Fil rouge - Block World

CODEN Erwann 22001714
LEMARTINEL Thomas 21905036

2023

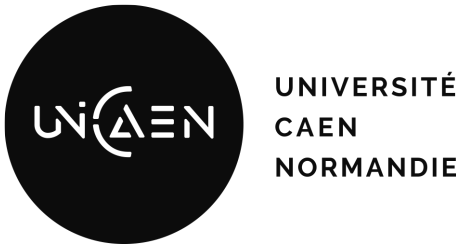

Table des matières

1 Introduction

Ce document présente un compte rendu détaillé du projet Fil Rouge, un travail de synthèse intégrant divers aspects de l'intelligence artificielle étudiés dans nos travaux pratiques (TP). Le projet s'articule autour de plusieurs axes majeurs, incluant la représentation des données, la gestion des contraintes, la planification, et l'extraction de connaissances. Chacune de ces parties s'inspire et fait référence à un TP spécifique, permettant ainsi une application concrète des concepts théoriques abordés en cours.

L'objectif de ce compte rendu est de mettre en lumière les méthodes et stratégies développées au cours du projet, en démontrant comment les enseignements tirés de chaque TP ont été appliqués pour résoudre des problèmes spécifiques et réaliser les objectifs du projet.

2 Représentation des Données - MainConstraintes.java

Dans le cadre de notre projet fil rouge, nous nous sommes concentrés sur la représentation des données dans le contexte du monde des blocs. Ce modèle consiste en un ensemble de blocs numérotés, organisés en piles, où l'on peut déplacer un bloc à la fois.

La représentation des configurations dans ce monde est cruciale pour le traitement des problèmes de satisfaction de contraintes et de planification. Nous avons utilisé trois ensembles de variables pour représenter les configurations de n blocs et m piles :

- **Variables de positionnement (*onb*) :** Pour chaque bloc b, une variable Onb indique la position du bloc, soit sur un autre bloc, soit sur une pile spécifique. Cette approche permet de décrire la localisation de chaque bloc.
- **Variables booléennes pour les blocs indéplaçables (*fixedb*) :** Chaque bloc b est associé à une variable booléenne Fib, qui prend la valeur *true* si le bloc est indéplaçable, c'est-à-dire s'il y a un autre bloc posé dessus et *false* si le bloc peut être déplacé.
- **Variables booléennes pour les piles libres (*freep*) :** Pour chaque pile p, une variable booléenne Frp indique si la pile est libre, c'est-à-dire sans aucun bloc dessus.

Les contraintes binaires appliquées à ces variables assurent la validité des configurations. Par exemple, aucun bloc ne peut être positionné sur lui-même, et deux blocs différents ne peuvent pas occuper la même position. De plus, si un bloc est sur un autre, ce dernier est considéré comme indéplaçable.

3 Planification - MainPlanner.java

La planification dans le cadre du projet BlocksWorld est abordée en définissant des états initiaux, des ensembles d'actions, et des buts à atteindre. Chaque action et but est représenté par des classes spécifiques, implémentant des interfaces définies pour ce projet.

3.1 Actions - BWActions.java

Les actions possibles consistent à déplacer un bloc d'une position à une autre, à condition que le bloc déplacé et la position cible soient disponibles. Ces actions sont classées en quatre types principaux :

1. Déplacer un bloc du dessus d'un autre bloc vers le dessus d'un troisième bloc.
2. Déplacer un bloc du dessus d'un autre bloc vers une pile vide.
3. Déplacer un bloc du dessous d'une pile vers le dessus d'un autre bloc.
4. Déplacer un bloc du dessous d'une pile vers une autre pile vide.

3.1.1 Préconditions

Les actions de planification sont soumises à des contraintes spécifiques, garantissant la validité des mouvements de blocs. Chaque action nécessite la vérification de trois conditions préalables :

— Le bloc concerné doit être à la position correspondant à l'action envisagée.
— Le bloc doit être déplaçable, c'est-à-dire qu'il ne doit pas avoir d'autres blocs posés dessus (fixed doit être à false).
— La position cible doit être disponible pour recevoir le bloc, ce qui signifie que le bloc en dessous de la position d'arrivée ne doit pas être fixé.

3.1.2 Effets des Actions

Les effets des actions de planification modifient l'état des blocs et des piles selon les règles suivantes :

— La nouvelle position du bloc déplacé est mise à jour (Onb change de valeur).
— Le statut de la position initiale du bloc est ajusté (le bloc qui était en dessous devient non fixé).
— Le statut de la position cible est également mis à jour (le nouveau bloc en dessous du bloc bougé est fixé).

3.2 Planificateurs

Nous avons créé différents types de planificateurs qui utilisent un état initial, un ensemble d'actions, et un but pour trouver une séquence d'actions menant à la réalisation du but.

