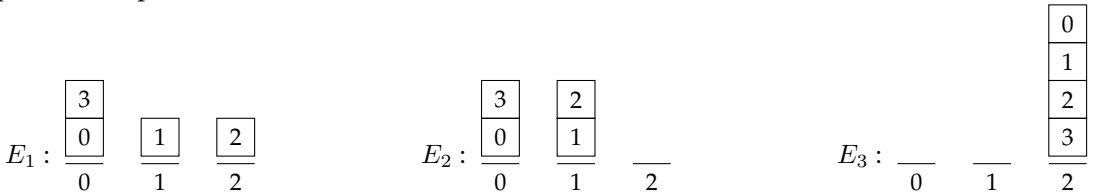
Q3 : La quantité de mémoire utilisée par `IDA*` est linéairement proportionnelle à la taille d'un chemin solution, au nombre de solutions partielles de coût inférieur à la solution optimale, ou à la taille de l'espace d'états ?

Q4 : Rappelons que IDA* effectue une succession de parcours en profondeur, chaque parcours correspondant à un appel à la procédure SEARCH. Cette procédure ne précise pas dans quel ordre les voisins de l'état courant sont explorés. Notons #rand(i) le nombre d'états explorés lors du i-ème appel à la procédure SEARCH dans le cas où les voisins de l'état courant sont explorés dans un ordre quelconque, et #best(i) le nombre d'états explorés lors du i-ème appel à la procédure SEARCH dans le cas où les voisins de l'état courant sont explorés par ordre croissant de la valeur heuristique h. Supposons que la solution est trouvée au n-ème appel à SEARCH. Parmi les affirmations suivantes, quelles sont celles qui sont nécessairement vraies :

- Si $i < n$, alors #best(i) $\leq$ #rand(i)
- Si $i < n$, alors #best(i) $=$ #rand(i)
- Si $i < n$, alors #best(i) $\geq$ #rand(i)
- Si $i = n$, alors #best(i) $\leq$ #rand(i)
- Si $i = n$, alors #best(i) $=$ #rand(i)
- Si $i = n$, alors #best(i) $\geq$ #rand(i)

2 Modélisation du problème du monde des blocs sous la forme d'un problème de planification

Nous supposerons qu'il y a n blocs (numérotés de 0 à $n-1$) et k piles (numérotées de 0 à $k-1$). Nous dirons d'une pile qu'elle est vide s'il n'y a aucun bloc posé dessus. Trois exemples d'états possibles avec $n = 4$ blocs et $k = 3$ piles sont représentés ci-dessous :



L'état E_2 a une pile vide (2), et l'état E_3 a deux piles vides (0 et 1).

Une action (aussi appelée mouvement) consiste à prendre un bloc au sommet d'une pile non vide p_1, et à le déposer au sommet d'une autre pile p_2. Une telle action est notée $p_1 \rightarrow p_2$. Nous noterons $actions(E)$ l'ensemble de toutes les actions possibles pour un état E donné. Par exemple, les actions possibles pour l'état E_2 sont :

$$actions(E_2) = \{0 \rightarrow 1, 0 \rightarrow 2, 1 \rightarrow 0, 1 \rightarrow 2\}$$

L'application d'une action $a \in actions(E)$ sur un état E permet d'obtenir un nouvel état noté $t(E, a)$. Par exemple, l'application de l'action $2 \rightarrow 1$ sur E_1 permet d'obtenir l'état $E_2 = t(E_1, 2 \rightarrow 1)$.

L'objectif du TP est d'écrire un programme pour trouver la plus petite séquence d'actions permettant de passer d'un état initial (où les blocs sont uniformément répartis sur les piles) à un état final (où tous les blocs sont empilés sur la dernière pile, du plus grand au plus petit). Par exemple, la plus petite séquence d'actions permettant de passer de l'état initial E_1 à l'état final E_3 est :

$$< 2 \rightarrow 1, 0 \rightarrow 2, 1 \rightarrow 2, 1 \rightarrow 2, 0 \rightarrow 2 >$$

Questions (réponses à donner sur Moodle) :

Q5 : Combien d'actions différentes sont-elles possibles pour l'état E_1 ?
Q6 : Etant donné un état E de n blocs sur k piles, quelle est la taille maximale de $actions(E)$?
Q7 : Etant donné un état E de n blocs sur k piles ayant v piles vides, quelle est la taille de $actions(E)$?
Q8 : Quel est l'ordre de grandeur du nombre total d'état différents possibles ?

3 Modélisation sous la forme de la recherche d'un plus court chemin dans un graphe

Etant donnés n et k, nous définissons le graphe $G = (S, A)$ tel que

— S est l'ensemble des états possibles, chaque état étant une configuration différente des n blocs sur les k piles ;
— $A = \{(E_i, E_j) \mid \exists a \in actions(E_i), E_j = t(E_i, a)\}$

Pour trouver la plus petite séquence d'actions permettant de passer d'un état initial E_0 à un état final E_f, nous pouvons rechercher le plus court chemin dans G allant de E_0 à E_f. Pour cela, nous pouvons faire une parcours en largeur (BFS) au départ de c_0.

