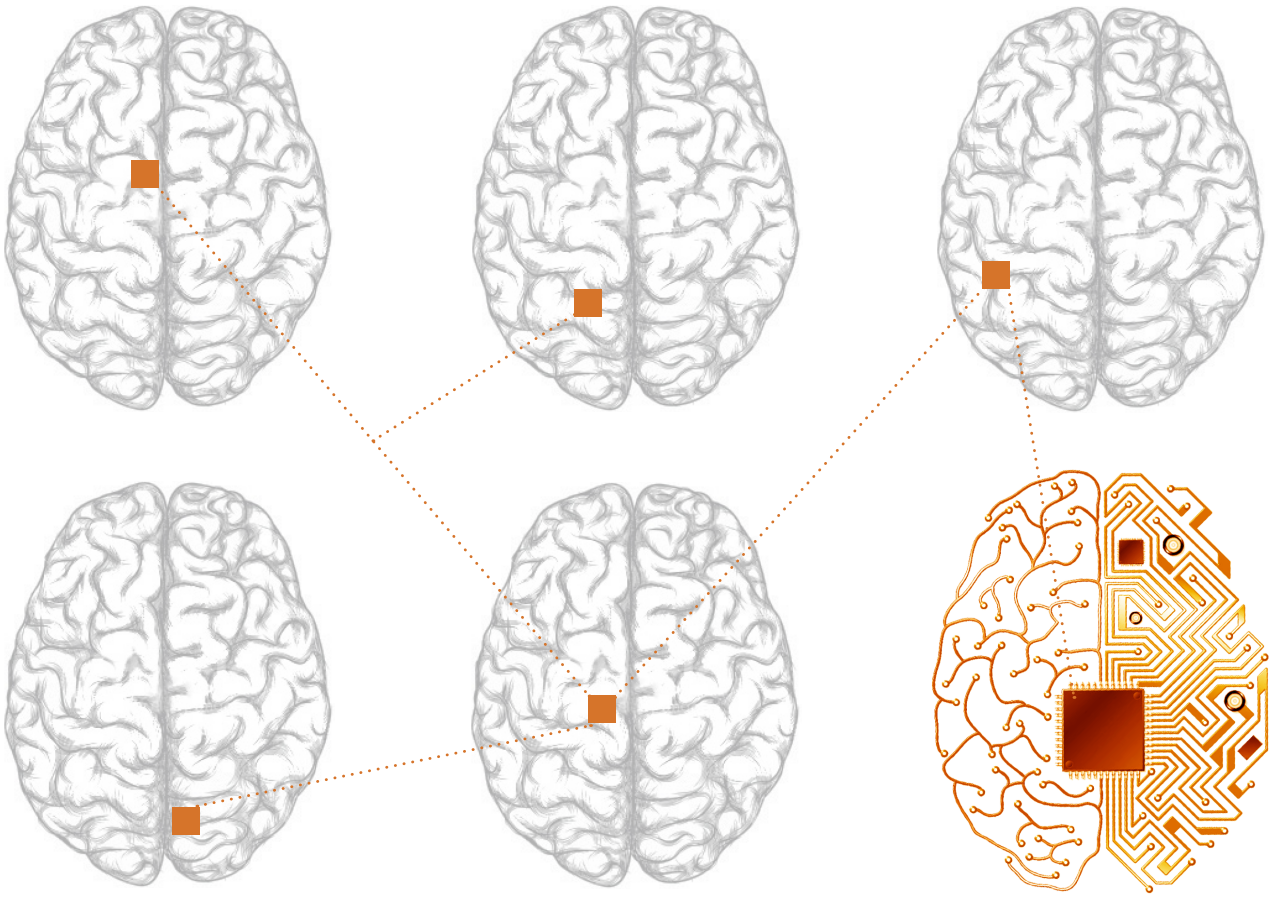
Actes des 10es Journées Francophones Planification, Décision, Apprentissage pour la conduite des systèmes

Frédéric Maris



PFIA 2015

Plate-forme Intelligence Artificielle

Rennes

Actes JFPDA

Président CP : Frédéric Maris

Inria
INVENTEURS DU MONDE NUMÉRIQUE

AfIA
Association française
pour l'Intelligence Artificielle

10es Journées Francophones Planification, Décision, Apprentissage

pour la conduite des systèmes

Présentation de la conférence

Les Journées Francophones sur la Planification, la Décision et l'Apprentissage pour la conduite de systèmes (JFPDA) ont pour but de rassembler la communauté de chercheurs francophones travaillant sur les problèmes d'intelligence artificielle, d'apprentissage par renforcement, de programmation dynamique et plus généralement dans les domaines liés à la prise de décision séquentielle sous incertitude et à la planification. Les travaux présentés traitent aussi bien d'aspects purement théoriques que de l'application de ces méthodes à la conduite de systèmes virtuels (jeux, simulateurs) et réels (robots, drones). Ces journées sont aussi l'occasion de présenter des travaux en cours de la part de doctorants, postdoctorants et chercheurs confirmés dans un cadre laissant une large place à la discussion constructive et bienveillante. Après Toulouse (2006), Grenoble (2007), Metz (2008), Paris (2009), Besançon (2010), Rouen (2011), Nancy (2012), Lille (2013) et Liège (2014) les journées se tiendront en 2015 à Rennes dans le cadre de la Plate-forme de l'Intelligence Artificielle. Thèmes principaux (liste non exhaustive)

- Processus décisionnels de Markov, totalement ou partiellement observables, factorisés ou hiérarchiques, centralisés ou décentralisés
- Programmation Dynamique approchée (ADP), apprentissage par renforcement (RL) :
  - RL Bayésien, RL inverse, RL batch, RL multi-agents, RL multi-objectifs
  - Convergence et bornes sur les performances des algorithmes RL/ADP
  - Complexité en RL/ADP
  - Apprentissage statistique, bornes PAC en RL/ADP
  - Méthodes de Monte Carlo et quasi Monte Carlo
  - Recherche directe de politiques, architectures acteur-critique
  - Apprentissage de fonctions de valeurs, généralisation, représentations parcimonieuses, méthodes à base de noyaux en RL/ADP
- Planification :
  - Planification classique, planification temporelle
  - Planification multi-agent, planification épistémique, conforme ou contingente
  - Complexité, classes traitables et transformations de problèmes de planification
  - Génération et exécution de plans flexibles
  - Replanification, planification en ligne
- Contrôle de systèmes continus ou discrets, réels ou simulés, mono ou multi-agents
- Approches d'inspiration biologique
- Applications et confrontations au monde réel

Conférencier Invité : Hector Geffner (ICREA & Universitat Pompeu Fabra)

Biographie

Après une thèse à UCLA et des recherches à l'IBM T.J. Watson Research Center puis à l'Université Simon Bolivar, Caracas (Vénézuela), Hector Geffner est maintenant chercheur à l'Université Pompeu Fabra, Barcelona (Espagne). H. Geffner s'intéresse aux modèles informatiques de raisonnement pour l'action, la planification, l'apprentissage et l'interaction sociale. Il est titulaire des AAAI et ECCAI fellows et il est éditeur associé de la revue Artificial Intelligence. Il est notamment co-auteur du livre « A Concise Introduction to Models and Methods for Automated Planning », paru en 2013 (Morgan-Claypool).

Progress and Challenges in Planning

Planning in AI represents the model-based approach to control where actions are selected to achieve goals given a predictive model of the actions and sensors. The main challenges in planning are computational, as all the models, whether accommodating feedback and uncertainty or not, are intractable in the general case. In this talk, I'll review some the models

considered in planning, and some recent ideas that have turned out to be useful computationally. I'll focus mainly on three types of methods from our own recent work : 1) width-based search, where we show results on a number of video games from the ALE and GVG-AI environments, 2) translations for planning with partial observability using classical planners, and 3) multiagent planning with nested beliefs. In all cases, we seek methods that can be both general and effective. This is joint work with students and collaborators, including Blai Bonet, Nir Lipovetzky, Miquel Ramirez, Filippos Kominis, and Tomas Geffner.

Comité de Programme

Présidente du comité de programme

Frédéric Maris, Institut de Recherche en Informatique de Toulouse (IRIT), Université Toulouse 3 - Paul Sabatier

Membres du comité de programme

— Lamia Belouaer (IRENAV, Brest)
— Olivier Buffet (LORIA/INRIA, Nancy)
— Humbert Fiorino (Univ. Grenoble 1, LIG, Grenoble)
— Raphaël Fonteneau (Univ. de Liège, Liège, Belgique)
— Jérôme Lang (LAMSADE, Paris)
— Vincent Lemaire (Orange Labs, Lannion)
— Rémi Munos (INRIA, Lille)
— Cyril Pain-Barre (Univ. Aix-Marseille, LSIS, Aix-Marseille)
— Damien Pellier (Univ. Grenoble 2, LIG, Grenoble)
— Olivier Pietquin (Univ. Lille 1, LIFL, Lille)
— Pierre Régnier (Univ. Toulouse 3, IRIT, Toulouse)
— Olivier Sigaud (Univ. Paris 6, ISIR, Paris)
— Vincent Vidal (ONERA, Toulouse)
— Bruno Zanuttini (Univ. Caen Basse-Normandie, GREYC, Caen)
— Aurélie Beynier (Univ. Paris 6, LIP6, Paris)
— Pierre Bisquert (INRA, Montpellier)
— Maroua Bouzid (Univ. Caen Basse-Normandie, GREYC, Caen)
— Lucian Busoniu (Technical University of Cluj Napoca, Romania)
— Martin Cooper (Univ. Toulouse 3, IRIT, Toulouse)
— Christos Dimitrakakis (Chalmers university of technology, Gothenburg, Sweden)
— Alain Dutech (LORIA/INRIA, Nancy)
— Aurélien Garivier (Univ. Toulouse 3, IMT, Toulouse)
— Stéphane Grandcolas (Univ. Aix-Marseille, LSIS, Aix-Marseille)
— Matthieu Geist (Supélec, Metz)
— Romain Guillaume (Univ. Toulouse 2, IRIT, Toulouse)
— Laurent Jeanpierre (Univ. Caen Basse-Normandie, GREYC, Caen)
— Guillaume Laurent (ENSMM, Besançon)
— Odalric-Ambrym Maillard (Technion Faculty of Electrical Engineering, Haifa, Israël)
— Laetitia Matignon (Univ. Lyon 1, LIRIS, Lyon)
— Cédric Pralet (ONERA, Toulouse)
— Philippe Preux (Univ. Lille 3, INRIA, Lille)
— Mathieu Serrurier (Univ. Toulouse 3, IRIT, Toulouse)
— Florent Teichteil-Königsbuch (ONERA, Toulouse)
— Thierry Vidal (Ecole Nationale d'Ingénieurs de Tarbes, LGP, Tarbes)

Table des matières

Articles longs

Résumés

Démonstration

ST-SMTPLAN : règles de codage SMT pour la planification spatio-temporelle

Lamia Belouaer[1], Frédéric Maris[2]

[1] IRENav – Ecole navale, Brest
lamia.belouaer@ecole-navale.fr

[2] IRIT - CNRS UMR 5505 – Université de Toulouse
maris@irit.fr

Résumé :

Pour résoudre des problèmes de planification dans le monde réel, il est nécessaire de considérer l'information spatiale et temporelle pour prendre en compte la durée des actions, les instants où les conditions sont requises, ...et l'espace dans lequel la mission est accomplie en définissant les différentes zones d'action, en prenant en compte les chemins entre ces zones, ...Dans cet article, nous nous intéressons à des problèmes de planification qui prennent en compte ces deux dimensions. Notre principale contribution est de présenter une approche permettant la résolution de problèmes de planification spatio-temporelle basée sur de nouvelles règles de codage SMT (SAT Modulo Theories) pour ce type de planification. Ces nouveaux codages permettent de compiler et de résoudre des problèmes de planification spatio-temporelle pour lesquels toutes les solutions nécessitent la concurrence des actions dans un espace 2D.

Introduction

Le cadre classique de planification ne permet de représenter que des aspects limités du monde réel. Nous cherchons à élargir ce cadre à des aspects spatiaux et temporels tout en maintenant de bonnes performances pour la résolution de problèmes. Dans des problèmes de planification réels, ces deux dimensions sont indispensables car il faut prendre en compte la durée des actions, les instants de production de leurs effets, les instants où les conditions sont requises...et l'espace dans lequel la mission est accomplie (en définissant les différentes zones d'action et les chemins entre ces zones). La manipulation d'objets dans une centrale nucléaire, l'évacuation d'un bâtiment en cas d'incendie, le service dans un restaurant...sont des exemples de tels problèmes. Pour résoudre des problèmes de ce type nous proposons une approche basée sur le raisonnement spatial et temporel. A notre connaissance, il n'existe aucun modèle permettant d'exprimer des problèmes de planification en considérant en même temps ces deux dimensions, et il n'existe aucun planificateur permettant de résoudre de tels problèmes.

Dans cet article, nous présentons une approche permettant la résolution de problèmes de planification spatio-temporelle en autorisant la synchronisation d'actions non instantanées dans l'espace. Pour cela, nous intégrons aux règles de codage de TLP-GP-2 (Maris & Régnier, 2008) de nouvelles règles basées sur l'ontologie spatiale *SpaceOntology* (Belouaer *et al.*, 2010), et qui permettent de coder la dimension spatiale du problème de planification.

Le planificateur temporel TLP-GP-2 est basé sur l'utilisation d'un graphe de planification simplifié et d'un solveur SAT Modulo Theories (SMT). Son langage de représentation de problème permet une plus grande flexibilité que PDDL2.1 (Fox & Long, 2003) pour définir des domaines et des problèmes de planification temporelle. Bien que sa puissance expressive soit identique à celle de PDDL2.1, les extensions mises en œuvre dans TLP-GP permettent à l'utilisateur d'exprimer des problèmes du monde réel beaucoup plus facilement (Cooper *et al.*, 2010). Pour fournir une représentation temporelle des actions plus riche, des *timepoints* peuvent être utilisés dans les actions, autres que *start* et *end*, avec des ensembles de contraintes binaires d'(in)égalités linéaires simples entre *timepoints*. D'autre part, des modalités temporelles de haut niveau permettent à l'utilisateur de définir des relations complexes entre une condition ou un effet et un

intervalle, notamment être valide sur tout l'intervalle, à un certain instant de l'intervalle, dans un sous-intervalle ou être soumis à une transition continue sur l'intervalle. TLP-GP peut également prendre en compte, d'une manière très naturelle, des évènements exogènes ainsi que des buts temporellement étendus. Il est complet pour les sous-langages temporellement-expressifs de PDDL2.1 en utilisant la méthode de transformation de (Cooper *et al.*, 2013) pour restaurer sa complétude lors de la manipulation de problèmes temporellement-cycliques.

L'ontologie *SpaceOntology* (Belouaer *et al.*, 2010) considère différentes représentations de l'espace : qualitative (relation topologique et distance floue), quantitative (distance numérique) et hiérarchique (inclusion). De plus, *SpaceOntology* définit un ensemble de règles pour déduire de nouvelles informations et compléter la description de l'environnement. Cela permet une description spatiale complexe et une gestion facile de différents aspects spatiaux. Dans cet article, nous considérons quatre des concepts importants définis dans *SpaceOntology* : l'entité spatiale, la distance numérique, la distance floue et le lien hiérarchique. Nous présentons une extension du planificateur temporel TLP-GP-2, appelée ST-SMTPLAN, qui prend en compte la dimension temporelle et les quatre aspects précédents de *SpaceOntology*.

Cet article est organisé de la manière suivante. Nous y présentons d'abord divers travaux concernant la planification spatiale et/ou temporelle. Nous introduisons ensuite, dans une partie préliminaire, les différentes notations et définitions nécessaires. Nous décrivons un exemple pour illustrer notre propos puis, nous présentons les principales règles de codage nécessaires à sa résolution. La section de discussion montre de manière théorique la correction et la complétude de ce codage à deux dimensions. La conclusion présente enfin les perspectives de recherche ouvertes par ce travail.

État de l'art

Pour résoudre des problèmes réels de planification, l'un des principaux défis consiste à considérer les dimensions spatiales et temporelles. Cependant, à notre connaissance, il n'y a pas de modèle permettant de représenter un problème de planification en tenant compte à la fois de la connaissance spatiale et temporelle et aucun planificateur permettant de résoudre un tel problème. Cette partie est un état de l'art des travaux qui portent sur les planificateurs temporels et spatiaux.

Planificateurs temporels

La plupart des systèmes de planification temporelle utilisent un solveur de contraintes supplémentaire pour l'ordonnancement des tâches. Dans le passé, de nombreux planificateurs ont utilisé un espace de plans hiérarchique (HTN). Ils ont utilisé une logique temporelle basée sur les instants et les intervalles, conjointement à un TMM (Time Map Manager) qui gère les contraintes temporelles. C'est le cas de planificateurs comme IxTeT (Ghallab & Alaoui, 1989), (Laborie & Ghallab, 1995) ou HSTS (Muscettola, 1993). Lorsque (Cushing *et al.*, 2007) ont publié leurs importants travaux sur la planification temporellement-expressive, la majorité des planificateurs temporels efficaces développés entre 1994 et 2007 étaient incapables de résoudre les problèmes temporellement-expressifs, bien que certains planificateurs aient depuis été améliorés pour résoudre ces problèmes. Outre les planificateurs de type HTN qui présentent des performances relativement faibles, seulement quelques-uns d'entre eux peuvent résoudre ce type de problème. Certains d'entre eux comme CRIKEY3 (Coles *et al.*, 2008), VHPOP (Younes & Simmons, 2003), LPGP (Long & Fox, 2003), TLP-GP-1 (Maris & Régnier, 2008) (Maris & Régnier, 2008), et la version la plus récente de LPG (Gerevini *et al.*, 2010), utilisent un algorithme de recherche couplé avec un solveur STN (Simple Temporal Network). D'autres, comme TM-LPSAT (Shin & Davis, 2004), BAT (Hu, 2007), TLP-GP-2 (Maris & Régnier, 2008) (Maris & Régnier, 2008), STEP (Huang *et al.*, 2009), utilisent une méthode similaire à celle utilisée par la famille des planificateurs classiques de type BLACKBOX (Kautz & Selman, 1999). Ils codent simultanément un graphe de planification (Blum & Furst, 1995) et des contraintes temporelles, puis font appel à un solveur (LP, CSP, SMT ou SAT) pour trouver une solution.

Planificateurs spatiaux

La connaissance spatiale est essentielle en planification, par exemple dans le cas de l'évacuation de bâtiments : pour accomplir cette mission les agents doivent se déplacer à travers l'environnement, entre les différentes zones. Il y a peu de planificateurs exploitant la connaissance spatiale et qui ont été décrits dans

la littérature. Trois exemples sont ASYMOV (Gravot *et al.*, 2005), un planificateur présenté dans (Guitton *et al.*, 2008) et SPOON (Belouaer *et al.*, 2011).

Le planificateur ASYMOV calcule des plans pour gérer des problèmes dans lesquels l'exécution d'une action a un effet important sur la représentation spatiale du problème. Par exemple, lorsqu'un agent doit transporter un objet, la forme de l'ensemble de l'agent avec l'objet est différente de celle de la charge de l'agent. Le planificateur décrit dans (Guitton *et al.*, 2008) se compose de deux modules de raisonnement : un module de raisonnement symbolique soutenu par un planificateur de tâches et un module de raisonnement soutenu par un planificateur de chemins. Ces deux planificateurs ne gèrent que l'information spatiale quantitative (distance numérique, coordonnées...). Ceci n'est pas toujours suffisant, par exemple se déplacer dans un environnement nécessite une connaissance spatiale pour le décrire. Dans ce type d'application, il est également nécessaire de connaître les relations d'adjacence, les distances entre les différentes régions pour calculer un chemin. La connaissance spatiale peut être quantitative (numérique), qualitative (topologique ou floue) et hiérarchique (simplifie la description de l'environnement global). En outre, ces deux planificateurs n'expriment pas les problèmes de planification en tenant compte de la dimension spatiale.

SPOON est un planificateur hybride combinant deux modules de raisonnement : un module de raisonnement symbolique soutenu par un planificateur de tâches et un module de raisonnement spatial soutenu par un planificateur de chemin et *SpaceOntology*. Cette ontologie fournit une connaissance structurée de l'environnement exploré dans le processus de planification. SPOON utilise space-PDDL (Belouaer *et al.*, 2011) comme langage pour exprimer un problème de planification en tenant compte de la dimension spatiale.

Notre principale contribution est d'intégrer certains concepts de *SpaceOntology* dans TLP-GP-2 afin de considérer à la fois les dimensions spatiale et temporelle dans les problèmes de planification.

Préliminaires

Cette section présente toutes les notations nécessaires à la définition des règles spatio-temporelles. Un $fluent$ est une proposition atomique positive ou négative. Nous définissons un ensemble de prédicats spatiaux produisant des fluents spatiaux lorsqu'ils sont instanciés. Nous considérons des conditions sur la valeur des fluents et les changements de cette valeur peuvent être soit instantanés soit imposés sur un intervalle.

Une action a est un quadruplet $< Cond(a), Eff(a), Constr(a), Mov(a) >$, où :

— $Cond(a)$ est un ensemble de fluents (y compris spatiaux) qui doivent être vrais pour que a soit exécutée,

— $Eff(a)$ est un ensemble de fluents (y compris spatiaux) qui sont établis ou détruits par l'exécution de a,

— $Constr(a)$ est un ensemble de contraintes entre les instants relatifs aux évènements qui interviennent pendant l'exécution de a,

— $Mov(a)$ est un ensemble de vecteurs de fonctions à valeurs réelles associés à chaque fluent spatial $Move(e) \in Eff(a)$. Ces vecteurs correspondent au mouvement de l'entité spatiale e produit par a.

Un évènement correspond à l'une des neuf possibilités suivantes :

$$\left\{\begin{array}{l}\text{l'instant exact,}\\ \text{ou le début d'un intervalle,}\\ \text{ou la fin d'un intervalle,}\end{array}\right. \quad \text{sur lequel un fluent est} \quad \left\{\begin{array}{l}\text{produit,}\\ \text{ou détruit,}\\ \text{ou requis,}\end{array}\right. \quad \text{par une action } a.$$

Quand un évènement intervient à un instant exact, nous utilisons les notations :

— $a \rightarrow f$ pour désigner l'évènement correspondant à l'établissement de f par l'action a,

— $a \rightarrow \neg f$ pour désigner l'évènement correspondant à la destruction de f par l'action a,

— $f \rightarrow a$ pour désigner l'évènement correspondant à la requête de f par l'action a.

Quand un évènement intervient sur un intervalle, nous utilisons les notations :

— $a \mid\!\rightarrow f$ (resp. $a \rightarrow\!\mid f$) pour désigner le début (resp. la fin) de l'intervalle correspondant à l'établissement de f par l'action a,

— $a \mid\!\rightarrow \neg f$ (resp. $a \rightarrow\!\mid \neg f$) pour désigner le début (resp. la fin) de l'intervalle correspondant à la destruction de f par l'action a,

— $f \mid\!\rightarrow a$ (resp. $f \rightarrow\!\mid a$) pour désigner le début (resp. la fin) de l'intervalle correspondant à la requête de f par l'action a.

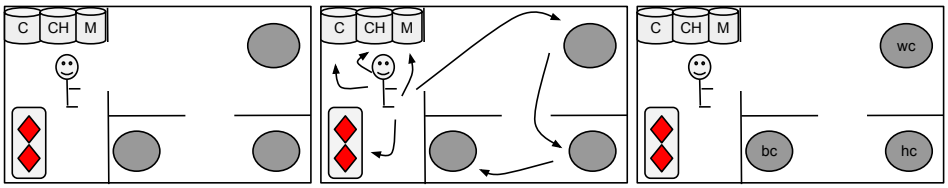
Nous utilisons la notation $\tau(E)$ pour représenter l'instant dans le plan auquel un évènement E intervient.

Pour une ation a donnée, $Events(a)$ représente les différents évènements qui constituent sa définition. Si A est un ensemble d'actions, alors $Events(A)$ est l'union des $Events(a)$ (pour toute action $a \in A$).

Pour toute entité spatiale e nous définissons $T\delta(e)$ comme l'ensemble des variables temporelles correspondant à un évènement spatial concernant l'entité e dans la définition des actions. Ces variables temporelles correspondent à une lecture de l'état spatial de e pour les conditions (position numérique, distance numérique, distance floue) ou une mise à jour de cet état pour les effets (mouvement, lien hiérarchique). L'ensemble de toutes les variables correspondant aux évènements spatiaux sur toutes les entités de SpE (l'ensemble de toutes les entités spatiales) est noté $T\delta$.

Exemple

Pour illustrer notre proposition, considérons l'exemple suivant. Un serveur travaille dans l'environnement représenté par la figure 1(a). Dans l'état initial, il se trouve dans la cuisine où il peut accéder à deux tables de cuisson (représentées par des losanges) et une étagère sur laquelle se trouve du café (C), du chocolat (CH) et du lait (M). Sa mission est de servir chaque client avec les boissons commandées : un café au lait (wc) dans le salon, un expresso (bc) et un chocolat chaud (hc) en terrasse (Figure 1(c)).



(a) Scenario (b) Chemins possibles (c) But du serveur

FIGURE 1 – Environnement et mission.

Pour réaliser ce travail le serveur considère la description spatiale de l'environnement dans lequel il opère, afin de trouver un plan qui lui permette de préparer et servir les boissons (Figure 1(b)). En outre, le succès de cette mission nécessite la satisfaction de la contrainte temporelle suivante : quand la boisson est servie elle doit être suffisamment chaude. Considérons que l'objectif du serveur soit de servir le café au lait. Dans la suite, nous représentons la description de sa mission en PDDL étendu avec des fluents spatiaux (Figures 2, 3).

La figure 2 présente les objets (ingrédient, ...), types, les prédicats symboliques (at, on, carry, ...), les prédicats spatiaux (Inside, ...) et les actions (Go, Pick, ...). La figure 3 présente la mission de l'agent.

Définition 1 (Problème de planification spatio-temporelle)
Un problème de planification spatio-temporelle $< I, A, G >$ est défini par un ensemble d'actions A, un état initial I et un but G, où I et G sont des ensembles de fluents.

Nous introduisons trois contraintes de base que tous les plans spatio-temporels doivent satisfaire :

— les **contraintes inhérentes** sur l'ensemble des actions A : pour toutes actions $a_i, a_j \in A$, a satisfait $Constr(a)$;

— les **contraintes d'effets contradictoires** sur l'ensemble des actions A : pour toutes actions $a_i, a_j \in A$, pour tout fluent f tel que $\neg f \in Eff(a_i)$ et $f \in Eff(a_j)$, $\tau(a_i \rightarrow| \neg f) < \tau(a_j \mid\rightarrow f) \vee \tau(a_j \rightarrow| f) < \tau(a_i \mid\rightarrow \neg f)$;

— les **contraintes de mouvements contradictoires** sur l'ensemble des actions A : pour toutes actions $a_i, a_j \in A$, pour toute entité spatiale $e \in SpE$, pour tout fluent spatial $Move(e)$ tel que $Move(e) \in Eff(a_i) \cap Eff(a_j)$, $\big(\tau(a_i \rightarrow| Move(e)) < \tau(a_j \mid\rightarrow Move(e))\big) \vee \big(\tau(a_j \rightarrow| Move(e)) < \tau(a_i \mid\rightarrow Move(e))\big)$.

Définition 2 (Plan spatio-temporel)
$P =< A, t >$ où A est un ensemble d'instances d'actions $\{a_1, \ldots, a_n\}$ et t est une fonction à valeurs réelles sur $Events(a)$, est un plan spatio-temporel pour le problème $< I, A', G >$ si et seulement si :

1. *$A \subset A'$*
2. *P satisfait les contraintes inhérentes et les contraintes d'effets contradictoires sur A ; quand P est exécuté (i.e. les fluents sont établis et détruits aux instants donnés par t) en partant de l'état initial I*

3. *pour toute action $a_i \in A$, chaque fluent $f \in Cond(a_i)$ est vrai quand il est requis*
4. *tous les fluents du but $g \in G$ sont vrais à la fin de l'exécution de P.*

Pour coder un problème de planification nous utilisons un graphe de planification simplifié *GP* (Blum & Furst, 1995). *NoeudsA* désigne l'ensemble des nœuds d'actions de *GP*. *ArcsCond* et $ArcEff$ désignent respectivement l'ensemble des arcs conditions et des arcs d'effets de *GP*. *SpE* désigne l'ensemble des entités spatiales définies dans le problème de planification. Dans la section suivante, nous présentons l'en-

```
(define (domain service)
 (:requirements …)
 (:types …)
 (:predicates …)
 (:durative-action Pick
    :parameters (?ingredient - ingredient ?area - area)
    :duration (= ?duration (pick-time ?ingredient))
    :condition (and (at start (on ?ingredient ?area))
                    (over [start end[ (at ?area))
                    (at start (FD_Near waiter ?area)))
    :effect (and (at end (carry ?ingredient)))

 (:durative-action Go
     :parameters (shelf table)
     :duration (= ?duration (travel-time shelf table))
     :condition (and (at start (at shelf))
                     (at start (Inside waiter kitchen))
     :effect (and (at end (at table)
                  (over [start (+ start (/ (?duration) 3))[
                 (Move_1 waiter : Delta_x[t]=2.0*t : Delta_y[t]=0.0*t))
                 (at (+ start (/ (?duration) 3)) (not (Inside kitchen)))
                 (at (+ start (/ (?duration) 3)) (Inside lounge))
                 (over [(+ start (/ (?duration) 3)) end[
                 (Move_2 waiter : Delta_x[t]=1.5*t : Delta_y[t]=1.0*t))
 ))

 (:durative-action  Heat …)
 (:durative-action Make_W_C …)
 (:durative-action Serve …))
```

FIGURE 2 – Définition du domaine en PDDL.

```
(:init
      (Inside waiter kitchen)
      (Inside table lounge)
      (= table.center.x 4.0)
      (= table.center.y 3.2)
      …
      (at shelf)
      (on coffee shelf)
      (on milk shelf)
      (= (pick-time coffee) 0.2)
      (= (pick-time milk) 0.2)
      (= (travel-time shelf table) 3.0)
...)
(:goal
      (at 20 (served table cup))
      (at 20 (contains2 cup coffee milk))
      (over [20 25] (hot coffee))
      (over [20 25] (hot milk))
)
```

FIGURE 3 – Définition de la mission en PDDL.

semble des règles qui définissent ST-SMTPLAN. Comme on ne connaît pas à l'avance la longueur d'un plan solution d'un problème, on ne peut pas créer un codage unique permettant de le résoudre puisqu'il faudrait créer une infinité de variables propositionnelles pour représenter toutes les actions de tous les plans possibles. Nous proposons un codage représentant tous les plans d'une longueur k fixée (la profondeur du graphe de planification). La base de contraintes ainsi obtenue est donnée en entrée à un solveur SMT qui retourne, lorsqu'il existe, un modèle de cette base. Le décodage de ce modèle permet alors d'obtenir un plan-solution. Si la résolution du codage ne donne pas de modèle, la valeur de k est augmentée (le graphe de planification est étendu) et le processus réitéré. Pour la complétude du procédé, tous ces modèles correspondent exactement à tous les plans solutions d'une longueur fixée du problème.

ST-SMTPLAN : règles de codage pour la planification spatio-temporelle

Pour coder un problème de planification spatio-temporelle, nous définissons trois types de règles :

1. les règles (R) codent la structure d'un plan solution du problème basée sur un graphe de planification temporellement étendu,
2. les règles (T) gèrent la connaissance temporelle
3. les règles (S) gèrent la connaissance spatiale et temporelle.

Formellement, $\mathcal{R}$ désigne l'ensemble des règles de ST-SMTPLAN ($\mathcal{R} = R \cup T \cup S$). Dans cet article, nous présentons un sous-ensemble des règles de $\mathcal{R}$ qui sont utilisées pour résoudre le problème de notre exemple. Les règles présentées ici sont classées par groupe : celles concernant la définition de l'état initial et du but, le codage du graphe de planification temporellement étendu, simultanément les dimensions spatiale et temporelle et les exclusions mutuelles temporellement étendues.

Codage de l'état initial et du but

Nous définissons deux actions factices *Init* et *Goal*. *Init* produit les fluents de l'état initial et *Goal* requiert les fluents du but. Dans un plan-solution, ces deux actions factices doivent être vraies.

$$Init \wedge Goal \tag{R1}$$

Pour coder la description de l'état initial et du but on a aussi besoin de certaines règles S1. D'un point de vue spatial, l'état initial considère la description des positions (Règle S1.1) et les définitions hiérachiques (Règle S1.2). La règle (T1) code le fait que l'instant auquel le but est atteint est après l'instant auquel l'état initial est vérifié. Elle peut être relaxée lorsque l'on souhaite modéliser des états initiaux datés ou des buts temporellement étendus.

(S1.1) : Position numérique des entités spatiales dans l'état initial

Dans *SpaceOntology* : une entité spatiale e est une entité localisée dans un espace donné. Elle peut être statique : par exemple, tout élément fixe dans l'espace dont la position ne peut pas être modifiée par l'action d'un agent. Ou elle peut être dynamique : toute entité dont la position spatiale change avec le temps. Une entité spatiale est définie par son centre.

$$\bigwedge_{e \in SpE} \big((x[\tau(Init)](e) = e.center.x) \wedge (y[\tau(Init)](e) = e.center.y)\big) \tag{S1.1}$$

Remarque 1
Nous supposons que nous sommes dans un espace 2D. x désigne l'axe x et y désigne l'axe y. Toutes les règles sont définies par rapport à ces deux axes. Dans cet article, nous détaillons uniquement les règles pour l'axe x. Ces règles peuvent être étendues à un espace 3D.

Exemple 1
Considérons l'espace défini dans la Figure 1(c). Nous nous concentrons sur la table qui est dans le salon.
Pour localiser cette table, nous considérons son centre et sa projection sur chacun des axes (Figure 4). Ceci nous permet d'instancier la règle (S1.1) de cette manière :

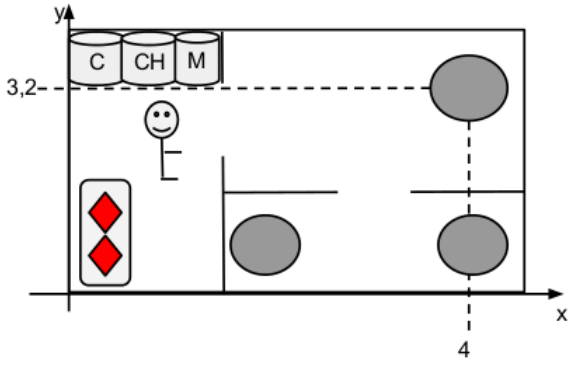


FIGURE 4 – Positions des entités spatiales.

$$\begin{cases} x[\tau(Init)](table) = 4 \\ y[\tau(Init)](table) = 3,2 \end{cases}$$

(S1.2) : Relation d'inclusion des entités spatiales dans l'état initial

L'ontologie *SpaceOntology* fournit une représentation hiérarchique qui permet de simplifier la description de l'environnement global. Pour représenter la relation hiérarchique, nous définissons le fluent spatial *Inside*. Ce fluent spatial exprime le lien d'inclusion entre deux entités spatiales. Formellement, pour chaque paire d'entités spatiales $\{e_1, e_2\}$ où la relation $e_1 \subset e_2$ est vraie, nous ajoutons la règle suivante :

$$\bigwedge_{(e_1,e_2)\in SpE^2/e_1\subset e_2} Inside[\tau(Init)](e_1,e_2) \qquad \text{(S1.2)}$$

Exemple 2
L'état initial de notre problème est représenté dans la Figure 1(a). $\tau(Init)$ est l'instant initial. $Inside[\tau(Init)](waiter, kitchen)$ exprime que, dans l'état initial, le serveur est dans la cuisine. Ainsi, pour exprimer l'inclusion spatiale dans l'état initial, nous instancions la règle S1.2 de la manière suivante :

$$\begin{cases} Inside[\tau(Init)](shelf, kitchen) \\ Inside[\tau(Init)](cooker, kitchen) \\ Inside[\tau(Init)](waiter, kitchen) \end{cases}$$

(T1) : Bornes inférieure et supérieure

L'instant inital (où les propositions de l'état initial sont vraies) précède tous les instants de début des conditions des autres actions. L'instant final (où toutes les propositions du but sont vraies) suit tous les instants de fin des autres actions.

$$(\tau(Init) \leq \tau(Goal)) \wedge \bigwedge_{a\in NodeA} \left((\tau(Init) \leq \min_{p\in Cond(a)} \{\tau(p \mid\rightarrow a)\}) \wedge (\max_{q\in Eff(a)} \{\tau(a \rightarrow\mid q)\} \leq \tau(Goal)) \right) \qquad \text{(T1)}$$

Codage du graphe de planification temporellement étendu

Les règles (*R*) codent le problème comme un graphe de planification temporellement étendu. Ces règles codent les liens causaux entre effets et conditions (R2) et la sélection d'actions du plan (R3).

