INSTITUT POLYTECHNIQUE DE PARIS
ENSTA PARIS

CSC_5RO16_TA, Planification et Contrôle

TP5, Planning Domain Definition Language

by
Guilherme NUNES TROFINO

supervised by
Philippe MORIGNOT
David FILLIAT

Confidentiality Notice
Non-confidential and publishable report

ROBOTIQUE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET COMMUNICATION

Paris, FR
20 janvier 2025

Table des matières

1. Question 1

1.1. Opérateurs

Ces opérateurs décrivent toutes les manipulations possibles dans un monde de cubes tout en respectant les contraintes physiques (e.g. un bloc doit être clair pour être manipulé).

1.1.1. `pick-up`

Résolution. Permet de saisir un bloc `?x` qui est clair, sur la table, et lorsque la main du robot est vide. Après cette action, le bloc est tenu par le robot, n'est plus sur la table, et n'est plus clair.

1.1.2. `put-down`

Résolution. Permet de déposer un bloc `?x` tenu par le robot sur la table. Après cette action, le bloc est sur la table, devient clair, et la main du robot redevient vide.

1.1.3. `stack`

Résolution. Permet de placer un bloc `?x` tenu par le robot sur un autre bloc `?y` qui est clair. Après cette action, `?x` est sur `?y`, `?y` n'est plus clair, et la main du robot est vide.

1.1.4. `unstack`

Résolution. Permet de retirer un bloc `?x` situé sur un bloc `?y`, si `?x` est clair et que la main du robot est vide. Après cette action, le bloc `?x` est tenu, `?y` devient clair, et `?x` n'est plus sur `?y`.

1.2. `put-down` vs `stack`

L'opérateur put-down permet de déposer un bloc sur la table, sans interagir avec d'autres blocs. L'opérateur stack permet de placer un bloc sur un autre bloc, impliquant une interaction directe entre les blocs, le bloc cible doit être clair.

Ces deux cas représentent des actions fondamentalement différentes dans le monde des blocs. En dissociant ces opérateurs, on distingue les contraintes liées à chaque cas :

1. put-down n'impose pas de vérifier la clarté d'un autre bloc, car la table est toujours disponible.
2. stack nécessite que le bloc cible soit clair, ajoutant une contrainte supplémentaire.

Cette distinction garantit une représentation précise des contraintes physiques et évite des ambiguïtés dans les plans générés.

1.3. `holding`

Le fluent `(holding ?x)` indique que le robot tient actuellement le bloc `?x`. Il est essentiel pour modéliser l'état de la main du robot et garantir la cohérence des actions.

Exemple 1.1. Une action comme put-down ou stack ne peut être réalisée que si le robot tient déjà un bloc. Cela évite qu'un bloc soit placé sans qu'il ait été préalablement saisi.

Si `(holding ?x)` n'existait pas, il faudrait redéfinir les opérateurs pour inclure des préconditions ou effets complexes afin de suivre indirectement l'état de la main du robot.

Exemple 1.2. Ajouter une variable implicite ou un état global décrivant le contenu de la main. Multiplier les conditions liées aux transitions entre l'état vide et occupé de la main.

En somme, l'absence de `(holding ?x)` rendrait la représentation moins intuitive et les opérateurs plus difficiles à interpréter.

2. Question 2

Afin de pourvoir explorer des algorithmes de **PDDL**, Planning Domain Definition Language, le commande suivant a été lancé sur un terminale Windows PowerShell :

```
.\cpt.exe -o .\domain-blocksaips.pddl -f .\blocksaips01.pddl
```

Listing 1 : Lancement `cpt.exe` pour `blocksaips01.pddl`

Qui a retourné comme résultat le suivant :

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

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 : 0.02
```

Listing 2 : Résultat `cpt.exe` pour `blocksaips01.pddl`

2.1. Plan-Solution

2.1.1. Longueur

Résolution. La longueur du plan-solution est de 6 actions, correspondant à une séquence minimale permettant d'atteindre l'objectif

```
Length : 6
```

2.1.2. Temps d'Exécution

Résolution. Le plan-solution a été trouvé en 0,00 seconde de temps de recherche, avec un temps total d'exécution y compris le parsing et les calculs initiaux de 0,02 seconde.

