TP5-Planification d'actions

Yan CHEN

Janvier 2021

1 Introduction

On utilise le planificateur d'actions CPT version 2 pour résoudre touts les problèmes dans le TP. Il a le même domaine de planification pour le monde des cubes pour exercice 1 à 4.

Listing 1 – domain-blocksaips.pddl

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; 4 Op-blocks world
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (domain BLOCKS)
  (:requirements :strips)
  (:predicates (on ?x ?y)
               (ontable ?x)
               (clear ?x)
               (handempty)
               (holding ?x)
               )

  (:action pick-up
             :parameters (?x)
             :precondition (and (clear ?x) (ontable ?x) (handempty))
             :effect
             (and (not (ontable ?x))
                   (not (clear ?x))
                   (not (handempty))
                   (holding ?x)))

  (:action put-down
             :parameters (?x)
             :precondition (holding ?x)
             :effect
             (and (not (holding ?x))
                   (clear ?x)
                   (handempty)
                   (ontable ?x)))
  (:action stack
```

```
32              :parameters (?x ?y)
33              :precondition (and (holding ?x) (clear ?y))
34              :effect
35              (and (not (holding ?x))
36                   (not (clear ?y))
37                   (clear ?x)
38                   (handempty)
39                   (on ?x ?y)))
40
41    (:action unstack
42              :parameters (?x ?y)
43              :precondition (and (on ?x ?y) (clear ?x) (handempty))
44              :effect
45              (and (holding ?x)
46                   (clear ?y)
47                   (not (clear ?x))
48                   (not (handempty))
49                   (not (on ?x ?y)))))
```

Listing 2 – blocksaips01.pddl

```
1  (define (problem BLOCKS-4-0)
2  (:domain BLOCKS)
3  (:objects D B A C )
4  (:INIT (CLEAR C) (CLEAR A) (CLEAR B) (CLEAR D) (ONTABLE C) (ONTABLE A)
5   (ONTABLE B) (ONTABLE D) (HANDEMPTY))
6  (:goal (AND (ON D C) (ON C B) (ON B A)))
7  )
```

2 Exercice 1

Il y a 4 opérations du fichier de domaine. Il sont respectivement **pick-up, put-down, stack, unstack**. La définition des opérations sont le suivant :

1. **pick-up :** ramasser le cube de la table à la main.
2. **put-down :** mettre le cube sur la table.
3. **stack :** mettre une cube sur l'autre cube.
4. **unstack :** ramasser une cube mis sur l'autre cube.

Il faut distinguer le **put-down** du **stack**. Ils ont l'action similaire. Mais la différence entre eux est ce que **put-down** mets une cube sur la table. Et le cube est séparé avec l'autre. Ils sont tous sur la table. Ce pendant, **stack** mets une cube sur l'autre. L'autre cube est sur la table. On suppose que la table était infinie et toujours disponible, le cube n'était pas toujours disponible. On doit juger la disponibilité de cube tout à bord avant d'opération. C'est pour cette raison que ces deux opération doivent être dissociées.

Le fluent **(holding ?x)** signifie que le robot occupe le cube x. Il est utilisé pour simplifier les opérations **put-dowm** et **stack**, et pour empêcher les deux

opérations **pick-up** and **unstack**. Si ce fluent n'était pas là, il faut juger si le main du robot était vide, si le cube était sur la table.

3 Exercice2

On exécute le commande `cpt.exe -o domain-blocksaips.pddl -f blocksaips01.pddl` lancer le planificateur sur ce problème avec ce domaine.

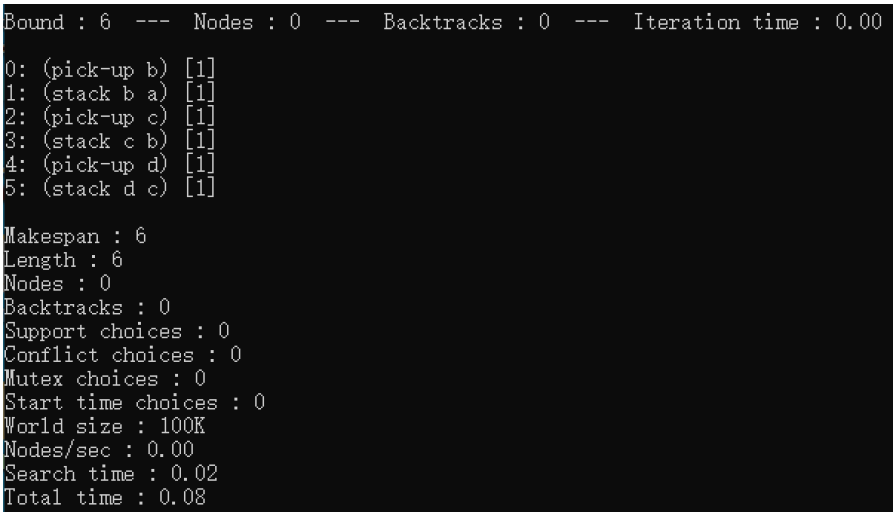

FIGURE 1 – Résultat de la planification

Comme on le voit, la longueur du plan-solution est 6. Il faut 0.02 s pour trouver. Le temps total est 0.08 s. Et il a eu une seule itération. Chaque action du plan-solution dure un pas de temps.

En fait, il existe nombreux de plan-solutions pour résoudre ce problèmes de cubes. Par exemple, tout à bord, on choie au hasard une cube. S'il est le cube b, on le met sur cube a, si non, on continue à choisir une cube jusqu'à le cube b. Et ainsi de suite jusqu'à la fin. Normalement, ce plan-solution prend plus de temps pour résoudre ce problème.

Ce plan-solution original est de **pick-up** b et **stack** b, puis c et d. Ce plan-solution est le meilleur. Il prend le moins de temps pour résoudre ce problème.

4 Exercice 3

On écrit le problème de planification suivant pour ce domaine des cubes.

Listing 3 – blocksaips02.pddl