(R2) : Production des conditions par liens causaux

Cette règle code le fait que si une action b est active dans le plan, alors pour chacune de ses conditions p, il existe au moins un lien causal (noté $Link(a,p,b)$) de l'action a (qui produit cette condition) vers b.

$$\bigwedge_{(p,b)\in ArcsCond} \left(b \Rightarrow \bigvee_{(a,p)\in ArcsEff} Link(a,p,b) \right) \tag{R2}$$

(R3) : Activation des actions et ordre partiel

Cette règle code le fait que s'il existe un lien causal entre une action a qui produit une précondition p pour une action b, alors a et b sont actives dans le plan. De plus, l'instant où a produit certainement p est antérieur ou égal à l'instant où b commence à nécessiter p.

$$\bigwedge_{(a,p)\in ArcsEff} \bigwedge_{(p,b)\in ArcsCond} \left(Link(a,p,b) \Rightarrow \left(a \wedge b \wedge (\tau(a \mid\!\rightarrow p) \leq \tau(p \mid\!\rightarrow b)) \right) \right) \tag{R3}$$

Codage de la connaissance spatiale

Une action peut exiger la satisfaction de relations spatiales entre deux entités spatiales pour être exécutée. Une action peut aussi modifier les définitions spatiales entre deux entités spatiales. L'information spatiale est intégrée dans les ensembles $Cond(a)$ et $Eff(a)$ lorsque l'action a est définie dans le domaine de planification.

Une entité spatiale peut être dynamique ou statique. Si l'entité spatiale e est dynamique, l'utilisation du prédicat $Move(e)$ dans les effets d'une action a correspond à un déplacement de l'entité spatiale e par l'action a sur un intervalle temporel donné $[t_1,t_2]$. Nous associons au fluent $Move(e)$ une fonction de déplacement. Nous définissons sur l'intervalle $[t_1,t_2]$ une fonction linéaire correspondant au déplacement de l'entité e pour chaque axe. Pour l'axe horizontal (ox) la fonction linéaire est désignée par $\Delta_x^a[t](e) = v_x^a(e) \times t$ (où $v_x^a(e)$ est une constante correspondant à la projection sur cet axe de la vitesse donnée à l'entité e par l'action a). Le déplacement global de l'entité e produit par une action a sur l'axe (ox) est désigné par $\Delta_x^a(e)$ $= \Delta_x^a[\tau(a \rightarrow\mid Move(e)) - \tau(a \mid\!\rightarrow Move(e))](e)$.

Conditions spatiales

Les règles (S2) codent les conditions spatiales. En effet, il peut être nécessaire de calculer des distances (S2.1) et de connaître les distances numériques exactes ou approximatives (S2.2) pour exécuter des actions.

(S2.1) : Distances numériques entre entités

Pour chaque paire d'entités spatiales $\{e_1,e_2\}$ nous désignons par $d_x[t]\{e_1,e_2\}$ la distance numérique entre e_1 et e_2 selon l'axe x.

$$\bigwedge_{(e_1,e_2)\in SpE^2/(e_1\neq e_2)} \bigwedge_{t\in T\delta(e_1)\cap T\delta(e_2)} \begin{pmatrix} (x[t](e_1) \leq x[t](e_2) \Rightarrow d_x[t]\{e_1,e_2\} = x[t](e_2) - x[t](e_1)) \\ (x[t](e_2) < x[t](e_1) \Rightarrow d_x[t]\{e_1,e_2\} = x[t](e_1) - x[t](e_2)) \end{pmatrix} \tag{S2.1}$$

(S2.2) : Distance numérique entre entités spatiales

Le fluent spatial $Distance(e_1,e_2) = value$, où $value$ est un nombre rationnel, peut être utilisé comme contrainte dans la définition de $Cond(a)$. Il représente une condition de distance numérique fixée entre deux entités spatiales e_1 et e_2. Une contrainte $d_x[t]\{e_1,e_2\} = value$ est alors ajoutée pour chaque action a du graphe à $t = \tau(Distance(e_1,e_2) = value \rightarrow a$. Nous pouvons généraliser en remplaçant l'égalité par une inégalité.

$$\bigwedge_{(e_1,e_2)\in SpE^2/(e_1\neq e_2)} \bigwedge_{a\in NodeA/(Distance(e_1,e_2)=value)\in Cond(a)} a \Rightarrow \left(d_x[\tau(Distance(e_1,e_2) = value \rightarrow a]\{e_1,e_2\} = value \right) \tag{S2.2}$$

(S2.3) : Conditions de distance floue entre entités

L'ontologie *SpaceOntology* définit quatre concepts {Near, NearEnough, FarEnough, Far} qui permettent d'exprimer des distances floues. Le fluent spécial $FD_{label}(e_1, e_2)$ représente une distance floue entre deux entités e_1 et e_2 telle que $label = \{Near, NearEnough, FarEnough, Far\}$. Chaque label est associé à un intervalle $[\alpha_{label}, \beta_{label}[$.

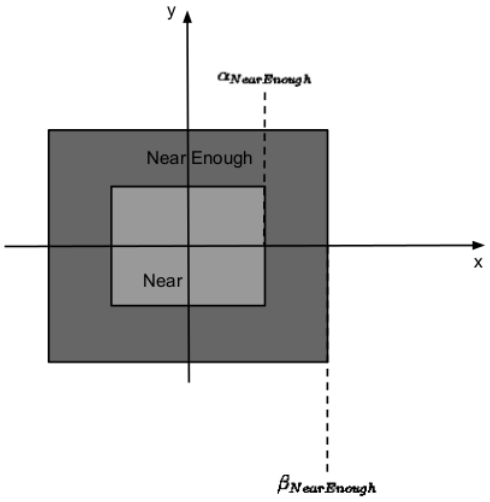


FIGURE 5 – Zones de distances floues relatives à une entité spatiale.

Considérons par exemple la distance floue *NearEnough*, $label = NearEnough$ dont l'intervalle associé est $[\alpha_{NearEnough}, \beta_{NearEnough}[$ (Figure 5).

$$\bigwedge_{(e_1,e_2)\in SpE^2/(e_1\neq e_2)} \bigwedge_{a\in NodeA/FD_{label}(e_1,e_2)\in Cond(a)} a \Rightarrow \left(\alpha_{label} \leq d_x[\tau(FD_{label}(e_1,e_2) \rightarrow a)]\{e_1,e_2\} < \beta_{label}\right) \quad \text{(S2.3)}$$

Exemple 3
Pour notre exemple, nous définissons les variables suivantes :

$$\begin{cases} a = Pick(milk, shelf) \\ t = \tau(FD_{Near}(waiter, shelf) \rightarrow Pick(milk, shelf) \\ e_1 = waiter \\ e_2 = shelf \end{cases}$$

L'application de la règle (S2.1) permet de calculer la distance entre le serveur et l'étagère :

$$\begin{cases} x[t](waiter) \leq x[t](shelf) \Rightarrow d_x[t]\{waiter, shelf\} = x[t](shelf) - x[t](waiter) \\ x[t](shelf) \leq x[t](waiter) \Rightarrow d_x[t]\{waiter, shelf\} = x[t](waiter) - x[t](shelf) \end{cases}$$

Considérons que l'exécution de l'action Pick nécessite que la distance entre le serveur et l'étagère soit exactement 0,3. L'application de la règle (S2.2) vérifie cette distance numérique entre le serveur et l'étagère : $Pick(milk, shelf) \Rightarrow d_x[\tau(Distance(waiter, shelf) = 0.3 \rightarrow Pick(milk, shelf))]\{waiter, shelf\} = 0.3)$. *Supposons maintenant que l'exécution de l'action Pick nécessite que le serveur soit "près" de l'étagère. L'application de la règle (S2.3) vérifie cette distance floue entre le serveur et l'étagère :* $Pick(milk, shelf) \Rightarrow \alpha_{Near} \leq d_x[\tau(FD_{Near}(waiter, shelf) \rightarrow Pick(milk, shelf))]\{waiter, shelf\} < \beta_{Near}$.

Effets spatiaux

Le calcul des distances entre entités spatiales nécessite de savoir leur position numérique. Il est donc nécessaire de connaître ces positions dans l'état initial, mais également de les recalculer quand les effets d'une action les modifient.

(S3) : Calcul des nouvelles positions

Considérons l'intervalle temporel $[t_1,t_2]$. La position de l'entité spatiale e_1 à l'instant t_2 est calculée à partir de sa position à l'instant t_1 et des déplacements $\delta_x^{Ind}[t_1,t_2](e_1,a,e_2)$ induits par toutes les entités e_2 qui peuvent être déplacées par une action a.

$$\bigwedge_{e_1 \in SpE} \bigwedge_{(t_1,t_2)\in T\delta^2} \Big((x[t_2](e_1) = x[t_1](e_1) + \sum_{(e_2,a)\in SpE\times NodeA/\Delta_x^a(e_2)\neq 0} \delta_x^{Ind}[t_1,t_2](e_1,a,e_2) \Big) \tag{S3}$$

Exemple 4
Considérons l'action $Go(shelf,table)$. Notons :
— $t_1 = \tau(Go(shelf,table) \mid\!\rightarrow Move_2(waiter))$
— $t_2 = \tau(Go(shelf,table) \rightarrow\!\mid Move_1(waiter))$
Appliquons la règle S3.

$$x[t_2](cup) = \begin{cases} x[t_1](cup) & + \\ \delta_x^{Ind}[t_1,t_2](cup, Go(shelf,cooker), waiter) & + \\ \delta_x^{Ind}[t_1,t_2](cup, Go(cooker,shelf), waiter) & + \\ \delta_x^{Ind}[t_1,t_2](cup, Go(shelf,table), waiter) & + \\ \dots & \end{cases}$$

Pour calculer le déplacement induit par l'entité e sur les autres entités ou sur l'entité e elle-même, il est nécessaire de connaître les relations hiérarchiques entre e et toutes les autres entités à l'instant où e commence ou termine son déplacement. Voici les règles qui décrivent les liens hiérarchiques entre entités.

(S4.1) : Production et suppression de lien hiérarchique

Si une action a qui produit l'inclusion d'une entité spatiale e_1 dans une entité spatiale e_2 à l'instant $t = \tau(a \rightarrow\!\mid Inside(e_1,e_2))$ est active dans le plan, alors le prédicat $Inside[t](e_1,e_2)$ (qui correspond à l'inclusion $e_1 \subset e_2$ à l'instant t) est vrai.

$$\bigwedge_{(e_1,e_2)\in SpE^2/e_1\neq e_2} \bigwedge_{a\in NodeA/Inside(e_1,e_2)\in Eff(a)} \big(a \Rightarrow Inside[\tau(a \rightarrow\!\mid Inside(e_1,e_2))](e_1,e_2)\big) \tag{S4.1.1}$$

Si une action a qui détruit l'inclusion $e_1 \subset e_2$ à l'instant $t = \tau(a \mid\!\rightarrow \neg Inside(e_1,e_2))$ est active dans le plan, alors le prédicat $Inside[t](e_1,e_2)$ est faux.

$$\bigwedge_{(e_1,e_2)\in SpE^2/e_1\neq e_2} \bigwedge_{a\in NodeA/\neg Inside(e_1,e_2)\in Eff(a)} \big(a \Rightarrow \neg Inside[\tau(a \mid\!\rightarrow \neg Inside(e_1,e_2))](e_1,e_2)\big) \tag{S4.1.2}$$

Exemple 5
Le déplacement du serveur entre l'étagère et la table $(Go(shelf,table))$ à deux effets spatiaux sur la hiérarchie. Le premier est la production du lien hiérarchique "le serveur est dans le salon" (en appliquant la règle (S4.1.1)) : $Go(shelf,table) \Rightarrow Inside[\tau(Go(shelf,table) \rightarrow\!\mid Inside(waiter,lounge))](waiter,lounge)$. Le second est la destruction du lien hiérarchique "le serveur est dans la cuisine" (en appliquant la règle (S4.1.2)) : $Go(shelf,table) \Rightarrow \neg Inside[\tau(Go(shelf,table) \mid\!\rightarrow \neg Inside(waiter,kitchen))](waiter,kitchen)$.

(S4.2) : Propagation des liens hiérarchiques dans le temps

Si un lien hiérarchique $e_1 \subset e_2$ est actif dans le plan à l'instant $t \in T\delta$, où un évènement spatial se produit, alors il existe au moins un lien hiérarchique positif de protection d'intervalle (noté $LinkInside(e_1,e_2,a,t)$) d'une action a qui produit $Inside(e_1,e_2)$, jusqu'à l'instant t.

$$\bigwedge_{(e_1,e_2)\in SpE^2/e_1\neq e_2} \bigwedge_{t\in T\delta} \Big(Inside[t](e_1,e_2) \Rightarrow \bigvee_{a\in NodeA/Inside(e_1,e_2)\in Eff(a)} LinkInside(e_1,e_2,a,t) \Big) \tag{S4.2.1}$$

$$\bigwedge_{(e_1,e_2)\in SpE^2/e_1\neq e_2}\ \bigwedge_{a\in NodeA/Inside(e_1,e_2)\in Eff(a)}\ \bigwedge_{t\in T\delta}\Big(LinkInside(e_1,e_2,a,t)\Rightarrow a\wedge(\tau(a\rightarrow|\ Inside(e_1,e_2))\leq t)\Big) \tag{S4.2.2}$$

De la même façon, si un lien hiérarchique $e_1\subset e_2$ n'est pas actif dans le plan à l'instant $t\in T\delta$, où un évènement spatial se produit, alors il existe au moins un lien hiérarchique négatif de protection d'intervalle (noté $LinkNotInside(e_1,e_2,a,t)$) d'une action a qui produit $\neg Inside(e_1,e_2)$, jusqu'à l'instant t.

Exemple 6
Nous supposons que t est l'instant où l'action de servir la tasse ($Serve(table,cup)$) nécessite que le serveur porte la tasse ($t=\tau(Inside(cup,waiter)\ |\!\!\rightarrow Serve(table,cup))$). Si à l'instant t le serveur porte la tasse alors il existe un lien pour propager cette propriété de l'action $Make\_W\_C(milk,coffee,cup)$ qui la produit.

$$\Big\{Inside[t](cup,waiter)\Rightarrow LinkInside(cup,waiter,Make\_W\_C(milk,coffee,cup),t)$$

Si un lien propage le fait que le serveur porte la tasse de l'action $Make\_W\_C(milk,coffee,cup)$, jusqu'à l'instant t, alors cette action doit être active dans le plan et produire le lien hiérarchique avant l'instant t.

$$\begin{cases} LinkInside(cup,waiter,Make\_W\_C(milk,coffee,cup),t) \\ \Rightarrow Make\_W\_C(milk,coffee,cup)\wedge\tau(Make\_W\_C(milk,coffee,cup)\rightarrow|\ Inside[(cup,waiter)\leq t \end{cases}$$

Dans notre cadre, imposons quelques contraintes supplémentaires que nous traduisons par plusieurs nouvelles règles. La règle (S5.1.1) impose qu'une entité ne puisse pas entrer dans une autre lorsque cette dernière est en mouvement. De la même manière, la règle (S5.1.2) impose qu'une entité ne puisse pas sortir d'une autre lorsque cette dernière est en mouvement. La règle (S5.2) indique que toute entité est incluse dans elle-même.

$$\bigwedge_{(e_1,e_2)\in SpE^2/e_1\neq e_2}\ \bigwedge_{(a,b)\in NodeA^2/Inside(e_1,e_2)\in Eff(a)\wedge\Delta^b_x(e_2)\neq 0}\begin{pmatrix} a\wedge b\Rightarrow \\ (\tau(a\rightarrow|\ Inside(e_1,e_2))<\tau(b\ |\!\!\rightarrow Move(e_2)) \\ \vee(\tau(b\rightarrow|\ Move(e_2)<\tau(a\ |\!\!\rightarrow Inside(e_1,e_2)) \end{pmatrix} \tag{S5.1.1}$$

$$\bigwedge_{e\in SpE}\ \bigwedge_{t\in T\delta}\big(Inside[t](e,e)\big) \tag{S5.2}$$

Exemple 7
Considérons les deux actions : préparer le café ($Make\_W\_C(milk,coffee,cup)$) et se déplacer de la table de cuisson vers l'étagère ($Go(cooker,shelf)$). Pour exécuter ces deux actions, nous devons respecter l'une de ces situations (en appliquant la règle (S5.1.1)) :

- *l'instant où l'action $Make\_W\_C(milk,coffee,cup)$ finit de produire le fait que le serveur porte la tasse est avant l'instant où l'action $Go(cooker,shelf)$ commence à produire le déplacement du serveur.*
- *l'instant où l'action $Go(cooker,shelf)$ finit de produire le déplacement du serveur est avant l'instant où l'action $Make\_W\_C(milk,coffee,cup)$ commence à produire le fait que le serveur porte la tasse.*

$$\begin{cases} Make\_W\_C(milk,coffee,cup)\wedge Go(cooker,shelf) \\ \Rightarrow \\ \begin{Bmatrix} \tau(Make\_W\_C(milk,coffe,cup\rightarrow|\ Inside(cup,waiter)) \\ <\tau(Go(cooker,shelf)\ |\!\!\rightarrow Move(waiter)) \end{Bmatrix} \\ \vee \\ \begin{Bmatrix} <\tau(Go(cooker,shelf)\rightarrow|\ Move(waiter)) \\ <\tau(Make\_W\_C(milk,coffe,cup\ |\!\!\rightarrow Inside(cup,waiter)) \end{Bmatrix} \end{cases}$$

Nous pouvons maintenant calculer les déplacements induits par une entité sur une autre.

(S6.1) : Déplacement induit avec lien hiérarchique

Supposons qu'une action a qui produit le déplacement d'une entité spatiale e_2 ($\Delta_x^a(e_2) \neq 0$) soit active dans le plan. Considérons une entité spatiale e_1 telle que $e_1 \subset e_2$. Considérons également l'intervalle temporel $I = [t_1, t_2]$. Soit $[t', t]$ l'intersection de I et de l'intervalle sur lequel le déplacement de e_2 est produit par a. Si cette intersection n'est pas vide alors le déplacement de e_1 induit par e_2 est $\delta_x^{Ind}[t_1,t_2](e_1,a,e_2) = \Delta_x^a[t'-t](e_2)$, sinon le mouvement induit est égal à zéro.

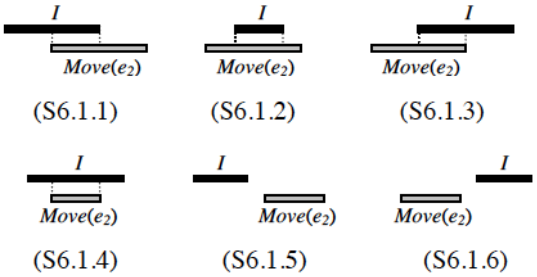


FIGURE 6 – Configurations possibles entre un intervalle I et un intervalle de déplacement d'entité spatiale.

Les différentes configurations et les règles associées sont illustrées par la Figure 6. Ici, nous ne détaillons que la règle (S6.1.1).

$$\bigwedge_{(e_1,e_2)\in SpE^2} \bigwedge_{(t_1,t_2)\in T\delta^2} \bigwedge_{a\in NodeA/\Delta_x^a(e_2)\neq 0} \left(\begin{array}{c} (a \wedge Inside[\tau(a \mid\!\rightarrow Move(e_2))](e_1,e_2) \\ \wedge t_1 \leq \tau(a \mid\!\rightarrow Move(e_2)) \wedge \tau(a \mid\!\rightarrow Move(e_2)) \leq t_2 \\ \wedge t_2 \leq \tau(a \rightarrow\!\mid Move(e_2)) \\ \Rightarrow (\delta_x^{Ind}[t_1,t_2](e_1,a,e_2) = \Delta_x^a[t_2 - \tau(a \mid\!\rightarrow Move(e_2))](e_2)) \end{array} \right) \quad \text{(S6.1.1)}$$

Exemple 8
Notons :
— $t_1 = \tau(Go(shelf, lounge) \rightarrow\!\mid Move_2(waiter))$
— $t_2 = \tau(Go(shelf, lounge) \mid\!\rightarrow Move_1(waiter))$
Nous sommes dans le cas de la règle (S6.1.4). Sur l'intervalle de déplacement global $[t_1,t_2]$ sur lequel le serveur se déplace de l'étagère à la table. Si le serveur porte la tasse, chacun de ses déplacements ($Move_1(waiter)$ et $Move_2(waiter)$) induit le déplacement de la tasse.

$$\left\{ \begin{array}{l} Go(shelf,table) \wedge \\ Inside[\tau(Go(shelf,table) \mid\!\rightarrow Move(waiter))](cup,waiter) \wedge \\ t_1 \leq \tau(Go(shelf,table) \mid\!\rightarrow Move(waiter)) \wedge \\ \tau(Go(shelf,table) \rightarrow\!\mid Move(waiter)) \leq t_2 \wedge \\ \Rightarrow (\delta_x^{Ind}[t_1,t_2](cup, Go(shelf,table), waiter) = \\ \quad \Delta_x^{Go(shelf,table)}[t_2 - \tau(Go(shelf,table) \mid\!\rightarrow Move(waiter))](waiter)) \end{array} \right.$$

(S6.2) : Déplacement induit sans liens hiérarchiques

Si une action a qui produit le déplacement d'une entité e_2, noté $\Delta_x^a(e_2)$, est active dans le plan, et étant donné une entité spatiale e_1 distincte de e_2 ($e_1 \neq e_2$) et qui n'est pas dans e_2 ($e_1 \not\subset e_2$) quand a commence à déplacer e_2, alors le déplacement de e_1 sur l'intervalle $[t_1,t_2]$ induit par le déplacement de e_2 produit par l'action a, noté $\delta_x^{Ind}[t_1,t_2](e_1,a,e_2)$, est égal à zéro.

$$\bigwedge_{(e_1,e_2)\in SpE^2/e_1\neq e_2} \bigwedge_{(t_1,t_2)\in T\delta^2} \bigwedge_{a\in NodeA/\Delta_x^a(e_2)\neq 0} \left((a \wedge \neg Inside[\tau(a \mid\!\rightarrow Move(e_2))](e_1,e_2) \Rightarrow (\delta_x^{Ind}[t_1,t_2](e_1,a,e_2) = 0)\right) \quad \text{(S6.2)}$$

Exemple 9
Dans notre exemple : le serveur se déplace de l'étagère à la table sans prendre la tasse ($cup \not\subset waiter$). L'application de la règle (S6.2) montre que le déplacement de la tasse induit par le déplacement du serveur est alors égal à zéro.

$$\left\{\begin{array}{l} Go(shelf,table)\wedge \\ \neg Inside[\tau(Go(shelf,table)\mid\!\rightarrow Move(waiter))](cup,waiter) \\ \qquad\qquad \Rightarrow (\delta_x^{Ind}[t_1,t_2](cup,Go(shelf,table),waiter)=0 \end{array}\right.$$

(6.3) : Déplacement induit par une action non selectionnée

Si une action a qui produit le déplacement d'une entité spatiale e_2, noté $\Delta_x^a(e_2)$, n'est pas active dans le plan et étant donnée une entité spatiale e_1 (par nécessairement distincte de e_2) alors le déplacement de e_2 sur l'intervalle temporel $[t_1,t_2]$ induit par le déplacement de e_2 produit par a, noté $\delta_x^{Ind}[t_1,t_2](e_1,a,e_2)$, est égal à zéro.

$$\bigwedge_{(e_1,e_2)\in SpE^2}\ \bigwedge_{(t_1,t_2)\in T\delta^2}\ \bigwedge_{a\in NodeA/\Delta_x^a(e_2)\neq 0} \left(\neg a \Rightarrow (\delta_x^{Ind}[t_1,t_2](e_1,a,e_2)=0\right) \tag{S6.3}$$

Exclusions mutuelles temporellement étendues

Deux propositions sont mutuellement exclusives quand :
— les propositions sont antagonistes, par exemple p et $\neg p$;
— ou les propositions représentent le mouvement de la même entité spatiale e (le prédicat spécial $Move(e)$ est utilisé par deux actions).
Dans ces cas, les propositions ne peuvent intervenir sur le même intervalle temporel.

(R4) : Exclusions mutuelles temporellement étendues

— Si un lien causal assure la protection d'une proposition p et qu'une action produisant sa négation est active dans le plan, alors l'intervalle temporel correspondant au lien causal et l'intervalle temporel correspondant à l'activation de $\neg p$ par l'action sont disjoints.

$$\bigwedge_{(a,p)\in ArcsEff}\ \bigwedge_{(p,b)\in ArcsCond}\ \bigwedge_{(c,\neg p)\in ArcsEff} \left(Link(a,p,b)\wedge c \Rightarrow \left(\begin{array}{l}(\tau(c\mid\!\rightarrow\neg p)<\tau(a\mid\!\rightarrow p))\\ \vee(\tau(p\mid\!\rightarrow b)<\tau(c\mid\!\rightarrow\neg p))\end{array}\right)\right) \tag{R4.1}$$

— Si deux actions produisant respectivement une proposition p et sa négation sont actives dans le plan, alors les intervalles temporels correspondants à l'activation de p et à l'activation de $\neg p$ sont disjoints.

$$\bigwedge_{(a,p)\in ArcsEff}\ \bigwedge_{(b,\neg p)\in ArcsEff} \left(a\wedge b \Rightarrow \left(\begin{array}{l}(\tau(b\mid\!\rightarrow\neg p)<\tau(a\mid\!\rightarrow p))\\ \vee(\tau(a\mid\!\rightarrow p)<\tau(b\mid\!\rightarrow\neg p))\end{array}\right)\right) \tag{R4.2}$$

(S7) : Exclusions mutuelles entre actions qui déplacent la même entité

— Si un lien causal assure la protection de la proposition $Move(e)$ qui représente le déplacement d'une entité spatiale e et qu'une action qui produit la proposition $Move(e)$ est active dans le plan, alors l'intervalle temporel correspondant au lien causal et l'intervalle temporel correspondant à l'activation de $Move(e)$ par l'action sont disjoints.
— Si deux actions produisant la proposition $Move(e)$ sont actives dans le plan, alors les intervalles temporels correspondants à l'activation de $Move(e)$ par ces actions sont disjoints. Pour obtenir respectivement les règles (S7.1) et (S7.2) correspondant à ces types d'exclusions mutuelles, il suffit de remplacer p et $\neg p$ par $Move(e)$ dans les règles (R4.1) and (R4.2).

(S8) : Exclusions mutuelles sur les intervalles de protection de lien hiérarchique

De manière similaire à la protection de lien causal (R4.1), nous ajoutons deux règles, respectivement (S8.1) et (S8.2), qui fournissent une protection de lien hiérarchique pour $LinkInside(e_1,e_2,a,t)$ et $LinkNotInside(e_1,e_2,a,t)$.

$$\bigwedge_{(e_1,e_2)\in SpE^2/e_1\neq e_2}\ \bigwedge_{a\in NodeA/Inside(e_1,e_2)\in Eff(a)}\ \bigwedge_{t\in T}\delta\, b\in NodeA/\neg Inside(e_1,e_2)\in Eff(b)$$
$$\left(LinkInside(e_1,e_2,a,t)\wedge b\Rightarrow\left(\begin{array}{c}(\tau(b\rightarrow|\ \neg Inside(e_1,e_2)<\tau(a\mid\rightarrow Inside(e_1,e_2))\\ \vee(t<\tau(b\mid\rightarrow\neg Inside(e_1,e_2)))\end{array}\right)\right) \tag{S8.1}$$

Discussion

Les règles de codage présentées dans cet article sont saines et complètes pour tous les problèmes de planification temporellement-expressifs exprimés dans le langage de représentation de TLP-GP étendu avec la définition de fluents spatiaux. Ce résultat est donné, pour les règles qui codent la structure du plan solution (R) et les contraintes temporelles (T), par le fait que TLP-GP est sain et complet, en utilisant la méthode de transformation de (Cooper *et al.*, 2013) pour restaurer sa complétude lorsqu'il doit résoudre des problèmes temporellement-cycliques. La validité des règles est conservée quand on ajoute les règles spatio-temporelles (S). Quand une solution est trouvée par le solveur de contraintes, les contraintes inhérentes et d'effets contradictoires sont satisfaites trivialement par les règles définies dans TLP-GP. Les contraintes de mouvements contradictoires sont également satisfaites trivialement par la règle (7.2). De plus, la propagation, codée par les règles, de la connaissance spatiale sur les entités (position et hiérarchie) à tous les instants où cette connaissance est utilisée par les actions dans le plan, garantit que les fluents spatiaux sont vrais quand ils sont requis. D'où, par la Définition 2, la solution donne un plan spatio-temporel valide. Nous définissons maintenant des versions spatialement-relaxées d'un plan spatio-temporel comme tous les plans solutions temporels du problème de planification temporelle résultant du problème de planification spatio-temporelle, après avoir supprimé tous les fluents spatiaux de l'état initial, des définitions d'actions et du but. Si un plan spatio-temporel $< A,t >$ existe pour un problème de planification spatio-temporelle $< I,A',G >$ alors toutes les solutions correspondant aux versions spatialement-relaxées de ce plan peuvent être trouvées en utilisant uniquement les règles (R) et (T), ce qui résulte de la complétude de TLP-GP. De toutes ces solutions, celles qui correspondent au problème de planification spatio-temporelle initial devraient satisfaire toutes les contraintes spatio-temporelles à tous les instants où un fluent spatial est requis, c'est à dire exactement ce qui est codé par toutes les règles (S). D'où, la complétude des règles de codage vient du fait que ces dernières solutions peuvent toujours être trouvées par le solveur.

Pour simplifier la présentation, nous n'avons considéré que certains aspects restreints de *SpaceOntology*. Par exemple, dans la définition de distance floue, les valeurs de α_{label} et β_{label} sont les mêmes pour tous les niveaux hiérarchiques. Nous pouvons facilement étendre les règles de codage pour prendre en compte différentes valeurs pour la distance floue en fonction des liens hiérarchiques entre entités.

Conclusion

Les règles de codage ST-SMTPLAN nous permettent donc de décrire et de résoudre des problèmes de planification spatio-temporelle en combinant les règles de codage décrites dans TLP-GP-2 et l'intégration de concepts de *SpaceOntology*. Cette ontologie permet d'obtenir une représentation spatiale simplifiée de l'état initial et de manipuler des distances floues entre les entités spatiales. De plus, le principe de TLP-GP, nous permet d'obtenir un système de planification temporelle capable de résoudre des problèmes qui nécessitent la concurrence des actions. Ce système utilise un solveur SMT et bénéficie directement des améliorations de ce type de solveur en terme de performances.

Cet article est un travail théorique. Afin de prouver l'efficacité de ce codage, nous devons définir des benchmarks de planification spatio-temporelle. Pour simplifier la représentation de l'espace, nous avons considéré toute entité spatiale comme un point (ce point définit le centre), ce qui réduit l'expressivité. Dans nos travaux futurs, nous nous concentrerons sur la taille et la forme de chaque entité spatiale. Une voie de recherche future est d'utiliser le principe de TLP-GP-1 qui effectue une recherche en arrière dans le graphe de planification. Ceci nous permettra de faire des appels à *SpaceOntology* et de coder la connaissance spatiale extraite à chaque sélection d'action. Ceci permet une très grande expressivité spatiale ajoutée à l'expressivité temporelle.

Références

BELOUAER L., BOUZID M. & MOUADDIB A. (2010). Ontology based spatial planning for human-robot interaction. In *TIME 2010 - 17th International Symposium on Temporal Representation and Reasoning, Paris, France, 6-8 September 2010*, p. 103–110.

BELOUAER L., BOUZID M. & MOUADDIB A.-I. (2011). Spatial knowledge in planning language. In *International Conference on Knowledge Engineering and Ontology Development*.

BLUM A. & FURST M. L. (1995). Fast planning through planning graph analysis. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, p. 1636–1642.

COLES A., FOX M., LONG D. & SMITH A. (2008). Planning with problems requiring temporal coordination. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, p. 892–897.

COOPER M. C., MARIS F. & RÉGNIER P. (2010). Compilation of a high-level temporal planning language into PDDL 2.1. In *22nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2010, Arras, France, 27-29 October 2010 - Volume 2*, p. 181–188.

COOPER M. C., MARIS F. & RÉGNIER P. (2013). Managing temporal cycles in planning problems requiring concurrency. *Computational Intelligence*, **29**(1), 111–128.

CUSHING W., KAMBHAMPATI S., MAUSAM & WELD D. S. (2007). When is temporal planning really temporal ? In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, p. 1852–1859.

FOX M. & LONG D. (2003). PDDL2.1 : an extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)*, **20**, 61–124.

GEREVINI A., SAETTI A. & SERINA I. (2010). Temporal planning with problems requiring concurrency through action graphs and local search. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010*, p. 226–229.

GHALLAB M. & ALAOUI A. M. (1989). Managing efficiently temporal relations through indexed spanning trees. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence. Detroit, MI, USA, August 1989*, p. 1297–1303.

GRAVOT F., CAMBON S. & ALAMI R. (2005). asymov : a planner that deals with intricate symbolic and geometric problems. In *Robotics Research. The Eleventh International Symposium*, p. 100–110 : Springer.

GUITTON J., FARGES J.-L. & CHATILA R. (2008). A planning architecture for mobile robotics. In *INTELLIGENT SYSTEMS AND AUTOMATION : 1st Mediterranean Conference on Intelligent Systems and Automation (CISA 08)*, volume 1019, p. 162–167 : AIP Publishing.

HU Y. (2007). Temporally-expressive planning as constraint satisfaction problems. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007*, p. 192–199.

HUANG R., CHEN Y. & ZHANG W. (2009). An optimal temporally expressive planner : Initial results and application to P2P network optimization. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*.

KAUTZ H. A. & SELMAN B. (1999). Unifying sat-based and graph-based planning. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*, p. 318–325.

LABORIE P. & GHALLAB M. (1995). Planning with sharable resource constraints. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, p. 1643–1651.

LONG D. & FOX M. (2003). Exploiting a graphplan framework in temporal planning. In *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003), June 9-13, 2003, Trento, Italy*, p. 52–61.

MARIS F. & RÉGNIER P. (2008). Tlp-gp : New results on temporally-expressive planning benchmarks. In *20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2008), November 3-5, 2008, Dayton, Ohio, USA, Volume 1*, p. 507–514.

MARIS F. & RÉGNIER P. (2008). TLP-GP : solving temporally-expressive planning problems. In S. DEMRI & C. S. JENSEN, Eds., *15th International Symposium on Temporal Representation and Reasoning, TIME 2008, Université du Québec à Montréal, Canada, 16-18 June 2008*, p. 137–144 : IEEE Computer Society.

MUSCETTOLA N. (1993). *HSTS : Integrating Planning and Scheduling*. Rapport interne CMU-RI-TR-93-05, Robotics Institute, Pittsburgh, PA.

SHIN J. & DAVIS E. (2004). Continuous time in a sat-based planner. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, p. 531–536.

YOUNES H. L. S. & SIMMONS R. G. (2003). VHPOP : versatile heuristic partial order planner. *J. Artif. Intell. Res. (JAIR)*, **20**, 405–430.

A simple account of multiagent epistemic planning

Martin C. Cooper, Andreas Herzig, Faustine Maffre, Frédéric Maris, Pierre Régnier

IRIT - CNRS UMR 5505 – University of Toulouse, France
cooper@irit.fr, herzig@irit.fr, maffre@irit.fr,
maris@irit.fr, regnier@irit.fr

Abstract A realistic model of multiagent planning must allow us to model notions which are absent in classical planning such as communication and knowledge. We investigate multiagent planning based on a simple logic of action and knowledge that is based on the visibility of propositional variables. Using such a formal logic allows us to deduce the validity of a plan from the validity of the individual actions which compose it. We present a coding of multiagent planning problems expressed in this logic into the classical planning language PDDL. Feeding the resulting problem into a PDDL planner provides a provably correct plan for the original multiagent planning problem. We use the gossip problem as a running example.

Keywords: Planning, multiagent system, epistemic logic

1 Introduction

Suppose there are n people, and each of them knows some item of gossip not known to the others. They communicate by telephone, and whenever one person calls another, they tell each other all they know at that time. How many calls are required before each item of gossip is known to everyone ? This can be viewed as a multiagent planning problem where contrarily to classical planning it is not the world that evolves but the agents' knowledge. In this paper we provide a simple multiagent epistemic logic allowing to reason about such problems. We also study how they can be encoded into PDDL, the classical Planning Domain Definition Language first introduced by (Mc Dermott *et al.*, 1998).

2 A simple epistemic logic

We start by defining a language and semantics of a simple logic of action and knowledge that is based on the visibility of propositional variables. Our logic is a dialect of Dynamic Logic of Propositional Assignments DL-PA (Balbiani *et al.*, 2013).