```
Search time : 0.00
Total time : 0.02
```

2.1.3. Itérations

Résolution. Le planificateur a effectué 1 itérations pour trouver la solution. Cela indique qu'aucune exploration ou backtracking n'a été nécessaire.

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

2.1.4. Durée d'Action

Résolution. Dans ce cas, chaque action est considérée comme prenant 1 unité de temps, ce qui est arbitraire et dépend de la manière dont le domaine est modélisé.

Le `Makespan` total est donc de 6 unités de temps, correspondant aux 6 actions du plan.

```
...
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
...
```

2.2. Longueur Solution

Résolution. Des plans plus longs pourraient exister si des actions supplémentaires ou inutiles étaient insérées.

Exemple 2.1. Ramener un bloc sur la table avant de le replacer sur une autre position :

```
(pick-up b) (put-down b) (pick-up b) (stack b a)
```

Changer l'ordre des blocs temporairement avant de revenir à l'ordre final.

Ces plans ne sont pas proposés car le planificateur optimise pour trouver une solution minimale en termes de longueur ou de `Makespan`. Cela est dû à l'utilisation d'heuristiques qui privilégient les solutions les plus efficaces en respectant les contraintes du domaine. Fournir des plans plus longs irait à l'encontre de cette logique d'optimalité.

Le planificateur génère un plan optimal en minimisant la longueur et le temps. Les plans plus longs, bien que possibles, ne sont pas proposés car ils n'ajoutent aucune valeur à la résolution du problème et ne respectent pas les objectifs d'efficacité.

3. Question 3

3.1. Algorithme

```
(define (problem BLOCKS-4-1)
    (:domain BLOCKS)
    (:objects A B C D)
    (:init (on B C) (on C A) (on A D) (ontable D)
        (clear B) (handempty))
    (:goal (and (on D C) (on C A) (on A B)))
)
```

Listing 3 : Algorithme `blocksaips02.pddl`

3.2. Analyse

```
.\cpt.exe -o .\domain-blocksaips.pddl -f .\blocksaips02.pddl
```

Listing 4 : Lancement `cpt.exe` pour `blocksaips02.pddl`

```
domain file : .\domain-blocksaips.pddl
problem file : .\blocksaips02.pddl

Parsing domain..... done : 0.00
Parsing problem..... done : 0.00
domain : blocks
problem : blocks-4-1
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
          6 init facts, 3 goals

Bound : 10  ---  Nodes : 0  ---  Backtracks : 0  ---  Iteration time : 0.00

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

Makespan : 10
Length : 10
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 : 0.02
```

Listing 5 : Résultat `cpt.exe` pour `blocksaips02.pddl`

3.2.1. Longueur

Résolution. La longueur du plan-solution est de 10 actions, correspondant à une séquence minimale permettant d'atteindre l'objectif

```
Length : 10
```

3.2.2. Temps d'Exécution

Résolution. Le plan-solution a été trouvé en 0,00 seconde de temps de recherche, avec un temps total d'exécution y compris le parsing et les calculs initiaux de 0,02 seconde.

```
Search time : 0.00
Total time : 0.02
```

3.2.3. Itérations

Résolution. Le planificateur a effectué 1 itérations pour trouver la solution. Cela indique qu'aucune exploration ou backtracking n'a été nécessaire.

```
Bound : 10  ---  Nodes : 0  ---  Backtracks : 0  ---  Iteration time : 0.00
```

4. Question 4

4.1. Algorithme

```
(define (problem BLOCKS-MULTI-TOWERS)
    (:domain BLOCKS)
    (:objects A B C D E F G H I J)
    (:init
        ;; Tour 1
        (on C G) (on G E) (on E I) (on I J) (on J A) (on A B) (ontable B)
        ;; Tour 2
        (on F D) (on D H) (ontable H)
        ;; Propriétés initiales
        (clear C) (clear F) (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)
        )
    )
)
```

Listing 6 : Algorithme `blocksaips03.pddl`

4.2. Analyse

