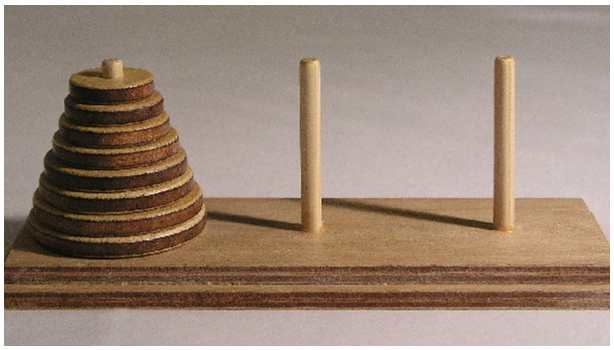
Planification d'actions

ROB316 - Planification et contrôle

Bastien HUBERT



ENSTA Paris - janvier 2023

1 Exercice 1 :

Les 4 opérateurs du fichier de domaine signifient respectivement :

- `pick-up` : prendre un block libre sur la table avec une main vide
- `put-down` : poser un bloc pris sur la table
- `stack` : poser un bloc pris sur un bloc libre
- `unstack` : prendre un block libre sur un cube avec une main vide

La différence entre les opérateurs `put-down` et `stack` est l'emplacement de dépose du cube, respectivement sur la table ou sur un bloc libre.

Le fluent `(holding ?x)` signifie que le bloc `x` est pris par la main, et n'est donc pas sur la table ni sur un bloc. Il sert à représenter l'état transitoire d'un bloc en cours de déplacement. Si ce fluent n'était pas là, il faudrait par exemple modifier les opérateurs afin d'introduire `puton ?b ?u ?t` (issu du cours). Dans notre cas, cela reviendrait à passer de $2 * 2 = 4$ opérateurs à $2^2 = 4$ opérateurs, mais dans le cas où on considère plusieurs types de blocs différents, la solution sans `(holding ?x)` force l'introduction de bien plus d'opérateurs.

2 Exercice 2 :

L'exécution de la commande `.\cpt.exe -o .\domain-blocksaips.pddl -f .\blocksaips01.pddl` dans un terminal conduit à la sortie suivante :

```
domain file : .\textbackslash domain-blocksaips.pddl.txt
problem file : .\textbackslash blocksaips01.pddl.txt

Parsing domain..... done : 0.00
Parsing problem..... done : 0.00
domain : blocks
problem : blocks-4-0
Instantiating operators..... done : 0.00
Creating initial structures..... done : 0.00
Computing bound..... done : 0.00
Computing e-deleters..... done : 0.00
Finalizing e-deleters..... done : 0.00
Refreshing structures..... done : 0.00
Computing distances..... done : 0.00
Finalizing structures..... done : 0.00
```

```
Variables creation..... done : 0.00
Bad supporters..... done : 0.00
Distance boosting..... done : 0.00
Initial propagations..... done : 0.00

Problem : 34 actions, 25 fluents, 79 causals
         9 init facts, 3 goals

Bound : 6 --- Nodes : 0 --- Backtracks : 0 --- Iteration time : 0.00

0: (pick-up b) [1]
1: (stack b a) [1]
2: (pick-up c) [1]
3: (stack c b) [1]
4: (pick-up d) [1]
5: (stack d c) [1]

Makespan : 6
Length : 6
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 : 0.00
Total time : 0.01
\end{partt}
```

Le plan solution est de longueur 6. Il est trouvé en 0.01s après 1 itération. Chaque action de ce plan solution dure 1 unité de temps.

Parmi les plans solutions plus longs, on peut imaginer

```
0: (pick-up b) [1]
1: (stack b a) [1]
2: (pick-up c) [1]
3: (stack c b) [1]
4: (pick-up d) [1]
5: (put-down d) [1]
6: (pick-up d) [1]
7: (stack d c) [1]
```

ou tout sur-plan solution qui prend puis dépose immédiatement un cube.

De tels plans ne sont pas fournis par le planificateur car celui-ci ne retourne que les plans solutions de longueur minimale.