2.1 Language

Let $\mathsf{PVar} = \{p_1, p_2, \ldots\}$ be a finite set of *propositional variables* and $\mathsf{Agt} = \{i_1, i_2, \ldots\}$ a finite set of *agents*. To each $p \in \mathsf{PVar}$ and each agent $i \in \mathsf{Agt}$ we associate one more propositional variable $\mathbf{Kw}_i p$, which reads "i knows whether p". We understand this as follows : when i knows whether p then either p is true and i knows that, or p is false and i knows that. The set of *atomic formulas* of our language is then

$$\mathsf{Fml}_0 = \mathsf{PVar} \cup \{\mathbf{Kw}_i p \ : \ i \in \mathsf{Agt} \text{ and } p \in \mathsf{PVar}\}.$$

We remark that our framework can be further extended to also contain information of the form $\mathbf{Kw}_i\mathbf{Kw}_j p$ ('i knows whether j knows whether p'). Note that the latter does not entail that i himself knows whether p.

Our language has two syntactical entities, formulas and programs. They are defined by mutual recursion as follows :

$$\varphi ::= \alpha \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\pi\rangle\varphi$$
$$\pi ::= +\alpha \mid -\alpha \mid \pi;\pi \mid \pi \sqcup \pi \mid \varphi? \mid \pi^*$$

where α ranges over the set of atomic formulas Fml_0.

$\langle\pi\rangle\varphi$ is the formula which results when the program π is applied to the formula φ. It reads "there exists an execution of π after which φ is true". The other boolean operators $\bot$, $\top$, $\vee$, $\rightarrow$ and $\leftrightarrow$ and the dual operator $[\pi]\varphi = \neg\langle\pi\rangle\neg\varphi$ (which reads "after every execution of π, φ is true") are defined as usual.

Programs are composed of instructions which are either assignments ($+\alpha$ or $-\alpha$), sequences of instructions ($\pi;\pi$), non-deterministic choice between instructions ($\pi \sqcup \pi$), tests ($\varphi?$) or repetitions (π^*).

We define the more standard modal operator 'knowing that' ranging over literals as follows :

$$\mathbf{K}_i p \stackrel{\text{def}}{=} p \wedge \mathbf{Kw}_i p$$
$$\mathbf{K}_i \neg p \stackrel{\text{def}}{=} \neg p \wedge \mathbf{Kw}_i p$$

A *boolean formula* is a formula without the dynamic operator $\langle.\rangle$. The set of all boolean formulas is noted $\mathsf{Fml}_{\mathsf{Bool}}$. We note $\mathsf{var}(\varphi)$ the set of atomic formulas appearing in the boolean formula φ. For example, $\mathsf{var}(q \wedge \mathbf{Kw}_i p) = \{q, \mathbf{Kw}_i p\}$. (Note that $p \notin \mathsf{var}(q \wedge \mathbf{Kw}_i p)$.) The set of *atomic programs* is

$$\mathsf{Pgm}_0 = \{+\alpha \ : \ \alpha \in \mathsf{Fml}_0\} \cup \{-\alpha \ : \ \alpha \in \mathsf{Fml}_0\}.$$

Two elements of Pgm_0 are *in conflict* if they assign different truth values to the same variable.

2.2 Semantics

A state is a subset of the set of atomic formulas (corresponding to exactly those atomic formulas that are true in this state). Formulas are interpreted as sets of states (those states in which the formula is true) and programs are interpreted as relations on states.

$$\begin{aligned}
||\alpha|| &= \{s \ : \ \alpha \in s\} \\
||\neg\varphi|| &= 2^{\mathsf{Fml}_0} \setminus ||\varphi|| \\
||\varphi \wedge \psi|| &= ||\varphi|| \cap ||\psi|| \\
||\langle\pi\rangle\varphi|| &= \{s \ : \ \text{there is a } s' \in ||\varphi|| \text{ such that } \langle s, s'\rangle \in ||\pi||\} \\
||+\alpha|| &= \{\langle s, s'\rangle \ : \ s' = s \cup \{\alpha\}\} \\
||-\alpha|| &= \{\langle s, s'\rangle \ : \ s' = s \setminus \{\alpha\}\} \\
||\pi_1;\pi_2|| &= ||\pi_1|| \circ ||\pi_2|| \\
&= \{\langle s, s'\rangle \ : \ \text{there is } s'' \text{ such that } \langle s, s''\rangle \in ||\pi_1|| \text{ and } \langle s'', s'\rangle \in ||\pi_2||\} \\
||\pi_1 \sqcup \pi_2|| &= ||\pi_1|| \cup ||\pi_2|| \\
||\varphi?|| &= \{\langle s, s\rangle \ : \ s \in ||\varphi||\} \\
||\pi^*|| &= (||\pi||)^*
\end{aligned}$$

In words, a propositional variable α is true in s if it belongs to it ; the negation of a formula $\neg\varphi$ is true in all worlds where φ is not true ; the conjunction $\varphi \wedge \psi$ is true in states where both φ and ψ are true ; and $\langle\pi\rangle\varphi$ is true if there is a state reachable by executing π where φ is true. Concerning programs, assignments update the state by adding or removing the corresponding variable ; the sequence $\pi_1;\pi_2$ reaches a state by executing π_1 and executes π_2 from this state ; the non-deterministic choice $\pi_1 \sqcup \pi_2$ represents the union of the relations for π_1 and for π_2 ; the test $\varphi?$ stays in the same state if it verifies φ (otherwise it fails and the program stops) ; and the repetition π^* reaches any state attainable if we repeat π any number of times by making the reflexive transitive closure of the relation $||\pi||$.

3 Action theories and planning tasks

A conditional action is a couple $a = \langle\mathsf{pre}(a), \mathsf{eff}(a)\rangle$ where $\mathsf{pre}(a)$ is a boolean formula (the *precondition* of a) and $\mathsf{eff}(a)$ is a set of *conditional effects* : couples of the form $\langle\mathsf{cond}_i, \pi_i\rangle$ such that $\mathsf{cond}_i \in \mathsf{Fml}_{\mathsf{Bool}}$ is a boolean formula and $\pi_i \in \mathsf{Pgm}_0$ is an assignment.

As an example, let us consider the conditional action $\mathsf{toggle}(p)$ of flipping the truth value of the propositional variable p. It is described as $\mathsf{toggle}_p = \langle\mathsf{pre}(\mathsf{toggle}_p), \mathsf{eff}(\mathsf{toggle}_p)\rangle$ with $\mathsf{pre}(\mathsf{toggle}_p) = \top$ and $\mathsf{eff}(\mathsf{toggle}_p) = \{\langle p, -p\rangle, \langle\neg p, +p\rangle\}$.

Example 1
The action call_{ij} *corresponds to agents* i *and* j *telling each other every secret they know among all* n *secrets. The secret of agent* i *is modelled by a propositional variable* s_i *and agent* j*'s knowledge of this secret is modelled by* $\mathbf{Kw}_j s_i$*. We have* $\mathsf{call}_{ij} = \langle \mathsf{pre}(\mathsf{call}_{ij}), \mathsf{eff}(\mathsf{call}_{ij}) \rangle$ *with* $\mathsf{pre}(\mathsf{call}_{ij}) = \top$ *and*

$$\begin{aligned}\mathsf{eff}(\mathsf{call}_{ij}) = \{&\langle \mathbf{Kw}_i s_1, +\mathbf{Kw}_j s_1 \rangle, \ldots, \langle \mathbf{Kw}_i s_n, +\mathbf{Kw}_j s_n \rangle,\\ &\langle \mathbf{Kw}_j s_1, +\mathbf{Kw}_i s_1 \rangle, \ldots, \langle \mathbf{Kw}_j s_n, +\mathbf{Kw}_i s_n \rangle\}.\end{aligned}$$

Intuitively, we check knowledge of both agents about every secret and add knowledge of a secret to one agent only if it is known to the other.

A given conditional action a determines a relation between states :

$$\begin{aligned}||a|| = \big\{\langle s, s' \rangle \ : \ & s \in ||\mathsf{pre}(a)|| \text{ and there is no } \langle \mathsf{cond}_1, \pi_1 \rangle, \langle \mathsf{cond}_2, \pi_2 \rangle \in \mathsf{eff}(a)\\ & \text{such that } \pi_1 \text{ and } \pi_2 \text{ are in conflict and } s \in ||\mathsf{cond}_1 \wedge \mathsf{cond}_2||\\ & \text{and } s' = \big(s \setminus \{\alpha \ : \ \text{there is } \langle \mathsf{cond}, -\alpha \rangle \in \mathsf{eff}(a) \text{ and } s \in ||\mathsf{cond}||\}\big)\\ & \qquad \cup \{\alpha \ : \ \text{there is } \langle \mathsf{cond}, +\alpha \rangle \in \mathsf{eff}(a) \text{ and } s \in ||\mathsf{cond}||\}\big\}\end{aligned}$$

Thus an action is executable if its precondition is currently verified and if there cannot be any conflict. When executed, for every conditional effect such that the condition is true, the action will remove every α such that the effect is $-\alpha$ and add every α such that the effect is $+\alpha$.

We say that a state s is *reachable* from a state s_0 via a set of actions Act if there is a sequence of actions $\langle a_1, \ldots, a_n \rangle$ and a sequence of states $\langle s_0, \ldots, s_n \rangle$ with $n \geq 0$ such that $s_n = s$ and $\langle s_{k-1}, s_k \rangle \in ||a_k||$ for every k such that $1 \leq k \leq n$.

A *planning task* is a triple $\langle \mathsf{Act}, s_0, \mathsf{Goal} \rangle$ where Act is a finite set of actions, $s_0 \subseteq \mathsf{Fml}_0$ is a state (the initial state) and $\mathsf{Goal} \in \mathsf{Fml}_{\mathsf{Bool}}$ is a boolean formula (the goal). It is solvable if at least one of the goal states in $||\mathsf{Goal}||$ is reachable from s_0 via Act ; else it is unsolvable.

Example 2 (Example 1, ctd.)
The planning task corresponding to the gossip problem is $\langle \mathsf{Act}, s_0, \mathsf{Goal} \rangle$ *with* $\mathsf{Act} = \{\mathsf{call}_{ij} : i, j \in \mathsf{Agt} \text{ and } i \neq j\}$, $s_0 = \{\mathbf{Kw}_i s_i : 1 \leq i \leq n\}$ *(every agent knows its own secret) and* $\mathsf{Goal} = \bigwedge_{i,j \in \mathsf{Agt}} \mathbf{Kw}_i s_j$ *(every agent knows every secret).*

4 Encoding into DL-PA

Consider an action $a = \langle \mathsf{pre}(a), \{\langle \mathsf{cond}_1, \pi_1 \rangle, \ldots, \langle \mathsf{cond}_n, \pi_n \rangle\} \rangle$. Action a can be directly coded into PDDL as we will show in Section 5. However, some extra work is required to code actions in DL-PA.

Before executing an action, we have to check that its effects are not in conflict ; if two effects are, the action can only be executed if these effects cannot both happen. The formula

$$\mathsf{conflict}_a = \bigvee_{\substack{\langle \mathsf{cond}_i, \pi_i \rangle, \langle \mathsf{cond}_j, \pi_j \rangle \in \mathsf{eff}(a),\\ \pi_i \text{ and } \pi_j \text{ are in conflict}}} \mathsf{cond}_i \wedge \mathsf{cond}_j$$

is true when there is a couple of effects of a whose effects are conflicting and whose conditions are both true. We suppose that the disjunction of an empty set is $\bot$. In our running example there are no conflicts : we have $\mathsf{conflict}_{\mathsf{call}_{ij}} = \bot$.

We also have to take care that the truth value of conditions cannot change because of effects. To achieve this, we copy our conditions at the beginning of the action and evaluate copies. Note that this technical problem is specific to DL-PA ; when coding a planning problem in PDDL this problem does not arise since in PDDL all conditions are checked before any effects are produced. We use copies of atomic variables α, noted α'. We suppose that α' is syntactically different from every $\beta \in \mathsf{Fml}_0$. Then the copy of a formula φ is noted φ' and is defined such that every α occurring in φ is replaced by its copy α'. Furthermore, we define the program assigning the value of α to its copy α' by

$$\mathsf{copy}(\alpha) = (\alpha?; +\alpha') \sqcup (\neg\alpha?; -\alpha').$$

Given a set of atoms $A = \{\alpha_1, \ldots, \alpha_m\}$, we define the program $\mathsf{copy}(A) = \mathsf{copy}(\alpha_1); \ldots; \mathsf{copy}(\alpha_m)$ storing the the truth values of the α_i by copying them. (Note that the order does not matter.) Remark that $\varphi \leftrightarrow [\mathsf{copy}(\mathsf{var}(\varphi))]\varphi'$ is valid.

We can finally associate to a the DL-PA program :

$$\mathsf{prog}_a = \mathsf{pre}(a)?; \neg\mathsf{conflict}_a?; \mathsf{copy}\Big(\bigcup_{\langle \mathsf{cond}_i, \pi_i\rangle \in \mathsf{eff}(a)} \mathsf{var}(\mathsf{cond}_i)\Big); \big((\mathsf{cond}_1{}'?; \pi_1) \sqcup \neg\mathsf{cond}_1{}'?\big); \ldots; \big((\mathsf{cond}_n{}'?; \pi_n) \sqcup \neg\mathsf{cond}_n{}'?\big).$$

Proposition 1
The program prog_a *is well-defined ; in other words,* $||a|| = ||\mathsf{prog}_a||$.

Proof 1
Take a state s.

If $s \notin ||\mathsf{pre}(a)||$, *then the program* prog_a *fails because of the first test* $\mathsf{pre}(a)?$, *thus is not related to any state, as specified by* $||a||$. *The next test* $\mathsf{conflict}_a?$ *exactly corresponds to the constraint on conflict in* $||a||$ *: if* $\mathsf{conflict}_a$ *is false in* s *then no couple of conditional effects of* a *is in conflict and the program continues.*

Furthermore, the ordering of the sequence prog_a *does not matter because we test copies of the atoms occurring in the conditions (thus we test the truth of every condition in* s*). By the semantics of the non-deterministic choice operator* $\sqcup$, *each instruction* $(\mathsf{cond}_i{}'?; \pi_i) \sqcup \neg\mathsf{cond}_i{}'?$ *is equivalent to an if-then control sequence : if the guard* $\mathsf{cond}_i{}'$ *is true then* π_i *is executed, whereas if* $\neg\mathsf{cond}_i{}'$ *is true then no instruction is executed and the program continues. Therefore, if the condition is verified in* s, *then, due to the semantics of* $+\alpha$ *and* $-\alpha$, *if* π_i *is of the form* $+\alpha$ *then* α *is added to* s, *while if* π_i *is of the form* $-\alpha$ *then* α *is removed from* s, *as specified in* $||a||$.

It is interesting to note that coding epistemic planning into a formal logic has allowed us to identify the problem of conflicting effects. This highlights the fact that an automatic planner designed to solve planning problems expressed in PDDL and consisting of actions with conditional effects must take into account the possibility of conflicting effects : if two conditions of an action a are true and their respective effects are conflicting then action a cannot be executed. The possibility of conflicting effects is not explicitly prohibited in PDDL and this potential problem must therefore be managed within the automatic planner.

Example 3 (Example 1, ctd.)
The action call_{ij} *is associated to the program :*

$$\begin{aligned}\mathsf{prog}_{\mathsf{call}_{ij}} = {} & \top?; \neg\bot?; \mathsf{copy}\big(\{\mathbf{Kw}_i s_k \ : \ 1 \le k \le n\} \cup \{\mathbf{Kw}_j s_k \ : \ 1 \le k \le n\}\big); \\ & \big((\mathbf{Kw}_i s_1{}'?; +\mathbf{Kw}_j s_1) \sqcup \neg\mathbf{Kw}_i s_1{}'?\big); \\ & \ldots; \\ & \big((\mathbf{Kw}_i s_n{}'?; +\mathbf{Kw}_j s_n) \sqcup \neg\mathbf{Kw}_i s_n{}'?\big); \\ & \big((\mathbf{Kw}_j s_1{}'?; +\mathbf{Kw}_i s_1) \sqcup \neg\mathbf{Kw}_j s_1{}'?\big); \\ & \ldots; \\ & \big((\mathbf{Kw}_j s_n{}'?; +\mathbf{Kw}_i s_n) \sqcup \neg\mathbf{Kw}_j s_n{}'?\big)\end{aligned}$$

Note that in this case, both $\mathsf{pre}(a)?$ *and* $\neg\mathsf{conflict}_a?$ *(which are respectively* $\top?$ *and* $\neg\bot?$*) are always true, and hence could clearly be omitted.*

Proposition 2
A planning task $\langle \mathsf{Act}, s_0, \mathsf{Goal}\rangle$ *is solvable if and only if* s_0 *satisfies* $\big\langle (\bigsqcup_{a\in\mathsf{Act}} \mathsf{prog}_a)^* \big\rangle \mathsf{Goal}$.

Proof 2
We know by Proposition 1 that prog_a *behaves correctly and produces the same effects as action* a.

We can read the formula $\big\langle (\bigsqcup_{a\in\mathsf{Act}} \mathsf{prog}_a)^* \big\rangle \mathsf{Goal}$ *as follows :* $\bigsqcup_{a\in\mathsf{Act}} \mathsf{prog}_a$ *means choosing one action from* Act, *then applying the star operator to this means a sequence of arbitrary length of arbitrary actions, i.e., a plan* π.

For a planning task to be solvable, at least one state satisfying the goal must be reachable from s_0 by actions from Act, *that is, there is a sequence of actions such that executing them leads to a state in* $||\mathsf{Goal}||$. *By the semantics, s_0 satisfies $\langle\pi\rangle$*Goal *if and only if there exists an execution of π starting at s_0 such that we end in a state satisfying* Goal *; this corresponds to our definition of reachable. Therefore the task is solvable if and only if the formula is satisfied at the initial state.*

5 Encoding into PDDL

In this section we present a method for automatically encoding planning problems defined in our logic into PDDL. As already observed, in PDDL we do not need to make copies of variables as we were obliged to do in DL-PA. We also make the assumption that the automatic planner does not allow the execution of an action with conflicting effects : this means that we do not explicitly code the test for conflicts as we did in DL-PA. Under this assumption, Proposition 2 guarantees that the planner will find a solution-plan if and only if one exists.

Consider the planning task $\langle \mathsf{Act}, s_0, \mathsf{Goal}\rangle$. In order to encode this planning problem into PDDL, some requirement flags should be set depending on the form of the formulas cond_i used in the effects $\langle \mathsf{cond}_i, \pi_i\rangle$ of actions from Act, and on the formula Goal. Since all these formulas are in $\mathsf{Fml}_{\mathsf{Bool}}$, they have to be encoded into PDDL using two boolean operators :

— `and` directly with default requirement flag `:strips`,

— `not` using the requirement flags `:negative-preconditions` if used only on atomic propositions, and `:disjunctive-preconditions` in addition if used on conjunctions.

Moreover, the operator `or` could also be used, when this latter requirement flag is set, in order to simplify writing. Given a boolean formula $\varphi \in \mathsf{Fml}_{\mathsf{Bool}}$, we define a recursive function $tr_{PDDL}(\varphi)$ which returns the encoding of φ into PDDL :

$$\begin{aligned}
tr_{PDDL}(p) &::= \texttt{(p)}\\
tr_{PDDL}(\mathbf{Kw}_i p) &::= \texttt{(Kw i p)}\\
tr_{PDDL}(\neg\varphi) &::= \texttt{(not } tr_{PDDL}(\varphi)\texttt{)}\\
tr_{PDDL}(\varphi_1 \wedge \varphi_2) &::= \texttt{(and } tr_{PDDL}(\varphi_1)\ tr_{PDDL}(\varphi_2)\texttt{)}\\
tr_{PDDL}(\varphi_1 \vee \varphi_2) &::= \texttt{(or } tr_{PDDL}(\varphi_1)\ tr_{PDDL}(\varphi_2)\texttt{)}
\end{aligned}$$

An atomic proposition $p \in \mathsf{PVar}$ is encoded as a fluent without any parameters by its name `(p)`, whereas $\mathbf{Kw}_i p$ with $i \in \mathsf{Agt}$ and $p \in \mathsf{PVar}$ is encoded by a special fluent with i and p as parameters `(Kw i p)`. The initial state s_0 is then trivially encoded, and the Goal and the preconditions of all actions in Act can be encoded using the function above. Moreover, the requirement flag `:conditional-effects` must be set, unless all the effects of the actions in Act are of the form $\langle\top, \pi\rangle$ (recall that $\pi \in \mathsf{Pgm}_0$ is an assignment). Consider an action $a = \langle \mathsf{pre}(a), \{\langle \mathsf{cond}_1, \pi_1\rangle, \ldots, \langle \mathsf{cond}_n, \pi_n\rangle\}\rangle$. For every $\langle \mathsf{cond}_i, \pi_i\rangle \in \mathsf{eff}(a)$, we add the following conditional effects to `:effect` of the action a in PDDL :

— if $\pi_k = +p$, then we add `(when` $tr_{PDDL}(\mathsf{cond}_k)$ `(p))` ;

— if $\pi_k = -p$, then we add `(when` $tr_{PDDL}(\mathsf{cond}_k)$ `(not p))` ;

— if $\pi_k = +\mathbf{Kw}_i p$, then we add `(when` $tr_{PDDL}(\mathsf{cond}_k)$ `(Kw i p))` ;

— if $\pi_k = -\mathbf{Kw}_i p$, then we add `(when` $tr_{PDDL}(\mathsf{cond}_k)$ `(not (Kw i p)))` ;

Example 4 (Example 1, ctd.)
The action call_{ij} *is coded in PDDL, using conditional effects as follows :*

```
(:action call
     :parameters (?i - gossip ?j - gossip)
     :effect (and (when (Kw ?i s1) (Kw ?j s1))
                  ...
                  (when (Kw ?i sn) (Kw ?j sn))
                  (when (Kw ?j s1) (Kw ?i s1))
                  ...
                  (when (Kw ?j sn) (Kw ?i sn))))
```

Almost all planners from last International Planning Competition (IPC 2014) [1] handle conditional effects and negative preconditions, and most of them handle disjunctive preconditions. Moreover, most of them also handle universal effects which allows us to encode problems more succinctly, and to model actions from a planning domain independently of the actual problem as we can see in the following example.

Example 5 (Example 1, ctd.)
Here is the action call_{ij} *redefined in PDDL using universal effects :*

```
(:action call
     :parameters (?i - gossip ?j - gossip)
     :effect (and (forall (?s - secret)
                    (and (when (Kw ?i ?s) (Kw ?j ?s))
                         (when (Kw ?j ?s) (Kw ?i ?s))))))
```

6 Variants

Several variants of the Gossip Problem have been studied in the literature, and a survey of these alternatives and the associated results was published by Hedetniemi *et al.* (1988). We enumerate some of them and give an intuition on how they could be encoded as actions.

A first variant is to consider a non-complete graph instead of a complete one, that is, some people cannot call each other. We can add a variable $linked_{ij}$ such that the precondition of the action call_{ij} is now $linked_{ij} \wedge linked_{ji}$. In this case, if the graph is connected, the minimal number of calls to obtain the solution is either $2n-4$ or $2n-3$, depending on the structure of the graph (Harary & Schwenk, 1974).

We can also restrict calls to one-way communications, such as mails, and thus considering directed graphs. This can be achieved by only providing information from i in a call, thus call_{ij} is cut in half and becomes $\langle \top, \{\langle \mathbf{Kw}_i s_1, +\mathbf{Kw}_j s_1\rangle, \ldots, \langle \mathbf{Kw}_i s_n, +\mathbf{Kw}_j s_n\rangle\}\rangle$. (Thus call_{ij} is not equivalent to call_{ji} anymore.) If the directed graph is strongly connected, the minimal number of calls is $2n-2$ (Harary & Schwenk, 1974).

A well-known variant of the gossip problem is the NOHO ("No One Hears Own") version (Baker & Shostak, 1972) where nobody should hear their own secret. In this version, assuming agents cannot hide secrets, a call can only take place between i and j if the precondition $\neg\mathbf{Kw}_j s_i \wedge \neg\mathbf{Kw}_i s_j$ is satisfied, that is, j cannot tell i the secret of agent i and vice versa. Such a problem can only have a solution if n is even, in which case the minimal number of calls is still $2n-4$ (West, 1982).

An interesting variant for temporal planning is to consider time instead of calls, and thus suppose that several calls are executed in parallel. If the number of participants n is even, the time taken is $\lceil \log_2 n \rceil$, and if n is odd, it is $\lceil \log_2 n \rceil + 1$ (Bavelas, 1950; Landau, 1954; Knödel, 1975).

Agents can be interested in learning at least k secrets without caring which secrets they access. This can be modelled by changing the goal into a disjunction of all possible outcomes (conjunctions of $\mathbf{Kw}_i s_j$) such that each agent learns k secrets. A complete study of the problem (called the *partial gossiping problem* Richards & Liestman (1988)) and results depending on k can be found in (Chang & Tsay, 1996).

Finally, a last variant could be to find a sequence of calls such that no agent is allowed to lie by omission (at each call, she always tells all the secrets she knows), but in the end, some people should not know some secrets. This can actually be pretty easily achieved by only changing the goal (actions remain the same) to allow negative literals in the conjunctions. As an example, take $n=3$, a goal could be $\mathbf{Kw}_1 s_1 \wedge \mathbf{Kw}_1 s_2 \wedge \mathbf{Kw}_1 s_3 \wedge \mathbf{Kw}_2 s_1 \wedge \mathbf{Kw}_2 s_2 \wedge \mathbf{Kw}_2 s_3 \wedge \mathbf{Kw}_3 s_1 \wedge \neg\mathbf{Kw}_3 s_2 \wedge \mathbf{Kw}_3 s_3$ (everyone knows every secret except 3 that does not know the secret of 2). This can be achieved by the sequence of actions $\mathsf{call}_{13}, \mathsf{call}_{12}$. To the best of our knowledge, this variant has not been studied. This problem is not always solvable as some goals may not be reachable. For example, $\mathbf{Kw}_1 s_1 \wedge \mathbf{Kw}_1 s_2 \wedge \mathbf{Kw}_1 s_3 \wedge \mathbf{Kw}_2 s_1 \wedge \mathbf{Kw}_2 s_2 \wedge \neg\mathbf{Kw}_2 s_3 \wedge \mathbf{Kw}_3 s_1 \wedge \neg\mathbf{Kw}_3 s_2 \wedge \mathbf{Kw}_3 s_3$ (2 does not know the secret of 3 and conversely) is not attainable since call_{23} must never happen, so if 1 calls 2, then calls 3, then 3 will know the secret of 2 and if 1 calls 3 then 2, 2 will know the secret of 3.

What we observe is that all these variants are easy to model by modifying planning tasks, and hence can be directly encoded as DL-PA programs and PDDL tasks. This gives a sound and efficient method to fully solve this problem.

1. http ://helios.hud.ac.uk/scommv/IPC-14/planners.html

7 Conclusion and future work

In this article we have made a first step towards a realistic and provably-correct method for multiagent epistemic planning. Our use of a logic of action and knowledge, which is known to be sound, together with an efficient automatic planner (which is assumed to be correct in the case of classical planning with conditional effects) provides an efficient method for producing plans which are guaranteed to be correct. We intend to continue this line of research by incorporating other important aspects of multiagent planning, namely control (i.e. which agents are allowed to change the value of which variables) and mutual exclusion (to guarantee that at most one agent has control of a variable at any instant), along with epistemic aspects such as higher-order observation (visibility on visibility of agents). In the long term, we also aim to generalise this approach to temporal planning in which actions are durative and may overlap.

We can note that, although we have only mentioned PDDL in this paper, alternative approaches exist. For example, it is possible to code a planning problem containing actions with conditional effects directly into SAT and then use an efficient SAT solver to find a plan (Rintanen *et al.*, 2006).

Références

BAKER B. & SHOSTAK R. (1972). Gossips and telephones. *Discrete Mathematics*, **2**(3), 191–193.

BALBIANI P., HERZIG A. & TROQUARD N. (2013). Dynamic logic of propositional assignments : a well-behaved variant of PDL. In O. KUPFERMAN, Ed., *Proceedings of the 28th Annual IEEE/ACM Symposium on Logic in Computer Science (LICS)*, p. 143–152.

BAVELAS A. (1950). Communication patterns in task-oriented groups. *The Journal of the Acoustical Society of America*, **22**(6), 725–730.

CHANG G. J. & TSAY Y.-J. (1996). The partial gossiping problem. *Discrete Mathematics*, **148**(1-3), 9–14.

HARARY F. & SCHWENK A. J. (1974). The communication problem on graphs and digraphs. *Journal of the Franklin Institute*, **297**(6), 491–495.

HEDETNIEMI S. M., HEDETNIEMI S. T. & LIESTMAN A. L. (1988). A survey of gossiping and broadcasting in communication networks. *Networks*, **18**(4), 319–349.

KNÖDEL W. (1975). New gossips and telephones. *Discrete Mathematics*, **13**(1), 95.

LANDAU H. G. (1954). The distribution of completion times for random communication in a task-oriented group. *The Bulletin of Mathematical Biophysics*, **16**(3), 187–201.

MC DERMOTT D., GHALLAB M., HOWE A., KNOBLOCK C., RAM A., VELOSO M., WELD D. & WILKINS D. (1998). *PDDL—The Planning Domain Definition Language*. Rapport interne CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

RICHARDS D. & LIESTMAN A. L. (1988). Generalizations of broadcasting and gossiping. *Networks*, **18**(2), 125–138.

RINTANEN J., HELJANKO K. & NIEMELÄ I. (2006). Planning as satisfiability : parallel plans and algorithms for plan search. *Artif. Intell.*, **170**(12-13), 1031–1080.

WEST D. B. (1982). A class of solutions to the gossip problem, part I. *Discrete Mathematics*, **39**(3), 307–326.

A Generative Game-Theoretic Framework for Adversarial Plan Recognition

Nicolas Le Guillarme[1,3], Abdel-Illah Mouaddib[1], Xavier Lerouvreur[2], Sylvain Gatepaille[3]

[1] Models, Agents, Decision team, GREYC laboratory (UMR 6072),
University of Caen Basse-Normandie - Caen
{nicolas.leguillarme,abdel-illah.mouaddib}@unicaen.fr

[2] Signal Processing, Algorithms & Performance,
Airbus Defence and Space - Toulouse
xavier.lerouvreur@astrium.eads.net

[3] Advanced Information Processing,
Airbus Defence and Space - Val-de-Reuil
{nicolas.leguillarme,sylvain.gatepaille}@cassidian.com

Abstract : Adversarial reasoning is of the first importance for defence and security applications since it allows to (1) better anticipate future threats, and (2) be proactive in deploying effective responses. In this paper, we address the two subtasks of adversarial reasoning, namely adversarial plan recognition and strategy formulation, from a generative, game-theoretic perspective. First, a set of possible future situations is computed using a contextual action model. This projected situation serves as a basis for building a set of Markov games modeling the planning strategies of both the defender and his adversary. Finally, a library of critical plans for the attacker and a library of best responses for the defender are generated automatically by computing a Nash equilibrium in each game. The adversarial plan recognition task therefore consists of inferring a probability distribution over the set of possible plans of the adversary, while the strategy formulation problem reduces to the selection of the most appropriate response. Initial results on a urban warfare scenario suggest that our framework can be useful to model complex strategic interactions inherent to plan recognition in adversarial situations.

Keywords : Threat assessment, Adversarial reasoning, Plan recognition, Multi-agent planning, Game theory, Stochastic games.

1 Introduction

Defence and security activities, such as military operations or cybersecurity to name a few, are adversarial by nature. In such situations, where a decision-maker ("the defender") is threatened by one or several intelligent opponents with conflicting goals ("the attackers"), a key enabler for the success of operations is the ability for the defender to "read the opponent's mind" (Kott & McEneaney, 2006). Adversarial reasoning generally consists of inferring the intentions of an adversary from the available evidence so as to make possible the prediction of his future actions, which then enables the planning of an effective response. Adversarial reasoning can therefore be divided into two main subtasks, namely *adversarial plan recognition*, or the identification of an adversary's plans and goals on the basis of observations of his actions, and *strategy formulation*, where the defender has to formulate his own plan of action taking into account possible counteractions of the adversary.

Compared to other types of plan recognition, namely intended plan recognition, where the observed agent is cooperative, and keyhole plan recognition, where the observed agent is not aware or indifferent to the recognition process (Geib & Goldman, 2001), adversarial plan recognition is characterized by the fact that the observer (defender) and the observed agent (attacker) evolve in the same environment and that the observed agent is at least not cooperative, if not actively hostile to the inference of his plans. These characteristics of adversarial plan recognition result in two types of adversarial interactions between the defender and the attacker. First, since both agents evolve in the same environment and pursue conflicting

goals, each agent will have to reason about the planning strategy of his opponent to plan his actions, keeping in mind that his adversary will do the same. Secondly, should the observed agent be actively hostile to the recognition process, he will plan his actions accordingly so as to minimize the probability for his plans to be recognized (e.g. using concealment and/or deceptive actions), while the plan recognition process will have to reason about the adversary's planning strategy to predict his action.

Game-theory is a natural and popular formalism to address problems involving interactions between agents ranging from total cooperation to full hostility (Burkov & Chaib-draa, 2008). In (Lisý *et al.*, 2012), the problem of adversarial plan recognition is defined as an imperfect information two-player zero-sum game in extensive form between an actor and an observer. From a game-theoretic perspective, the solution is the strategies for both players that optimize the tradeoff between plan utility and plan detectability. The drawbacks of this method are that a plan library must be provided explicitly, the model does not handle simultaneous decisions, and the observer can only perform activity recognition actions. In (Braynov, 2006), the interaction between adversarial planning and adversarial plan recognition is modeled as a two-player zero-sum game over an attack graph representing the possible plans of the attacker. This approach addresses both types of strategic interactions between the observer and the observed agent. However, it relies on a plan recognition algorithm as a plugin. Closest to the present paper is (Chen *et al.*, 2007) where a set of optimal courses of action (COAs) for the enemy and friendly forces is generated using a decentralized Markov game. The state space is defined as the set of all the possible COAs for enemy, friendly, and neutral forces. The effectiveness of the approach has been demonstrated through combat simulation, but the authors are not explicit about the way their method can be used for goal/plan recognition.

In this work, we propose a generative game-theoretic framework to tackle the problem of adversarial plan recognition in a fully observable, multi-agent setting where actions have stochastic effects. The adversarial plan recognition task therefore consists of inferring a probability distribution over the set of possible plans of an agent whose behavior results from a finite two-player zero-sum stochastic game, where the players are respectively the attacker/observed agent and the defender/observer. The problem of strategy formulation then reduces to the selection of the most likely best response. Stochastic games are used to model the strategic interactions between the defender and the attacker planning strategies due to the common environment and their conflicting goals. The use of deception and concealment by the attacker is out of the scope of this paper and is left for future work. Since generative plan recognition techniques require the definition of an action model to encode the possible behaviors of the observed agent, we define a contextual action model based on the COI model of intentional action (Steinberg, 2007) which enables the automatic assessment of all opportunities for action available to each player in the current situation.

The rest of this paper is organized as follows. In Section 2, we provide a brief background on generative plan recognition and stochastic game theory. Section 3 contains an extensive description of PRESAGE, our generative game-theoretic framework for adversarial plan recognition. Section 4 applies this to a urban warfare scenario and presents preliminary experimental results. In Section 5 we discuss the current progress of our work and the planned extensions.

2 Background

2.1 Generative Plan Recognition

Most of previous research in plan recognition relies on an a priori, handcrafted plan library specifying all the possible plans of the observed agent (Avrahami-Zilberbrand & Kaminka, 2007; Geib & Goldman, 2009; Lisý *et al.*, 2012). This library is generally assumed to be exhaustive, which is quite impractical in real-world applications. Recently, a new paradigm known as *generative plan recognition* (also called *model-based* plan recognition or plan recognition *by inverse planning*) has been proposed by Baker et al. (Baker *et al.*, 2009) and further studied in a series of work by Ramírez and Geffner (Ramırez & Geffner, 2009, 2010, 2011). In plan recognition by inverse planning, observed agents are assumed to be rational, i.e. they will plan optimally to achieve their goals. The first advantage of generative methods is that the need for an explicit declaration of the possible agent behaviors as a plan library is replaced by an implicit encoding, using an agent action model and a set of possible goals, thus making generative methods far more flexible and less dependent on the availability of expert knowledge. The second advantage is that the set of optimal plans can be generated automatically using state-of-the-art planners, including classical planners (Ramırez & Geffner, 2009, 2010), Markov Decision Processes (MDPs) (Baker *et al.*, 2009), and partially-observable MDPs (Ramırez & Geffner, 2011).

2.2 Stochastic Games

Definition

Stochastic games - also called Markov games - have been introduced by Lloyd Shapley (Shapley, 1953) as an extension of MDPs to the multi-agent case. In a stochastic game, the players perform joint actions that determine both the new state of the environment, according to transition probabilities, and the reward obtained by each agent. Formally, a stochastic game is defined as a tuple $\langle Ag, S, \mathbf{A}, \{R^i, i = 1,...,|Ag|\}, T\rangle$ where

- $Ag = \{1,...,|Ag|\}$ is a finite set of players.
- S is a finite set of states.
- $A_i = \{a^i_1,...,a^i_{|A_i|}\}$ is a finite set of actions available to player i.
- $\mathbf{A} \equiv \times_{i\in Ag} A_i$ is the set of joint actions
- $R^i : S \times \mathbf{A} \to \mathbb{R}$ is the payoff function of player i.
- $T : S \times \mathbf{A} \times S \to [0,1]$ is the transition function.