```
(define (problem BLOCKS-4-0)
(:domain BLOCKS)
(:objects D B A C )
(:INIT (CLEAR B) (ON B C) (ON C A) (ON A D)(ONTABLE D) (HANDEMPTY))
```

```
(:goal (AND (ON D C) (ON C A) (ON A B)))
)
```

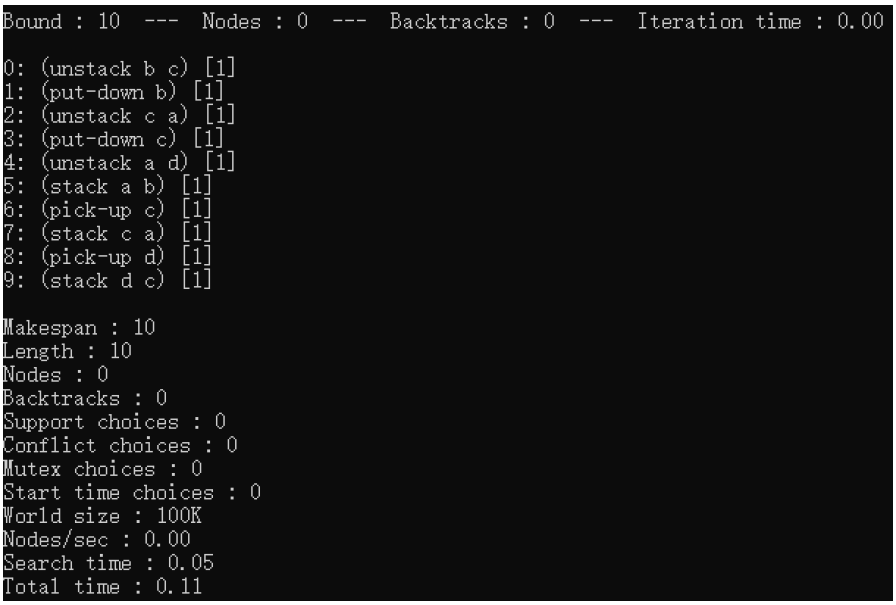

FIGURE 2 – Résultat de la planification

Comme on le voit, le longueur du plan-solution est 10. Il faut 0.05 s pour le trouver. Le temps total est 0.11 s. Et une seule itération est exécutée.

5 Exercice 4

On écrit le problème de planification suivant pour ce domaine des cubes :

Listing 4 – blocksaips03.pddl