3 Exercice 3 :

Le problème demandé se traduit en :

```
(define (problem BLOCKS-EX3)
(: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) (ONTABLE B)))
)
```

Le plan solution correspondant est de taille 10. Il est trouvé en 0.01s après 1 itération.

4 Exercice 4 :

Le problème demandé se traduit en :

```
(define (problem BLOCKS-EX4)
(: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) (ONTABLE J)))
)
```

Le plan solution correspondant est de taille 32. Il est trouvé en 0.36s après 5 itérations.

5 Exercice 5 :

Le domaine peut être décrit comme suit :

```
(define (domain PATH)
```

```
(:requirements :strips)
(:predicates
       (at ?from)
       (edge ?from ?to)
)

(:action move
       :parameters (?x ?y)
       :precondition (and (at ?x) (edge ?x ?y))
       :effect
              (and (not(at ?x)) (at ?y))
)
)
```

On peut le mettre en application avec le problème suivant :

```
(define (problem PATH-EX5)
(:domain PATH)
(:objects A B C D E )
(:INIT (at A) (edge A B) (edge B C) (edge C E) (edge A D) (edge D E))
(:goal (at E))
)
```

Dans ces conditions, le plan solution est de taille 2. Il est trouvé en 0.01s après 1 itération.

La méthode de planification employée est inefficace car le planificateur va effectuer un parcours en largeur d'abord, ce qui peut ralentir la recherche.

6 Exercice 6 :

Le domaine peut être décrit comme suit :

```
(define (domain MONKEY)
(:requirements :strips)
(:predicates
       (at ?x ?y)
       (height ?x ?y)
       (handempty)
       (hold)
       (monkey ?x)
       (box ?x)
       (bananas ?x)
```

```
        (plafond ?x)
        (sol ?x)
)

(:action aller
        :parameters (?x ?y ?m ?s ?h)
        :precondition
                (and (monkey ?m) (at ?m ?x) ;;; le singe est en x
                (sol ?s) (height ?s ?h) (height ?m ?h)) ;;; le singe est au niveau du sol
        :effect
                (and (not(at ?m ?x)) (at ?m ?y)) ;;; le singe se déplace de x vers y
)

(:action pousser
        :parameters (?x ?y ?m ?b ?s ?h)
        :precondition
                (and (monkey ?m) (at ?m ?x) (box ?b) (at ?b ?x) ;;; le singe et la boîte sont en x
                (sol ?s) (height ?s ?h) (height ?m ?h) (height ?b ?h)
                        ;;; le singe et la boîte sont au niveau du sol
                (handempty)) ;;; le singe a les mains libres
        :effect
                (and (not(at ?m ?x)) (at ?m ?y) ;;; le singe se déplace de x vers y
                (not(at ?b ?y)) (at ?b ?y)) ;;; la boîte se déplace de x vers y
)

(:action monter
        :parameters (?x ?m ?b ?s ?p ?h1 ?h2)
        :precondition
                (and (monkey ?m) (at ?m ?x) (box ?b) (at ?b ?x)
                        ;;; le singe et la boîte sont en x
                (sol ?s) (height ?s ?h1) (height ?m ?h1) (height ?b ?h1)
                        ;;; le singe et la boîte sont au niveau du sol
                (handempty) ;;; le singe a les mains libres
                (plafond ?p) (height ?p ?h2))
        :effect
                (and (not(height ?m ?h1)) (height ?m ?h2))
                        ;;; le singe monte au niveau du plafond
)

(:action descendre
        :parameters (?m ?s ?p ?h1 ?h2)
        :precondition
                (and (monkey ?m) (plafond ?p) (height ?p ?h1) (height ?m ?h1)
```

```
                ;;; le singe est au niveau du plafond
            (handempty) ;;; le singe a les mains libres
            (sol ?s) (height ?s ?h2))
        :effect
            (and (not(height ?m ?h1)) (height ?m ?h2)) ;;; le singe descend au niveau du sol
)

(:action attraper
        :parameters (?x ?m ?ba ?h)
        :precondition
            (and (monkey ?m) (at ?m ?x) (bananas ?ba) (at ?ba ?x)
                ;;; le singe est au même endroit que les bananes
            (height ?m ?h) (height ?ba ?h) ;;; le singe est au même niveau que les bananes
            (handempty)) ;;; le singe a les mains libres
        :effect
            (and (not(handempty)) (hold)) ;;; le singe a les mains prises
)

(:action lacher
        :parameters (?x ?m ?ba ?s ?p ?h1 ?h2)
        :precondition
            (and (monkey ?m) (at ?m ?x) ;;; le singe est en x
            (hold) ;;; le singe a les mains prises
            (sol ?s) (height ?s ?h1) (plafond ?p) (height ?p ?h2))
        :effect
            (and (not(hold)) (handempty) ;;; le singe a les mains libres
            (not(height ?ba ?h2)) (height ?ba ?h1)) ;;; les bananes tombent au niveau du sol
)
)
```