The game is played in discrete time steps. In each time step t, the players choose their actions simultaneously and the joint action $\mathbf{a} = \{a^1,...,a^{|A_g|}\}$ is obtained. Each agent i receives a reward $R^i(s_t, \mathbf{a})$ depending on the current state of the environment and the joint action, and the players are transferred to the next state s_{t+1} according to the transition function T. A policy $\pi_i : S \to \Delta A_i$ for agent i defines for each state of the game a local, mixed strategy, i.e. a probability distribution over the set of available actions.

Solution concepts

Given a joint strategy $\pi = \langle \pi_i, \pi_{-i}\rangle$ where π_{-i} is the joint strategy of all players except player i, the expected utility of π for player i is defined in each $s \in S$ as the expected value of the utility function of normal form games (Burkov & Chaib-draa, 2008)

$$u^\pi_i(s) = E_{\mathbf{a}\in\mathbf{A}}[R^i(s,\mathbf{a})] \tag{1}$$

The state utilities $U^\pi_i(s)$ for player i associated with joint policy π can be quantified using the same performance criteria than for MDPs, i.e. the long-term expected value over i's reward. For instance, using the γ-discounted criterion

$$\begin{aligned} U^\pi_i(s) &= E\left[\sum_{t=0}^{\infty} \gamma^t u^\pi_i(s_t) | s_0 = s\right] \\ &= u^\pi_i(s) + \gamma \sum_{\mathbf{a}\in\mathbf{A}} \sum_{s'\in S} T(s,\mathbf{a},s')\pi^{\mathbf{a}}(s) U^\pi_i(s'), \end{aligned} \tag{2}$$

where $\pi^{\mathbf{a}}(s)$ is the probability of joint action $\mathbf{a}$ in s given joint policy π, s_0 is the initial state of the game and $\gamma \in [0,1]$ is the discount factor. For other performance criterion, see (Garcia & Rachelson, 2008).

The concept that is most commonly used as a solution of non-cooperative stochastic games is the one of Nash equilibrium, i.e. a combination of strategies where each player selects the best response to the other players' strategies, and no player can improve its utility by unilaterally deviating from this strategy. Formally, a joint strategy $\pi^* = \langle \pi^*_i, \pi^*_{-i}\rangle$ is a Nash equilibrium if

$$\forall s \in S,\ \forall i \in Ag,\ \forall \pi_i \in \Pi_i, \qquad U^{\langle \pi^*_i, \pi^*_{-i}\rangle}_i(s) \geq U^{\langle \pi_i, \pi^*_{-i}\rangle}_i(s), \tag{3}$$

where Π_i is the set of policies available to agent i. The existence of at least one Nash equilibrium for 2-player stochastic games has been demonstrated by Shapley (Shapley, 1953).

3 Plan Recognition using Stochastic Games

In this section, we describe PRESAGE (Plan REcognition using StochAstic GamEs), a generative game-theoretic framework for adversarial plan recognition. This framework models the strategic interactions between the decision-making strategies of an attacker (observed agent) and a defender (observer) as Markov games. As any other generative method, our framework requires the definition of an action model. We

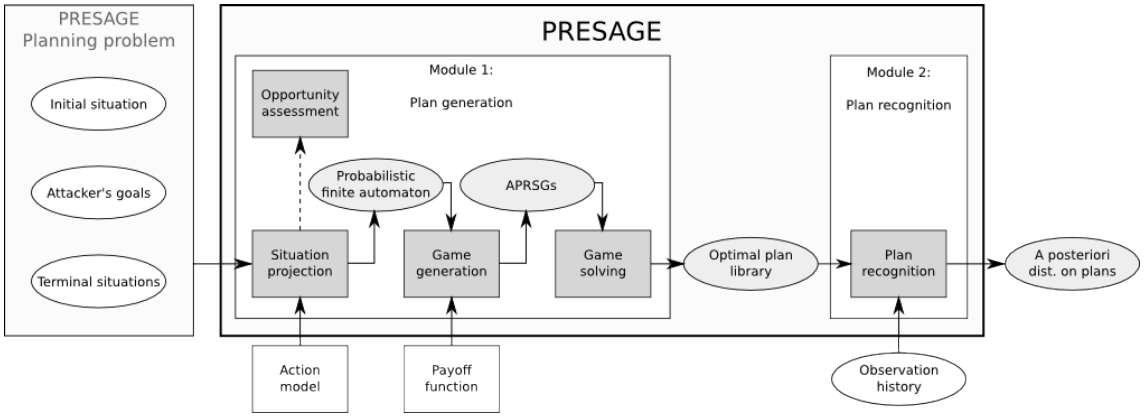


Figure 1: PRESAGE general architecture.

propose an adaptation of the COI model of action (Steinberg, 2007) to the domain of automated planning so that the opportunities (equivalent to MDPs' actions) available to each player can be automatically assessed from the situation and used to generate valid plans. Finally, we design a probabilistic plan recognition method whose aim is to create and maintain a belief state over the set of optimal strategies of the attacker from observations of his behavior.

3.1 General Architecture

Figure 1 is a representation of PRESAGE functional architecture. PRESAGE is divided into two independent modules. The first module (Module 1) is dedicated to the automatic generation of the optimal plan library. Starting from the definition of a planning problem P, including the initial situation and the set of possible goals of the attacker, the plan generation module executes a three-step procedure to build the library of possible plans for the attacker:

1. **Situation projection:** first, the set of possible future situations is generated and modeled as a *probabilistic finite automaton* Σ (Stoelinga, 2002), whose transitions are labeled by joint opportunities of the attacker and the defender. Opportunities for action available to each actor are assessed in each state of Σ using a generic opportunity assessment engine.

2. **Game generation:** for each possible goal g of the attacker, we build an Adversarial Plan Recognition Stochastic Game (APRSG) Γ_g by considering each state of Σ where g is achieved as a terminal state, and by representing each remaining state as a two-player zero-sum static game using a predefined payoff function.

3. **Game solving:** each APRSG is solved independently of the others using off-the-shelf solving algorithms so as to obtain an optimal plan library Π_P^* containing one optimal strategy for each possible goal of the attacker.

The second module (Module 2) encapsulates a plan recognition engine which, given a plan library and an observation history of both actors' actions, returns an a posteriori distribution over the set of possible plans of the attacker.

3.2 Assessing Opportunities for Action

Our action model is an adaptation of the COI model of threat to the planning domain. This model was first proposed in (Steinberg, 2005; Little & Rogova, 2006) and has been further extended in (Steinberg, 2007) to represent any intentional action. It considers the *capability*, *opportunity*, and *intent* of an agent to carry out actions on some targets to be the necessary and sufficient factors to characterize these actions:

Capability the possession of the resources and/or skills required to perform an action.

Opportunity "the right context" to perform an action, i.e. the presence of an operating environment in which potential targets are susceptible to be acted upon.

Intent the plans and goals an agent wants to achieve.

These action components are tightly interrelated. For instance, the intent of an agent can evolve in light of his capabilities and current opportunities, e.g. by changing his target. Inversely, the agent's intent can motivate the acquisition of new capabilities/opportunities, etc. Since our goal is to infer the intent of an agent from observations of his actions as restrained by its capabilities and opportunities, intent is implicit in our model and we postulate that it is constrained by the rationality principle. We also assume that we have full and certain knowledge of the capabilities of each agent. We therefore focus on the opportunity assessment task which consists of determining which action(s) can be performed on which target(s) in the current situation and estimating the expected outcome of these actions. To understand the concept of opportunity assessment, let us consider the example depicted on Figure 2.

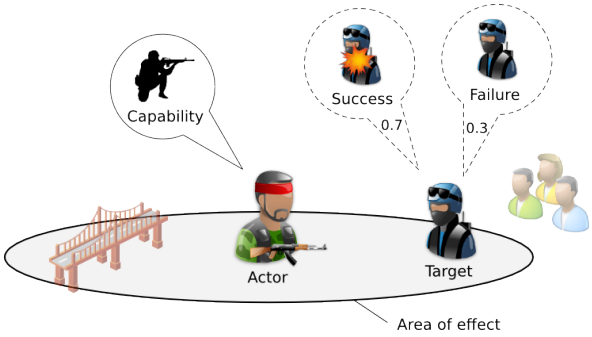


Figure 2: An example of opportunity assessment.

In this situation, the actor owns a resource *assault rifle* which provides him with the capability to perform action *shooting*. However, if the agent wants this action to have an effect on a target t, he must first acquire the opportunity to act upon t. Given a situation s, we will consider that an agent has the opportunity to perform action a on a target t if:

- the agent has the capability to perform action a in s
- the agent has access to target t in s,
- target t is vulnerable to action a in s.

We propose a simple, yet expressive opportunity model that takes into account the three aforementioned conditions and associates probabilities to the possible consequences of an action, namely success or failure. Enhanced action definitions may include more than two outcomes.

Definition 1. *An opportunity o^i for actor i is a tuple $\langle a,t,p\rangle$, where a is an action, t is the target of the action, and p is the probability of success of action a when performed on t.*

Given a situation s, the goal of the opportunity assessment engine is to compute for each actor i, the set $O_i(s)$ of opportunities available to i in s. In our model, each capability (feasible action) is associated with one area of effect (AoE).

Definition 2. *The area of effect associated with a capability a is a tuple $AoE(a) = \langle \mathscr{P}, P_a(success)\rangle$ where $\mathscr{P}$ is a set of preconditions necessary for a target t to be considered accessible (e.g. lying within a spatial area), and $P_a(success)$ is the probability of success of action a.*

Definition 3. *Assuming that actor i has the capability to perform action a, there exists an opportunity o^i for i to perform action a on a target t iff*

1. *t satisfies the accessibility conditions listed in $\mathscr{P}$,*
2. *the probability of success $p > 0$.*

p is determined by both the probability of success of the action $P_a(success)$ and the vulnerability of the target t to action a when t satisfies the accessibility conditions (written $t \in AoE(a)$).

$$\begin{aligned} p &= P(success(a), t \in AoE(a), vulnerable(t,a)) \\ & \quad P_a(success)P(vulnerable(t,a)|success(a), t \in AoE(a)) \end{aligned} \tag{4}$$

The vulnerability of a target to an action may be seen as its ability to thwart some possible effects of this action. For instance, considering the two possible effects of the *shooting* action to be *target killed* (success) and *target missed* (failure), we can assume that a target wearing a bulletproof vest will be less vulnerable to this action, i.e. the probability to obtain effect *target killed* on this particular target will be smaller than with an unprotected target.

Definition 4. *The vulnerability of a target t to an action a is the conditional probability $V_t(a)$, where $V_t(a) = 0$ means that t is not vulnerable to action a and $V_t(a) = 1$ represents the highest level of vulnerability.*

The overall probability of success p is therefore given by

$$p = P_a(success) \times V_t(a) \tag{5}$$

Example 1. *In the situation depicted in Figure 2, the only opportunity for the attacker is to attack the vulnerable ($V_{blueforce}(shooting) = 1.0$) blue force with a probability of success $P_{shooting}(success) = 0.7$; the civilian group cannot be acted upon because it does not lie within the (spatial) AoE, and the attacker does not possess the capability bomb-bridge required to act upon the bridge. Therefore, we have $O_i(s) = \{\langle shooting, blueforce, 0.7\rangle\}$.*

3.3 PRESAGE Planning Problem

A PRESAGE planning problem is defined as a tuple $P = \langle Ag, I, G, \mathscr{T}\rangle$, where Ag is a finite set of actors, I is the initial situation, G is the set of goals of the attacker, and $\mathscr{T}$ is a set of terminal situations. Elements of G are not necessarily elements of $\mathscr{T}$, thus offering the possibility to account for an adversary trying to achieve several goals sequentially.

3.4 Situation Projection

The goal of the situation projection engine is, given the definition of a PRESAGE planning problem, to generate a probabilistic finite automaton Σ which aims at formally representing the possible future situations and their dynamics. Σ is defined as a tuple $\langle s_0, S, S^{\mathscr{T}}, \mathbf{O}, T\rangle$. The definitions for the different parameters are compiled in Table 1.

Par.	Description	Expression
S	A finite set of states	$S = \{s_0, ..., s_{\|S\|}\}$
s_0	The initial state	$s_0 \in S$
$S^{\mathscr{T}}$	A finite set of terminal states	$S^{\mathscr{T}} \subset S$
O_i	A mapping assigning to each $s \in S \setminus S^{\mathscr{T}}$ the set of opportunities for player i in s	$O_i(s) = \{o_1^i, ..., o_n^i\}$ with $o_j^i = (a_j^i, t_j^i, p_j^i)$
O(s)	The set of possible joint opportunities in state $s \in S \setminus S^{\mathscr{T}}$	$\mathbf{O(s)} \equiv \times_{i \in Ag} O_i(s)$
SO	The set of all possible joint opportunity profiles	$SO = \{(s, \mathbf{o}) \| s \in S \setminus S^{\mathscr{T}}, \mathbf{o} \in \mathbf{O(s)}\}$
T	A transition function	$T : SO \times S \to [0,1]$

Table 1: Definitions for the parameters of Σ

Given a PRESAGE planning problem P and an opportunity assessment engine $\mathtt{OA}$ with $\forall s \in S \setminus S^{\mathscr{T}}, i \in Ag$, $\mathtt{OA}(s,i) \to O_i(s)$, we build Σ using the situation projection algorithm shown in Algorithm 1. The algorithm first checks the terminal conditions (lines 5-6). If the current state s is not terminal, opportunities available to each player in s are assessed using the opportunity assessment engine so as to build the set of possible

joint opportunities $\mathbf{O}(\mathbf{s})$ (line 8-10). Line 11 loops over all possible joint opportunities $\mathbf{o}$ in $\mathbf{O}(\mathbf{s})$. Given a state s and a joint opportunity $\mathbf{o}$, the `findSuccessors` function (whose algorithm is not detailed here due to a lack of space) computes the set of successor states S' obtained when executing joint action $\mathbf{o}$ in s, as well as the transition probability $P(s'|s,\mathbf{o})$ for each $s' \in S'$ (line 12). Each successor state s' in S', if not already processed, is added to S and the corresponding transition probability is used to update the transition function T (lines 13-16). The recursive situation projection procedure is applied to each state until no new state can be added to S (line 17).

Algorithm 1: Situation projection algorithm

```
   Data: P = ⟨Ag, I, G, 𝒯⟩, OA
   Result: Σ
1  begin
2      s_0 ← I, S ← {s_0}, S^𝒯 ← {}, T ← {}
3      project (P, s_0, S, S^𝒯, T)
4  procedure project (P, s, S, S^𝒯, T)
5      if s ∈ 𝒯 then
6          S^𝒯 ← S^𝒯 ∪ {s}
7      else
8          foreach i ∈ Ag do
9              O_i(s) ← OA (s, i)
10         O(s) ← ×_{i∈Ag} O_i(s)
11         foreach o ∈ O(s) do
12             S' ← findSuccessors (s, o)
13             foreach s' ∈ S' such as P(s'|s,o) ≠ 0 do
14                 T(s, o, s') ← P(s'|s, o)
15                 if s' ∉ S then
16                     S ← S ∪ {s'}
17                     project (Ag, P, s', S, S^𝒯, T)
```

3.5 Stochastic Games Generation and Solving

The projected situation Σ is used as a basis for building one Adversarial Plan Recognition Stochastic Game (APRSG) planner per possible goal of the attacker. An APRSG for goal $g \in G$ is a finite two-player zero-sum stochastic game which is formally defined as a tuple $\Gamma_g = \langle \Sigma_g, \{R_g^i, i = 1,...,|Ag|\}\rangle$ where $\Sigma_g = \langle Ag, S_g, S_g^{\mathscr{T}}, \mathbf{O_g}, T_g\rangle$ and $R_g^i : SO_g \to \mathbb{R}$ is the payoff function of player i. The only addition to the classical definition of stochastic games (as discussed above) is the set $S_g^{\mathscr{T}} \subset S_g$ of terminal states for goal g.

Players

An APRSG is a two-player game between an attacker (player 1, the observed agent), which tries to maximize his payoff, and a defender (player 2, the observer), which aims at minimizing player 1's payoff. Therefore we have $Ag = \{1,2\}$.

State space and transition function

The game generation procedure is shown in Algorithm 2. First, S_g and $S_g^{\mathscr{T}}$ are initialized as copies of their respective equivalent set in Σ. Then, each state s in S_g corresponding to a situation where goal g is achieved is marked as a terminal state of game Γ_g (lines 3-5). The `pruneDisconnectedStates` function (line 6) acts by removing all the edges originating from terminal states ($\forall s \in S_g^{\mathscr{T}}, \forall \mathbf{o} \in \mathbf{O_g}(\mathbf{s}), \forall s' \in S_g, T(s,\mathbf{o},s') \leftarrow 0$). States in S_g and $S_g^{\mathscr{T}}$, which are no more connected to the initial state s_0 due to the fact that goal states are now considered as terminal, are pruned using depth-first-search algorithm. The state space S_g of game Γ_g is therefore at most as large as the initial state space S. Finally, T_g and $\mathbf{O_g}(\mathbf{s})$ are respectively defined as the restriction of T and $\mathbf{O}(\mathbf{s})$ to the set $S_g \setminus S_g^{\mathscr{T}}$ (lines 7-8). This algorithm is applied for each attacker's goal $g \in G$. A simple example illustrating this game generation procedure is depicted in Figure 3.

Algorithm 2: Game generation algorithm

Data: Σ, g, $\{R_g^i, i = 1,...,|Ag|\}$
Result: Γ_g
begin
  $S_g \leftarrow copy(S)$, $S_g^{\mathscr{T}} \leftarrow copy(S^{\mathscr{T}})$
  foreach $s \in S_g$ **do**
    if *g is achieved in s* **then**
      $S_g^{\mathscr{T}} \leftarrow S_g^{\mathscr{T}} \cup \{s\}$
  `pruneDisconnectedStates`$(s_0, S_g, S_g^{\mathscr{T}}, T)$
  $T_g \leftarrow T_{|S_g \setminus S_g^{\mathscr{T}}}$
  $\mathbf{O_g}(\mathbf{s}) \leftarrow \mathbf{O}_{|S_g \setminus S_g^{\mathscr{T}}}(\mathbf{s})$
  $\Sigma_g \leftarrow \langle Ag, S_g, S_g^{\mathscr{T}}, \mathbf{O_g}, T_g \rangle$
  $\Gamma_g \leftarrow \langle \Sigma_g, \{R_g^i, i = 1,...,|Ag|\} \rangle$
end

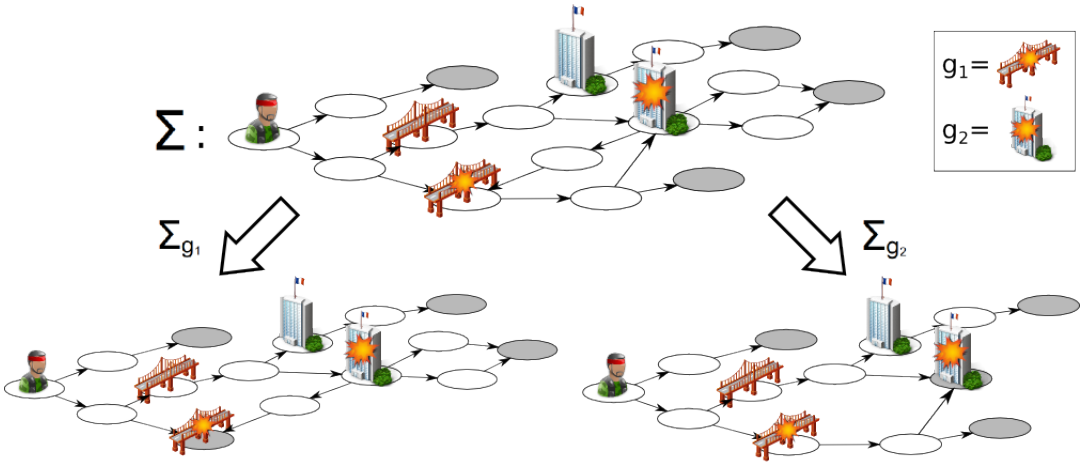


Figure 3: Illustration of the game generation procedure. In this example, the attacker has two possible goals: bombing the bridge (g_1) and/or attacking the embassy (g_2). Terminal states are represented as grey ellipses and do not necessarily corresponds to goal states. By defining terminal situations distinct from goal states, we allow an attacker to achieve a sequence of goals. From the projected situation Σ resulting from the situation projection procedure, the game generation algorithm derives two Markov games Σ_{g_1} and Σ_{g_2} associated to goal g_1 and g_2 respectively, by converting their respective goal states into terminal states and pruning unreachable successor states. The attacker can achieve the sequence of goal (*bombBridge, attackEmbassy*) by executing the optimal strategy π^*_{1,g_1}, then switch to strategy π^*_{1,g_2} once he reach the first goal state.

Payoffs and optimal strategy

The payoffs for player i playing game Γ_g are given by the application-dependent payoff function R_g^i which associates to each joint opportunity and in every state of the game, a gain in terms of utility for player i. Each state of an APRSG can be seen locally as a two-player zero-sum static game. Consequently, the payoff function must satisfy the following constraint: $\forall s \in S_g, \forall \mathbf{o} \in \mathbf{O_g}(\mathbf{s})$, $\sum_{i \in Ag} R_g^i(s, \mathbf{o}) = 0$. A two-player zero-sum static game can be solved using the Minimax algorithm (Neumann, 1928). Assuming that player 1 wants to maximize its payoff while player 2 wants to minimize player 1's payoff, the optimal strategy for player 1 (resp. player 2) is called the *maximin* (resp. *minimax*) strategy, and an equilibrium is reached if *maximin* = *minimax*. Such an equilibrium always exists in mixed strategy. Let $\mathbf{R_g}(\mathbf{s})$ be a $m \times n$ matrix where m (resp. n) is the number of opportunities available to player 1 (resp. player 2) in s, and $\mathbf{R_g}(\mathbf{s})_{i,j} = R_g^1(s, o_i^1, o_j^2)$, the mixed maximin strategy $x^* = (x_1^*, ..., x_m^*)$ in s is obtained by solving the following linear program (Nisan *et al.*, 2007):

$$\begin{array}{rl} \textit{maximize} & v \\ \textit{s.t} & \sum_i x_i \, \mathbf{R_g(s)}_{i,j} \geqslant v, \quad for \; 1 \leqslant j \leqslant n \\ \textit{and} & \sum_i x_i = 1. \end{array}$$

The mixed minimax strategy $y^* = (y_1^*, ..., y_n^*)$ in state s is obtained by solving the analogous linear program:

$$\begin{array}{rl} \textit{minimize} & v \\ \textit{s.t} & \sum_i y_i \, \mathbf{R_g(s)}_{i,j}^T \leqslant v, \quad for \; 1 \leqslant j \leqslant m \\ \textit{and} & \sum_i y_i = 1. \end{array}$$

An APRSG Γ_g is played in discrete time steps. In each time step t, each $i \in Ag$ chooses an opportunity o_t^i from his set of available opportunities $O_i(s_t)$, where s_t is the current state of the game. Decisions are simultaneous and the joint opportunity $\mathbf{o_t} \in \mathbf{O_g(s_t)}$ is obtained. Each player i received a reward $R_g^i(s_t, \mathbf{o_t})$ and the players are transferred to the next state s_{t+1}. A policy $\pi_{i,g} : SO_g \rightarrow [0,1]$ for agent i defines for each state of Γ_g a mixed strategy, i.e. a probability distribution over the set of available opportunities.

We use Shapley's algorithm (Shapley, 1953), an extension of the Value-Iteration algorithm to the case of stochastic games, to find one Nash equilibrium $\pi_g^* = (\pi_{1,g}^*, \pi_{2,g}^*)$ in each Γ_g. Shapley's algorithm iteratively builds a normal form game $M(s)$ for each state $s \in S_g$ by using a value function which accounts for both the immediate utility of choosing an action in s and the long-term utility of playing the equilibrium starting from the next state. The optimal joint strategy $\pi^*(s)$ is then obtained by calculating the maximin and minimax strategies for the two-player zero-sum static game $M(s)$. Once a Nash equilibrium π_g^* is known for each Γ_g, we can finally build the set $\Pi_P^* = \{\pi_{1,\,g}^* | \forall g \in G\}$ of optimal plans for the attacker and the set $br(\Pi_P^*) = \{\pi_{2,\,g}^* | \forall g \in G\}$ of best responses for the defender given problem P.

3.6 Plan Recognition

The aim of the plan recognition module is to infer the current intent (goal and plan) of the attacker from observations of his actions. The past behaviors of both the attacker and the defender are represented as an observation history H_h which keeps record of the last h states visited by the players and of the joint opportunities that were played in these states: $H_h(t) = \{(s_{t-(h-1)}, \mathbf{o_{t-(h-1)}}), ..., (s_t, \mathbf{o_t})\}$.

According to the rationality principle underlying the generative plan recognition paradigm, if the attacker intends to reach the desired end-state $g \in G$, then he will follow the optimal policy $\pi_g^* \in \Pi_P^*$ to achieve g. Our plan recognition procedure acts by creating and updating at each time step t a belief state defined over the set Π_P^* of optimal plans for the attacker. This belief state is represented by a belief function $b_t : \Pi_P^* \longrightarrow [0,1]$ with $b_t(\pi_g^*) = P\left(\pi_t = \pi_g^* | H_h(t), s_t\right)$, where π_t is the policy of the attacker at time t. Hence $b_t(\pi_g^*)$ represents the belief of the defender that, at time t, the attacker is following policy $\pi_g^* \in \Pi_P^*$ given the observation history $H_h(t)$.

Proposition 1. *Let o_t be the opportunity played by the attacker in state s_t. Given history $H_h(t)$, the belief that the attacker is following policy π_g^* at t is given by:*

$$b_t(\pi_g^*) \propto P(o_t | \pi_g^*, s_t) \times \prod_{i=1}^{h-1} P(o_{t-i} | \pi_g^*, s_{t-i}) T_g(s_{t-i}, \mathbf{o_{t-i}}, s_{t-i+1}),$$

with the constraint that b_t is a probability distribution over Π_P^*.

Proof. From Bayes' rule:

$$P\left(\pi_g^* | H_h(t), s_t\right) = P\left(H_h(t) | \pi_g^*, s_t\right) P\left(\pi_g^*, s_t\right) / P\left(H_h(t), s_t\right),$$

Let $H_n = H_{h-n}(t)$ (i.e. the last $h-n$ observations at t), and $\tau = t-(h-1)$. The expression for $P\left(H_h(t) | \pi_g^*, s_t\right)$ results from the following recursive decomposition

$$P\left(H_n | \pi_g^*, s_{\tau+n}\right) = P\left(o_{\tau+n} | \pi_g^*, s_{\tau+n}\right) T_g\left(s_{\tau+n}, \mathbf{o_{\tau+n}}, s_{\tau+(n+1)}\right) P\left(H_{n+1} | \pi_g^*, s_{\tau+(n+1)}\right)$$

with $H_{n+1} = H_n \setminus \{(s_{\tau+n}, \mathbf{o_{\tau+n}})\}$. Therefore, starting from state $s_{t-(h-1)}$ ($n = 0$), we obtain

$$P\left(H_h(t) | \pi_g^*, s_t\right) = P(o_t | \pi_g^*, s_t) \times \prod_{i=1}^{h-1} P(o_{t-i} | \pi_g^*, s_{t-i}) T_g(s_{t-i}, \mathbf{o_{t-i}}, s_{t-i+1}).$$

□

The use of a finite length history allows us to handle the fact that the adversary may change his intent during the execution of the plan recognition procedure. By keeping only "up-to-date" information, we prevent old observations inconsistent with the new intent from deteriorating the solution of the inference process. Another important remark is that the plan recognition algorithm introduced in Proposition 1 does not require the plan library to contain optimal strategies, but can be used to recognize any type of plan, with the constraint however that these plans are represented as MDP policies.

4 Experimental results

4.1 Scenario description

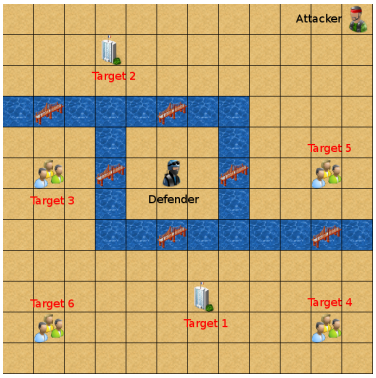


Figure 4: Initial situation of our urban warfare scenario.

We perform our experiments on the scenario depicted in Figure 4. In this scenario, the defender is the Blue force (bf) with capabilities "move to", "attack red force", and "do nothing", and the attacker is the Red force (rf) with capabilities "move to", "attack embassy", "attack civilians", "attack blue force" and "do nothing". Each capability is associated with an area of effect specifying the type of target which can be acted upon, the action range, and the probability of success of the action. The set of possible goals of the attacker is $G = \{g_i = destroyed(target_i) | i = 1...6\}$. Of course, the true goal of the attacker is hidden to the defender. The mission of the defender is to determine which target has been chosen and to protect this target by eliminating the attacker. In this particular set of experiments, we assume that the attacker has only one single goal (hence he will not try to achieve several goals sequentially) and therefore the set of terminal situations is $\mathscr{T} = G \cup \{destroyed(rf), destroyed(bf)\}$. We also define a simple application-dependent payoff function $R_g^i : S \to \mathbb{R}$ which associates immediate rewards to each state of a game given a goal $g \in G$:

$$R_g^i(s) = \begin{cases} = & 200, & \text{if } s \models g, \\ = & 50, & \text{if } s \models destroyed(bf), \\ = & -100, & \text{if } s \models destroyed(rf), \\ = & \alpha d(rf, target) - \beta d(rf, bf), & \text{otherwise,} \end{cases}$$

with $d(rf, target)$ the distance between the attacker and his target, $d(rf, bf)$ the distance between the attacker and the defender, and $\alpha + \beta = 1$.

4.1.1 Experiments

We use the scenario defined in the previous section to evaluate the ability of our generative game theoretic framework to recognize the current plan of a rational adversary. We also verify that the recognition of the attacker's plan is rapid enough for the defender to select and implements the appropriate response before the attacker reaches his goal. With this in mind, we define two possible strategy selection policies for the defender. Let $b_t(\pi_g^*)$ be the belief state of the defender at time t:

WAIT : the defender waits until $\exists g \in G$ such as $b_t(\pi_g^*) = 1$, then executes the best response $br(\pi_g^*)$.

MAX : the defender executes the best response $br(\pi_g^*)$ with $g = argmax_{g' \in G}\, b_t(\pi_{g'}^*)$, and chooses randomly between goals with the same maximum likelihood.

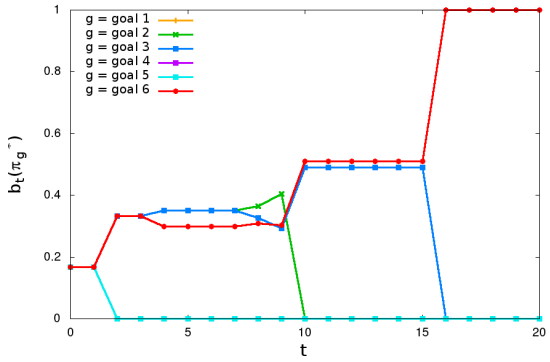


Figure 5: Evolution of the defender's belief about the real goal of the attacker.

Figure 5 shows the belief state $b_t(\pi_g^*)$ for $g = destroyed(target_6)$ as a function of time. As we can see from this plot, the true goal of the attacker is correctly recognized by our plan recognition algorithm at $t = 16$. This is not a surprise: since the attacker is assumed to be strictly rational, we are guaranteed to recognize his real goal in finite time (in the worst case when the attacker executes the final action leading to the achievement of the goal), unless a terminal situation such as $destroyed(bf)$ or $destroyed(rf)$ is achieved before the true goal has been recognized. The real question is therefore to know if our plan recognition engine is able to recognize the real goal of the attacker before it is fulfilled.

Policy	Goal 1	Goal 2	Goal 3	Goal 4	Goal 5	Goal 6
MAX (%)	98.9 ± 1.0	75.5 ± 2.6	100	80.4 ± 3.8	89.2 ± 3.8	100
WAIT (%)	100	42.1 ± 4.5	100	17.6 ± 2.7	100 ± 0	100

Table 2: Mean percentage of instances during which the real goal of the attacker was recognized before it was achieved.

Table 2 contains the results of the evaluation of the ability of our plan recognition method to infer the real goal of the adversary before he achieves this goal. For instance, when the true goal of the adversary is goal 1 and the strategy selection policy of the defender is **MAX**, our approach is able to recognize the goal of the attacker before the destruction of the target in 98.9% of cases (mean over 100 runs of 100 scenario instances). Our method performs better on average when associated to the **MAX** selection policy, since it seems to lead the attacker in being less unclear about the goal he is chasing. This is particularly true for ambiguous goals such as goal 2 (whose optimal policy is very similar to those of goals 3 and 6) and goal 4 (similar to goal 5 and 1).

From Table 2, we saw that our plan recognition method is able to infer the real goal of an adversary before the achievement of this goal in most cases. But does this inference occurs soon enough for the defender to implement the appropriate response and defend the target? To evaluate this point, we define several strategy formulation policies which will serve as baseline values for comparison with the **WAIT** and **MAX** policies defined above:

RO : the defender selects an opportunity randomly in each state of the game (uniform distribution).

RS* : the defender selects a goal g randomly in each state of the game and executes the best response $br(\pi_g^*)$.

RS : the defender selects a goal g randomly at the beginning of the game and executes the best response $br(\pi_g^*)$.

CLOSE : the defender selects the goal g which is closest from the attacker in each state of the game and executes the best response $br(\pi_g^*)$.

BR : the defender always executes the best response $br(\pi_g^*)$ where g is the true goal of the attacker.

The quality of each policy is evaluated by averaging the number of scenario instances during which the attacker successfully achieved his goal over a total of 100 runs of 100 instances for each goal. From the

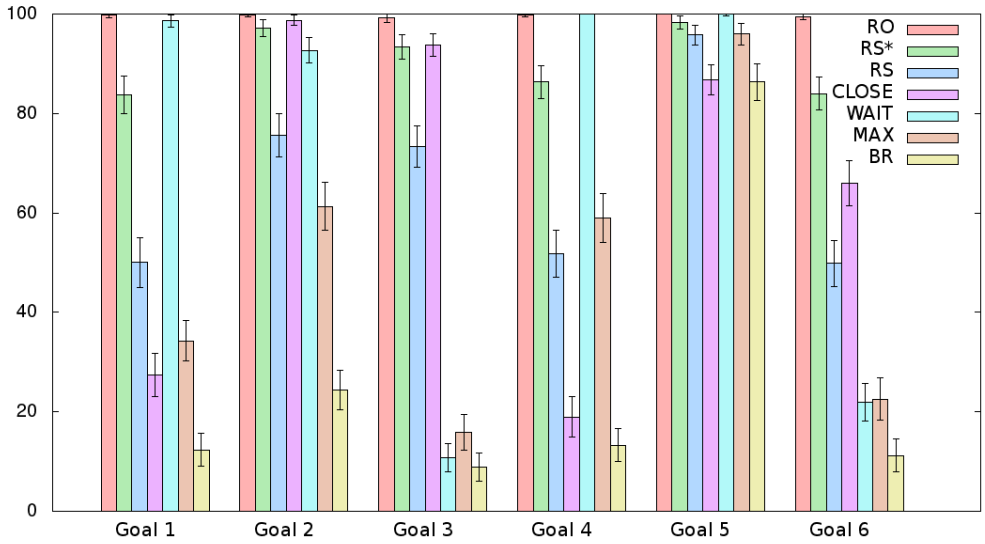


Figure 6: Mean attacker's percentage of success when using **MAX** and **WAIT**, compared to **BR** and other baselines.

results depicted in Figure 6, we can see that the **MAX** strategy formulation policy, which relies on our plan recognition engine, performs better on average than every other baseline (except **BR** of course). In a few cases however (goal 3 and 6), it may be preferable for the defender to wait to be certain about the real goal of the attacker before acting. But the disappointing results of the **WAIT** policy on the other 4 goals tend to confirm this famous quote from Prussian general and military theorist Carl Von Clausewitz

> The greatest enemy of a good plan is the dream of a perfect plan (Clausewitz, 1832).

5 Conclusion and Future Work

In this paper, we presented PRESAGE, a generative game theoretic framework for adversarial plan recognition in stochastic multi-agent environments. This framework generates a library of optimal plans for a rational attacker from the definition of a planning problem. We proposed an algorithm for situation projection relying on a simple yet expressive contextual action model. We showed how a set of Markov games can be generated automatically from this projected situation to model the planning processes of both the defender and the attacker. By computing one Nash equilibrium for each one of these games, we built a library of optimal plans for the attacker and a set of best responses for the defender. This library was later exploited by a probabilistic plan recognition algorithm to infer the current plan of the adversary from observations of his behavior. Finally, we demonstrated the capability of our adversarial plan recognition to assist a decision-maker in selecting the most appropriate response to a given threat.

