TP AAIA numéro 2 : A* et IDA*

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 ces algorithmes pour résoudre les problèmes du taquin et du monde des blocs.

Attention : Vous devez répondre aux questions Q1 à Q20 sur Moodle [1].

1 Exercice 1 : Le Taquin

L'énoncé de cette partie et le code source associé peuvent être retrouvés à la section 9 du cours sur les heuristiques [2].

Vous devez implémenter la procédure `SEARCH` de l'algorithme IDA^* en partant d'un code source fourni [3]. Dans la fonction `main`, vous trouverez décrites trois configurations initiales (`C0`, `C1` et `C2`) que vous utiliserez pour vos expérimentations. Par ailleurs, vous utiliserez également le code source déjà développé en cours pour l'algorithme `A*` [4].

Questions (réponses à donner sur Moodle) :

Q1 : Si `A*` ne parvient pas à résoudre le taquin à partir de `C2`, est-ce d'abord par manque de temps ou de mémoire ?

Q2 : La quantité de mémoire utilisée par `A*` est-elle linéairement proportionnelle (1) à la taille d'un chemin solution, (2) au nombre de solutions partielles de coût inférieur à la solution optimale, ou (3) à la taille de l'espace d'états ?

Q3 : Combien de recherches en profondeur effectue `IDA*` pour résoudre le taquin à partir de `C0` ?

Q4 : Combien de recherches en profondeur effectue `IDA*` pour résoudre le taquin à partir de `C1` ?

Q5 : La quantité de mémoire utilisée par `IDA*` est linéairement proportionnelle (1) à la taille d'un chemin solution, (2) au nombre de solutions partielles de coût inférieur à la solution optimale, ou (3) à la taille de l'espace d'états ?

Q6 : Quelle est la longueur (en nombre d'actions) de la solution optimale pour `C2` ?

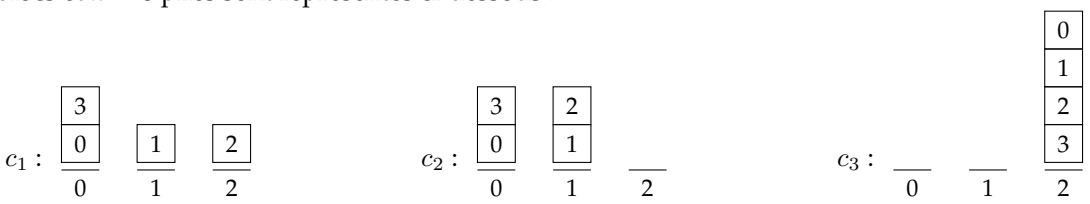
Q7 : En comparaison avec une version de l'algorithme pour laquelle les voisins d'un état sont explorés dans un ordre quelconque, la version meilleur d'abord (explorant les voisins d'un état par ordre croissant de la valeur heuristique h) pourra explorer : (1) moins d'états, (2) plus d'états, ou (3) au moins autant d'états que ceux vus lors des premiers n-1 parcours en profondeur (la solution étant trouvée au cous du n-ème parcours) ?
Attention : Plusieurs réponses possibles

2 Exercice 2 : Le monde des blocs

Le monde des blocs 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.

2.1 Modélisation du problème 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 de configurations possibles avec $n = 4$ blocs et $k = 3$ piles sont représentés ci-dessous :



La configuration c_2 a une pile vide (2), et la configuration c_3 a deux piles vides (0 et 1).

1. http://moodle2.insa-lyon.fr/mod/quiz/view.php?id=43650
2. http://liris.cnrs.fr/pierre-edouard.portier/teaching_2016_2017/ia/heur/heur.html
3. http://liris.cnrs.fr/pierre-edouard.portier/teaching_2016_2017/ia/heur/ida_tp.cpp
4. http://liris.cnrs.fr/pierre-edouard.portier/teaching_2016_2017/ia/heur/taquin15.cpp

Une action consiste à prendre un bloc au sommet d'une pile non vide p_1, et le déposer au sommet d'une autre pile p_2, et une telle action est notée $p_1 \rightarrow p_2$. Nous noterons $actions(c)$ l'ensemble de toutes les actions possibles pour une configuration c donnée. Par exemple, les actions possibles pour la configuration c_2 sont :

$$actions(c_2) = \{0 \rightarrow 1, 0 \rightarrow 2, 1 \rightarrow 0, 1 \rightarrow 2\}$$

L'application d'une action $a \in actions(c)$ sur une configuration c permet d'obtenir une nouvelle configuration notée $t(c, a)$. Par exemple, l'application de l'action $2 \rightarrow 1$ sur c_1 permet d'obtenir la configuration $c_2 = t(c_1, 2 \rightarrow 1)$.