Avec ce domaine, le problème qu'on cherche à résoudre est le suivant :

```
(define (problem MONKEY-EX6)
(:domain MONKEY)
(:objects A B C Singe Boite Bananes Sol Plafond bas haut)
(:INIT
        (monkey Singe) (at Singe A) (height Singe bas)
        (box Boite) (at Boite B) (height Boite bas)
        (bananas Bananes) (at Bananes C) (height Bananes haut)
        (sol Sol) (height Sol bas)
        (plafond Plafond) (height Plafond haut)
        (handempty)
)
```

```
(:goal (AND (height Bananes bas) (hold)))
)
```

Le plan solution correspondant est de taille 7. Il est trouvé en 0.12s après 2 itérations :

```
0: (aller a b singe sol bas) [1]
1: (pousser b c singe boite sol bas) [1]
2: (monter c singe boite sol plafond bas haut) [1]
3: (attraper c singe bananes haut) [1]
4: (lacher c singe bananes sol plafond bas haut) [1]
5: (descendre singe sol plafond haut bas) [1]
6: (attraper c singe bananes bas) [1]
```

7 Exercice 7:

Le domaine peut être décrit comme suit :

```
(define (domain HANOI)
(:requirements :strips)
(:predicates
       (disque ?x)
       (pic ?x)
       (sur-disque ?x ?y)
       (sur-pic ?x)
       (rien-sur ?x)
       (main-vide)
       (dans-main ?x)
       (plus-grand ?x ?y)
)

(:action transitivite-tailles ;;; simplifie la description du monde
       :parameters (?gd ?md ?pd)
       :precondition
              (and (disque ?gd) (disque ?md) (disque ?pd)
              (plus-grand ?gd ?md) (plus-grand ?md ?gd))
                     ;;; si gd est plus grand que md, qui est plus grand que pd
       :effect
              (plus-grand ?gd ?pd) ;;; alors gd est plus grand que pd
)

(:action prendre-sur-pic
```

```
        :parameters (?d ?p)
        :precondition
                (and (disque ?d) (pic ?p)
                (sur-pic ?d ?p) (rien-sur ?d)
                (main-vide))
        :effect
                (and (not(sur-pic ?d ?p)) (rien-sur ?p)
                (not(main-vide)) (dans-main ?d))
)

(:action prendre-sur-disque
        :parameters (?d ?ssd)
        :precondition
                (and (disque ?d) (disque ?ssd)
                (sur-disque ?d ?ssd) (rien-sur ?d)
                (main-vide))
        :effect
                (and (not(sur-disque ?d ?ssd)) (rien-sur ?ssd)
                (not(main-vide)) (dans-main ?d))
)

(:action poser-sur-pic
        :parameters (?d ?p)
        :precondition
                (and (disque ?d) (pic ?p)
                (rien-sur ?p) (dans-main ?d))
        :effect
                (and (not(rien-sur ?p)) (sur-pic ?d ?p)
                (not(dans-main ?d)) (main-vide))
)

(:action poser-sur-disque
        :parameters (?d ?ssd)
        :precondition
                (and (disque ?d) (disque ?ssd)
                (rien-sur ?ssd) (dans-main ?d)
                (plus-grand ?ssd ?d))
        :effect
                (and (not(rien-sur ?ssd)) (sur-disque ?d ?ssd)
                (not(dans-main ?d)) (main-vide))
)
)
```