An interesting direction for future works would be to relax the attacker's rationality assumption and to adapt our plan recognition algorithm to the case of an adversary with bounded rationality. We also plan to extend our framework in order to deal with an adversary which would be actively hostile to the inference of his plans. This would require the ability to detect deceptive actions performed by the attacker. We would also have to relax the assumption of full observability of the opponent's actions, since he may use concealment. Currently, our APRSGs are defined as two-player zero-sum stochastic games and therefore, they can only model the strategic interactions between one defender and one attacker. Our intuition is that our framework can be generalized quite directly to the 1 vs N and N vs N cases.

6 Acknowledgments

We thank Bruno Zanuttini, assistant professor with habilitation at the University of Caen Basse-Normandie, for helpful comments and corrections.

References

AVRAHAMI-ZILBERBRAND D. & KAMINKA G. A. (2007). Incorporating observer biases in keyhole plan recognition (efficiently!). In *AAAI*, volume 7, p. 944–949.

BAKER C. L., SAXE R. & TENENBAUM J. B. (2009). Action understanding as inverse planning. *Cognition*, **113**(3), 329–349.

BRAYNOV S. (2006). Adversarial planning and plan recognition: Two sides of the same coin. In *Secure Knowledge Management Workshop*.

BURKOV A. & CHAIB-DRAA B. (2008). Stochastic games. In *Markov Decision Processes in Artificial Intelligence*, volume 1, p. 229–276. Wiley Online Library.

CHEN G., SHEN D., KWAN C., JR. J. B. C., KRUGER M. & BLASCH E. (2007). Game theoretic approach to threat prediction and situation awareness. *Journal of Advances in Information Fusion*, **2**(1), 35–48.

CLAUSEWITZ C. V. (1832). *On War*. Ed./trans. Michael Howard and Peter Paret. Princeton University Press, 1976, revised 1984.

GARCIA F. & RACHELSON E. (2008). MDPs: models and methods. In *Markov Decision Processes in Artificial Intelligence*, volume 1, p. 1–38. Wiley Online Library.

GEIB C. W. & GOLDMAN R. P. (2001). Plan recognition in intrusion detection systems. In *Proceedings of DARPA Information Survivability Conference & Exposition II, 2001. DISCEX'01.*, volume 1, p. 46–55: IEEE.

GEIB C. W. & GOLDMAN R. P. (2009). A probabilistic plan recognition algorithm based on plan tree grammars. *Artificial Intelligence*, **173**(11), 1101–1132.

KOTT A. & MCENEANEY W. M. (2006). *Adversarial reasoning: computational approaches to reading the opponentÂŠs mind*. CRC Press.

LISÝ V., PÍBIL R., STIBOREK J., BOSANSKÝ B. & PECHOUCEK M. (2012). Game-theoretic approach to adversarial plan recognition. In *ECAI*, p. 546–551.

LITTLE E. G. & ROGOVA G. L. (2006). An ontological analysis of threat and vulnerability. In *9th International Conference on Information Fusion*, p. 1–8: IEEE.

NEUMANN J. V. (1928). Zur theorie der gesellschaftsspiele. *Mathematische Annalen*, **100**(1), 295–320.

NISAN N., ROUGHGARDEN T., TARDOS E. & VAZIRANI V. V. (2007). *Algorithmic game theory*, volume 1. Cambridge University Press Cambridge.

RAMIREZ M. & GEFFNER H. (2009). Plan recognition as planning. In *Proceedings of the 21st international joint conference on Artifical intelligence. Morgan Kaufmann Publishers Inc*, p. 1778–1783.

RAMIREZ M. & GEFFNER H. (2010). Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI 2010)*.

RAMIREZ M. & GEFFNER H. (2011). Goal recognition over pomdps: Inferring the intention of a pomdp agent. In *IJCAI*, p. 2009–2014.

SHAPLEY L. S. (1953). Stochastic games. *Proceedings of the National Academy of Sciences of the United States of America*, **39**(10), 1095.

STEINBERG A. N. (2005). An approach to threat assessment. In *8th International Conference on Information Fusion*, p. 1–8: IEEE.

STEINBERG A. N. (2007). Predictive modeling of interacting agents. In *10th International Conference on Information Fusion*, p. 1–6: IEEE.

STOELINGA M. (2002). An introduction to probabilistic automata. *Bulletin of the EATCS*, **78**(176-198), 2.

IMDDI et SPIMDDI : apprentissage incrémental de diagrammes de décisions pour une architecture SDYNA

Jean-Christophe Magnan and Pierre-Henri Wuillemin

Sorbonne Universites, UPMC, Univ Paris 06, UMR 7606, LIP6, Paris, France

Résumé : En théorie de la Décision, dans le cadre de la planification, l'approche factorisée des Processus Décisionnels Markoviens (FMDP) a débouché sur des algorithmes efficaces utilisant les Arbres de Décision (SVI, SPI) ou les Diagrammes de Décision Algébriques (SPUDD). Pouvoir apprendre de tels modèles factorisés d'environnements stochastiques offre un avantage certain lors de la recherche de politique optimale. Des approches telles que l'architecture SDYNA proposent d'intégrer l'apprentissage et la planification. Toutefois, l'état de l'art pour l'apprentissage incrémental (Utgoff *et al.* (1997)), pour la planification théorique de la décision structurée (Hoey *et al.* (1999)) ou pour l'apprentissage par renforcement (Degris *et al.* (2006a); Strehl *et al.* (2007); Chakraborty & Stone (2011)) requièrent que le problème soit caractérisé à l'aide de variables binaires et/ou utilise des structures de données améliorables en termes de compacité. Magnan & Wuillemin (2013) propose l'utilisation de Diagrammes de Décision MultiModaux (MDDs) comme structures de données plus efficaces et décrit des algorithmes de planification dédiés à ces structures. Cet article s'intéresse à l'apprentissage de tels modèles factorisés. Son principal résultat est l'algorithme IMDDI d'apprentissage en-ligne de MDDs qui offre des améliorations significatives de compacité du modèle et de temps d'apprentissage. De plus, aucune connaissance *a priori* sur le nombre de modalités des variables aléatoires n'est requise – propriété intéressante pour un algorithme d'apprentissage en-ligne qui le démarque également de l'état de l'art. Enfin, cet algorithme permet une toute nouvelle instanciation de l'architecture SDYNA exploitant les MDDs : l'instance SPIMDDI.

1 Introduction

En planification, le Processus Décisionnel de Markov (MDP) formalise les actions d'un agent dans un environnement stochastique. L'objectif d'un MDP est de trouver une politique optimale qui donne la meilleure action à effectuer dans chaque configuration de l'environnement (état). Les algorithmes VI (Itération de la Valeur, Bellman (1957)) et PI (Itération de la Politique, Howard (1960)) permettent d'obtenir ces politiques optimales.

Chaque étape de ces deux algorithmes a une complexité linéaire en la taille de l'espace d'états. Néanmoins, pour des problèmes réalistes, l'espace d'états a tendance à être de très grande taille car ils sont en effet souvent multidimensionnels (la "Malédiction de la Dimensionnalité" (Bellman, 1961)). Il devient alors inenvisageable de rechercher une politique optimale à l'aide de VI ou PI, malgré leur complexité linéaire.

Dans la littérature, plusieurs solutions existent pour contourner ce problème. Par exemple, la formalisation d'abstractions dans l'espace d'état permet de traiter simultanément de grands ensembles d'états dans lesquels l'agent se comporte fondamentalement de la même façon. En conséquence, le nombre d'états à visiter à chaque itération de VI ou PI diminue considérablement. Exploitant cette idée, les MDPs factorisés ont prouvé leur efficacité grâce à des algorithmes structurés. Ainsi, Boutilier *et al.* (1999) suggère d'utiliser des arbres de décision (DTs) comme base pour le processus d'abstraction (SVI et SPI : *Structured* VI et PI). Plus récemment, des structures de données plus compactes comme les Diagrammes de Décisions Algébriques (ADDs) ou les Diagrammes de Décision Multimodale (MDDs) ont permis d'obtenir des algorithmes plus efficaces : SPUDD (Hoey *et al.*, 1999) et SPUMDD (Magnan & Wuillemin, 2013).

Toutefois ces algorithmes impliquent en prérequis de connaître le modèle factorisé du problème à résoudre. Une telle connaissance est difficile à obtenir, en particulier pour les problèmes avec un très grand nombre d'états. De plus, pour les problèmes pratiques, il peut être difficile d'obtenir les données nécessaires à l'élaboration de ces modèles. C'est pourquoi il est intéressant d'avoir à disposition des algorithmes d'apprentissage incrémentals (ou en-ligne) qui utilisent des informations sur l'environnement obtenues en cours d'exploitation.

Dans le domaine de l'Apprentissage par Renforcement, l'agent a la charge de découvrir une représentation du monde et d'en déduire une politique optimale. Dans des approches telles que DYNA et DYNA-Q (Sutton, 1990), l'agent apprend itérativement un MDP et applique soit VI soit PI pour obtenir une politique optimale. Bien entendu, ces algorithmes font face à des difficultés lors du passage à l'échelle et il devient nécessaire de fournir à ces algorithmes la capacité de manipuler des représentations factorisées.

SPITI (Degris *et al.*, 2006a) étend l'apprentissage par renforcement aux représentations factorisées en se servant de DTs. Un algorithme d'apprentissage incrémental (ITI, Utgoff *et al.* (1997)) améliore les DTs définissant le MDP à chaque action de l'agent.

Pour la planification, les DTs sont néanmoins moins efficaces en termes de taille et, par conséquence, de manipulation que les ADDs ou les MDDs (Hoey *et al.* (1999) et Magnan & Wuillemin (2013)). Toutefois, il n'existe pas d'algorithme permettant de les apprendre de manière incrémentale. La principale contribution de cet article est de présenter un algorithme traitant ce problème pour les MDDs.

La Section 2 de cet article décrit les différents algorithmes de planification et d'apprentissage avec des modèles factorisés. La Section 3 décrit IMDDI, notre proposition pour l'apprentissage incrémental de MDDs. Les expérimentations de la Section 4 illustrent l'efficacité de IMDDI en termes de qualité des modèles appris et en termes de durée d'apprentissage. Cette section montre aussi l'intérêt de cet algorithme dans une instance de l'architecture SDYNA pour les MDDs : SPIMDDI.

2 Plannification et Apprentissage avec des modèles factorisés

2.1 Processus Décisionnel de Markov Factorisé

Un Processus Décisionnel de Markov Factorisé (FMDP, Puterman (2005)) est un processus de contrôle en temps discret qui modélise un environnement stochastique dans lequel un agent est amené à effectuer séquentiellement des actions. Un ensemble de variables $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$ est utilisé pour caractériser sans ambiguïté cet environnement. Ainsi chaque état s du système est une instanciation unique $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$ de $\mathbf{X}$ (avec $x_i \in D_{X_i}$, domaine de valeurs de la variable X_i).

L'agent dispose d'un ensemble A d'actions possibles. Effectuer une action $a \in A$ dans un état s (représenté par $\mathbf{x}$) affecte le système qui transite vers un nouvel état s' (représenté par $\mathbf{x}'$). Les probabilités $P(\mathbf{X}'|a, \mathbf{X})$ décrivent ces transitions en donnant pour tout état $\mathbf{x}'$ du système la probabilité de l'atteindre en effectuant l'action a dans l'état $\mathbf{x}$.

Une fonction récompense $r(\mathbf{X}', a, \mathbf{X})$ permet d'évaluer la pertinence d'avoir rejoint l'état $\mathbf{x}'$ en ayant effectué a en l'état $\mathbf{x}$. Sans perte de généralité, nous nous restreindrons dans cet article aux fonctions de récompense de la forme $r(\mathbf{X})$ qui ne prennent en compte que la pertinence d'être en l'état $\mathbf{x}$.

Une politique est une fonction $\pi(\mathbf{X}) \in A$ qui fournit dans tout état du système une action à effectuer. L'objectif est de trouver une politique optimale qui nous indique la meilleure action à effectuer en chaque état $\mathbf{x}$ afin de maximiser un critère, par exemple (dans cet article) la récompense totale espérée. VI et PI sont des algorithmes fréquemment utilisés pour identifier ces politiques optimales avec une complexité linéaire en fonction de la taille de l'espace d'états. Mais le cadre factorisé implique que cette complexité est exponentielle en fonction du nombre de dimensions de l'espace d'état. La résolution d'un problème réaliste devient donc souvent impossible.

Pour contourner cette difficulté, le cadre factorisé se donne pour objectif d'exploiter les indépendances conditionnelles et structurelles : pour une fonction donnée (probabilité de transition, récompense ou politique), plusieurs états sont similaires et produisent donc la même sortie pour cette fonction. L'idée est donc de les agréger et de les traiter comme un seul état abstrait. La caractérisation de l'espace d'état S à l'aide de l'ensemble de variables $\mathbf{X}$ permet d'exprimer les probabilités de transition, la fonction récompense, et la politique optimale comme fonctions de $\mathbf{X}$. À partir de là, trouver et exploiter les indépendances se fait naturellement en utilisant les graphes de fonction associés à ces fonctions.

Soit f une fonction de variables $X_1, \ldots, X_n$. Notons $f|_{X_i=b}$ la restriction de f à $X_i = b$. Le *support* de f est l'ensemble des variables dont f dépend réellement :

$$support(f) = \{X_i \,|\, \exists u, v \in D_{X_i} \text{ t.q. } f|_{X_i=u} \neq f|_{X_i=v}\}$$

Définition 1 (Graphe de fonction)
Soit f une fonction de $\{X_1, \ldots, X_n\}$. Un graphe dirigé acyclique (DAG) $G_f(N, E)$ est un graphe de fonction pour f ssi :

— *si f est constante alors G_f a un unique nœud $r \in N$. On note $r.val = f$*
— *si f est non constante alors G_f a un unique nœud $r \in N$ sans parent. De plus,*
  — *on note $r.var \in support(f)$ une variable associée à n,*
  — *$\forall u \in D_{r.var}, \exists! n_u \in N$ tel que :*
    ◦ *(r, n_u) est un arc de G_f, donc $(r, n_u) \in E$,*
    ◦ *subgraph$(n_u) = G_{f|_{r.var=u}}$*

Si un nœud n est terminal (sans enfant) alors n est lié à une valeur constante ($n.val$) sinon n est associé à une variable ($n.var$).

Puisque $\forall$ nœud n non terminal, subgraph$(n_u) = G_{f|_{n.var=u}}$, l'arc (n, n_u) peut être vu comme l'instanciation de $n.var$ suivant la modalité u. Il en découle que tout chemin dans le graphe est une instanciation partielle des variables de $\mathbf{X}$.

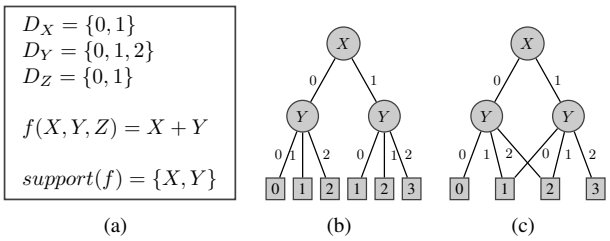


FIGURE 1 – Deux graphes de fonctions représentant la même fonction $f(X, Y, Z) = X + Y$

La Figure 1 montre qu'il peut y avoir plusieurs graphes de fonction pour une même fonction. Toutefois, ils ont tous en commun une propriété de compacité : aucune variable superflue n'est présente dans le graphe grâce à l'usage du $support(.)$. De plus, plusieurs chemins (donc plusieurs instanciations de variables) peuvent amener au même sous-graphe, caractérisant le fait que plusieurs restrictions peuvent être égales. Par exemple, dans la Figure 1(c), $f|_{\text{Nœud terminal 2}} = f|_{X=0,Y=2} = f|_{X=1,Y=1} = 2$.

Les graphes de fonctions permettent donc de représenter de manière compacte et efficace une fonction sur un espace d'état très grand. Ajoutons que des algorithmes dédiés aux opérations de combinaison (addition, multiplication, maximisation) sur ces graphes de fonctions permettent de créer des fonctions composées sans passer par une énumération explicite de l'espace d'état.

SVI et SPI (Boutilier *et al.*, 1999) sont les deux premiers algorithmes à exploiter des représentations sous forme de graphes de fonction. La structure utilisée est alors l'arbre décision (DT, cf. Figure 1(b)). Comme le montre la Figures 1(b), les DTs peuvent avoir des sous-graphes isomorphes. Cette redondance augmente sans nécessité la taille de la structure, et donc les temps de calcul.

SPUDD (Hoey *et al.*, 1999) exploite une structure de données plus compacte : les Diagrammes de Décision Algébriques (ADDs, Bahar *et al.* (1993)) (cf. Figure 1(c)). Plus récemment encore, Magnan & Wuillemin (2013) proposent d'utiliser les Diagrammes de Décision Multimodaux (MDDs) afin de mieux exploiter les variables multimodales. Les ADDs et les MDDs ont, en plus des propriétés communes aux graphes de fonction, des propriétés de réduction et d'ordonnancement.

Définition 2 (Graphe de fonction réduit)
G_f est réduit $\iff$ $\forall$ nœuds $n \neq n'$, $f|_n \neq f|_{n'}$.

Quand un graphe de fonction est réduit, deux sous-parties isomorphes sont nécessairement fusionnées.

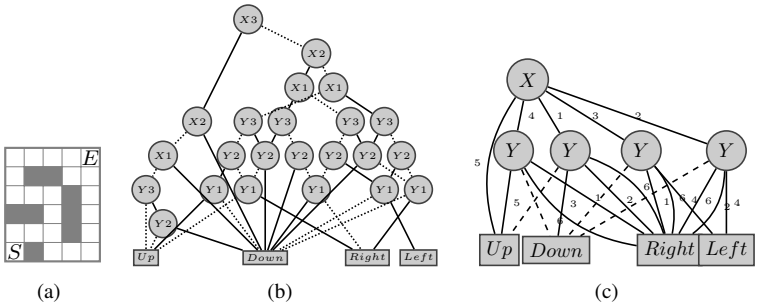
Définition 3 (Graphe de fonction ordonné)
Un graphe de fonction G_f est ordonné $\Leftrightarrow \exists \succ_{G_f}$ ordre total sur le $support(f)$, t.q. $\forall$ nœuds n_1, n_2 non terminaux de G_f,

$$n_2 \in desc(n_1) \Rightarrow n_1.var \succ_{G_f} n_2.var$$

Quand un graphe de fonction est ordonné, l'algorithme pour le réduire est polynomial.

2.2 Planification et apprentissage conjoints

Le manque de données (certaines régions de l'espace d'état n'ont pas encore été explorées) ou la nature même de ces données (typiquement pour des processus en-ligne tels que la prédiction de séries temporelles)



(a) (b) (c)

FIGURE 2 – La même politique optimale pour aller de S à E dans le labyrinthe (a) construite avec SPUDD (b) et avec SPUMDD (c). Un état est la position (X, Y) dans le labyrinthe. Les arcs en pointillé dans (c) indiquent des arcs par "défaut" qui représentent toutes les autres valeurs que peut assumer la variable. La compacité de même que la lisibilité de la solution est clairement améliorée sur les MDDs.

amènent à l'implémentation d'algorithmes en-ligne qui doivent pouvoir apprendre à partir d'un flux d'observations [1] expérimentales ξ. L'apprentissage par Renforcement se propose d'apprendre ces fonctions par essai-erreur (Dyna, Dyna-Q, Sutton (1990)).

Algorithm 1: Architecture globale de SDYNA (Degris *et al.*, 2006b)

1 **foreach** *pas de temps* t **do**
2 $s \leftarrow$ état courant;
3 $a \leftarrow$ Décision en fonction de $\pi_t(s)$; `// πt est la stratégie actuelle`
4 Effectuer a et observer s' et r;
5 Apprentissage "Factorisé" incrémental d'une observation $\xi = (s, a, s')$;
6 Planification "Factorisée" $\rightarrow \pi_{t+1}$;
7 **end**

Dans Degris *et al.* (2006b), l'architecture générale SDYNA (cf. algorithme 1) intègre l'apprentissage, la planification et la prise de décision en exploitant les représentations factorisées. Une instance de cette architecture nécessite une structure de données pour modéliser le problème (DT, ADDs, etc.), une version de PI ou VI qui manipule des instances de cette structure et un algorithme d'apprentissage d'instances de cette structure. Il est à noter que l'abstraction induite dans SDYNA par l'utilisation de représentations factorisées favorise la généralisation : par exemple, lors de la planification, une action "optimale" sera attribuée à des états qui n'ont pas encore été visités. SPITI, une instanciation de cette architecture, proposée dans le même article, utilise les DTs comme représentation en s'appuyant sur l'algorithme SPI (Boutilier *et al.*, 1999) pour l'étape de planification (ligne 6) et sur l'algorithme ITI (Utgoff *et al.*, 1997) pour l'induction des DTs durant la phase d'apprentissage (ligne 5). Dans SPITI, la prise de décision (ligne 3) est un ϵ-*greedy* qui, au lieu d'utiliser la politique optimale courante (exploitation), peut essayer de nouvelles actions non optimales afin d'améliorer la connaissance du modèle (exploration). Il est à noter que le compromis exploration-exploitation peut être bien mieux pris en compte, par exemple dans des approches telles que sur Factored-E^3 (Kearns & Koller, 1999) , Factored-RMAX (Strehl *et al.* (2007), Strehl (2007)), Met-RMAX (Diuk *et al.*, 2009), LSE-RMAX (Chakraborty & Stone, 2011).

Les sous-sections 2.3 et 2.4 couvrent brièvement SPITI. La sous-section 2.5 s'intéressera à une telle implémentation pour les MDDs.

2.3 ITI : apprentissage incrémental d'arbre

Plusieurs algorithmes d'apprentissage des DT depuis un corpus d'observations ont été proposés : CART (Breiman *et al.*, 1984), C4.5 (Quinlan, 1993), etc. Ces algorithmes apprennent à partir d'une base de données d'apprentissage complète et n'offrent donc pas la possibilité de mettre à jour l'arbre appris lors de l'acquisition d'une nouvelle observation (mise à jour incrémentale). Des approches fenêtrées pourraient être mises en place mais elles impliquent la reconstruction complète de l'arbre pour chaque nouvelle ob-

1. Dans cet article, une observation est un vecteur $\xi = (z_1, \cdots z_k)$ d'observations de k variables aléatoires. Le flux de données en-ligne prend alors la forme d'une séquence $(\xi_t)_{t\in\mathbb{N}}$.

servation. Par contre, ITI (Utgoff *et al.*, 1997) propose un apprentissage incrémental de cette structure de données. Nous rappelons ici brièvement les principes de l'algorithme ITI.

Dans l'algorithme ITI, à chaque nœud N du DT appris est associé un ensemble Ω_N d'observations acquises qui sont compatibles avec l'instanciation des variables en N. Ainsi, les ensembles d'observations placés aux feuilles de l'arbre forment une partition de l'ensemble de toutes les observations Ω. De même, l'ensemble d'observation associée à la racine est Ω. Ajouter une nouvelle observation ξ signifie l'intégrer à tous les ensembles compatibles avec ξ.

La structure de l'arbre dépend uniquement de ces ensembles d'observations : au nœud interne N est associée la variable qui subdivise le "mieux" l'ensemble d'observations Ω_N. Pour sélectionner cette variable, ITI maximise le ratio du gain d'information (*Information Gain Ratio*) comme critère entre les différentes distributions crées par la sélection de chaque variable disponible en ce nœud. Chaque nouvelle observation ξ induit une mise à jour des ensembles associés aux nœuds compatibles avec ξ et donc une possible remise en cause des variables sélectionnées en ces nœuds.

Nous renvoyons les lecteurs à Utgoff *et al.* (1997) pour une présentation plus complète de l'algorithme.

2.4 SPITI : Planification et apprentissage avec des DTs

SPITI est une instanciation de l'architecture SDYNA où les DTs sont utilisés pour effectuer la factorisation du modèle. Ainsi pour chaque $P(X'_i|A,\mathbf{X})$ et pour la fonction récompense $r(\mathbf{X})$, un DT doit être appris incrémentalement. Degris *et al.* (2006a) utilisent dans SPITI une version de ITI où le critère de sélection est le test du χ^2 au lieu du ratio de gain d'information et indiquent deux raisons majeurs pour ce choix : White & Liu (1994) montre que le test du χ^2 n'a pas de biais vis-à-vis des variables multimodales. De plus le test du χ^2 permet à SPITI de faire du pré-élagage : une feuille ne sera pas développée si aucune variable ne satisfait au test du χ^2.

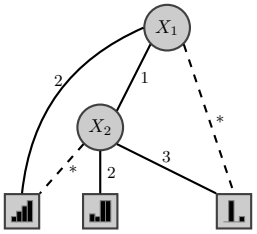


FIGURE 3 – Un MDD représentant une distribution de probabilité $P(Y|X_1,X_2)$. Les feuilles contiennent les distributions de probabilité Y. Ce MDD indique que $P(Y|X_1=2) = P(Y|X_1=1, X_2 \notin \{2,3\})$ et que $P(Y|X_1 \notin \{1,2\}) = P(Y|X_1 = 1, X_2 = 3)$. Ces égalités représentent des indépendances contextuelles dans P (Boutilier *et al.*, 1996).

Ainsi chaque variable de $\mathbf{X}$ peut être insérée dans les DTs. Dans les DTs représentant les $P(X'_i|A,\mathbf{X})$, les feuilles sont associées à une distribution de probabilité sur X'_i (voir Figure 3). La distribution de probabilité de X'_i en une feuille N est estimée à partir des fréquences de X'_i dans Ω_N. Dans le DT représentant $r(\mathbf{X})$, les feuilles sont associées à une valeur réelle. Enfin, les politiques sont représentées par des DTs dont les feuilles contiennent des actions.

2.5 Architecture SDYNA avec MDDs

Plusieurs articles s'intéressent à l'apprentissage de MDDs. L'approche proposée dans Oliver (1993) et Oliveira & Sangiovanni-Vincentelli (1994) repose sur l'induction d'un arbre en utilisant l'un des algorithmes de l'état de l'art (CART, C4.5) dans un premier temps. Puis cet arbre est réordonné de telle manière que la recherche de sous-parties isomorphes soit simplifiée. Enfin, ces sous-parties isomorphes sont fusionnées en utilisant comme critère le principe de Minimum Description Lenght.

Kohavi (1994); Kohavi & hsin Li (1995) propose une approche récursive : en suivant un ordre fixé à l'avance des variables, il procède à une décomposition des observations de la base d'apprentissage suivant les modalités que peut prendre la variable de l'appel récursif courant. Cette décomposition se poursuit jusqu'à ce que, en toutes feuilles, la fonction représentée assume la même valeur pour toutes les observations présentes. Une opération de réduction est alors effectuées pour obtenir un diagrammes de décision.

Ces algorithmes ne peuvent s'appliquer à un apprentissage en-ligne puisque les arbres sont déduits de bases de données fixes. Ainsi, ajouter de nouvelles observations à la base de données demande d'apprendre

un tout nouvel arbre. Ces approches ne sont donc pas de bons candidats pour une architecture SDYNA. Être capable de mettre à jour l'arbre, et sa version réduite en MDD, sans avoir à parcourir à nouveau toute la base de données serait un grand atout qui n'a pas encore été proposée à notre connaissance. Dans la section suivante, nous proposons un tel algorithme.

3 Apprentissage en-ligne de Diagrammes de Décision Multimodale

Dans cette section, nous décrivons IMDDI, un nouvel algorithme pour l'apprentissage incrémental de Diagrammes de Décision Multimodale. Sans perte de généralité, cette présentation se focalisera sur l'estimation de la distribution de probabilités d'une variable multimodale Y d'après un ensemble de variables multimodales $\mathbf{X} = \{X_1, \ldots, X_n\}$ (cf figure 3). En effet, apprendre le modèle factorisé de transition d'un état $(X_1, \cdots, X_n)$ consiste à apprendre plusieurs probabilités de transition (une par variable caractérisant le système) (Boutilier *et al.*, 1999). De plus, l'apprentissage d'autres fonctions telle que la fonction Récompense repose sur des algorithmes similaires : seule la fusion de feuilles "identiques" est à spécifier.

Dans une architecture SDYNA, l'agent récupère un flot continu d'informations sur son environnement au fur et à mesure qu'il l'expérimente. Apprendre un nouvel MDD pour chaque nouvelle observation serait évidemment trop coûteux, ce qui implique la mise en place d'un apprentissage incrémental afin d'exploiter de manière optimale ce flux. Notre algorithme doit donc être capable de mettre à jour la structure de données qu'il apprend au lieu de la reconstruire complètement lorsque de nouvelles informations arrivent.

Il y a deux différences majeures entre un DT (Figure 4(a)) et un MDD (Figure 4(c)) : premièrement, un MDD est contraint par un ordre global sur les variables. Les variables doivent apparaître sur chaque branche du MDD en accord avec cet ordre global. Cet ordre a naturellement un fort impact sur la compacité du MDD. Nous appelons Arbre de Décision Ordonné (ODT) un DT ayant cette contrainte d'ordre global sur ses variables (Figure 4(b)).

Deuxièmement, un MDD peut réduire sa taille en fusionnant des sous-arbres "identiques". Ainsi, les nœuds de la Figure 4(c) qui ont plusieurs parents représentent des sous-arbres fusionnés. La complexité de réduction d'un ODT T en un MDD est en $O(|T| \cdot \log |T|)$ où $|T|$ est le nombre de nœuds dans T (Bryant, 1986).

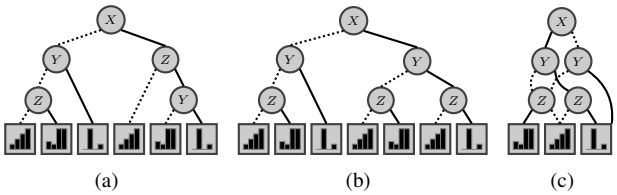


FIGURE 4 – Une distribution de probabilité représentée à l'aide d'un DT, (b) un ODT ($X \succ Y \succ Z$) et (c) un MDD (utilisant le même ordre global).

Dans le cadre de l'apprentissage non-incrémental de MDD, l'approche proposée par Oliver (1993) consiste à simplement mixer un algorithme d'apprentissage de DT avec un algorithme de transformation de DT en MDD. Un premier algorithme incrémental (nommé ITI+DD dans le reste de l'article) pourrait suivre cette approche : i) simplement utiliser un algorithme incrémental comme ITI pour apprendre un DT, ii) choisir un ordre global sur les variables et construire l'ODT et finalement iii) construire le MDD par fusion de nœuds. Dans ITI+DD, les étapes ii) et iii) sont donc effectuées à chaque nouvelle observation. L'étape iii) est en général peu coûteuse. Par contre, l'étape ii) (recherche d'un ordre global pour un MDD de plus petite taille) est NP-difficile (Friedman & Supowit, 1990).

Un point décisif de notre algorithme consistera donc en une heuristique incrémentale d'estimation de l'ordre optimal qui permettra de se passer de cette étape coûteuse : nous proposons une version modifiée de ITI qui manipule un ODT en lieu et place d'un DT. Ensuite, et seulement si nécessaire, le MDD sera déduit de cet ODT maintenu incrémentalement (étape iii). L'algorithme 2 présente ces étapes pour une observation ξ produite par un flux en-ligne.

Dans la sous-section suivante, nous décrivons IOTI, un algorithme qui produit de façon incrémentale ODT. Puis nous décrivons l'induction du MDD à partir de ce ODT.

Algorithm 2: Induction Incrémental de MDD (IMDDI) pour $P(Y|X_1,\ldots,X_n)$

Data: une observation $\xi = (X_1,\ldots,X_n,Y)$

```
1 begin
2   | Mettre à jour l'ODT avec ξ;          // IOTI : cf. algorithmes 3 et 4
3   | if mise à jour requise then
4   |   | Réduire l'arbre en MDD ;          // étape iii) : cf. algorithme 5
5   | end
6 end
```

3.1 IOTI : Induction Incrémentale d'Arbre de Décision Ordonné

L'algorithme IOTI est basé sur ITI. Toutefois cet algorithme a été modifié afin de mettre à jour non seulement la structure d'arbre mais aussi l'ordre global. Ces modifications sont décrites dans les sous-sections suivantes : étant donnée une observation ξ à intégrer dans le modèle, comment sélectionner la variable en chaque nœud de l'arbre, comment mettre à jour la structure en accord avec ξ, et enfin comment ξ modifie-t-elle l'ordre global.

3.1.1 Sélection de Variable

L'algorithme ITI n'a pas de restriction concernant la mesure utilisée pour la sélection de variable. Dans notre cas d'étude, nous devons gérer des variables multimodales. La mesure employée ne doit donc pas avoir de biais envers ces variables. Nous proposons d'utiliser la statistique G (Mingers, 1989) qui n'a en effet pas de biais pour les variables multimodales (White & Liu, 1994). De plus, le test G est similaire au test du χ^2 si le nombre d'échantillons est suffisamment grand. Enfin, d'après Dunning (1993), la statistique G gère mieux les événements rares que la statistique du χ^2.

Soit N le nœud auquel nous voulons soit installer une variable (si N est une feuille), soit nous assurer que la variable actuellement installée est la plus pertinente. Soit $\mathbf{V}_N$ l'ensemble des variables potentiellement candidates en ce nœud N (rappelons que chaque variable ne peut apparaître qu'une fois dans tout chemin). Pour chaque variable $X_i \in \mathbf{V}_N$, N garde un tableau de contingence comptant le nombre d'observations passées en ce nœud pour chaque combinaison de modalités de X_i et Y. La statistique G suit alors la distribution du χ^2 ayant un degré de liberté $\nu = (|X_i|-1)(|Y|-1)$

$$G(X_i) = 2 \cdot \sum_{x_i \in DomX_i} \sum_{y \in DomY} n_{x_i,y} \ln \frac{n_{x_i,y}}{e_{x_i,y}} \tag{1}$$

où $e_{x_i,y} = \frac{n_{.,y} n_{x_i,.}}{N}$ avec $n_{.,y}$ et $n_{x_i,.}$ étant les totaux marginaux ($n_{.,y} = \sum_{x_i} n_{x_i,y}$). Dans ce cas-ci, la statistique G mesure à quel point les observations que nous avons faites sont décorrélées des observations que nous aurions pu faire si X_i et Y étaient indépendants. Ainsi, plus cette statistique G est grande, plus X_i et Y sont liés ; plus X_i semble pertinent à installer dans le nœud N.

Ajoutons que, comme la statistique du χ^2, la statistique G a une propriété intéressante : elle peut être utilisée comme critère de pré-élagage pour empêcher l'arbre de trop grandir. Pour se faire, il suffit de regarder la p-valeur associée à la statistique G calculée. La p-valeur sert en effet de niveau de validité. Si la p-valeur n'excède pas un certain seuil, cela implique que l'hypothèse nulle ne devrait pas être rejetée. Dans notre cas, l'hypothèse nulle est que X_i et Y ne sont pas fortement dépendants : X_i ne devrait pas être installé en N.

IOTI utilise donc la p-valeur comme mesure pour la sélection de variables. La variable ayant la plus forte p-valeur sera retenue. Si cette p-valeur est supérieure à un seuil fixé, nous installons la variable au nœud N. Dans le cas contraire, et si le nœud N n'est pas une feuille, N est transformé en feuille. Soulignons le fait que ce seuil a le même sens que le niveau de fiabilité pour un test d'indépendance du χ^2 (ou G).

3.1.2 Ajouter une nouvelle observation ξ

Une observation ξ consiste en une instanciation de toutes les variables observées $\langle X_1,\cdots,X_n,Y\rangle$. Par construction, il existe un unique chemin de la racine à une feuille du ODT qui est compatible avec ξ. Ajouter ξ au ODT consiste à mettre à jour les bases Ω_N et les statistiques de chaque nœud N de ce chemin. Avec X_N la variable installée en un nœud interne N, $p_G^N(X_i)$ la p-valeur de la variable $X_i \in \mathbf{V}_N$ au nœud N,

$c_N(v)$ le nœud fils de N pour la modalité v de X_N et enfin $\xi(A)$ la modalité prise par la variable A dans ξ, l'algorithme 3 décrit formellement cette mise à jour de la structure de l'ODT.

Algorithm 3: Ajout d'un nouvelle observation ξ pour $P(Y|X_1,\ldots,X_n)$

Data: l'observation $\xi = \{x_1,\ldots,x_n,y\}$ ajouter dans l'ODT T

```
1  begin
2      Nœud N ← racine de T;
3      repeat
4          Ajouter ξ à Ω_N;
5          foreach variable X_i ∈ V_N do
6              Mettre à jour p_G^N(X_i);
7          end
8          if N n'est pas une feuille then
9              Nœud N ← c_N(ξ(X_i));
10         end
11     until N est une feuille;
12 end
```

3.1.3 Mettre à jour l'ODT

Une fois que les statistiques ont été mises à jour, l'étape suivante consiste à réviser la topologie de l'arbre. En effet, du fait de l'insertion de la nouvelle observation, les variables précédemment installées peuvent ne plus être les variables pertinentes à installer.