Questions (réponses à donner sur Moodle) :

Q9 : Le graphe G est-il orienté ?
Q10 : Quels sont les algorithmes qui peuvent être utilisés pour résoudre ce problème ?
Q11 : Quel est l'algorithme le plus efficace pour résoudre ce problème ?
Q12 : Quelle est la complexité en temps de cet algorithme par rapport à $|S|$ et $|A|$?
Q13 : Quelle est la complexité en temps de cet algorithme par rapport à n et k ?

4 Heuristiques possibles pour l'algorithme IDA*

Nous vous proposons d'utiliser l'algorithme IDA* pour rechercher le plus court chemin allant de E_0 à E_f. Cet algorithme utilise une heuristique pour estimer la distance séparant chaque sommet E_i du graphe de l'état final E_f. Nous décrivons ci-dessous plusieurs heuristiques possibles (nous rappelons que l'état final est l'état où tous les blocs sont empilés sur la dernière pile, du plus grand au plus petit, comme illustré dans l'état E_3 en première page) :

— h_0 : Pas d'heuristique...

$$h_0(E_i) = 0$$

— h_1 : Nombre de blocs ne se trouvant pas sur la dernière pile.

$$h_1(E_i) = |\{b \in [0, n-1] \mid E_i.pile(b) \neq k-1\}|$$

où $E_i.pile(b)$ retourne le numéro de la pile où se trouve le bloc b dans l'état E_i.

— h_2 : Nombre de blocs ne se trouvant pas sur la dernière pile, plus deux fois le nombre de blocs b tels que b se trouve sur la dernière pile mais la liste de blocs se trouvant sous b est différente de $\langle b+1, b+2, \ldots, n-1\rangle$.

$$\begin{aligned} h_2(E_i) &= |\{b \in [0, n-1] \mid E_i.pile(b) \neq k-1\}| \\ &+ 2 * |\{b \in [0, n-1] \mid E_i.pile(b) = k-1 \text{ et } E_i.sous(b) \neq \langle b+1, b+2, \ldots, n-1\rangle\}| \end{aligned}$$

où $E_i.sous(b)$ retourne la liste des blocs se trouvant sous le bloc b dans E_i.

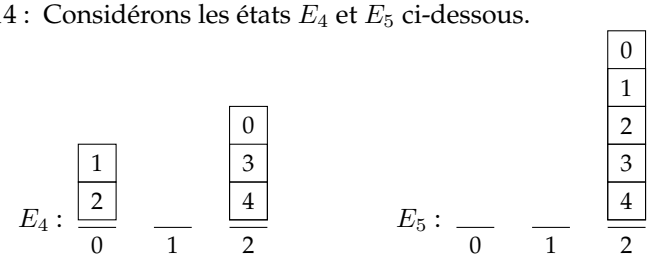
— h_3 : Nombre de blocs de E_i ne se trouvant pas sur la dernière pile, plus le nombre de blocs se trouvant au dessus de chaque bloc ne se trouvant pas sur la dernière pile.

$$h_3(E_i) = \sum_{b \in [0,n-1], E_i.pile(b) \neq k-1} 1 + E_i.nbSur(b)$$

où $E_i.nbSur(b)$ retourne le nombre de blocs se trouvant sur le bloc b dans E_i.

Questions (réponses à donner sur Moodle) :

Q14 : Considérons les états E_4 et E_5 ci-dessous.



Quelles sont les valeurs de $h_1(E_4)$, $h_2(E_4)$ et $h_3(E_4)$, sachant que l'état final est E_5 ?

Q15 : L'heuristique h_1 est-elle admissible ?

Q16 : L'heuristique h_2 est-elle admissible ?

Q17 : L'heuristique h_2 est elle plus informée, moins informée, ou incomparable par rapport à l'heuristique h_1

Q18 : L'heuristique h_3 est-elle admissible ?

Q19 : L'heuristique h_3 est elle plus informée, moins informée, ou incomparable par rapport à l'heuristique h_2

5 Programmation de l'algorithme IDA* pour le monde des blocs

Vous trouverez sur `http://liris.cnrs.fr/christine.solnon/state.cpp` une implémentation en C++ de la classe State permettant, par exemple, de créer des états, de les afficher, de les transformer en appliquant des actions, etc.

Vous trouverez sur `http://liris.cnrs.fr/christine.solnon/exampleBlocs.cpp` un exemple d'utilisation de cette classe pour créer les états E_1 et E_3 (cf première page), ainsi que les états intermédiaires permettant de passer de E_1 à E_3 en appliquant la séquence d'actions

$$< 2 \rightarrow 1, 0 \rightarrow 2, 1 \rightarrow 2, 1 \rightarrow 2, 0 \rightarrow 2 >$$

Votre travail : Reprenez le code de l'algorithme IDA* étudié en cours [1]. Adaptez-le pour résoudre le problème du monde des blocs. Comparez différentes heuristiques pour ce problème.

Questions (réponses à donner sur Moodle) :

Q20 : Quelle est la longueur (en nombre d'actions) de la solution optimale pour $n = 8$ blocs et $k = 3$ piles ?

Q21 : Quelle est la longueur (en nombre d'actions) de la solution optimale pour $n = 16$ blocs et $k = 4$ piles ?

Q22 : Quelle est la longueur (en nombre d'actions) de la solution optimale pour $n = 20$ blocs et $k = 4$ piles ?

1. Code source : `https://github.com/peportier/ia07-idastar-delta-nei`
Support de cours : `http://p6e7p7.freeshell.org/teaching_2017_2018/ia/heuristics/`