```
.\cpt.exe -o .\domain-blocksaips.pddl -f .\blocksaips03.pddl
```

Listing 7 : Lancement `cpt.exe` pour `blocksaips03.pddl`

```
domain file : .\domain-blocksaips.pddl
problem file : .\blocksaips03.pddl

Parsing domain..... done : 0.00
Parsing problem..... done : 0.00
domain : blocks
problem : blocks-multi-towers
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 : 202 actions, 121 fluents, 499 causals
          13 init facts, 9 goals

Bound : 26  ---  Nodes : 0  ---  Backtracks : 0  ---  Iteration time : 0.00
Bound : 27  ---  Nodes : 0  ---  Backtracks : 0  ---  Iteration time : 0.00
Bound : 28  ---  Nodes : 29  ---  Backtracks : 29  ---  Iteration time : 0.01
Bound : 29  ---  Nodes : 36  ---  Backtracks : 36  ---  Iteration time : 0.01
Bound : 30  ---  Nodes : 782  ---  Backtracks : 782  ---  Iteration time : 0.24
Bound : 31  ---  Nodes : 787  ---  Backtracks : 787  ---  Iteration time : 0.23
Bound : 32  ---  Nodes : 166  ---  Backtracks : 129  ---  Iteration time : 0.04

0: (unstack c g) [1]
1: (put-down c) [1]
2: (unstack g e) [1]
3: (put-down g) [1]
4: (unstack e i) [1]
5: (put-down e) [1]
6: (unstack i j) [1]
7: (put-down i) [1]
8: (unstack j a) [1]
9: (put-down j) [1]
```

```
    10: (pick-up g) [1]
    11: (stack g j) [1]
    12: (unstack f d) [1]
    13: (put-down f) [1]
    14: (unstack d h) [1]
    15: (put-down d) [1]
    16: (pick-up h) [1]
    17: (stack h g) [1]
    18: (pick-up e) [1]
    19: (stack e h) [1]
    20: (unstack a b) [1]
    21: (stack a e) [1]
    22: (pick-up i) [1]
    23: (stack i a) [1]
    24: (pick-up f) [1]
    25: (stack f i) [1]
    26: (pick-up d) [1]
    27: (stack d f) [1]
    28: (pick-up b) [1]
    29: (stack b d) [1]
    30: (pick-up c) [1]
    31: (stack c b) [1]

    Makespan : 32
    Length : 32
    Nodes : 1800
    Backtracks : 1763
    Support choices : 510
    Conflict choices : 1290
    Mutex choices : 0
    Start time choices : 0
    World size : 300K
    Nodes/sec : 3333.33
    Search time : 0.54
    Total time : 0.56
```

Listing 8 : Résultat `cpt.exe` pour `blocksaips03.pddl`

4.2.1. Longueur

Résolution. La longueur du plan-solution est de 32 actions, correspondant à une séquence minimale permettant d'atteindre l'objectif

```
    Length : 32
```

4.2.2. Temps d'Exécution

Résolution. Le plan-solution a été trouvé en 0,54 seconde de temps de recherche, avec un temps total d'exécution y compris le parsing et les calculs initiaux de 0,56 seconde.

```
    Search time : 0.54
    Total time : 0.56
```

4.2.3. Itérations

Résolution. Le planificateur a effectué 7 itérations pour trouver la solution. Cela indique qu'aucune exploration ou backtracking n'a été nécessaire.

```
    Bound : 26  ---  Nodes : 0  ---  Backtracks : 0  ---  Iteration time : 0.00
    Bound : 27  ---  Nodes : 0  ---  Backtracks : 0  ---  Iteration time : 0.00
    Bound : 28  ---  Nodes : 29  ---  Backtracks : 29  ---  Iteration time : 0.01
    Bound : 29  ---  Nodes : 36  ---  Backtracks : 36  ---  Iteration time : 0.01
    Bound : 30  ---  Nodes : 782  ---  Backtracks : 782  ---  Iteration time : 0.24
    Bound : 31  ---  Nodes : 787  ---  Backtracks : 787  ---  Iteration time : 0.23
    Bound : 32  ---  Nodes : 166  ---  Backtracks : 129  ---  Iteration time : 0.04
```

5. Question 5

5.1. Algorithme

5.1.1. `move`

Voici un exemple d'opérateur pour un domaine PDDL permettant à l'agent de se déplacer d'un nœud from vers un nœud to si un arc existe entre ces deux nœuds :