Cette opération est rendue plus complexe par l'existence d'un ordre global. Pour mettre à jour l'ODT, IOTI doit donc i) trouver un (aussi bon que possible) ordre global, ii) assurer que cet ordre est respecté sur chaque branche et iii) assurer que la meilleure variable possible soit installée en chaque nœud de l'ODT. Pour toute observation qui ne change pas la structure, cette opération doit être aussi peu coûteuse que possible.

Pour remplir ces conditions, la stratégie que nous proposons est de vérifier variable par variable la pertinence de sa position dans l'ordre global courant. Tant que les variables gardent leurs positions, aucune modification structurelle n'est à faire dans l'ODT.

Pour décider quelle variable devrait être la suivante dans l'ordre global, un score est attribué à toutes les variables restantes. Ces scores se servent des statistiques G maintenues en chaque nœud mais les agrègent sur plusieurs nœuds regroupés dans une frontière $\mathcal{B}$. $\mathcal{B}$ est l'ensemble des nœuds où une variable doit actuellement être installée en accord avec l'ordre global. Cette frontière est initialisée au singleton qu'est la racine de l'arbre ; elle contiendra les feuilles de l'ODT à la fin de la mise à jour (cf. algorithme 4).

IOTI calcule un score agrégé en sommant les p-valeur obtenues en chaque nœud $N \in \mathcal{B}$ pondérées par les proportions d'observations aux nœuds N ($|\Omega_N|$) comparées au nombre total d'observation ajoutées à l'ODT ($|\Omega|$). La variable avec le plus haut score est alors choisie comme variable suivante dans l'ordre global.

Pour chaque nœud de la frontière, la variable choisie sera installée en ce nœud si la p-valeurs est au-delà du seuil fixé. Cette installation se fait de la même manière que dans l'algorithme ITI. Si la variable est installée, le nœud est alors retiré de la frontière, remplacé par tous ses fils. Notons qu'une fois cette itération terminée, tout nœud de la frontière (présent ou à venir) ne pourra plus choisir d'installer cette variable.

Le critère d'arrêt de cet algorithme 4 doit prendre en compte deux possibilités : ou bien toutes les variables de $\mathbf{X}$ ont été insérées dans l'ordre global ; ou bien aucune des variables restantes ne peut être installée en aucun des nœuds de la frontière courante (i.e. toutes les p-valeurs sont en deçà du seuil). Quand l'algorithme se termine, la frontière contient toutes les feuilles de l'ODT. Par conséquence, tout nœud de la frontière qui n'est pas une feuille à ce moment-là doit être élagué.

Il est simple de vérifier que si la nouvelle observation ne change pas la structure de l'ODT, les seuls calculs effectués par l'algorithme 4 sont les estimations de poids en chaque nœud de l'arbre.

3.2 De IOTI à IMDDI : génération des diagrammes de décision

Une fois l'ODT généré, il faut le réduire en un MDD. Cette étape va donc fusionner les sous-parties isomorphes du graphe. Cette fusion se fait à l'aide d'un algorithme (bottom-up) polynomial.

Algorithm 4: Mise à jour de la structure de l'ODT pour $P(Y|a, X_1, \ldots, X_n)$

Data: un ODT T après l'ajout d'un ξ (avec l'Algorithme 3)

begin
  $\mathcal{B} = \{$ racine R de $T\}$; // frontière
  $\mathcal{F} = \mathbf{X}$; // Variables non insérées dans la mise à jour de l'ordre global
  repeat
    foreach *variable* $X_i \in \mathcal{F}$ **do**
      $p_G(X_i) = \sum_{N \in \mathcal{B}} \frac{|\Omega_N|}{|\Omega|} \cdot p_G^N(X_i)$;
    end
    $V \leftarrow \arg\min_{X_i \in \mathcal{F}} p_G(X_i)$;
    $\mathcal{B}' \leftarrow \mathcal{B}$;
    foreach $N \in \mathcal{B}$ **do**
      if $p_G^N(V) \geq \tau^1$ **then**
        Installer V au nœud N;
        $\mathcal{B}' \leftarrow \mathcal{B}' \setminus \{N\} \cup \bigcup_{v \in Dom(V)} c_N(v)$
      end
    end
    $\mathcal{B} \leftarrow \mathcal{B}'$;
    $\mathcal{F} \leftarrow \mathcal{F} \setminus \{V\}$;
  until $\mathcal{F} = \emptyset$ *or* $\nexists X_i \in \mathcal{F}$ *et* $N \in \mathcal{B}$ *t.q.* X_i *peut être installée en* N;
end

Cet algorithme commence donc par regrouper ensemble les feuilles qui ont des distributions de probabilités similaires. Ensuite, pour chaque variable X_i, en remontant dans l'ordre global, et pour chaque nœud N_{X_i} lié à X_i, si $\exists N'_{X_i}$ tel que $\forall x_i \in Dom(X_i), c(N_{X_i}, x_i) = c(N'_{X_i}, x_i)$ alors N_{X_i} et N'_{X_i} sont fusionnés. De plus, si $\forall$ modalité $x_i^k, x_i^l \in Dom(X_i), c(N_{X_i}, x_i^k) = c(N_{X_i}, x_i^l)$ alors N_{X_i} est un nœud redondant. Il est alors remplacé par des arcs reliant ses parents à son unique fils.

La première phase de cet algorithme, qui consiste à agréger les feuilles semblables ensemble doit être analysée plus spécifiquement. Si les feuilles du MDDs étaient des valeurs scalaires (comme pour l'utilité), fusionner ces feuilles serait simple. Toutefois, pour des distributions de probabilités conditionnelles, chaque feuille est une distribution sur la variable Y. Il nous faut donc un test pour déterminer si oui ou non les probabilités de distribution de tout couple de feuilles sont suffisamment similaires pour être fusionnés. Ici encore nous utiliserons la statistique G, non pour un test de dépendance forte mais pour un test d'ajustement.

Soit donc U et V les deux feuilles que nous envisageons de fusionner, soit $n_{U,y}$ (resp. $n_{V,y}$) le nombre d'observations en U (resp. V) pour la modalité $y \in Dom(Y)$ et $N_U = |\Omega_U|$ (resp. $N_V = |\Omega_V|$) le nombre total d'observations en U (resp. V). Agréger U et V produirait une nouvelle feuille W. Il vient que $\forall y \in Dom(Y), n_{W,y} = n_{U,y} + n_{V,y}$ et $N_W = N_U + N_V$.

Pour déterminer si oui ou non U et V devraient être regroupées en W, nous calculons une statistique G pour les deux feuilles.

$$\forall L \in \{U, V\}, G_L = 2 \cdot \sum_y n_{L,y} \ln \frac{n_{L,y}}{e_{L,y}} \tag{2}$$

où $e_{L,y} = n_{W,y} \frac{N_L}{N_W}$. Notons que nous devons réduire proportionnellement les quantités $n_{W,y}$ pour être comparable aux quantités $n_{L,y}$. Ces statistiques mesurent non pas combien les deux distributions en U et V sont similaires, mais combien elles sont similaires à la distribution W que nous obtiendrions si nous prenions la décision de les regrouper.

Là encore, nous avons recours aux p-valeurs p_G^U et p_G^V associées à ces statistiques G, le degré de liberté étant la taille du domaine Y (-1). Ainsi, si p_G^U et p_G^V sont inférieurs à un certain seuil, alors leurs distributions de probabilité sont suffisamment similaires à la distribution de probabilité de W. Il est alors pertinent de remplacer U et V par W.

Ainsi, nous avons conçu un algorithme glouton sur les feuilles de l'ODT : un couple de p-valeur (p_G^U, p_G^V) est calculée pour chaque couple possible de feuilles (U, V). Il est facile de trouver le meilleur candidat (U^*, V^*) à la fusion (par exemple à l'aide d'un tas). Si $p_G^{U^*}$ et $p_G^{V^*}$ sont l'un et l'autre plus petit que le seuil

donné, alors les nœuds sont groupés. Ensuite, pour tout couple (W, T) les p-valeur sont calculées. Puis le processus est répété. Le critère d'arrêt est l'absence de fusion lors d'une itération.

Algorithm 5: Fusionner les sous-graphe isomorphe d'un arbre représentant $P(Y|X_1, \cdots, X_N)$

Data: un ODT T

begin
  repeat
    $(U^*, V^*) = \arg\min_{(U,V)\text{feuilles}}(\max(p_U^G, p_V^G))$;
    if $\max(p_G^{U^*}, p_G^{V^*}) \leq \tau^2$ **then**
      Fusionner U^* et V^*;
    end
  until *∄ deux feuilles qui peuvent être fusionnées*;
  foreach $X_i \in \mathbf{X}$, *en remontant dans l'ordre global* **do**
    foreach *pair de nœuds internes* N_{X_i}, N'_{X_i} *ayant* X_i *comme variable installée* **do**
      if $\forall x_i \in Dom(X_i), c_{N_{X_i}}(x_i) = c_{N'_{X_i}}(x_i)$ **then**
        N_{X_i} et N'_{X_i} sont fusionnés ;
      end
    end
    foreach N_{X_i} *ayant* X_i *comme variable installée* **do**
      if $\forall x_i^k, x_i^l \in Dom(X_i)$, $c_{N_{X_i}}(x_i^k) = c_{N_{X_i}}(x_i^l)$ **then**
        Remplacer N_{X_i} par des arcs allant des parents à l'unique fils.
      end
    end
  end
end

L'algorithme 5 présente la réduction complète d'un ODT en MDD. Il faut noter que cet algorithme est assez gourmand en temps. Toutefois, il est utilisé assez rarement car, très régulièrement, l'ODT ne changera pas (comme décrit dans l'Algorithme 2) ou l'algorithme changera marginalement la structure laissant l'ordre global inchangé. Les expérimentations confirment ce comportement attendu.

3.3 Étude de la complexité d'IMDDI

Étant un algorithme d'apprentissage incrémental composite, IMDDI a une complexité globale difficile à évaluer. Toutefois quelques propriétés peuvent être analysées. Avec Y la variable d'intérêt, et $\mathbf{X}$ l'ensemble des variables expliquant Y, IMDDI essaye d'apprendre $P(Y|\mathbf{X})$ sous la forme d'un MDD en utilisant un ODT T via IOTI.

Propriété 1 (Ajouter une observation – algorithme 3)
*La complexité pour prendre en compte une nouvelle observation ξ dans l'*ODT *est en $\mathcal{O}(|\mathbf{X}|^2)$.*

Preuve 1
*Pour ajouter une observation ξ à l'*ODT*, l'algorithme effectuera une recherche profondeur d'abord jusqu'à atteindre une feuille. La hauteur de l'arbre est en $\mathcal{O}(|\mathbf{X}|)$ puisque chaque variable n'apparaît qu'au plus une fois seulement sur chaque chemin liant la racine à une feuille. Toutefois, durant l'ajout de ξ, en chaque nœud du chemin, l'algorithme met à jour la statistique G pour chaque variable de V_N avec $|V_N| \leq |\mathbf{X}|$.* ■

Le comportement d'IOTI dépend fortement de la nature de l'observation ξ : il peut arriver qu'ajouter ξ à l'arbre T n'induise pas de changement structurel de T. Toutefois certaines insertions d'observation ξ impliquent de telles altérations. La complexité est alors différente. Il faut noter que même si la modalité d'une variable V dans ξ n'a jamais été observée, la complexité donnée ci-dessus ne change pas.

Propriété 2 (Mettre à jour T sans modification – algorithme 4)
La complexité pour mettre à jour l'arbre sans changement structurel est en $\mathcal{O}(|T| \odot |\mathbf{X}|)$.

Preuve 2
*Puisque aucun changement structurel ne se produira sur la frontière de nœud à chaque itération, chaque nœud N de l'*ODT *T participera une fois seulement au calcul des scores agrégés aux frontières. N fera un calcul pour chaque variable V_N avec (comme ci-dessus) $|V_N| \leq |\mathbf{X}|$.* ■

Propriété 3 (Installer une nouvelle variable en un nœud – algorithme 4)
La complexité d'installer une nouvelle variable en un nœud N donné est en $\mathcal{O}(|\Omega_N| + |\mathbf{X}|)$.

Preuve 3
L'installation d'une nouvelle variable V en un nœud N requiert de créer un nouvel ensemble de $(Dom(V))$ fils au nœud N. Soit $c_N(i)$ le fils pour la i-ème modalité que peut assumer la nouvelle variable V installée en N. Ce fils $c_N(i)$ inclut maintenant toutes les observations d'Ω_N où V est égale à sa i-ème modalité. Pour trouver ces observations, il faut parcourir Ω_N. Il faut aussi calculer ($\leq |\mathbf{X}|$) p-valeur pour chaque nœud fils et pour N. ■

Notons qu'il n'est pas nécessaire de subdiviser la base Ω_N en tout nœud N du graphe. En effet, il n'y a besoin que des tables de contingence pour calculer les statistiques G. Ainsi, celles du nœud N s'obtiennent directement puisqu'elles restent inchangée (elles ne dépendent pas de la variable installée). De plus, les tables des nœuds fils s'obtiennent facilement en sommant les tables des nœuds petits-fils. C'est uniquement lors de la subdivision de nœuds feuilles qu'il est nécessaire de parcourir les bases d'observations associées. En effet, par définition, ces nœuds feuilles n'ont pas de petits-fils ; il faut donc parcourir la base de donnée pour construire les tables de contingence des fils (un seul parcours suffit alors).

Une autre partie importante d'IMDDI est la construction du MDD depuis l'ODT.

Propriété 4 (Réduire T en un MDD – algorithme 5)
*La complexité de la réduction de l'*ODT *est en $\mathcal{O}(|T|^2)$.*

Preuve 4
Comme Bryant (1986) le démontre, la fusion d'un ODT *où les feuilles ont des valeurs exactes est en $\mathcal{O}(|TlogT|)$. Toutefois* IMDDI *agrège des distributions de probabilités similaires sur les feuilles et non des valeurs exactes. Ce regroupement doit donc être découpé en deux parties. Une partie est la fusion des nœuds internes. Cette parties reste en $\mathcal{O}(|TlogT|)$. La fusion des feuilles est différente. En effet, initialement, il faut évaluer deux p-valeurs pour chaque pair de feuilles ($2|leaves|^2$ valeurs sont alors calculées). Ensuite, à chaque regroupement, il faut estimer des p-valeurs à nouveau pour la feuille nouvellement créée et ce pour toutes les autres feuilles restantes ($|leaves|$ valeurs sont alors calculées). Toutefois il n'y aura au plus que $|leaves|$ fusions (on ne peut pas faire plus de fusion qu'il n'existe de feuille). Ainsi, cette étape calcule au plus $2|leaves|^2 + |leaves| * |leaves|$ valeurs. Tout compte fait, la complexité de cette phase est en $|leaves|^2$. Or comme $|T|/2 \leq |leaves| \leq |T|$, la complexité totale de cet partie est en $\mathcal{O}(|T|log|T|) + \mathcal{O}(|T|^2) = \mathcal{O}(|T|^2)$.* ■

Cette analyse en complexité montre que IMDDI est un algorithme dédié à l'apprentissage en-ligne : la taille de la base d'observations Ω n'entre jamais en compte dans les complexités sus-évoqué. Seule la complexité d'un sous-algorithme dépend d'une sous-partie Ω_N installée en ce nœud. Notons aussi que les deux sous-algorithmes les plus coûteux (installer et fusionner) ne se déroulent pas nécessairement à chaque fois qu'une nouvelle observation ξ est prise en compte. Comme la section suivante le démontrera, la réévaluation du MDD ne se produit effectivement que marginalement.

3.4 SPIMDDI dans un cadre SDYNA

L'algorithme de planification SPUMDD (Magnan & Wuillemin, 2013) et IMDDI qui vient d'être présenté sont conçus pour pouvoir s'intégrer naturellement dans l'architecture SDYNA. Il est aisé de mixer ces deux algorithmes itératifs pour obtenir un algorithme d'apprentissage par renforcement qui manipule exclusivement des MDDs. De même que pour SPITI, il est à noter que l'architecture permet aisément de prendre en compte des méthodes différentes de gestion du compromis exploration-exploitation : Factored-E^3 (Kearns & Koller, 1999) , Factored-RMAX (Strehl *et al.* (2007), Strehl (2007)), Met-RMAX (Diuk *et al.*, 2009), LSE-RMAX (Chakraborty & Stone, 2011) sont ainsi facilement intégrables dans cette approche.

TABLE 1 – Comparaison en taille et vraisemblance (moyenne ± écart-type) entre IMDDI et ITI et ITI+DD

	IMDDI vs ITI		IMDDI vs ITI+DD	
	Taille	Log vraisemblance	Taille	Log vraisemblance
MDDs	63.34% ±9, 51%	100.96% ±1, 53%	101.15% ±3, 46%	100.01% ±0.04%
DTs	69.44% ±15.44%	101.93% ±1.89%	101.52% ±15.22%	100.03% ±0.09%
BNs	60.51% ±8.73%	100.69% ±1.25%	105.53% ±5.83%	99.94% ±0.39%

4 Experiences et Observations

Pour évaluer les performances de IMDDI, nous l'avons comparé avec ITI, qui est l'architecure SDYNA basé sur des arbres (DTs). La version classique d'ITI utilisée dans SPITI est un algorithme qui produit des DTs non-ordonnés. Nous proposons donc aussi de comparer IMDDI à une version 'augmentée' de ITI : ITI+DD qui réordonne en fin d'apprentissage l'arbre appris par ITI en suivant la même heuristique que dans IMDDI puis le réduit en MDD.

Ces trois algorithmes ont été implémentés en Python. Ces algorithmes sont testés ici en leur faisant apprendre des représentations factorisées de distributions $P(Y|\mathbf{X})$. Il faut donc dans un premier temps fixer la méthode utilisée pour obtenir des flux d'observations (des bases de données) suivant ces distributions. Dans ces expériences, trois approches ont été suivis pour générer aléatoirement ces fonctions $P(Y|\mathbf{X})$:

Arbre de décision : la fonction $P(Y|\mathbf{X})$ est générée aléatoirement directement sous la forme d'un arbre de décision, les nœuds internes étant associés aux variables de $\mathbf{X}$ et les feuilles sont des distributions sur la variable Y. La tâche d'apprentissage est donc de retrouver cet arbre.

Diagrammes de décision : De même que pour l'approche *Arbre de décision*, cette approche génère aléatoirement des diagrammes de décisions. La tâche d'apprentissage est donc de retrouver ce diagramme ou un arbre l'approchant.

Réseau bayésien : Toutes les variables Y et $\mathbf{X}$ sont reliés dans un réseau bayésien généré aléatoirement. Cette approche perturbe la tâche d'apprentissage car il n'est pas certain que la distribution de Y par rapport à ses parents dans le réseau bayésien possède des indépendances contextuelles : l'apprentissage aboutira peut-être à un arbre complet.

Il est à noter que seules les fonctions $P(Y|\mathbf{X})$ suivant les deux premières approches présenteront des dépendances contextuelles. Toutefois, toutes ces approches induisent des dépendances conditionnelles : Y ne dépend pas nécessairement de toutes les variables de $\mathbf{X}$.

Pour chaque approche, 20 instances ont été aléatoirement générées (soit 20 DTs, 20 MDDs et 20 BNs). Avant chaque création de fonction, l'ensemble des variables $\mathbf{X}$ est réinitialisé aléatoirement afin que le nombre de modalités de chaque variable de $\mathbf{X}$ varie d'une fonction à l'autre. Le nombre de modalités de ces variables varient entre 2 et 10. Enfin, les fonctions ainsi générées n'utilisent qu'une sous-partie de $\mathbf{X}$: les variables restantes agissent donc comme du bruit dans l'apprentissage.

Afin d'obtenir des observations à partir de ces fonctions (et donc générer les bases d'apprentissages), la troisième approche est la plus simple puisque le réseau bayésien propose la loi jointe de l'ensemble des variables. Il suffit donc de générer des observations en suivant cette loi jointe.

En ce qui concerne les deux premières approches, il est d'abord nécessaire d'instancier $\mathbf{X}$. Une modalité est donc tirée aléatoirement pour chaque variable de $\mathbf{X}$ et ce de manière uniforme et indépendante. Ensuite, une modalité pour Y est tirée aléatoirement suivant la distribution de $P(Y|\mathbf{X})$ correspondant à l'instanciation obtenue de $\mathbf{X}$.

Ainsi, pour chaque distribution créée, 2 bases de 20 000 et 10 000 observations ont été constituées. La première base est utilisée pour l'apprentissage du modèle alors que la seconde est utilisée pour la validation : il s'agira principalement de comparer les vraisemblances des modèles appris. Les résultats suivants sont obtenus en moyennant sur les 20 modèles de chaque approche.

4.1 Qualité des modèles appris

Le premier critère de comparaison proposé dans cet article est la log-vraisemblance du modèle appris par rapport à la base de validation. Cette vraisemblance nous assure qu'IMDDI ne dégrade pas trop la qualité de la distribution de probabilité apprise par rapport à ITI et ITI+DD. Les résultats de la table 1 montrent qu'en moyenne, il n'y a pas de perte de la qualité des solutions obtenues. Ainsi, la stratégie IMDDI est aussi

pertinente que ITI ou ITI+DD de ce point de vue.

Le second paramètre de comparaison est la taille (en nombre de nœuds) des MDDs et DTs appris. En effet, il s'agit de montrer que IMDDI fournit des représentations plus compactes que les arbres qu'ITI génère. Ce critère est important car les complexités des algorithmes de calcul de politique optimale dépende cruciale-ment de ce paramètre. Les résultats de la table 1 montrent qu'en moyenne les modèles appris par IMDDI sont en effet plus compacts. Avec une diminution de 40% du nombre de nœuds et une vraisemblance quasiment identique, ces expériences nous semblent être une validation expérimentale de l'intérêt de la modélisation de domaines structurés en MDDs plutôt qu'en DTs.

Il est à noter que même si l'utilisation des MDDs semble validée, les résultats très proches de IMDDI et ITI+DD suivant les 2 critères précédents ne permet pas de montrer la pertinence de notre algorithme d'apprentissage incrémental IMDDI par rapport à la méthode plus basique de construction du MDD à chaque apprentissage incrémental d'arbre ITI+DD.

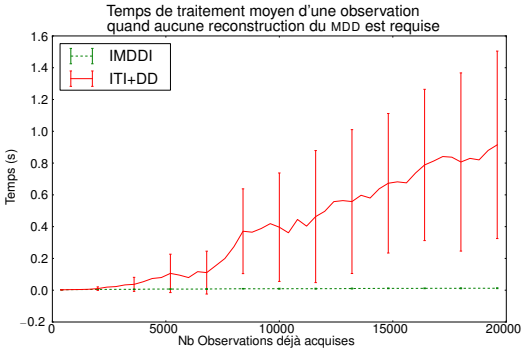


FIGURE 5 – Temps de prise en compte d'une nouvelle observation par IMDDI et ITI+DD. Avant de créer ce graphe, les itérations où l'ajout de l'observation a conduit à une modification du MDD final ont été écartées. Pour les deux algorithmes, ces temps sont mesurés lors des apprentissages des bases générées à partir de BNs. On trace alors les moyennes et écart-types des temps de prises en compte des i^{emes} observations durant les 20 apprentissages de chaque algorithme.

Le dernier critère sur lequel IMDDI et ITI+DD ont été confrontés est donc le temps de calcul. Il s'agit de comparer les temps de prise en compte de nouvelles observations. Dans une première étape, la figure 5 montre le temps de prise en compte d'une nouvelle observation lorsque celle-ci n'induit pas de modification de la structure finale. Ce graphe montre clairement que le temps passé à réordonner l'arbre dans la stratégie ITI+DD le rend complètement inefficace. La phase de ré-ordonnancement de l'arbre coûte en effet de plus en plus de temps à ITI+DD au fur et à mesure que sa taille augmente.

Toutefois, comme cela a été mentionné dans la section précédente, la phase de réduction a la plus forte complexité. Toutefois, en moyenne, seulement 15% des ajouts d'observations implique une reconstruction du MDD pour IMDDI et ITI+DD durant l'apprentissage de bases générées avec les BNs. Cette moyenne tombe à 7% quand l'apprentissage est fait sur des bases obtenues à partir de DTs et MDDs.

On pourrait objecter que la révision du MDD supprimerait complètement l'avantage gagné à éviter les réordonnancements de ITI+DD. La figure 6 décrit l'écart relatif entre les temps totaux pris par IMDDI et ITI+DD pour acquérir une nouvelle observation. La courbe centrale montre qu'en moyenne, en dépit de la reconstruction du MDD, l'algorithme IMDDI reste jusqu'à cinquante fois plus rapide que l'algorithme ITI+DD.

La conclusion de cette première phase expérimentale nous semble donc être double : d'une part, la modélisation en MDD permet effectivement une compacité plus importante du modèle, gain qui aura un impact fort au moment de la recherche de la stratégie optimale. D'autre part, l'algorithme d'apprentissage incrémental direct du MDD se revèle beaucoup plus rapide qu'un algorithme basé sur un apprentissage incrémental d'arbre suivi d'une induction du MDD à partir de cet arbre.

4.2 SPIMDDI : premières expériences d'intégration dans une architecture SDYNA

IMDDI s'avère donc être un algorithme efficace d'apprentissage incrémentale de MDDs. L'étape suivante consiste à illustrer la pertinence de son intégration dans une architecture SDYNA. Comme cela a été indiqué plus haut, il est possible et certainement efficace d'intégrer dans SPIMDDI des méthodes sophistiquées de gestion du compromis exploration-exploitation. Dans le cadre de cet article, nous nous sommes limité à

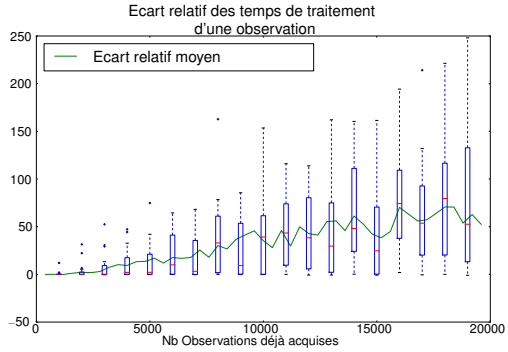


FIGURE 6 – Évolution de l'écart relatif $\frac{T_{ITI+DD}-T_{IMDDI}}{T_{IMDDI}}$ (où T_X est le temps total pour l'algorithme X pour intégrer la nouvelle observation). Pour établir ce graphe, nous nous sommes servi des modèles appris à partir des bases BNs.

une approche ϵ-*greedy* : il s'agit en effet ici de démontrer et valider l'intérêt d'une approche basée sur des MDDs. Par conséquent toute surcouche algorithmique qui pourrait brouiller l'interprétation des observations est plutôt à proscrire.

Cette section propose de premiers tests de SPIMDDI, l'instance de SDYNA basé sur nos algorithmes, sur trois exemples classiques de la littérature : *Coffee Robot*, *Factory* (Dearden & Boutilier, 1997) et *Taxi* (Dietterich, 1998) . *Coffee Robot* consiste pour l'agent à chercher du café dans une boutique pour le fournir à son propriétaire en évitant la pluie. Dans *Factory*, le but est de façonner et d'assembler deux pièces ensembles suivant une séquence ordonnée d'opérations. Enfin *Taxi* consiste à réussir à embarquer un passager dans une 'case' donnée pour le déposer dans une autre. *Coffee Robot* est un problème de petite taille. *Factory* est un FMDP binaire, mais de grande taille. Enfin, *Taxi* est un FMDP multimodal de grande taille.

Afin de pouvoir comparer notre approche au plus juste, nous avons implémenté SPIMDDI et SPITI dans la même infrastructure logicielle en C++. La figure 7 présente 3 critères de comparaison entre SPIMDDI et SPITI sur ces 3 exemples. Ces courbes sont obtenues en moyennant sur 20 expériences de 4000 trajectoires. Une trajectoire représente au plus 25 prises de décisions avant une réinitialisation aléatoire de l'état du système.

Le graphe 7(a) compare la taille des modèles appris qui est égale au nombre de nœuds de l'ensemble des représentations factorisées du modèle (toutes les distributions de probabilités et la fonction récompense). Il s'agit donc ici de tester si l'objectif d'une meilleure compacité de la représentation est atteinte. Il s'avère que sur les 3 exemples, IMDDI parvient à trouver une représentation toujours plus compacte du problème.

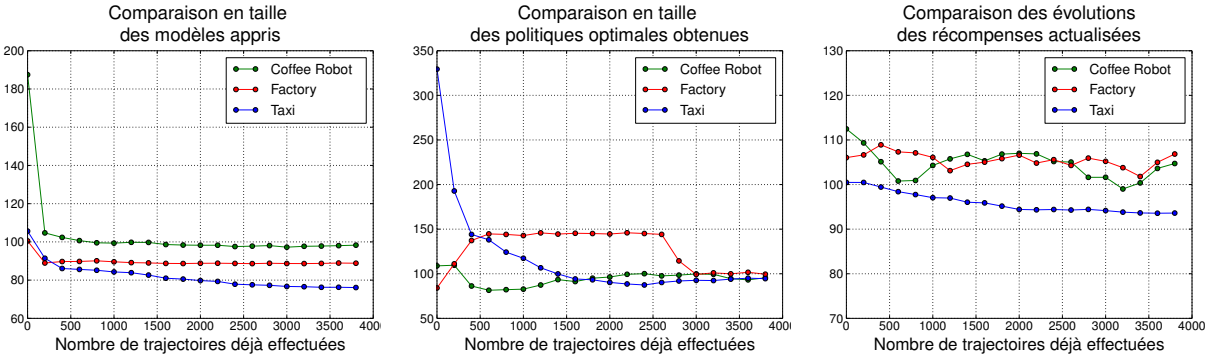


FIGURE 7 – Comparaison en % de SPIMDDI par rapport à SPITI suivant (a) la taille des modèles appris (transitions et récompenses), (b) la taille de la politique optimale plannifiée, et (c) la récompense actualisée obtenue au cours de l'apprentissage.

Le graphe 7(b) compare la taille des politiques optimales obtenues (en nombre de nœuds). Pour toutes les expériences, les politiques finales obtenues sont de taille comparable. Toutefois, le nombre d'actions possibles dans les exemples utilisés est peut-être trop faible pour permettre une différenciation entre une politique optimale exprimée par un MDD ou par un DT. Par ailleurs, dans le graphe 7(c), la similarité des gains actualisés tend à indiquer que SPIMDDI apprend une politique aussi efficace que SPITI.

5 Conclusion

L'intérêt des MDDs sur les DTs est démontré depuis longtemps en Planification Théorique de la Décision (par exemple avec SPUDD). Cet article décrit IMDDI, un algorithme d'apprentissage incrémental de MDD, qui se combine naturellement avec l'algorithme de planification SPUMDD pour fournir une instanciation de l'architecture SDYNA basée sur les MDDs : SPIMDDI. Il propose une analyse en complexité des différentes phases de IMDDI (ajout d'une observation, mise à jour de la structure, et réduction en diagramme de décision).

Les expérimentations décrites valident la compacité et la précision des modèles appris, les comparant aux structures induites par ITI et ITI+DD. Elles montrent également que IMDDI est plus rapide que l'approche basique ITI+DD. Il est à noter qu'IMDDI ne requiert aucun *a priori* sur les variables : le nombre de variables ou la taille de leurs domaines ne sont pas nécessaires à l'initiation de l'algorithme et peuvent être découverts dynamiquement durant l'apprentissage. L'intégration de IMDDI dans l'instance SPIMDDI de l'architecture SDYNA donne des résultats encourageants : malgré la faible capacité d'exploration de ϵ-*greedy*, IMDDI s'avère capable d'apprendre un modèle plus compact que ne l'aurait fait SPITI. Cette réduction en taille se fait sans perte d'optimalité au vu de la similarité dans l'évolution des récompenses actualisées.

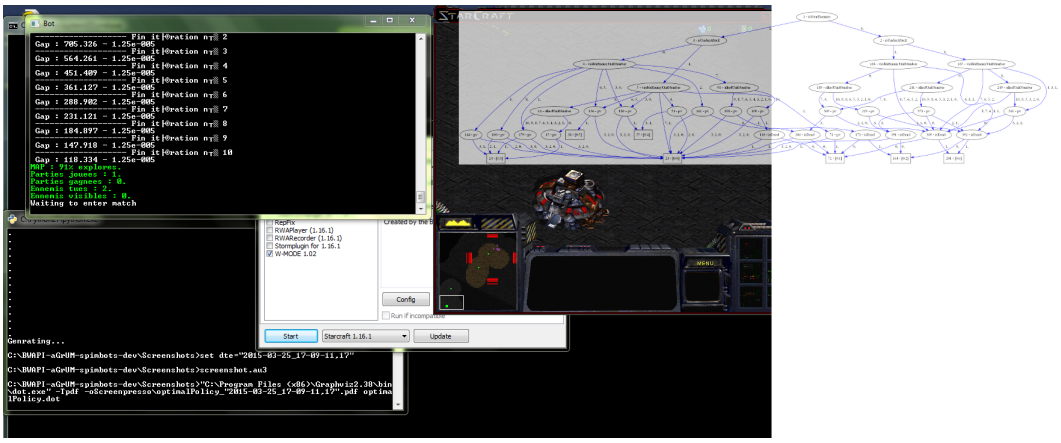


FIGURE 8 – Un aperçu de SPIMDDI utilisé dans le jeu vidéo StarCraft. En surimpression, la politique optimale obtenue après une cinquantaine de parties.

Dans les développements futurs, nos objectifs seront d'intégrer de meilleurs algorithmes de gestion du compromis exploration-exploitation, d'analyser plus finement le comportement de SPIMDDI, en particulier en le confrontant à des applications réelles telles que le jeu vidéo comme nous avons commencé à le faire dans une première expérimentation avec StarCraft [2] (voir figure 8).

Références

BAHAR R. I., FROHM E. A., GAONA C. M., HACHTEL G. D., MACII E., PARDO A. & SOMENZI F. (1993). Algebraic decision diagrams and their applications.

BELLMAN R. (1957). *Dynamic Programming*. Princeton, NJ, USA : Princeton University Press, 1 edition.

BELLMAN R. (1961). *Adaptive Control Processes*. Princeton University Press.

BOUTILIER C., DEAN T. & HANKS S. (1999). Decision-theoretic planning : Structural assumptions and computational leverage. *JAIR*, **11**, 1–94.

BOUTILIER C., FRIEDMAN N., GOLDSZMIDT M. & KOLLER D. (1996). Context-specific independence in bayesian networks. p. 115–123.

BREIMAN L., FRIEDMAN J., OLSHEN R. & STONE C. (1984). *Classification and Regression Trees*. Monterey, CA : Wadsworth and Brooks.

BRYANT R. E. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, **35**, 677–691.

2. StarCraft 2, Blizzard Entertainment (2010), http ://eu.battle.net/sc2/fr/

CHAKRABORTY D. & STONE P. (2011). Structure learning in ergodic factored mdps without knowledge of the transition function's in-degree. In *Proceedings of the Twenty Eighth International Conference on Machine Learning (ICML'11)*.

DEARDEN R. & BOUTILIER C. (1997). Abstraction and approximate decision-theoretic planning. *Artificial Intelligence*, **89**(1–2), 219 – 283.

DEGRIS T., SIGAUD O. & WUILLEMIN P.-H. (2006a). Chi-square Tests Driven Method for Learning the Structure of Factored MDPs. In *Proceedings of the 22nd conference on Uncertainty in Artificial Intelligence*, p. 122–129 : AUAI Press.

DEGRIS T., SIGAUD O. & WUILLEMIN P.-H. (2006b). Learning the Structure of Factored Markov Decision Processes in Reinforcement Learning Problems. In *Proceedings of the 23rd International Conference on Machine Learning*, p. 257–264 : ACM.

DIETTERICH T. G. (1998). The MAXQ method for hierarchical reinforcement learning. In J. W. SHAVLIK, Ed., *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998), Madison, Wisconsin, USA, July 24-27, 1998*, p. 118–126 : Morgan Kaufmann.

DIUK C., LI L. & LEFFLER B. R. (2009). The adaptive k-meteorologists problem and its application to structure learning and feature selection in reinforcement learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, p. 249–256 : ACM.

DUNNING T. (1993). Accurate methods for the statistics of surprise and coincidence. *Comput. Linguist.*, **19**(1), 61–74.

FRIEDMAN S. J. & SUPOWIT K. J. (1990). Finding the optimal variable ordering for binary decision diagrams. *IEEE Trans. Comput.*, **39**(5), 710–713.

HOEY J., ST-AUBIN R., HU A. & BOUTILIER C. (1999). Spudd : Stochastic planning using decision diagrams. In *In Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, p. 279–288 : Morgan Kaufmann.

HOWARD R. A. (1960). Dynamic programming and markov processes.

KEARNS M. & KOLLER D. (1999). Efficient reinforcement learning in factored mdps. In *IJCAI*, volume 16, p. 740–747.