```
(define (problem BLOCKS-4-0)
(:domain BLOCKS)
(:objects A B C D E F G H I J)
(:INIT (CLEAR C) (ON C G) (ON G E) (ON E I) (ON I J) (ON J A) (ON A B) (ONTABLE
    B)
(CLEAR F) (ON F D) (ON D H) (ONTABLE H) (HANDEMPTY))
(:goal (AND (ON C B) (ON B D) (ON D F) (ON F I) (ON I A) (ON A E) (ON E H) (ON
   H G) (ON G J)))
)
```

Comme on le voit, le longueur du plan-solution est 32. Il faut 0.94 s pour le trouver. Le tmeps total est 1.03 s. Et 7 itérations sont exécutées.

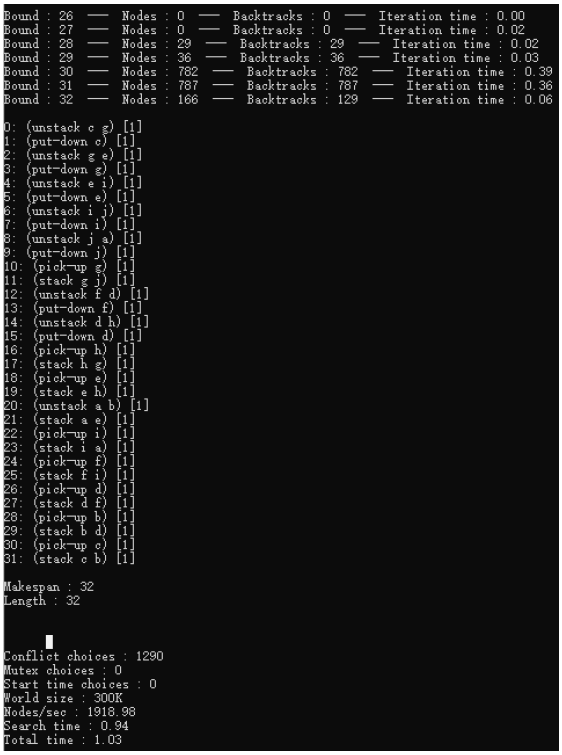

FIGURE 3 – Résultat de la planification

6 Exercice 5

On défini a graph acyclique orienté les nœuds comme la Figure 4. Puis, on souhaite trouver un chemin pour aller d'un nœud à un autre. Le chemin commence au point A et se termine au point H.

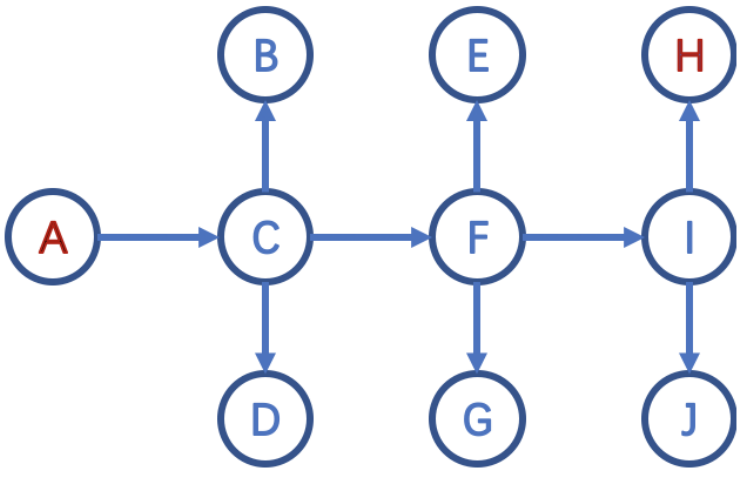

FIGURE 4 – Graph acyclique

On écrit un opérateur d'un domaine PDDL et un problème PDDL. Le code est indiqué comme le suivant.

Listing 5 – domain-path.pddl

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;graph acyclique
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(define (domain graph)
        (:requirements : strips)
        (:predicates (arc ?from ?to)    ;direction of arc
                (pose ?from))               ;current position

        (:action go
                :parameters (?from ?to)
                :precondition (and (pose ?from) (arc ?from ?to))
                :effect (and (pose ?to) (not (pose ?from))))
)
```