```
    (:action move
        :parameters (?from ?to)
        :precondition (and (agent-at ?from) (arc ?from ?to))
        :effect (and
            (not (agent-at ?from))
            (agent-at ?to)
        )
    )
```

Listing 9 : Algorithme `move.pddl`

5.1.2. `domain-graphs`

```
    (define (domain GRAPH)
        (:requirements :strips)
        (:predicates
            (agent-at ?node)
            (arc ?from ?to)
        )
        (:action move
            :parameters (?from ?to)
            :precondition (and (agent-at ?from) (arc ?from ?to))
            :effect (and
                (not (agent-at ?from))
                (agent-at ?to)
            )
        )
  )
```

Listing 10 : Algorithme `domain-graphs.pddl`

5.1.3. `blocksaips05`

```
    (define (problem SMALL-GRAPH)
        (:domain GRAPH)
        (:objects A B C D E)
        (:init
            (agent-at A)

            (arc A B)
            (arc B C)
            (arc C D)
            (arc D E)
        )
        (:goal (agent-at E))
    )
```

Listing 11 : Algorithme `blocksaips05.pddl`

5.2. Analyse

5.2.1. Avantages

Simplicité : La modélisation sous forme d'actions PDDL est intuitive pour les graphes orientés. Optimalité : Avec un bon planificateur, le chemin calculé est souvent minimal en termes de longueur.

5.2.2. incovénients

Échelle : Pour des graphes de grande taille, cette méthode devient inefficace car les planificateurs doivent explorer un espace d'état potentiellement exponentiel. Dépendance au planificateur : L'efficacité dépend fortement de l'algorithme sous-jacent (heuristiques, recherche en largeur/profondeur, etc.). Pas de gestion de coût : Si les arcs ont des poids, cette méthode nécessite des adaptations pour inclure une minimisation des coûts. Conclusion

La méthode est efficace pour des petits graphes avec des objectifs simples, mais elle est peu adaptée aux grands graphes ou aux problèmes avec des coûts d'arcs. Dans ces cas, des algorithmes spécialisés comme Dijkstra ou A* sont plus performants.

6. Question 6

(:init (singe-a A) ; ; Le singe est en A (caisse-a B) ; ; La caisse est en B (bananes-a C) ; ; Les bananes sont en C (hauteur-singe Bas) ; ; Le singe est à hauteur Bas (vide) ; ; Le singe n'a rien en main)

(:goal (and (singe-a C) ; ; Le singe est en C (caisse-a C) ; ; La caisse est en C (hauteur-singe Haut) ; ; Le singe est monté sur la caisse (attrape bananes) ; ; Le singe a attrapé les bananes))

(:action aller :parameters (?from ?to) :precondition (singe-a ?from) :effect (and (not (singe-a ?from)) (singe-a ?to)))

(:action pousser :parameters (?objet ?from ?to) :precondition (and (singe-a ?from) (caisse-a ?from)) :effect (and (not (caisse-a ?from)) (caisse-a ?to) (not (singe-a ?from)) (singe-a ?to)))

(:action monter :parameters (?objet ?lieu) :precondition (and (singe-a ?lieu) (caisse-a ?lieu) (hauteur-singe Bas)) :effect (and (not (hauteur-singe Bas)) (hauteur-singe Haut)))

(:action descendre :parameters (?objet ?lieu) :precondition (and (singe-a ?lieu) (caisse-a ?lieu) (hauteur-singe Haut)) :effect (and (not (hauteur-singe Haut)) (hauteur-singe Bas)))

(:action attraper :parameters (?objet ?lieu) :precondition (and (singe-a ?lieu) (bananes-a ?lieu) (hauteur-singe Haut) (vide)) :effect (and (not (vide)) (attrape ?objet)))