KOHAVI R. (1994). Bottom-up induction of oblivious read-once decision graphs. In F. BERGADANO & L. RAEDT, Eds., *Machine Learning : ECML-94*, volume 784 of *Lecture Notes in Computer Science*, p. 154–169. Springer Berlin Heidelberg.

KOHAVI R. & HSIN LI C. (1995). Oblivious decision trees, graphs, and top-down pruning. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, p. 1071–1077 : Morgan Kaufmann.

MAGNAN J.-C. & WUILLEMIN P.-H. (2013). Improving Decision Diagrams for Decision Theoretic planning. In *Florida Artificial Intelligence Research Society Conference*, p. 621–626.

MINGERS J. (1989). An empirical comparison of selection measures for decision-tree induction. *Machine Learning*, **3**(4), 319–342.

OLIVEIRA A. L. & SANGIOVANNI-VINCENTELLI A. (1994). Inferring reduced ordered decision graphs of minimal description length. In *proceedings of the 12th international conference on machine learning*, p. 421–429 : Morgan Kaufmann.

OLIVER J. J. (1993). Decision graphs - an extension of decision trees.

PUTERMAN M. (2005). *Markov decision processes : discrete stochastic dynamic programming*. Wiley series in probability and statistics. Wiley-Interscience.

QUINLAN J. R. (1993). *C4.5 : Programs for Machine Learning*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc.

STREHL A. (2007). Model-based reinforcement learning in factored-state mdps. In *Approximate Dynamic Programming and Reinforcement Learning, 2007. ADPRL 2007. IEEE International Symposium on*, p. 103–110.

STREHL E. L., DIUK C. & LITTMAN M. L. (2007). Efficient structure learning in factored-state mdps. In *Proceedings of the Twenty-Second National Conference on Artificial Intelligence (AAAI-07*.

SUTTON R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, p. 216–224.

UTGOFF P. E., BERKMAN N. C. & CLOUSE J. A. (1997). Decision tree induction based on efficient tree restructuring. *Machine Learning*, **29**(1), 5–44.

WHITE A. P. & LIU W. Z. (1994). Technical note : Bias in information-based measures in decision tree induction. *Mach. Learn.*, **15**(3), 321–329.

Exploration et Exploitation dans des MDPs Cybernétiques

Filipo Studzinski Perotto

Université de Toulouse 1 - Capitole
Institut de Recherche en Informatique de Toulouse
Équipe LILaC
filipo.perotto@irit.fr

Résumé : Dans cet article on présente l'algorithme "average engaged climber" (AEC), une version modifiée de la méthode d'itération sur les valeurs pour l'estimation de la fonction d'utilité dans un *processus de décision markovien* (MDP) à horizon infini. Plus précisément, la méthode présentée vise à résoudre le dilemme entre exploration et exploitation dans une classe particulière de problèmes appelée *MDP cybernétique* (C-MDP). Dans ces problèmes, l'agent est doté d'une énergie interne dont la valeur est affectée par les récompenses reçues au long de son cycle de vie, et il doit la maintenir au-dessus d'un seuil de viabilité, afin d'assurer sa propre survie. Ainsi, l'agent doit chercher à optimiser son comportement, tout en tenant compte des coûts des actions exploratoires. L'originalité de la méthode proposée repose sur le fait de construire explicitement deux politiques différentes (une pour explorer, l'autre pour exploiter), et de déterminer les bons moments pour changer de stratégie en utilisant une notion d'engagement. L'algorithme estime l'utilité des paires état-action en approchant la récompense moyenne espérée de chacune. L'intérêt potentiel de l'idée est appuyé par des résultats expérimentaux préliminaires.

Mots-clés : Adaptation et Apprentissage Automatique, Apprentissage par Renforcement, Processus de Décision Markovien, Dilemme Exploration-Exploitation

1 Introduction

Trouver l'équilibre entre les comportements d'exploration et d'exploitation est un dilemme essentiel affronté par tout agent qui doit s'adapter à son environnement de façon autonome. Le défi de trouver un bon compromis entre l'exploration (apprendre de nouvelles choses) et l'exploitation (agir de manière optimale en fonction de ce qui est déjà connu) constitue un problème largement étudié dans les domaines de la prise de décision séquentielle et de l'apprentissage par renforcement (Puterman, 1994; Sutton & Barto, 1998; Sigaud & Buffet, 2010; Wiering & Otterlo, 2012; Feinberg & Shwartz, 2002). L'agent est censé favoriser les actions bien récompensées, mais il doit à la fois expérimenter des actions inconnues (ou méconnues) pour pouvoir découvrir de nouvelles façons d'atteindre de meilleures récompenses.

En *apprentissage par renforcement avec modèle* (*MBRL*), l'agent doit simultanement (a) apprendre un modèle du monde (c'est-à-dire, approcher progressivement les fonctions de récompense et de transition à partir de ses observations) et (b) profiter de ses connaissances pour optimiser son comportement (c'est-à-dire, définir une politique d'actions à partir des modèles afin de maximiser l'espérance de récompenses futures). Prises ensemble, ces deux tâches créent une dépendance cyclique : la façon dont l'agent se comporte conditionne ce qu'il peut apprendre, et ce qu'il sait conditionne la manière dont il se comporte. Telle est l'origine du *dilemme exploration-exploitation*, et cela pose deux défis supplémentaires à l'agent : (c) trouver un équilibre entre l'exploration et l'exploitation, et (d) réaliser l'exploration de manière efficace.

Dans cet article, nous nous intéressons à l'ensemble des problèmes représentés par des *processus de décision markoviens* (MDPs) ergodiques, et qui se déroulent sur un horizon temporel infini. Ces sont des MDPs non-épisodiques, qui ne présentent pas des buts ou des états terminaux, où le cycle de vie de l'agent n'est pas limité dans le temps. Plus précisément, nous nous intéressons aux *MDPs cybernétiques* (C-MDPs), dans lesquels l'agent essaie d'apprendre une politique optimale, mais en tenant compte des coûts associés à ses actions.

Dans un C-MDP, l'agent possède une variable ω qui représente son "énergie". L'énergie n'est pas une

observation au sens classique, mais elle doit être considerée au moment de la prise de décision. Le processus de décision cybernétique se constitue ainsi comme un problème de survie. L'énergie de l'agent se modifie comme suit, où $\{r \in \mathbb{R}\}$ est la récompense immédiate reçue par l'agent :

$$\omega' \leftarrow \omega + r \tag{1}$$

Le niveau d'énergie de l'agent ω est initialisé avec une valeur positive (l'énergie initiale $\{\omega_0 \in \mathbb{R} \mid \omega_0 > 0\}$), et ce niveau évolue comme la récompense cumulative. L'énergie est définie comme suit :

$$\omega_n = \omega_0 + \sum_{t=1}^{n} r_t \tag{2}$$

Le problème prend tout son sens quand on considère les MDPs dans lesquels les récompenses peuvent être positives et négatives, c'est-à-dire $\{\exists s \exists a \mid \Pr(R(s,a) > 0) > 0\} \wedge \{\exists s \exists a \mid \Pr(R(s,a) < 0) > 0\}$. L'agent est tenu de maintenir son niveau d'énergie, que nous pouvons interpréter comme une variable essentielle (Ashby, 1956), dans des limites de viabilité. Ainsi, dans un C-MDP, l'agent doit apprendre une politique optimale (ou quasi-optimale), tout en préservant $\omega > 0$. Autrement dit, l'agent doit à la fois apprendre et survivre.

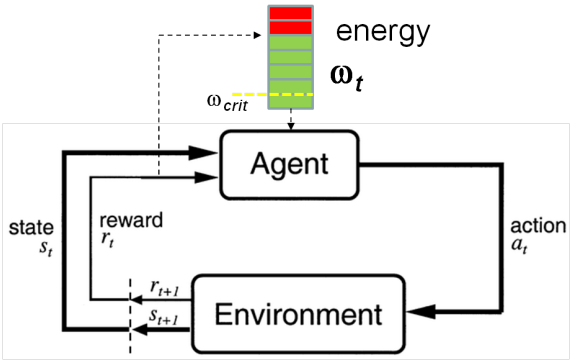


FIGURE 1 – C-MDP : la version cybernétique du problème classique d'apprentissage par renforcement.

Dans cet article, on aborde le problème de trouver le bon moment pour basculer entre une stratégie d'exploration et une stratégie d'exploitation. Même si le dilemme exploration-exploitation est étudié depuis plusieurs années, de notre point de vue, les approches standards mélangent le choix de la stratégie et le choix de l'action dans une même étape de décision. Dans cet article on veut analyser le potentiel avantage de distinguer ces deux choix plus clairement.

Cette recherche trouve sa motivation dans la notion intuitive que, en général, lorsque l'agent suit une politique, il enchaîne des actions spécifiques afin d'atteindre une certaine récompense intéressante à la fin de la séquence. Une fois engagé dans une séquence d'actions, il n'est pas très intéressant de changer de stratégie avant d'atteindre la récompense, sinon l'effort investi sera perdu. Le même *insight* s'applique à l'exploration : lorsque l'agent dépense son temps et son énergie à voyager à travers l'espace d'états en cherchant une certaine situation spécifique qui lui permettrait d'apprendre quelque chose d'important sur le monde, l'occasion ne devrait pas être manquée à cause d'un soudain changement de stratégie.

Dans (Perotto, 2015), la méthode *engaged climber* (EC) a été proposée, le concept d'*engagement* a été introduit, ainsi que l'idée du maintien explicite de deux politiques (l'une pour exploiter, l'autre pour explorer) issues de deux estimations d'utilité différentes : les *récompenses attendues* et les *progrès d'apprentissage espérés*. L'idée est que l'agent reste concentré sur la même stratégie afin d'atteindre le prochain "pic" (une récompense importante ou une grande découverte), comme un "grimpeur" qui s'engage avec un objectif, poursuivant sa démarche jusqu'à l'arrivée au sommet.

Cependant, la méthode EC calcule l'utilité des actions en fonction des récompenses cumulées actualisées (pondérées par un facteur de réduction γ). Ce critère d'optimalité n'est pas toujours adapté aux problèmes à horizon illimité. En plus, la version préliminaire de la méthode EC nécessite d'un réglage manuel de certains paramètres comme le temps moyen d'engagement avec une stratégie ou le seuil d'identification d'un pic de récompense, valeurs qui sont fortement dépendantes du contexte de chaque scenario.

En vue de cela, la contribution de cet article est une version améliorée de la méthode EC. Dans cette nouvelle version, appelée *average engaged climber* (AEC), les *temps d'engagement* ainsi que les états considérés comme des *pics* sont dynamiquement définis et associés à chaque paire état-action dans chacune des fonctions d'utilité (exploration / exploitation) afin d'estimer combien de temps l'agent doit suivre la même politique s'il veut concrétiser l'utilité espérée pour la stratégie choisie. Ces paramètres sont appris par l'agent en même temps qu'il estime les fonctions d'utilité.

Différemment de la version précédente, l'algorithme présenté dans cet article détermine l'utilité d'une action par la moyenne des récompenses prévues, au lieu d'utiliser la somme actualisée des récompenses. Pour certains problèmes d'apprentissage par renforcement non-épisodiques, cette dernière approche peut piéger l'agent en évaluant des politiques sous-optimales comme étant meilleures que les politiques optimales (Tadepalli, 2010).

Cet article est organisé comme suit : la section 2 revisite les concepts principaux de l'apprentissage par renforcement et des processus de décision markoviens, en se concentrant sur les méthodes standards pour équilibrer l'exploration et l'exploitation, et sur les méthodes standards pour estimer la valeur d'une politique par récompense moyenne ; la section 3 introduit les méthodes EC et AEC, et montre des résultats expérimentaux ; la section 4 présente les conclusions.

2 Processus de Décision Markoviens

Les *Processus de Décision Markoviens* (MDPs) sont au centre d'un *framework* largement utilisé pour représenter les problèmes de *contrôle automatisé*, de *prise de décision*, de *planification*, et d'*apprentissage par renforcement* (Puterman, 1994; Sutton & Barto, 1998; Sigaud & Buffet, 2010; Wiering & Otterlo, 2012; Feinberg & Shwartz, 2002; Uther, 2010). Un MDP est typiquement représenté par une machine à états finis discrète et stochastique : à chaque pas de temps la machine est dans un état s ; l'agent observe l'état et il choisi une action a à réaliser ; ensuite la machine change vers un nouvel état s' et donne à l'agent la récompense r correspondante. Il n'y a pas une phase d'entrainement séparée, et l'agent doit apprendre à coordonner ses actions par essais et erreurs.

D'une façon générale, un MDP peut être formellement défini comme un quadruplet $\{S, A, T, R\}$ où $S = \{s_1, s_2, ..., s_{|S|}\}$ est l'ensemble des états du système, $A = \{a_1, a_2, ..., a_{|A|}\}$ est l'ensemble des actions de l'agent, $T(s, a, s') = \Pr(s'|s, a)$ est la fonction de transition, et $R(s, a, s', r) = \Pr(r|s, a, s')$ est la fonction de récompense. La fonction de transition T définit la dynamique du système en déterminant l'état suivant s' à partir de l'état actuel s et de l'action exécutée a. La fonction de récompense R définit la récompense immédiate après la transition de s à s' avec a. Une politique déterministe π établit une correspondance entre les états et les actions sous la forme $\pi(s) = a$, ce qui permet de définir le comportement de l'agent en indiquant une action à effectuer pour chaque état du système.

Résoudre un MDP signifie trouver la politique d'actions qui maximise les récompenses reçues par l'agent au cours du temps. Comme la politique optimale peut être difficile à apprendre, dans la pratique on espère d'un bon algorithme qu'il soit capable d'apprendre une politique quasi-optimale avec une forte probabilité (Valiant, 1984).

Lorsque les fonctions de récompense et de transition sont données, le MDP peut être résolu par *programmation dynamique*. En MBRL, l'agent approche graduellement les paramètres de ces deux fonctions. Après chaque observation, les modèles sont mis à jour de façon incrémentielle. Ensuite, à partir de ces modèles, l'agent résout le MDP en estimant l'utilité de chaque paire état-action, et finalement en associant les états avec les meilleures actions, de manière à définir une politique (Sutton & Barto, 1998).

2.1 Conditions d'Optimalité

Quand l'agent a un horizon de temps infini ou illimité, son expérience ne peut pas être découpée en épisodes. En vue de cela, l'évaluation des politiques se fait, généralement, en utilisant la récompense cumulée actualisée, où un facteur de pondération $\{\gamma \in \mathbb{R} \mid 0 \leq \gamma < 1\}$ réduit le poids des récompenses futures par rapport aux récompenses immédiates. La récompense cumulative actualisée est définie comme suit :

$$v = \sum_{t=1}^{\infty} \gamma^t . r_t \tag{3}$$

Une fonction d'utilité V peut être itérativement approchée par *itération sur les valeurs* (*value-iteration*) à partir des modèles T et R, puis une politique optimale peut être extrait de V comme suit :

$$Q'(s,a) \leftarrow R(s,a) + \gamma \sum_{s'} T(s,a,s')V(s') \tag{4}$$

$$V'(s) \leftarrow \max_a [Q'(s,a)] \tag{5}$$

$$\pi'(s) \leftarrow \arg\max_a [Q'(s,a)] \tag{6}$$

La valeur cumulée des récompenses actualisées est toujours finie, ce qui assure la convergence des méthodes itératives vers une solution optimale. Cependant, dans de nombreux domaines, il n'y a aucune interprétation naturelle pour le facteur de réduction γ. Encore plus problématique, dans des domaines récurrents (où les récompenses tardives ont la même importance que les récompenses précoces) un tel critère peut fausser l'utilité réelle de certaines séquences d'actions. Un critère d'optimalité alternatif est la récompense moyenne obtenue par pas de temps (Tadepalli & Ok, 1998), généralement appelé *gain*.

Sous quelques suppositions raisonnables, la récompense moyenne (ou le gain) d'une politique donnée π converge vers une valeur unique $\rho(\pi)$ indépendamment de l'état de départ (Puterman & Patrick, 2010), où :

$$\rho(\pi) = \lim_{n\to\infty} 1/n \sum_{t=1}^{n} r_t \tag{7}$$

De cette façon, pour une politique π donnée, le gain pour tout état est équivalent à la récompense moyenne, $g(s) = \rho(\pi)$. Une politique π est dite optimale sur le gain si elle maximise $\rho(\pi)$, c'est-à-dire, si à long terme elle offre la moyenne des récompenses la plus élevée.

L'utilisation de la récompense moyenne (gain) comme critère d'optimalité amène l'agent à un circuit optimal dans le MDP. Cependant, même si le gain $\rho(\pi)$ d'une politique π est indépendant de l'état initial du processus, la récompense totale reçue dans un instant donné ne l'est pas, soit $n.\rho(\pi) \neq \sum_{t=1}^{n} r_t$. Un deuxième critère d'optimalité est alors nécessaire afin de comparer les états qui ne font pas partie du circuit optimal. Cette différence, appelée *biais*, est notée $h(s)$. Les politiques biais-optimales sont, parmi toutes les politiques gain-optimales, celles qui optimisent également les récompenses transitoires initiales. La combinaison de biais et de gain permet à l'agent d'atteindre de manière optimale le circuit optimal (Tadepalli & Ok, 1998).

La méthode standard en apprentissage par renforcement utilisant la récompense moyenne est l'algorithme *H-Learning* (Tadepalli, 2010). Dans cet algorithme, les valeurs de gain et de biais sont itérativement mises à jour comme suit, où α est le taux d'apprentissage et r est la récompense immédiate reçue :

$$\rho' \leftarrow (1-\alpha)\rho + \alpha(r - h(s) + h(s')). \tag{8}$$

$$h'(s) \leftarrow R(s,a) + h(s') - \rho \tag{9}$$

Estimer la moyenne des récompenses ρ pendant l'apprentissage n'est pas évidant, car sa valeur est déformée par les actions exploratoires. Pour cette raison, cette estimation doit être révisée uniquement lorsque l'agent exécute des actions d'exploitation. En outre, parce que la récompense moyenne ρ et le biais h sont calculés simultanément, la convergence ne peut être garantie que si ρ est actualisé à une échelle de temps beaucoup plus lente que h.

2.2 Dilemme Exploration-Exploitation

La méthode la plus basique pour équilibrer l'exploration et l'exploitation est l'algorithme ε-*greedy*, qui définit des proportions fixes de temps pour explorer ou pour exploiter. Dans cet algorithme, à chaque pas de temps, l'agent peut soit exécuter une action aléatoire avec une probabilité ε, soit il peut suivre la politique optimale courante. Une autre méthode classique est l'algorithme *softmax*, où l'agent sélectionne parmi les actions possibles proportionnellement à leur utilité estimée. Ainsi, la fonction d'utilité estimée sert à construire une politique stochastique. Dans *softmax*, les actions les plus bien évaluées sont exécutées avec des probabilités plus élevées, mais les actions moins récompensées peuvent être aussi choisies de temps à autre. Un paramètre de *température* τ est utilisé pour définir comment la différence entre les utilités estimées

affecte le choix de l'action. Concernant ces deux algorithmes, ε ou τ sont, à la base, des paramètres réglés à la main. Plus la valeur de ε ou de τ est haute, plus l'agent aura tendance à présenter des comportements exploratoires, tandis que des valeurs faibles entraînent une sélection d'actions dirigée vers l'exploitation.

Le problème avec ces méthodes basiques est que le paramètre d'exploration n'est pas ajusté au cours du processus d'apprentissage. Pour rendre ce paramètre évolutif, et donc les algorithmes plus souples et adaptables, un méta-paramètre peut être inclus afin de réguler le paramètre d'exploration (Auer *et al.*, 2002). C'est le cas de *ε-first*, méthode qui effectue une exploration pleine pendant une période de temps prédéterminée, et passe à l'exploitation totale dans un deuxième temps. Alternativement, l'algorithme *decreasing-ε* propose la réduction progressive de la valeur du paramètre ε, de sorte que la probabilité d'exploration diminue à un taux de $1/n$, où n est le temps écoulé (en cycles). La méthode *softmax* peut être modifiée de la même manière, en faisant la valeur de τ diminuer au fil du temps. Toutefois, dans tous ces cas, la valeur du méta-paramètre est fixe.

Afin de réguler dynamiquement le taux de décroissance de ε, l'algorithme *VDBE* (Tokic, 2010; Tokic & Palm, 2011) utilise le paramètre $\varepsilon(s)$ comme une fonction qui mesure l'incertitude associée à l'estimation de l'utilité de chaque état. Cette incertitude est donnée par les fluctuations de la propre estimation. L'idée est que l'agent doit être plus exploratoire dans les régions de l'environnement où son modèle est encore instable.

Une autre approche standard pour résoudre le dilemme exploration-exploitation est l'*optimisme face à l'incertitude*. Cette approche consiste à ajouter un bonus sur l'utilité des paires état-action moins fréquentées (Meuleau & Bourgine, 1999). Cela peut être fait par une initialisation optimiste des estimations dans le modèle de récompenses, couplée à une sélection d'actions basée sur une politique unique d'exploitation (*greedy*). Ainsi les actions qui conduisent l'agent à des situations peu connues seront probablement préférées. Plus une paire état-action est expérimentée, plus la récompense estimée pour elle s'approchera de sa vraie valeur. Ces algorithmes présentent une phase initiale plus exploratoire, qui est progressivement remplacée par un comportement d'exploitation, suivant la convergence des modèles de récompense et de transition.

Cette approche, qui pousse l'agent vers l'inconnu, est utilisée dans l'algorithme *R-Max* (Brafman & Tennenholtz, 2002), une des méthodes standards pour résoudre le dilemme exploration-exploitation en MBRL. *R-Max* généralise E^3 (Kearns & Singh, 2002), qui constitue une autre méthode aussi bien établie. Directement ou indirectement, ces algorithmes prennent en compte le nombre de fois qu'une paire état-action a été visitée. Les paires moins visitées ont un bonus d'exploration supplémentaire, qui augmente l'utilité estimée de ses récompenses. Les modèles estimés sont *probablement approximativement corrects* (PAC) après un nombre suffisant d'expériences.

Une fois que la méthode pour équilibrer l'exploration et l'exploitation est choisie, un autre problème est de définir la meilleure façon d'explorer l'environnement. Les méthodes de la famille ε ou *softmax* supposent la réalisation d'une exploration non-dirigée (*undirected*) (Kaelbling *et al.*, 1996), simplement en rajoutant une part de hasard dans le processus de sélection d'actions. Performer une exploration non-dirigée signifie exécuter des actions aléatoires. Cependant, en procédant ainsi (exécutant des actions au hasard), l'agent peut finir par gâcher son effort d'exploration en expérimentant des paires état-action déjà bien modélisées, ou par s'auto-pénaliser en se dirigeant vers des récompenses dangereusement négatives connues.

Contrairement, les méthodes d'exploration dirigées (*directed exploration*) préfèrent essayer les paires état-action les moins visitées, en pondérant cette préférence avec leur utilité (les récompenses futures prévues). Ce type d'exploration est réalisé par des algorithmes comme *R-Max* (Brafman & Tennenholtz, 2002), *UCB* (Auer *et al.*, 2002), or E^3 (Kearns & Singh, 2002). Des estimations à priori optimistes sont associées aux paires état-action moins fréquentes afin d'induire l'agent à les essayer. Donc, si un état à découvrir n'est pas trop loin, et qu'il n'y a pas d'évidence sur de possibles récompenses excessivement négatives sur le chemin, l'agent sera amené à voyager à travers l'espace pour atteindre cet état méconnu, tout simplement en suivant sa politique d'exploitation. Les méthodes d'exploration dirigée ajoutent le bonus d'exploration dans l'estimation de la récompense, ce qui conduit l'agent à une phase initiale plutôt tournée vers l'exploration, et qui se transforme progressivement en exploitation, à mesure que le monde devient suffisamment connu.

3 Le Grimpeur Engagé

Dans (Perotto, 2015) une version préliminaire de la méthode *engaged climber* (EC) a été présentée. L'idée de créer explicitement deux politiques séparées (une pour l'exploration et l'autre pour l'exploitation) y était déjà présente, ainsi que l'idée de rester engagé avec la stratégie choisie pour un certain temps avant de réévaluer le choix. La métaphore pour illustrer l'*insight* est celle d'un "grimpeur engagé" qui, à un moment donné, voit deux pics intéressants à escalader. Une fois que le pic est choisi, le grimpeur restera engagé avec la mission de l'atteindre, en persévérant dans sa stratégie. Le choix n'est pas réévalué ou remis en question au cours de la montée, à moins qu'une situation inattendue et critique se produise. Cette réflexion est illustrée dans la figure 2.

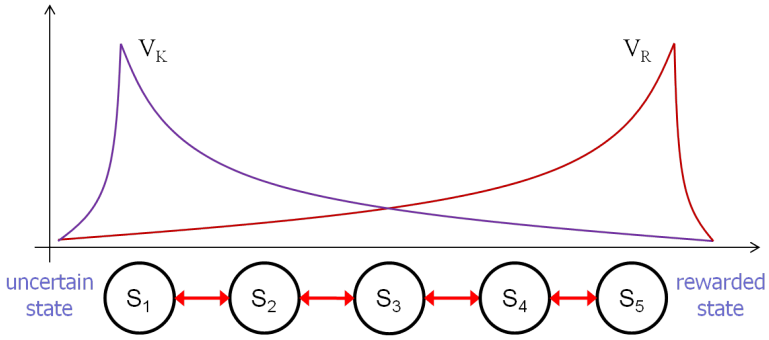


FIGURE 2 – Deux pics en vue : à gauche un état intéressant à explorer, et à droite un état intéressant à exploiter.

Cependant, la version de la méthode EC proposée en (Perotto, 2015) suppose un réglage manuel des certains paramètres pour être en mesure de fonctionner. En outre, cette version calcule l'utilité en utilisant les récompenses cumulées actualisées. Dans des MDP à horizon infini, comme les MDPs cybernétiques utilisés dans les expériences présentées à la fin de la section, une condition d'optimalité fondée sur la moyenne des récompenses correspond mieux au comportement attendu de l'agent.

Dans un MDP stationnaire et ergodique à horizon temporel non-limité, toute politique optimale contient au moins un ensemble fermé et cyclique de paires état-action. Cet ensemble forme un "circuit optimal", vers où conduisent tous les autres états, comme illustre la figure 3.

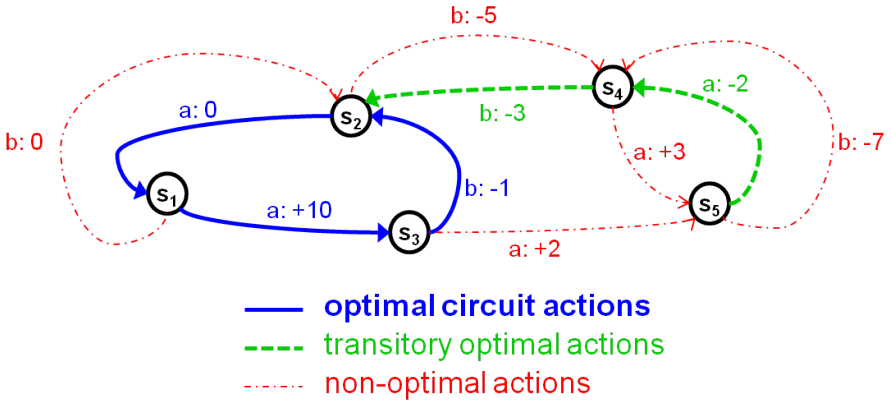


FIGURE 3 – Le circuit optimal est celui qui a le meilleur gain (récompense moyenne par cycle) dans le MDP. Les actions transitoires optimales sont celles qui ont le meilleur biais pour atteindre le circuit optimal.

Dans cet article, on présente l'algorithme *Average Engaged Climber* (AEC), une version modifiée de l'algorithme EC (Perotto, 2015), qui estime le gain et le biais de chaque état, similairement à ce qui est fait dans l'algorithme *H-learning* (Tadepalli, 2010). AEC fonctionne par itération de valeurs, calculant

les fonctions d'utilité à partir des modèles de transition et de récompense. Pendant le calcul, la politique d'exploitation issue de la fonction d'utilité calculée conduira l'agent, à partir de chaque état, soit vers un état intéressant (un " pic " de récompense) dans un avenir proche, soit vers un circuit. Le gain g représente la récompense moyenne de la politique dans une limite de temps à l'infini. Après la convergence, cette valeur doit correspondre à la récompense moyenne dans le circuit optimal. Le biais h représente la différence liée aux récompenses transitoires, et il est défini par la récompense moyenne du meilleur chemin vers le circuit optimal.

Pour être en mesure d'estimer ces moyennes, chaque état est associé à un état cible $t(s) = s^\star$ (l'état de plus grande utilité sur un horizon proche) et à un temps d'engagement $e(s) \in \mathbb{N}$ (le nombre estimé de pas de temps pendant lesquels la politique doit être suivie pour que l'agent puisse atteindre l'état $s^\star$ à partir de s).

Les modèles de transition et de récompense ($\widehat{T}$ et $\widehat{R}$) sont associés à une mesure de confiance $\{\sigma \in \mathbb{R} \mid 0 \leq \sigma \leq 1\}$. $\sigma_{\widehat{T}}(s,a)$ représente la confiance sur la probabilité estimée des transitions à partir de l'état s avec l'action a, donnée par $\widehat{T}$. De la même façon, $\sigma_{\widehat{R}}(s,a)$ représente la confiance sur la récompense prévue par $\widehat{R}$ pour l'exécution de l'action a dans l'état s.

L'algorithme maintient simultanément deux fonctions d'utilité. La fonction $V_R(s,a)$ représente l'*utilité de récompense*, calculée à partir du modèle de récompenses $\widehat{R}$. La fonction $V_K(s,a)$ représente l'*utilité d'apprentissage*, calculée à partir de la confidence dans les modèles, $\sigma_{\widehat{T}}$ et $\sigma_{\widehat{R}}$. Les utilités estimées par ces deux fonctions sont accompagnées d'une valeur qui correspond à l'engagement nécessaire $e_R(s,a)$ et $e_K(s,a)$ (en pas de temps), et de l'indication d'un état-cible $t_R(s,a)$ et $t_K(s,a)$, qui représente le "pic" à atteindre. Ainsi, V_K est l'utilité qui guide l'agent vers des possibles améliorations de ses connaissances, tandis que V_R est l'utilité classique, qui guide l'agent vers les bonnes récompenses.

Les fonctions $V(s,a)$ sont composées d'une paire de fonctions $V(s,a) = \langle g(s,a), h(s,a)\rangle$. Pendant le calcul par itération de valeurs, $V(s,a)$ est équivalente à $g(s,a)$, à moins que $g(s,a)$ soit encore indéfini. Dans ce cas, la valeur de $V(s,a)$ est équivalente à $h(s,a)$. Dans la comparaison entre deux valeurs, si le gain est équivalent, le biais est utilisé pour briser l'égalité. Les politiques $\pi_R(s)$ et $\pi_K(s)$ sont définies pour un état donné s comme l'action a qui maximise l'utilité respective.

Les fonctions d'utilité sont mises à jour par les méthodes $UpdateV_R(\widehat{T},\widehat{R})$ et $UpdateV_K(\widehat{T},\sigma)$ par *itération sur les valeurs* suivant les équations suivantes :

$$V_R(s) = \max_a \left[\sum \widehat{T}(s,a,s').\frac{(\widehat{R}(s,a) + (V_R(s').e(s'))}{e(s')+1}\right] \tag{10}$$

$$V_K(s) = \max_a \left[\sum \widehat{T}(s,a,s').\frac{(\sigma(s,a) + (V_K(s').e(s'))}{e(s')+1}\right] \tag{11}$$

La boucle principale de fonctionnement de la méthode AEC reste assez standard : l'agent interagit avec l'environnement, ensuite il se base sur l'expérience récente pour mettre à jour ses modèles, puis il calcule les utilités, et finalement il redéfinit les politiques. La différence par rapport aux algorithmes classiques est le maintien de deux fonctions d'utilité et de deux politiques (explorer/exploiter). Une autre différence est la séparation de la prise de décisions en deux étapes distinctes : le choix de la stratégie, et le choix de l'action.

L'algorithme 1 présente la boucle principale de AEC. À chaque pas de temps, après avoir observé l'état actuel du processus et son propre niveau d'énergie, l'agent choisit (ou maintient) une stratégie (explorer ou exploiter). Lorsque la stratégie est choisie, une action peut être sélectionnée à partir de la politique correspondante (π_K ou π_R). Une fois que l'action est exécutée, l'agent reçoit une récompense et observe l'état suivant du processus. Après cette phase d'interaction, la phase d'apprentissage intervient dans l'algorithme. D'abord les modèles de transition et de récompense ($\widehat{T}$ et $\widehat{R}$), sont mis à jour, ensuite les fonctions d'utilité (V_K et V_R) et de confiance (σ_K et σ_R), et enfin les politiques (π_K et π_R).

Algorithme 1 (Lifecycle())

$Initialize()$

loop

$s \leftarrow ObserveState()$
$\omega \leftarrow ObserveEnergy()$
$\pi \leftarrow ChooseStrategy()$

$a \leftarrow \pi(s)$
$ExecuteAction(a)$
$r \leftarrow GetReward()$
$s' \leftarrow ObserveState()$

$\widehat{T} \leftarrow UpdateT(\widehat{T}, s, a, s')$
$\widehat{R} \leftarrow UpdateR(\widehat{R}, s, a, s', r)$

$V_R \leftarrow UpdateV_R(\widehat{T}, \widehat{R})$
$V_K \leftarrow UpdateV_K(\widehat{T}, \sigma(\widehat{T}), \sigma(\widehat{R}))$

$\pi_R \leftarrow UpdatePolicy(V_R)$
$\pi_K \leftarrow UpdatePolicy(V_K)$

end loop

Construire un modèle du monde à partir de l'interaction constitue un défi à part entière, et il est en dehors de la portée de cet article. Dans la littérature scientifique en MBRL, plusieurs algorithmes sont proposés pour approcher les fonctions de transition et de récompense. Dans nos expériences, on utilise des méthodes simples et largement utilisées : l'estimation du *maximum de vraisemblance* (*maximum likelihood*) pour apprendre $\widehat{T}$, et la *moyenne mobile exponentielle* (*exponential moving averaging*), pour apprendre $\widehat{R}$.

Dans la méthode AEC, lorsque l'agent choisit une stratégie (l'exploration ou l'exploitation), il reste engagé avec elle pendant le nombre de cycles défini par la valeur de $e(s)$, qui indique le temps d'engagement associé à l'état courant au moment du choix. L'engagement est censé donner à l'agent suffisamment de temps pour atteindre le pic ciblé, qui correspond à l'état $t(s)$. Le mouvement est dit réussi si l'agent, après un temps $e(s)$, à partir de l'état s, en suivant sans hésitation la stratégie choisie, arrive à l'état $t(s)$. Lorsque la durée de l'engagement est terminée, l'agent redéfinit sa stratégie et prend un nouvel engagement.

La méthode $ChooseStrategy()$ (algorihm 2) identifie trois cas différents : (1) lorsqu'une situation critique est détectée, la stratégie est définie en fonction du type de situation ; (2) sinon, si l'agent est toujours engagé, la stratégie reste inchangé ; (3) finalement, si l'engagement précédant est terminé, une nouvelle stratégie est choisie de manière stochastique selon un paramètre ε.

Trois différents types de situation critique peuvent être identifiés : (a) il n'y a rien à exploiter à partir de l'état actuel, $V_R(s) < 0$, dans ce cas l'exploration doit être préférée ; (b) il n'y a plus rien à explorer à partir de l'état actuel, $V_K(s) = 0$, alors l'exploitation doit être préférée ; (c) l'énergie ω de l'agent est en dessous du niveau critique ω_{crit}, donc l'exploitation doit être préférée.

La variable ξ indique le nombre de cycles que l'agent doit rester encore engagé. La valeur de ξ est décrémentée à chaque pas de temps, et elle est mise à jour lorsque l'agent choisit une nouvelle stratégie de façon à correspondre à la valeur de $e(s)$ défini pour la stratégie choisie, où s est l'état actuel au moment de la prise de décision.

Algorithme 2 (ChooseStrategy())
if $V_R(s) < 0.0$ ***and*** $V_K(s) > 0.0$ ***then***
{nothing to exploit from here, change to exploration (case 1a)}
$\pi \leftarrow \pi_K$; $\xi \leftarrow e_K(s)$
else if $V_K(s) = 0.0$ ***then***
{nothing to explore from here, change to exploitation (case 1b)}
$\pi \leftarrow \pi_R$; $\xi \leftarrow e_R(s)$
else if $\omega < \omega_{crit}$ ***and*** $V_R(s) > 0$ ***then***
{energy level is critical, change to exploitation (case 1c)}
$\pi \leftarrow \pi_R$; $\xi \leftarrow e_R(s)$
else if $\xi = 0$ ***then***
{engagement is over, choose strategy using epsilon (case 2)}
if $rnd() > \varepsilon$ ***then***
$\pi \leftarrow \pi_R$; $\xi \leftarrow e_R(s)$
else
$\pi \leftarrow \pi_K$; $\xi \leftarrow e_K(s)$
end if

else
{otherwise preserve engaged strategy decrementing engagement time (case 3)}
$\xi \leftarrow \xi - 1$
end if

Comme nous l'avons déjà mentionné, parfois la stratégie du moment, avec laquelle l'agent est engagé, ne fait plus de sens (explorer quand une alerte d'énergie existe, explorer quand il n'y a plus rien à découvrir à partir de l'état actuel, ou exploiter lorsque la politique courante prévoit des récompenses négatives), dans ces cas, l'engagement doit être rompu.