Listing 6 – problem-path.pddl

```
;;;;;;;;;;;;;;
;;;problem graph
;;;;;;;;;;;;;;

(define (problem graph)
        (:domain graph)
        (:objects A B C D E F G H I J K)
        (:INIT (pose A) (arc A C) (arc C B) (arc C D) (arc C F) (arc F E) (arc
            F G) (arc F I) (arc I H) (arc I J))
        (:goal (AND (pose H)))
)
```

```
Problem : 6 actions, 5 fluents, 5 causals
          1 init facts, 1 goals

Bound : 4  ——  Nodes : 0  ——  Backtracks : 0  ——  Iteration time : 0.00

0: (go a c) [1]
1: (go c f) [1]
2: (go f i) [1]
3: (go i h) [1]

Makespan : 4
Length : 4
Nodes : 0
Backtracks : 0
Support choices : 0
Conflict choices : 0
Mutex choices : 0
Start time choices : 0
World size : 100K
Nodes/sec : 0.00
Search time : 0.02
Total time : 0.08
```

FIGURE 5 – Résultat de la planification

Pour le graph acyclique au dessus, cette méthode de planification de chemin est efficace car il a trouvé le plus court chemin accès à destination. Comme on le

voit, le longueur du plan-solution est 4. Il a pris 0,02 s pour trouver une chemin. Le temps total est 0.08 s. Et une seule itération est exécuté.

7 Exercice 6

Dans ce exercice, on discute le problème le singe et les bananes. Un singe est dans une salle d'expérience avec des bananes accrochées au plafond hors d'atteinte. Dans cette salle il y a une caisse, sur laquelle le singe peut monter. Il peut aussi la pousser. On écrit une opération d'un domain en PDDL et le problème en PDDL. Les six opérations aller, pousser, monter, descendre, attraper et lâcher sont définis.