Avec ce domaine, le problème qu'on cherche à résoudre est le suivant (cas avec 3 disques) :

```
(define (problem HANOI-EX7)
(:domain HANOI)
(:objects Pic1 Pic2 Pic3 A B C D E)
(:INIT
       (disque A) (disque B) (disque C) (disque D) (disque E) (pic Pic1) (pic Pic2) (pic Pic3)
       (plus-grand E D) (plus-grand D C) (plus-grand C B) (plus-grand B A)
       (rien-sur A) (sur-disque A B) (sur-disque B C) (sur-pic C Pic1)
       (rien-sur Pic2) (rien-sur Pic3) (main-vide))
(:goal (AND (rien-sur A) (sur-disque A B) (sur-disque B C) (sur-pic C Pic3)))
)
```

Pour 1 tour, le plan solution est de taille 2. Il est trouvé en 0.01s après 1 itération :

```
0: (prendre-sur-pic a pic1) [1]
1: (poser-sur-pic a pic3) [1]
```

Pour 2 tours, le plan solution est de taille 6. Il est trouvé en 0.04s après 1 itération :

```
0: (prendre-sur-disque a b) [1]
1: (poser-sur-pic a pic2) [1]
2: (prendre-sur-pic b pic1) [1]
3: (poser-sur-pic b pic3) [1]
4: (prendre-sur-pic a pic2) [1]
5: (poser-sur-disque a b) [1]
```

Pour 3 tours, le plan solution est de taille 14. Il est trouvé en 0.07s après 5 itérations :

```
0: (prendre-sur-disque a b) [1]
1: (poser-sur-pic a pic3) [1]
2: (prendre-sur-disque b c) [1]
3: (poser-sur-pic b pic2) [1]
4: (prendre-sur-pic a pic3) [1]
5: (poser-sur-disque a b) [1]
6: (prendre-sur-pic c pic1) [1]
7: (poser-sur-pic c pic3) [1]
8: (prendre-sur-disque a b) [1]
9: (poser-sur-pic a pic1) [1]
10: (prendre-sur-pic b pic2) [1]
11: (poser-sur-disque b c) [1]
12: (prendre-sur-pic a pic1) [1]
13: (poser-sur-disque a b) [1]
```