(:action lacher :parameters (?objet ?lieu) :precondition (and (attrape ?objet) (singe-a ?lieu)) :effect (and (not (attrape ?objet)) (vide)))

Si l'objet à pousser est trop lourd, l'opérateur "pousser" n'aurait pas d'effet, car la précondition n'est pas satisfaite.

Type de problème identifié : Qualification Ceci est un exemple du problème de la qualification. En effet, les préconditions du modèle ne spécifient pas toutes les conditions nécessaires pour que l'opérateur réussisse (comme le poids de l'objet ou la force du singe). Ajouter de telles conditions rendrait le modèle très complexe, et certaines limitations doivent être implicitement comprises par l'utilisateur du modèle.

7. Question 7

(define (problem hanoi-3disques) (:domain hanoi) (:objects disk1 disk2 disk3 - disk ; ; Les disques de plus petit (disk1) au plus grand (disk3) peg1 peg2 peg3 - peg ; ; Les trois pics : départ, intermédiaire, arrivée) (:init (on disk1 disk2) ; ; disk1 sur disk2 (on disk2 disk3) ; ; disk2 sur disk3 (on disk3 peg1) ; ; disk3 sur le pic1 (base) (clear disk1) ; ; disk1 est libre (clear peg2) ; ; pic2 vide (clear peg3) ; ; pic3 vide (hand-empty) ; ; La pince est libre (smaller disk1 disk2) ; ; Les relations d'ordre entre disques (smaller disk2 disk3)) (:goal (and (on disk1 disk2) ; ; Recréer la tour sur peg3 (on disk2 disk3) (on disk3 peg3))))

(:action prendre :parameters (?disk ?peg) :precondition (and (on ?disk ?peg) (clear ?disk) (hand-empty)) :effect (and (holding ?disk) (clear ?peg) (not (on ?disk ?peg)) (not (hand-empty))))

(:action poser-sur-pic-vide :parameters (?disk ?peg) :precondition (and (holding ?disk) (clear ?peg)) :effect (and (on ?disk ?peg) (hand-empty) (clear ?disk) (not (holding ?disk))))

(:action poser-sur-disque :parameters (?disk1 ?disk2) :precondition (and (holding ?disk1) (clear ?disk2) (smaller ?disk1 ?disk2)) :effect (and (on ?disk1 ?disk2) (hand-empty) (clear ?disk1) (not (holding ?disk1))))

(:action prendre-sur-disque :parameters (?disk1 ?disk2) :precondition (and (on ?disk1 ?disk2) (clear ?disk1) (hand-empty)) :effect (and (holding ?disk1) (clear ?disk2) (not (on ?disk1 ?disk2)) (not (hand-empty))))

PROCEDURE Hanoi(NDisques, PicDepart, PicIntermediaire, PicArrivee) SI NDisques différent de 0 ALORS Hanoi(NDisques – 1, PicDepart, PicArrivee, PicIntermediaire) Afficher("Déplacer le disque numéro " + NDisques + " de " + PicDepart + " à " + PicArrivee) Hanoi(NDisques – 1, PicIntermediaire, PicDepart, PicArrivee) FINSI FINPROCEDURE

Complexité algorithmique Nombre de mouvements : Correspondance avec CPT Les résultats de CPT pour 1 à 4 disques sont cohérents avec cette fonction. Cependant, pour 5 disques ou plus, CPT peut nécessiter plus de temps à cause de la taille exponentielle de l'espace d'état.

Différence entre fonction récursive et CPT Fonction récursive : Exploite une structure mathématique spécifique au problème et génère directement la solution optimale. Planificateur CPT : Explore un espace d'états plus général en appliquant des opérateurs et peut nécessiter plus d'itérations pour identifier la solution optimale. Conclusion CPT est généraliste, mais moins efficace pour des problèmes hautement structurés comme les tours de Hanoi. La fonction récursive est beaucoup plus efficace ici grâce à une résolution ciblée et préconçue, mais elle manque de flexibilité pour d'autres types de problèmes.