3.2.1 Expérimentations

Nous avons testé les différents algorithmes de planification dans notre classe `MainPlanner.java`. D'après ces tests, nous avons observé que chaque planificateur présente des avantages et des inconvénients distincts en fonction de la complexité des scénarios du monde des blocs.

DFSPlanner Le `DFSPlanner`, utilisant la recherche en profondeur, a montré une efficacité remarquable dans les scénarios où la solution est profondément enfouie dans l'arbre de recherche. Cependant, dans les situations où de nombreuses actions non productives sont possibles, ce planificateur peut s'engager dans des chemins très longs avant de trouver une solution ou de conclure qu'aucune solution n'est possible.

BFSPlanner Le `BFSPlanner`, basé sur la recherche en largeur, s'est avéré plus efficace pour identifier rapidement les solutions moins profondes. Toutefois, son inconvénient majeur réside dans sa consommation de mémoire, surtout dans les scénarios avec un grand nombre d'états intermédiaires, ce qui limite son applicabilité dans les cas de grande envergure.

DijkstraPlanner et AStarPlanner Les `DijkstraPlanner` et `AStarPlanner`, axés sur l'optimisation des coûts, ont offert des performances supérieures en termes d'efficacité et de rapidité, surtout dans les cas où les coûts des actions varient. Le `AStarPlanner`, avec son heuristique, a particulièrement bien performé en réduisant considérablement l'espace de recherche. Cependant, la qualité de la solution trouvée dépend fortement de la pertinence de l'heuristique choisie.

3.2.2 Conclusion des Tests

En conclusion, nos expérimentations ont démontré qu'il n'existe pas de planificateur universel optimal pour tous les scénarios. Le choix du planificateur le plus adapté dépend de beaucoup de paramètres tel que la nature spécifique du problème de planification, incluant le nombre d'états, la profondeur de la solution dans l'arbre de recherche, et la variabilité des coûts des actions.

4 Extraction de Connaissances - MainDatamining.java

L'extraction de connaissances visait à découvrir des motifs et des règles d'association significatifs à partir d'un corpus de 10 000 états d'un monde des blocs avec le même nombre de piles. La propositionnalisation des variables, une technique cruciale dans ce processus, a été utilisée pour transformer les états complexes du monde des blocs en ensembles de variables booléennes simples.

4.1 Nouvelles variables booléennes

Pour simplifier l'utilisation des algorithmes, on traduira ces variables de bases en variables booléennes. La "propositionnalisation" des variables a été réalisée dans la classe BWDataBoolVars.java comme suit :

- Pour chaque couple de blocs différents {b, b'}, une variable booléenne *On_bb'* indiquant si le bloc b est directement sur le bloc b'.
- Pour chaque bloc b et chaque pile p, une variable *OnTable_bp* déterminant si le bloc b est sur la table dans la pile p.
- Les variables *Fi_b* et *Fr_p*, reprises de la partie représentation des données, indiquant respectivement si un bloc est indéplaçable et si une pile est libre.

La phase d'extraction de connaissances à partir de la base de données booléenne générée a révélé des motifs et des règles d'association intéressants dans le contexte du monde des blocs.

4.2 Motifs et Règles d'Association

L'algorithme de fouille d'association a exploré l'ensemble des combinaisons d'items dans la base de données, révélant des motifs récurrents du monde des blocs. Les règles d'association extraites fournissent des indications sur les influences et les interactions entre différentes variables.

4.2.1 Fréquence et Confiance

Les règles d'association sont caractérisées par leur fréquence et leur confiance. La fréquence indique la proportion d'occurrences de la règle dans la base de données, tandis que la confiance mesure la probabilité conditionnelle que la conclusion de la règle se produise compte tenu des prémisses.

Exemple de Règle d'Association :

```
AssociationRule [premise=[Variable [domain=[false, true], name=On1_2],
Variable [domain=[false, true], name=Fi1],
Variable [domain=[false, true], name=On0_1]]
conclusion=[Variable [domain=[false, true], name=Fi0]]
frequency=0.70886
```

`confidence=1.0]`

D'après ce message qu'on affiche en console, on peut dire que :

- Le bloc 2 est sur le bloc 1
- Le bloc 1 est fixé
- Le bloc 1 est sur le bloc 0

Et si on a cette configuration, ce qui arrive 71% des cas, le bloc 0 est fixé avec une certitude de 100%.

4.3 Bilan

Au total, 26 règles d'association ont été extraites mais pour ce nombre de blocs on obtient en général entre 25 et 30 règles, fournissant une vision approfondie des relations entre les variables booléennes du monde des blocs. Ces résultats démontrent la capacité de la fouille d'association à révéler des structures significatives à partir de données complexes.

5 Conclusion

Ce projet nous a permis de comprendre en profondeur l'application des concepts d'intelligence artificielle comme la planification, la satisfaction de contraintes, et l'extraction de connaissances dans un contexte pratique. Les défis rencontrés et les solutions apportées ont enrichi notre compréhension théorique et pratique de ces concepts.