L'objectif de ce deuxième exercice est d'écrire un programme pour trouver la plus petite séquence d'actions permettant de passer d'une configuration initiale à une configuration finale. Par exemple, la plus petite séquence d'actions permettant de passer de c_1 à c_3 est :

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

Questions (réponses à donner sur Moodle) :

Q8 : Combien d'actions différentes sont-elles possibles pour la configuration c_1 ?
Q9 : Etant donnée une configuration c de n blocs sur k piles, quelle est la taille maximale de $actions(c)$?
Q10 : Etant donnée une configuration c de n blocs sur k piles ayant v piles vides, quelle est la taille de $actions(c)$?
Q11 : Quel est l'ordre de grandeur du nombre total de configurations différentes possibles ?

2.2 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 configurations possibles des n blocs sur les k emplacements ;
— $A = \{(c_i, c_j) \mid \exists a \in actions(c_i), c_j = t(c_i, a)\}$

Pour trouver la plus petite séquence d'actions permettant de passer d'une configuration initiale c_0 à une configuration finale c_f, nous pouvons rechercher le plus court chemin dans G allant de c_0 à c_f. Pour cela, nous pouvons faire une parcours en largeur (BFS) au départ de c_0.

Questions (réponses à donner sur Moodle) :

Q12 : Le graphe G est-il orienté ?
Q13 : Quelle est la complexité en temps de BFS par rapport à $|S|$ et $|A|$?
Q14 : Quelle est la complexité en temps de BFS par rapport à n et k ?

2.3 Heuristiques possibles pour l'algorithme A*

Nous vous proposons d'utiliser l'algorithme A* pour rechercher le plus court chemin allant de c_0 à c_f. Cet algorithme utilise une heuristique pour estimer la distance séparant chaque sommet c du graphe de la configuration finale c_f. Nous décrivons ci-dessous plusieurs heuristiques possibles.

— h_0 : Pas d'heuristique...

$$h_0(c, c_f) = 0$$

— h_1 : Nombre de blocs se trouvant sur une pile différente dans c et c_f.

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

où $c.pile(b)$ retourne le numéro de la pile où se trouve le bloc b dans la configuration c.

— h_2 : Nombre de blocs se trouvant sur un emplacement différent dans c et c_f, plus deux fois le nombre de blocs qui sont au même emplacement dans c et c_f mais ont des sous-piles différentes sous eux.

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

où $c.sous(b)$ retourne la sous-pile de blocs se trouvant sous le bloc b dans c.

— h_3 : Nombre de blocs b se trouvant sur un emplacement différent dans c et c_f, plus le nombre de blocs se trouvant au dessus de b dans c.

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

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

— h_4 : Nombre de blocs se trouvant sur un emplacement différent dans c et c_f et n'ayant pas un même bloc sous eux, plus deux fois le nombre de blocs se trouvant sur un emplacement différent dans c et c_f et ayant au moins un même bloc sous eux, plus deux fois le nombre de blocs qui sont au même emplacement dans c et c_f mais qui ont des piles de blocs différentes sous eux.

$$\begin{aligned} h_4(c, c_f) &= |\{b \in [0, n-1] \mid c.pile(i) \neq c_f.pile(b) \text{ et } \forall b_2 \in c.sous(b), b_2 \notin c_f.sous(b)\}| \\ &+ 2 * |\{b \in [0, n-1] \mid c.pile(b) \neq c_f.pile(b) \text{ et } \exists b_2 \in c.sous(b), b_2 \in c_f.sous(b)\}| \\ &+ 2 * |\{b \in [0, n-1] \mid c.pile(b) = c_f.pile(b) \text{ et } c.sous(b) \neq c_f.sous(b)\} \end{aligned}$$

Questions (réponses à donner sur Moodle) :
Q15 : L'heuristique h_1 est-elle admissible ?
Q16 : L'heuristique h_2 est-elle admissible ?
Q17 : L'heuristique h_3 est-elle admissible ?
Q18 : L'heuristique h_4 est-elle admissible ?

2.4 Programmation des algorithmes A* et IDA* pour le monde des blocs

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

Vous trouverez dans `http://liris.cnrs.fr/christine.solnon/exempleBlocs.cpp` un exemple d'utilisation de cette classe pour créer les configurations c_1 et c_3 de la figure de la page précédente, ainsi que les configurations intermédiaires permettant de passer de c_1 à c_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 les programmes A* et IDA* de l'exercice 1 et adaptez-les pour résoudre le problème du monde des blocs. Comparez différentes heuristiques pour ce problème. Comparez les performances de A* et IDA*.

Questions (réponses à donner sur Moodle) :
Q19 : Quelle est la longueur (en nombre d'actions) de la solution optimale pour $n = 8$ blocs et $k = 3$ piles ?
Q20 : Quelle est la longueur (en nombre d'actions) de la solution optimale pour $n = 16$ blocs et $k = 4$ piles ?