Listing 7 – domain-singe.pddl

```
;;;;;;;;;;;;
;;; @author: DavideLuigiBrambilla
;;; Modified by Yan CHEN
;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Domain Singe attraper babanes
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (domain Singe-banane)
        (:requirements :strips)
        (:predicates (Singe ?x)     ;singe in the position x
            (Caisse ?x)         ;caisse in the position x
                (Banane ?x)                 ;banane in the position x
                (Bas)                   ;low position
                (Haut)                  ;high position
                (handBananas)           ;bananas in the hand
                (handEmpty))            ;nothing in the hand

        (:action Aller
                :parameters (?from ?to)
                :precondition (and (Singe ?from) (Bas))
                :effect (and (Singe ?to) (not (Singe ?from))))

        (:action Pousser
                :parameters (?from ?to)
                :precondition (and (Singe ?from) (Caisse ?from) (Bas))
                :effect (and (Caisse ?to) (not (Caisse ?from)) (Singe ?to)))

        (:action Monter
                :parameters (?obj)
                :precondition (and (Singe ?obj) (Caisse ?obj) (Banane ?obj) (
                    Bas))
                :effect (and (Haut) (not(Bas))))

        (:action Descendre
                :parameters (?obj)
                :precondition (and (Singe ?obj)(Caisse ?obj) (Haut))
```

```
                :effect (and (Bas) (not(Haut))))

        (:action Attraper
                :parameters (?obj)
                :precondition (and (Banane ?obj) (handEmpty) (Haut))
                :effect (and (handBananas) (not(handEmpty))))

        (:action Lacher
                :parameters (?obj)
                :precondition (and (handBananas) (Banane ?obj) (Haut))
                :effect (and (not(handBananas)) (handEmpty)))
)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; problem singe attraper babanes
;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (problem singe-bananes01)
        (:domain Singe-banane)
        (:objects A B C)
        (:INIT (Singe A)
                (Caisse B)
                (Banane C)
                (Bas)
                (not(Haut))
                (not(handBananas))
                (handEmpty))
        (:goal (AND (handBananas)))
)
```

```
Problem : 18 actions, 10 fluents, 41 causals
          4 init facts, 1 goals

Bound : 4  ——  Nodes : 0  ——  Backtracks : 0  ——  Iteration time : 0.00

0: (aller a b) [1]
1: (pousser b c) [1]
2: (monter c) [1]
3: (attraper c) [1]

Makespan : 4
Length : 4
Nodes : 0
Backtracks : 0
Support choices : 0
Conflict choices : 0
Mutex choices : 0
Start time choices : 0
World size : 100K
Nodes/sec : 0.00
Search time : 0.01
Total time : 0.08
```

FIGURE 6 – Résultat de la planification

Comme on le voit, le longueur du plan-solution est 4. Il a pris 0,01 s pour attraper les bananes. Le temps total est 0.08 s. Et une seule itération est exécuté.

On n'a pas considéré la situation dans le programme. C'est la limite de poids du singe. si la caisse à pousser est trop lourd, le singe ne pourra pas pousser la caisse.

À mon avis, c'est un problème du qualification, et un problème de la ramification. La simplification du question est suivante une stratégie du programme.

8 Exercice 7

Dans cette exercice, on fait le jeu les tours de Hanoi. On écrit un opération d'un domain en PDDL l'état initial pour une tour à 1, 2, 3, 4 et 5 disques, initialement sur le pic1 à déplacer sur le pic3 avec le pic2 comme intermédiaire. Et on écrit le problème en PDDL les 5 opérateurs pour le domaine des tours de Hanoi pour 1 à 5 disques. On n'affiche que le code pour 3 disques comme exemple. Les problèmes en pddl pour 1 à 5 disques sont similaires.

Listing 8 – domain-tour.pddl

```
;;;;;;;;;;;;
;;; @author: DavideLuigiBrambilla
;;; Modified by Yan CHEN
;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Tower of Hanoi
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (domain Hanoi)
        (:requirements :strips)
        (:predicates (clear ?x)         ;determine if the top of x is empty
                (onDisk ?x ?y)  ;determine if the x is on the y
                (onPeg ?x ?y)   ;determine if the x on the table
                (smaller ?x ?y) ;x<y
                (handEmpty)     ;nothing in the hand
                (holding ?x)    ;x is in the hand
                (onTable ?x)    ;x is on the table
                (notOnTable ?x))        ;x is not on the table

        (:action pick_up_from_disk
                :parameters (?disk1 ?disk2)
                :precondition (and (onDisk ?disk1 ?disk2) (clear ?disk1) (
                    handEmpty))
                :effect (and (not (onDisk ?disk1 ?disk2)) (clear ?disk2) (
                    holding ?disk1) (not (handEmpty)) (not (clear ?disk1))))

        (:action pick_up_from_peg
                :parameters (?disk ?peg)
                :precondition (and (onPeg ?disk ?peg) (clear ?disk) (handEmpty)
                    )
                :effect (and (holding ?disk) (not(handEmpty)) (clear ?peg) (not
                    (onPeg ?disk ?peg)) (not(clear ?disk))))

        (:action put_on_disk
                :parameters (?disk1 ?disk2)
```

```
                :precondition (and (smaller ?disk1 ?disk2) (clear ?disk2) (
                    holding ?disk1) (notOnTable ?disk2))
                :effect (and (onDisk ?disk1 ?disk2) (handEmpty) (not (holding ?
                    disk1)) (clear ?disk1) (not (clear ?disk2))))

        (:action put_on_peg
                :parameters (?disk ?peg)
                :precondition (and (clear ?peg) (holding ?disk) (onTable ?peg))
                :effect (and (not(clear ?peg)) (onPeg ?disk ?peg) (not (holding
                     ?disk)) (handEmpty) (clear ?disk)))

)
```