Pour 4 tours, CPT ne trouve pas de plan solution en temps raisonnable :

```
domain file : .\domain-hanoi.pddl.txt
problem file : .\hanoi.pddl.txt
timer : 100000

Parsing domain..... done : 0.00
Parsing problem..... done : 0.01
domain : hanoi
problem : hanoi-ex7
Instantiating operators..... done : 0.00
Creating initial structures..... done : 0.00
Computing bound..... done : 0.00
Computing e-deleters..... done : 0.00
Finalizing e-deleters..... done : 0.00
Refreshing structures..... done : 0.01
Computing distances..... done : 0.00
Finalizing structures..... done : 0.00
Variables creation..... done : 0.00
Bad supporters..... done : 0.00
Distance boosting..... done : 0.00
Initial propagations..... done : 0.00

Problem : 32 actions, 30 fluents, 83 causals
         11 init facts, 5 goals

Bound : 14 --- Nodes : 0 --- Backtracks : 0 --- Iteration time : 0.00
Bound : 15 --- Nodes : 0 --- Backtracks : 0 --- Iteration time : 0.00
Bound : 16 --- Nodes : 1 --- Backtracks : 1 --- Iteration time : 0.00
Bound : 17 --- Nodes : 1 --- Backtracks : 1 --- Iteration time : 0.00
Bound : 18 --- Nodes : 4 --- Backtracks : 4 --- Iteration time : 0.00
Bound : 19 --- Nodes : 4 --- Backtracks : 4 --- Iteration time : 0.00
Bound : 20 --- Nodes : 6 --- Backtracks : 6 --- Iteration time : 0.00
Bound : 21 --- Nodes : 6 --- Backtracks : 6 --- Iteration time : 0.00
Bound : 22 --- Nodes : 49 --- Backtracks : 49 --- Iteration time : 0.01
Bound : 23 --- Nodes : 49 --- Backtracks : 49 --- Iteration time : 0.00
Bound : 24 --- Nodes : 97 --- Backtracks : 97 --- Iteration time : 0.01
Bound : 25 --- Nodes : 97 --- Backtracks : 97 --- Iteration time : 0.01
Bound : 26 --- Nodes : 177 --- Backtracks : 177 --- Iteration time : 0.02
Bound : 27 --- Nodes : 180 --- Backtracks : 180 --- Iteration time : 0.02
Bound : 28 --- Nodes : 472 --- Backtracks : 472 --- Iteration time : 0.05
Bound : 29 --- Nodes : 473 --- Backtracks : 473 --- Iteration time : 0.04
Bound : 30 --- Nodes : 1298 --- Backtracks : 1298 --- Iteration time : 0.14
```

```
Bound : 31 --- Nodes : 1302 --- Backtracks : 1302 --- Iteration time : 0.14
Bound : 32 --- Nodes : 3421 --- Backtracks : 3421 --- Iteration time : 0.36
Bound : 33 --- Nodes : 3427 --- Backtracks : 3427 --- Iteration time : 0.35
Bound : 34 --- Nodes : 4491 --- Backtracks : 4491 --- Iteration time : 0.51
Bound : 35 --- Nodes : 4493 --- Backtracks : 4493 --- Iteration time : 0.51
Bound : 36 --- Nodes : 9539 --- Backtracks : 9539 --- Iteration time : 1.17
Bound : 37 --- Nodes : 9590 --- Backtracks : 9590 --- Iteration time : 1.15
Bound : 38 --- Nodes : 18295 --- Backtracks : 18295 --- Iteration time : 2.34
Bound : 39 --- Nodes : 18315 --- Backtracks : 18315 --- Iteration time : 2.34
Bound : 40 --- Nodes : 47736 --- Backtracks : 47736 --- Iteration time : 6.54
Bound : 41 --- Nodes : 50420 --- Backtracks : 50420 --- Iteration time : 6.95
Bound : 42 --- Nodes : 139472 --- Backtracks : 139472 --- Iteration time : 20.67
Bound : 43 --- Nodes : 138272 --- Backtracks : 138272 --- Iteration time : 21.34
Bound : 44 --- Nodes : 286734 --- Backtracks : 286734 --- Iteration time : 47.26
Bound : 45 --- Nodes : 301520 --- Backtracks : 301520 --- Iteration time : 49.71
Bound : 46 --- Nodes : 642873 --- Backtracks : 642873 --- Iteration time : 112.13
Bound : 47 --- Nodes : 637020 --- Backtracks : 637020 --- Iteration time : 110.99
Bound : 48 --- Nodes : 1493222 --- Backtracks : 1493222 --- Iteration time : 273.78
Bound : 49 --- Nodes : 1494983 --- Backtracks : 1494983 --- Iteration time : 272.86
Bound : 50 --- Nodes : 2583596 --- Backtracks : 2583596 --- Iteration time : 486.98
Bound : 51 --- Nodes : 2645879 --- Backtracks : 2645879 --- Iteration time : 494.16
Bound : 52 ---
```

CPT n'en trouve pas non plus pour 5 tours.

Chaque appel de la fonction décrite en pseudo-code fait 2 appels à elle-même avec une taille décrémentée de 1. Puisque le cas de base est en $\Theta(1)$, cette fonction a une complexité en $\Theta(2^n)$ (de complexité exacte $2^n - 1$).

Les solutions obtenues avec l'algorithme en pseudo-code et celles calculées avec CPT sont bien de même taille (en considérant que chaque coup de l'algorithme correspond en réalité à une prise et une dépose, soit 2 coups avec notre modèle). Sachant que l'algorithme présenté est optimal, et que le planificateur CPT retourne toujours le plus court plan solution s'il le trouve, il est logique que les 2 solutions soient de même taille à chaque fois (au facteur 2 mentionné près).

La principale différence entre la résolution du problème des tours de Hanoï avec cette fonction et avec un planificateur d'actions comme CPT est le temps de calcul de la solution. En effet, la fonction connaît déjà la démarche à suivre afin de résoudre le problème, tandis que CPT effectue une recherche dans l'espace des possibles sans savoir au préalable quelle direction prendre.