Listing 9 – problem-tour3.pddl

```
;;;;;;;;;;;;;;;;;;;;;;
;;; problem with 3 disk
;;;;;;;;;;;;;;;;;;;;;;

(define (problem Tower3)
        (:domain Hanoi)
        (:objects pic1 pic2 pic3 disk1 disk2 disk3)
        (:init (smaller disk1 pic1) (smaller disk1 pic2) (smaller disk1 pic3) (
            smaller disk2 pic1)
                (smaller disk2 pic2) (smaller disk2 pic3) (smaller disk3 pic1)
                    (smaller disk3 pic2)
                (smaller disk3 pic3) (smaller disk1 disk2) (smaller disk1 disk3
                    ) (smaller disk2 disk3)
                (clear disk1) (clear pic2) (clear pic3) (onDisk disk1 disk2) (
                    onDisk disk2 disk3) (onPeg disk3 pic1)
                (notOnTable disk1) (notOnTable disk2) (notOnTable disk3) (
                    onTable pic1) (onTable pic2)
                (onTable pic3) (handEmpty))
        (:goal (and (onDisk disk1 disk2) (onDisk disk2 disk3) (onPeg disk3 pic3
            )))
)
```

Les résultats pour 1 à 5 disques sont affichés le suivants. On peut voir que le nombre de disque est plus grand, la complicité accroît et il est plus difficile de trouver la solution. Lorsque que le nombre de disque est 5, le cpt ne peut pas trouver une solution.

Numbre of disk	Length of plan-solution	Search time / s	Iteration
1	2	0.02	1
2	6	0.02	1
3	14	0.05	5
4	30	1.57	17
5	NoN	NoN	NoN

TABLE 1 – Résultat avec nombre de disques différents

En fait, il y a actuellement plusieurs solutions pour résoudre le problème du tour de Hanoi. Le suivant est un algorithme récursif.

Algorithm 1 algorithme récursif pour le problèm du tour Hanoi

```
1: PROCEDURE Hanoi(NDisques, PicDepart, PicIntermediaire, PicArrivee)
2: if NDisques different de 0 ALORS then
3:     Hanoi(NDisques – 1, PicDepart, PicArrivee, PicIntermediaire) ;
4:     Afficher("Deplacer le disque numero " + NDisques + « du pic » + PicDepart
   + « au pic » + PicArrivee + « \n ») ;
5:     Hanoi(NDisques – 1, PicIntermediaire, PicDepart, PicArrivee) ;
6: end if
7: FINPROCEDURE
```

L'idée de base de la solution est récursive. Supposons qu'il y ait trois tours A, B et C. La tour A comporte N disques, et le but est de déplacer tous ces disques vers la tour C. Ensuite, déplacez d'abord N-1 disques du haut de la tour A vers la tour B, puis déplacez les grands disques restants de la tour A vers la tour C, et enfin déplacez N-1 disques de la tour B vers la tour C. En utilisant cette méthode de manière récursive, la solution peut être résolue [1]. Par contre, CPT algorithme essaie différentes solutions possibles en fonction du nombre de prédicats et choisit la meilleure d'entre elles. La complexité de l'algorithme est élevée.

Références

[1] https://zh.wikipedia.org/wiki/Tower_of_Hanoi Accessed January 20, 2021.