L'algorithme AEC a été conçu pour traiter les C-MDPs où l'énergie ω de l'agent évolue suivant la récompense cumulée. En général, lorsque l'énergie devient inférieure à un certain niveau critique, la stratégie d'exploitation doit être activée afin de faire l'énergie revenir à un état non-critique, ce qui assure la survie de l'agent.

L'algorithme 3 montre comment l'utilité est calculée. Pour chaque paire état-action, cinq valeurs doivent être mises à jour à chaque itération : le gain $g(s,a)$, ce qui représente la récompense moyenne prévue à l'intérieur du circuit à être atteint, la taille $e_g(s,a)$ de tel circuit, le biais $h(s,a)$, qui représente la récompense moyenne dans le chemin transitoire qui amène à ce circuit, la taille $e_h(s,a)$ de ce chemin, et l'état cible $t(s,a)$, qui représente un pic d'utilité au but d'un chemin transitoire qui n'a pas encore trouvé un circuit, ou un point d'entrée dans un circuit.

Au début, toutes les paires état-action sont initialisées avec un gain indéfini, noté par un engagement nul $e_g(s,a) = 0$, mais avec un biais défini équivalent à la récompense attendue. À chaque nouvelle itération, la trajectoire transitoire est augmentée d'un état, et le biais (récompense moyenne dans la trajectoire) recalculé. Lorsque la paire état-action atteint un circuit, alors le gain est défini avec la récompense moyenne dans le circuit.

Ainsi, au cours des itérations, une paire état-action sera dans l'une de ces quatre situations : (a) simplement initialisée avec la récompense immédiate attendue comme biais, (b) une partie d'une trajectoire vers un pic, (c) une partie d'un circuit, ou (d) une partie d'une trajectoire vers un circuit. À la convergence, toutes les paires état-action doivent faire partie d'un circuit optimal ou d'un chemin optimal vers un circuit optimal.

Algorithme 3 (UpdateValues())
for all *(s,a)* ***do***
$s' \leftarrow ExpectedNextState(s, a, \widehat{T})$
$r \leftarrow ExpectedNextReward(s, a, s', \widehat{R})$
if $e_h(s') = 0$ ***and*** $e_g(s') = 0$ ***then***
{first visit (case a)}
$h(s,a) \leftarrow r$
$e_h(s,a) \leftarrow 1$
$g(s,a) \leftarrow 0.0$
$e_g(s,a) \leftarrow 0$
$t(s,a) \leftarrow s'$
else if $e_h(s') > 0$ ***and*** $e_g(s') = 0$ ***then***
{transient trajectory (case b)}
$h(s,a) \leftarrow ((h(s') * e_h(s')) + r)/(e_h(s') + 1)$
$e_h(s,a) \leftarrow e_h(s) + 1$
$g(s,a \leftarrow 0.0$
$e_g(s,a) \leftarrow 0$
$t(s,a) \leftarrow t(s')$
else if $e_h(s') = 0$ ***and*** $e_g(s') > 0$ ***then***
{immediate next state is in a circuit (case c)}
$h(s,a) \leftarrow r$
$e_h(s,a) \leftarrow 1$
$g(s,a) \leftarrow g(s')$
$e_g(s,a) \leftarrow e_g(s')$
$t(s,a) \leftarrow t(s')$
else if $e_h(s') > 0$ ***and*** $e_g(s') > 0$ ***then***

{trajectory toward a circuit (case d)}
$h(s,a) \leftarrow ((h(s') * e_h(s')) + r)/(e_h(s') + 1)$
$e_h(s,a) \leftarrow e_h(s) + 1$
$g(s,a) \leftarrow g(s')$
$e_g(s,a) \leftarrow e_g(s)$
$t(s,a) \leftarrow t(s')$
end if
end for
for all (s) ***do***
$V(s) \leftarrow \max_a \langle g(s,a), h(s,a) \rangle$
end for

3.1 Résultats Expérimentaux

L'expérience de la chaîne en boucle, version récurrente du problème proposé dans (Kaelbling *et al.*, 1996), configure un scénario de récompense retardée où l'agent doit découvrir la séquence unique d'actions qui mène au bout de la chaîne, comme illustre la figure 4. Une simple mauvaise action au milieu de la séquence suffit pour renvoyer l'agent de retour au point de départ. Une exploration non-dirigée demande un nombre de cycles qui s'élève à $O(2^{|S|})$ rien que pour atteindre la grande récompense une première fois, alors qu'une exploration dirigée nécessite seulement O($|S|^2$) cycles. Si'l n'y a pas de coût d'exploration, ni de limite de temps, une méthode comme *R-Max*, optimiste face à l'incertitude, suffit pour résoudre le problème : l'agent procédera à une exploration intensive de l'environnement avant de commencer progressivement à l'exploiter. Cependant, dans un C-MDP, un équilibre plus raffiné entre l'exploration et de l'exploitation se fait nécessaire.

Dans le scénario expérimental proposé, les fonctions de transition et de récompense sont déterministes. L'agent connaît le modèle de transition mais il doit apprendre un modèle de récompenses. Une chaîne comportant 10 états donne une récompense négative (-1,0) lorsque l'agent avance vers le dernier état, et une petite récompense positive (+0,5) quand il prend l'action qui lui renvoie au premier état. Quand l'agent effectue la séquence correcte d'actions, traversant la chaîne en parcourant tous ses états, il reçoit la grande récompense (+20,0). Ainsi, pour atteindre la grande récompense l'agent doit persévérer dans la marche, et accepter de recevoir une série de récompenses négatives avant de trouver son bonheur.

Dans cette expérience, le paramètre de confiance σ est mis à 1 quand une paire état-action est observée, autrement il a la valeur 0. L'énergie $\omega = 20 + \sum_{t=1}^{n} r_t$ est équivalent à la récompense cumulée ajoutée d'une énergie initiale positive, $\omega_0 = +20$. Le niveau d'énergie critique est fixé à $\omega_{crit} = +10$, ainsi des niveaux d'énergie inférieurs à ce seuil représentent une situation d'alerte. Le défi pour l'agent est de trouver la politique optimale en restant en vie.

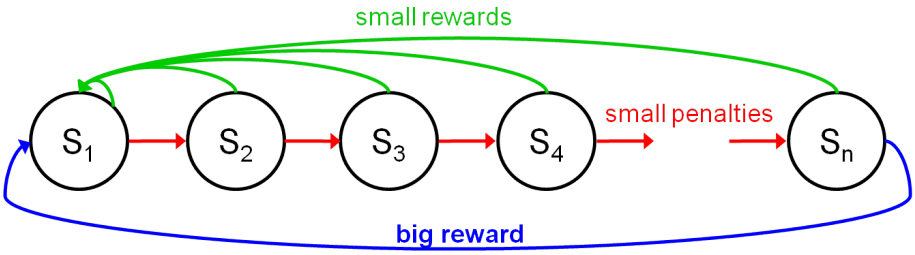


FIGURE 4 – Expérience de la châine en boucle

Dans des résultats préliminaires, AEC a démontré une performance intéressante. Concernant les C-MDPs, il bat la méthode ε-*greedy*, vu que cette dernière s'appuie sur une exploration non-dirigée, qui n'est pas adaptée aux problèmes de décision séquentielle en chaîne. Lorsque comparé à une méthode *optimiste face à l'incertitude*, AEC présente l'avantage de gérer les coûts d'exploration. Le tableau 1 montre les performances moyennes des trois méthodes sur 20 simulations. Pour la méthode ε-*greedy* nous avons réglé $\varepsilon = 0.99$ (exploration presque complètement aléatoire), sinon elle mettrait un temps démesurément long pour arriver à la fin de la chaîne, perturbée par les petites récompenses. La méthode optimiste en face à

TABLE 1 – Résultats expérimentaux

METHOD	TIME TO BIG REWARD	MINIMUM ENERGY	SURVIVAL RATE
ε-Greedy	≈ 1200	≈ -400	0%
Optimistic-Greedy	≈ 60	≈ -30	0%
Average Engaged Climber	≈ 180	≈ 0	100%

l'incertain a été implémentée simplement par une initialisation du modèle de récompenses avec des valeurs très positives. Cette méthode peut rapidement atteindre la grande récompense, vu qu'elle fera une exploration dirigée et intensive au début de la simulation, mais justement, cette façon d'explorer ne prend pas en compte les coûts, amenant l'agent systématiquement à la mort. Ainsi, due à la gestion des coûts d'exploration, AEC nécessite plus de temps que la méthode optimiste pour atteindre la grande récompense, mais il évite d'atteindre des niveaux d'énergie négatifs.

En outre, la manière standard pour le calcul de l'utilité, fondée sur la somme des récompenses actualisées, peut amener l'agent à une politique non-optimale. Dans l'expérience proposée, lorsque $\gamma = 0.8$, l'utilité de rester dans le premier état devient supérieure à l'utilité de suivre la chaîne jusqu'au bout. Le problème est que la grande récompense est trop loin, compte tenu du facteur de réduction. La méthode AEC calcule l'utilité en estimant la récompense moyenne, n'étant donc pas vulnérable à ce piège.

4 Conclusion et Perspectives

Quand l'agent est en mesure d'estimer la valeur de l'information, en comparant son importance à la valeur des récompenses à venir, il peut mieux équilibrer le temps consacré à l'exploration ou à l'exploitation. En MBRL, les méthodes standards suivent l'une des deux procédures : (a) soit l'agent réévalue la stratégie à chaque cycle (c'est le cas des méthodes de la famille ε), (b) soit l'agent construit une politique unique où l'utilité de l'exploration et l'utilité de l'exploitation sont confondues. La méthode proposée dans cet article sépare explicitement le processus de prise de décision en deux étapes : (1) choisir la stratégie (d'explorer ou d'exploiter), et persister dans le choix fait, et, à partir de ce premier choix, (2) sélectionner l'action à effectuer suivant la politique relative à la stratégie choisie.

L'utilisation des bonus d'exploration pour les états moins visités (l'optimisme face à l'incertitude) fait une compensation inhérente entre la valeur des récompenses futures et la valeur des apprentissages possibles. Des méthodes qui suivent ce principe mélangent l'*utilité de récompense* avec l'*utilité d'apprentissage* afin de pouvoir les comparer sur la base d'une "monnaie commune". Cependant, l'optimisme face à l'incertitude provoque généralement une longue phase d'exploration préliminaire. Selon le scenario, il n'est pas possible de se le permettre, par exemple, lorsqu'il y a des enjeux de sécurité, ou des risques à prendre en compte, ou des coûts d'exploration qui doivent respecter un certain budget.

Des expériences préliminaires ont montré que les choix faits par l'algorithme AEC peuvent constituer une voie prometteuse de recherche pour le cas des C-MDPs, même si des expériences dans de scenarios plus complexes sont encore nécessaires, ainsi qu'une analyse théorique plus approfondie de l'algorithme, pour les aspects de complexité et de robustesse.

Références

ASHBY W. (1956). *Introduction to Cybernetics*. Chapman & Hall.

AUER P., CESA-BIANCHI N. & FISCHER P. (2002). Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, **47**(2-3), 235–256.

BRAFMAN R. & TENNENHOLTZ M. (2002). R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *J. Mach. Learn. Res.*, **3**, 213–231.

FEINBERG E. & SHWARTZ A. (2002). *Handbook of Markov Decision Processes : methods and applications*. Kluwer.

KAELBLING L., LITTMAN M. & MOORE A. (1996). Reinforcement learning : A survey. *J. Artif. Int. Res.*, **4**(1), 237–285.

KEARNS M. & SINGH S. (2002). Near-optimal reinforcement learning in polynomial time. *Mach. Learn.*, **49**(2-3), 209–232.

MEULEAU N. & BOURGINE P. (1999). Exploration of multi-state environments : Local measures and back-propagation of uncertainty. *Mach. Learn.*, **35**(2), 117–154.

PEROTTO F. (2015). Looking for the right time to shift strategy in the exploration/exploitation dilemma. *Schedae Informaticae*, **24**. to appear.

PUTERMAN M. (1994). *Markov Decision Processes : Discrete Stochastic Dynamic Programming*. Wiley, 1 edition.

PUTERMAN M. & PATRICK J. (2010). Dynamic programming. In C. SAMMUT & G. WEBB, Eds., *Encyclopedia of Machine Learning*, p. 298–308. Springer.

O. SIGAUD & O. BUFFET, Eds. (2010). *Markov Decision Processes in Artificial Intelligence*. iSTE - Wiley.

SUTTON R. & BARTO A. (1998). *Introduction to Reinforcement Learning*. MIT Press.

TADEPALLI P. (2010). Average-reward reinforcement learning. In C. SAMMUT & G. WEBB, Eds., *Encyclopedia of Machine Learning*, p. 64–68. Springer.

TADEPALLI P. & OK D. (1998). Model-based average reward reinforcement learning. *Artif. Int.*, **100**(1-2), 177–224.

TOKIC M. (2010). Adaptive epsilon-greedy exploration in reinforcement learning based on value difference. In *Proc. of the 33rd KI*, p. 203–210 : Springer-Verlag.

TOKIC M. & PALM G. (2011). Value-difference based exploration : Adaptive control between epsilon-greedy and softmax. In *Proc. of the 34th KI*, p. 335–346 : Springer-Verlag.

UTHER W. (2010). Markov decision processes. In C. SAMMUT & G. WEBB, Eds., *Encyclopedia of Machine Learning*, p. 642–646. Springer.

VALIANT L. (1984). A theory of the learnable. *Commun. ACM*, **27**(11), 1134–1142.

WIERING M. & OTTERLO M. (2012). Reinforcement learning and markov decision processes. In *Reinforcement Learning : State-of-the-Art*, p. 3–42. Springer.

Résolution de PDMF^3 : processus décisionnels de Markov à transitions, récompenses et politiques stochastiques factorisées*

Julia Radoszycki, Nathalie Peyrard, Régis Sabbadin

INRA-MIAT (UR 875), F-31326, Castanet-Tolosan, France. prenom.nom@toulouse.inra.fr

Abstract : Les processus décisionnels de Markov à espaces d'état et d'action factorisés, généralement appelés PDMF-AF, fournissent un cadre riche pour modéliser les problèmes de décision séquentielle sous incertitude dont l'espace d'état et d'action sont de grande dimension et hautement structurés, comme en robotique, en biologie de la conservation, ou en épidémiologie du paysage. Cependant, même les algorithmes de résolution dédiés (exacts ou approchés) ne s'appliquent pas quand les dimensions des espaces d'état et d'action excèdent toutes les deux 20 ou 30, sauf sous des hypothèses fortes sur les transitions des états ou sur la fonction de valeur. Dans cet article, nous introduisons le cadre PDMF^3 et des algorithmes de résolution approchée associés qui peuvent traiter des problèmes de beaucoup plus grande dimension. Un PDMF^3 est un PDMF-AF dont la fonction de récompense se décompose de manière additive et dont les politiques solutions sont contraintes à être factorisées et peuvent être stochastiques. Les algorithmes que nous proposons appartiennent à la famille des algorithmes d'itération de la politique et exploitent des outils d'optimisation continue. Nous les validons sur des expériences numériques approfondies. Sur des petits problèmes, pour lesquels la politique optimale est disponible, ils fournissent des politiques proches de la politique optimale. Sur des problèmes de plus grande taille de la sous-classe des PDMG, ils fournissent des résultats de qualité similaire à ceux obtenus avec des algorithmes de l'état de l'art. Enfin, nous montrons que nos algorithmes peuvent traiter des PDMF^3 de grande taille. Ils permettent en effet de résoudre des problèmes d'épidémiologie agricole dont l'espace d'état et d'action sont tous les deux de taille 2^{100}.

Mots-clés : processus décisionnels de Markov, méthodes de *policy gradient*, inférence dans les modèles graphiques.

1 Introduction

Les processus décisionnels de Markov (PDM) fournissent un outil intéressant pour modéliser et résoudre des problèmes de décision séquentielle sous incertitude (Puterman, 1994). Cependant, une application à des domaines comme la robotique, la gestion environnementale, etc. n'est pas directe quand les représentations de l'état et de l'action sont factorisées. Plusieurs algorithmes ont été proposés pour les PDM dont les espaces d'état et d'action sont factorisés (PDMF-AF, (Guestrin *et al.*, 2001; Kim & Dean, 2002)). Cependant, en général ils ne s'appliquent pas à des problèmes dont le nombre de variables d'état et d'action excèdent tous les deux 20 ou 30, sauf sous des hypothèses drastiques sur les transitions des états ou la fonction de valeur (voir section 2).

Dans cet article, nous proposons un nouveau cadre, appelé PDMF^3, pour la résolution approchée de PDMF-AF généraux à récompense additive. F^3 signifie triplement factorisé, puisque nous considérons des politiques stochastiques factorisées en plus d'une fonction de transition et d'une fonction de récompense factorisées. Pour résoudre les PDMF^3, nous proposons un algorithme générique de type itération de la politique. Nous montrons que l'évaluation de politiques stochastiques factorisées revient à calculer des probabilités marginales dans un réseau Bayésien, ce qui peut être fait de manière exacte ou approchée, en utilisant des outils de l'état de l'art. L'étape de mise à jour de la politique revient à effectuer de petits changements dans les paramètres des politiques stochastiques, soit selon une seule coordonnée, soit selon le

*Ce travail a été financé par le projet ANR-13-AGRO-0001-04.

gradient de la valeur de la politique. Ce dernier peut se calculer soit par différences finies, soit en calculant les marginales d'un réseau Bayésien modifié.

Dans la Section 2, nous donnons quelques repères sur les PDMF-AF et sur la résolution de PDM factorisés ou de Dec-POMDP factorisés, en lien avec notre approche. Dans la Section 3, nous présentons notre principale contribution. Nous introduisons le cadre PDMF^3 et l'algorithme générique de type itération de la politique, basé sur une méthode de gradient. Dans la Section 4, nous présentons des expériences numériques conséquentes sur des PDMF-AF, aléatoires ou inspirés par un problème d'épidémiologie agricole.

2 Contexte

2.1 PDMF-AF

La méthode que nous proposons dans cet article est dédiée à la résolution de PDM stationnaires à espaces d'état et d'action factorisés (PDMF-AF), à horizon fini ou infini, dont la fonction de transition est représentée par un réseau Bayésien dynamique (Murphy, 2002). Nous considérons le cas de PDMF-AF dont la fonction de récompense est une somme de fonctions de faible arité. Un tel PDMF-AF est un PDM $< \mathcal{S}, \mathcal{A}, P, R, T >$ avec :

- Espace d'état factorisé: $\mathcal{S} = \prod_{i=1}^{n} \mathcal{S}_i$, où chaque $\mathcal{S}_i$ est un ensemble fini. L'état du système au temps $t \in \{0, \ldots, T\}$ est noté: $s^t = (s_1^t, ..., s_n^t) \in \mathcal{S}$.
- Espace d'action factorisé: $\mathcal{A} = \prod_{j=1}^{m} \mathcal{A}_j$, où chaque $\mathcal{A}_j$ est un ensemble fini. L'action au temps $t \in \{0, \ldots, T-1\}$ est notée: $a^t = (a_1^t, ..., a_m^t) \in \mathcal{A}$.
- Fonction de transition factorisée: $\forall t \in \{0, \ldots, T-1\}$, $P(s^{t+1}|s^t, a^t) = \prod_{i=1}^{n} P_i(s_i^{t+1}|pa_P(s_i^{t+1}))$ où $pa_P(s_i^{t+1}) = pa_P^S(s_i^{t+1}) \bigcup pa_P^A(s_i^{t+1})$, $pa_P^S(s_i^{t+1}) \subset \{s_j^t, j = 1...n, s_{j'}^{t+1}, j' = 1...n, j' \neq i\}$ et $pa_P^A(s_i^{t+1}) \subset \{a_k^t, k = 1...m\}$. Les arcs synchrones sont autorisés, mais le graphe dirigé sous-jacent doit être acyclique.
- Fonction de récompense additive: $R(s^t, a^t) = \sum_{\alpha=1}^{r} R_\alpha \left(pa_R(R_\alpha^t)\right)$, où $pa_R(R_\alpha^t) \subset \{s_j^t, j = 1...n\} \cup \{a_k^t, k = 1...m\}$. $R(s^t, a^t)$ est supposée positive et bornée par $R_{\max}$.
- $0 < T \leq +\infty$ l'horizon du problème.

Une politique stationnaire pour un PDM est définie comme une fonction $\delta : \mathcal{S} \mapsto \mathcal{A}$, qui décide de l'action globale a^t selon l'état global s^t. Sa valeur pour un état initial donné $s^0 \in \mathcal{S}$ est $V_\delta^{R,T}(s^0) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R^t \mid s_0, \delta\right]$, où $0 < \gamma \leq 1$ est un *facteur d'amortissement* donné[1]. Résoudre un PDM revient à trouver la politique δ^* de valeur maximale pour tout état initial : $\forall s^0 \in \mathcal{S}, \forall \delta : \mathcal{S} \mapsto \mathcal{A}, V_{\delta^*}(s^0) \geq V_\delta(s^0)$ (il existe toujours une telle politique déterministe (Puterman, 1994)).

Comme les solutions optimales des PDMF-AF sont hors de portée, nous n'essaierons pas de trouver une politique qui soit optimale pour tout état initial. Nous considérerons une distribution initiale donnée sur les états, $P^0(s^0)$, et rechercherons une politique maximisant la valeur espérée pour la distribution initiale P^0 : $V_\delta^{R,T}(P^0) = \sum_{s^0} P^0(s^0) V_\delta^{R,T}(s^0)$. Pour pouvoir être stockée en mémoire, la distribution initiale doit être représentée comme un réseau bayésien. De plus, nous limiterons notre recherche aux politiques stationnaires, même dans le cas d'un horizon fini où la politique optimale n'est pas forcément stationnaire, puisqu'elles peuvent être décrites de manière plus concise et sont plus faciles à interpréter et à calculer.

2.2 État de l'art

Dans les PDMF-AF, les politiques optimales, déterministes, sont décrites par une liste de fonctions δ_j^*: $\mathcal{S} \to \mathcal{A}_j$. Calculer, et même représenter, de telles politiques globales peut rapidement devenir difficile quand n augmente. C'est pourquoi la plupart des approches de résolution de PDMF-AF de grande taille ont tenté de remédier à cela en calculant des politiques approchées. Ces politiques sont soit (i) décrites par des arbres de décision ou des diagrammes de décision, (ii) des politiques factorisées, $\left(\prod_{i \in I_j} \mathcal{S}_i\right) \to \mathcal{A}_j$, où

[1] Dans le cas d'un horizon fini, on peut avoir $\gamma = 1$.

tous les $I_j \subset \{1, \ldots, n\}$ sont de faible cardinal, ou (iii) des politiques factorisées paramétrées, dont chaque politique locale est en plus paramétrée.

Dans la première famille d'approches, des algorithmes efficaces ont été proposés pour les PDMs à espace d'état factorisé (SPUDD (Hoey *et al.*, 1999) et APRICODD (St-aubin *et al.*, 2000)) mais ils ne passent pas à l'échelle quand l'espace d'action est aussi factorisé et de grande dimension. Moins d'attention a été portée au cas plus difficile où à la fois l'espace d'état et l'espace d'action sont factorisés. (Raghavan *et al.*, 2012) a proposé un algorithme de programmation dynamique symbolique pour PDMF-AF, qui a le même inconvénient : retourner une politique globale, représentée sous forme d'arbre, qui peut potentiellement prendre un espace exponentiel pour être représentée.

Dans la seconde famille d'approches, le cadre des POMDP décentralisés (Dec-POMDPs, (Bernstein *et al.*, 2002)) considère un espace d'action factorisé et des observations locales, conduisant à des politiques factorisées. Cependant, l'espace d'état est le plus souvent *non factorisé* et de petite taille, comparé à l'espace d'action (Dibangoye *et al.*, 2014). Il y a quelques exceptions, cependant, comme Dibangoye *et al.* (2012), qui considère des observations et des transitions indépendantes. Kumar *et al.* (2011) suppose que $V_\delta^{R,T}(s^0)$ s'écrit comme une somme de fonctions de faible arité par rapport à l'état initial, une propriété qui est vraie pour certaines sous-classes de Dec-POMDPs, principalement quand la fonction de transition présente une propriété d'indépendance. Ils proposent un algorithme EM qui passe à l'échelle, et résout des problèmes où $|\mathcal{S}| \approx 2^{58}$, $|\mathcal{A}| \approx 2^{40}$, $|\Omega| \approx 2^{47}$.

Toujours dans la seconde famille, Oliehoek *et al.* (2013) propose une approche basée sur les jeux Bayésiens collaboratifs graphiques. Le facteur limitant de cette approche est l'horizon T (l'horizon le plus grand considéré est $T = 6$, pour 100 variables d'action). Les PDM sur graphe (PDMG, (Sabbadin *et al.*, 2012; Cheng *et al.*, 2013)) forment une sous-classe de PDMF-AF qui appartient aussi à la seconde famille. Dans le cadre des PDMG, une seule variable d'action et une seule fonction de récompense sont associées à chaque variable d'état, et récompense, transition et politique sont supposées avoir la même structure de factorisation. Les algorithmes proposés dans le cadre PDMG font soit une approximation de la fonction de valeur comme une somme de fonctions de faible arité par rapport à l'état initial (Forsell & Sabbadin, 2006; Peyrard & Sabbadin, 2006), soit une approximation multiplicative (Cheng *et al.*, 2013).

Dans la troisième famille d'approches, Sallans & Hinton (2004) et Buffet & Aberdeen (2009) proposent des algorithmes d'apprentissage par renforcement qui optimisent des politiques paramétrées pour les PDMF-AF. L'approche de Sallans & Hinton (2004) a été appliquée à des problèmes avec au plus 40 variables d'action binaires (et moins de variables d'état). L'algorithme de gradient basé sur des simulations proposé par Buffet & Aberdeen (2009) peut s'appliquer à des problèmes de plus grande taille. Cependant, il ne permet pas de résoudre des problèmes avec des milliers de paramètres, ce qui est le cas quand on considère des politiques stochastiques factorisées comme dans notre approche.

L'approche que nous proposons est similaire dans l'esprit aux deux dernières approches, puisque nous traitons des PDMF-AF généraux, et que nous utilisons des approximations des probabilités marginales du réseau Bayésien dynamique sous-jacent pour évaluer les politiques. Mais, tandis que l'algorithme EM de Kumar *et al.* (2011) est de type *planification par inférence*, puisqu'il modélise directement le problème d'optimisation de la politique comme un problème d'inférence, nous utilisons seulement les méthodes d'inférence pour évaluer les politiques et maintenons des phases de mise à jour des paramètres. Autant que nous le sachions, aucune approche de planification par inférence n'a été proposée qui permette de résoudre des PDMF-AF généraux avec autant de variables que ce que nous faisons dans nos expérimentations numériques.

Dans la section suivante, nous proposons une nouvelle approche, appartenant à la seconde famille, où on considère des *politiques stochastiques factorisées* avec une structure définie *a priori*.

3 PDMF^3

3.1 Politiques stochastiques factorisées et PDMF^3

Un PDMF^3 est un PDMF-AF à fonction de récompense additive et politiques stochastiques factorisées (PSF). Les PSF généralisent les politiques factorisées déterministes. Elles sont définies par : $\delta(a^t|s^t) = \prod_{j=1}^m \delta_j(a_j^t|pa_\delta(a_j^t))$ où les $\delta_j(\cdot|pa_\delta(a_j^t))$ sont des distributions conditionnelles de probabilité et $pa_\delta(a_j^t) \subset \{s_i^t, i = 1...n\} \cup \{a_k^t, k = 1...m, k \neq j\}$. Nous considérons uniquement les politiques factorisées pour lesquelles le graphe des dépendances correspondant à la fois à la transition et à la PSF est acyclique (voir

Figure 1). Le problème que nous cherchons à résoudre est celui de l'optimisation de PSF de structure donnée. Ce type de problème émerge naturellement quand les agents ont une observabilité partielle de l'état joint. Dans le cas d'une observabilité totale, choisir la meilleure structure de politique, *i.e.* définir $\{pa_\delta(a_j^t), j = 1...m\}$ est une question importante et complexe que nous laissons pour de futures recherches.

La Figure 1 donne un exemple de structure de transition, de politique et de récompense pour un PDMF^3 à deux variables d'état et d'action où $pa_\delta(a_1^t) = \{s_1^t\}$ et $pa_\delta(a_2^t) = \{a_1^t, s_2^t\}$.

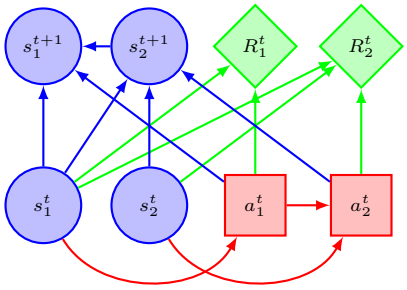


Figure 1: Exemple de structure de transition (en bleu), de récompense (en vert) et de politique (en rouge) pour un PDMF^3 à deux variables d'état et deux variables d'action. $pa_P(s_1^{t+1}) = \{s_1^t, a_1^t, s_2^{t+1}\}$, $pa_P(s_2^{t+1}) = \{s_1^t, s_2^t, a_2^t\}$, $pa_R(R_1^t) = \{s_1^t, a_1^t\}$, $pa_R(R_2^t) = \{s_1^t, s_2^t, a_2^t\}$, $pa_\delta(a_1^t) = \{s_1^t\}$ et $pa_\delta(a_2^t) = \{a_1^t, s_2^t\}$.

Une PSF donnée δ, une fonction de transition factorisée et une distribution initiale P^0 données, définissent conjointement un réseau Bayésien dynamique dans lequel la probabilité d'une trajectoire $(s,a)^{0:T} =< s^0, a^0, ...a^{T-1}, s^T >$ est:

$$P_\delta^T((s,a)^{0:T}) = P^0(s^0) \times \prod_{t=0}^{T-1} \left(\prod_{i=1}^{n} P_i(s_i^{t+1} | pa_P(s_i^{t+1})) \prod_{j=1}^{m} \delta_j(a_j^t | pa_\delta(a_j^t)) \right).$$

La valeur d'une PSF donnée δ pour une distribution initiale donnée P^0, dans le cas d'un horizon fini T[2] est définie par:

$$V_\delta^{R,T}(P^0) = \sum_{(s,a)^{0:T}} P_\delta^T((s,a)^{0:T}) \left(\sum_{t=0}^{T} \gamma^t R(s^t, a^t) \right).$$

Calculer cette espérance nécessite d'évaluer seulement certaines marginales de $P_\delta^T((s,a)^{0:T})$:

$$V_\delta^{R,T}(P^0) = \sum_{t=0}^{T} \gamma^t \sum_{\alpha=1}^{r} \sum_{pa_R(R_\alpha^t)} b_\alpha^t(pa_R(R_\alpha^t)) R_\alpha(pa_R(R_\alpha^t)),$$

où $b_\alpha^t(\cdot)$ est la distribution marginale des variables influençant la fonction de récompense α au temps t. Elle se calcule en marginalisant $P_\delta^T((s,a)^{0:T})$ par rapport à toutes les autres variables d'état et d'action dans la trajectoire $(s,a)^{0:T}$. Remarquons que la valeur s'écrit comme une somme de fonctions de faible arité de portées $\{pa_R(R_\alpha^t)\}_{\alpha \in \{1...r\}, t \in \{0,...,T\}}$. Cette décomposition est exacte et découle directement des hypothèses du cadre.

Il exitse un certain nombre d'algorithmes implémentés pour calculer de manière exacte ou approchée les marginales $b_\alpha^t(\cdot)$, comme l'algorithme exact *Junction Tree* (JT, voir par exemple (Yedidia *et al.*, 2005)), pour des problèmes de faible largeur d'arbre, ou l'algorithme approché *Loopy Belief Propagation* (LBP, (Frey & MacKay, 1998)). Des algorithmes de type Monte-Carlo (MC) peuvent aussi être utilisés pour approcher directement l'espérance.

[2]Dans le cas d'un horizon infini avec facteur d'amortissement, la somme infinie peut être approchée par une somme sur un horizon fini T suffisamment grand.

3.2 Optimisation de politiques stochastiques factorisées dans les PDMF^3

Quand on cherche des PSF optimales, on est face à un problème d'optimisation continue sous contraintes:

Problème d'optimisation dans un PDMF^3.

$$\begin{aligned} &\underset{\bar{\delta}\in(\mathbb{R}^+)^N}{\text{maximiser}} \quad V_{\bar{\delta}}^{R,T}(P^0) \\ &\text{s. c.} \qquad\qquad \sum_{a_j\in\mathcal{A}_j} \delta_j(a_j|pa_\delta(a_j)) = 1,\ \forall j, \forall pa_\delta(a_j) \end{aligned}$$

où $\bar{\delta}$ est le vecteur de coordonnées$\{\delta_j(a_j|pa_\delta(a_j)),\ \forall j, \forall pa_\delta(a_j)\}$.
Le nombre de coordonnées de $\bar{\delta}$ est:

$$N = \sum_{j=1}^{m} |\mathcal{A}_j| \prod_{k/A_k\in pa_\delta^A(a_j)} |\mathcal{A}_k| \prod_{k/S_k\in pa_\delta^S(a_j)} |\mathcal{S}_k| \tag{1}$$

Quand $\forall j = 1...m, |\mathcal{A}_j| = |\mathcal{A}_1|, |pa_\delta^A(a_j)| = |pa_\delta^A(a_1)| = z$ et $\forall i = 1...n, |\mathcal{S}_i| = |\mathcal{S}_1|$, $|pa_\delta^S(a_j)| = |pa_\delta^S(a_1)| = y$, on a

$$N = m|\mathcal{A}_1|^{z+1}|\mathcal{S}_1|^y$$

ce qui est beaucoup plus petit que le nombre de paramètres d'une politique globale déterministe ($m|\mathcal{S}| = m|\mathcal{S}_1|^n$) quand $z << m$ et $y << n$.

Théoreme. *Le problème de décision associé au problème d'optimisation dans un PDMF^3 est NP^{PP}-difficile.*
La preuve est omise pour des raisons de place. Elle utilise une réduction du problème $EMAJSAT$, qui est connu pour être NP^{PP}-difficile (Littman *et al.*, 1998). $EMAJSAT$ est le problème qui consiste, à partir d'une formule SAT sur les variables $\{X_i\}$ et $\{Y_j\}$, à décider si il existe une instanciation x^* pour laquelle la majorité des instaciations de Y satisfait la formule.

3.3 Algorithmes d'itération de la politique approchée

Le problème d'optimisation dans un PDMF^3 ne pourra pas être résolu de manière exacte en pratique puisque (i) nous aurons seulement accès la plupart du temps à une approximation de $V_{\bar{\delta}}^{R,T}(P^0)$ et (ii) le problème peut avoir plusieurs maxima locaux puisqu'il n'est *a priori* pas convexe (la plupart des algorithmes d'optimisation continue sont des algorithmes d'optimisation locale qui n'ont la garantie de converger vers un optimum global que quand la fonction objectif est convexe). Nous proposons donc une famille d'algorithmes de type itération de la politique, alternant des *évaluations approchées* et des *améliorations locales* des politiques:

- Etape d'évaluation: étant donnée la PSF courante δ^q, $V_{\bar{\delta}^q}^{R,T}(P^0)$ est évaluée de manière exacte ou approchée, en utlisant par exemple l'algorithme JT, LBP ou MC.
- Etape de mise à jour: δ^q est améliorée en δ^{q+1}, en utilisant soit une montée de gradient (GA pour *gradient ascent* (Luenberger & Ye, 2008)) soit une montée par coordonnées (CA pour *coordinate ascent* (Luenberger & Ye, 2008)).

3.3.1 Algorithmes de montée de gradient (GA)

Les algorithmes de montée de gradient fournissent un argument maximum local $\bar{\delta}$ de la valeur (éventuellement approchée) $V_{\bar{\delta}}^{R,T}(P^0)$[3]. Une simple reparamétrisation permet de se débarasser des contraintes:

$$\forall j, \forall a_j, \forall pa_\delta(a_j)\ \delta_j(a_j|pa_\delta(a_j)) = \frac{e^{\theta_j(a_j|pa_\delta(a_j))}}{\sum_{a'_j\in\mathcal{A}_j} e^{\theta_j(a'_j|pa_\delta(a_j))}},$$

[3]Ils fournissent en réalité un point critique, *i. e.* un point d'annulation du gradient, ce qui est une condition nécessaire mais pas suffisante pour avoir un maximum local.

où les coordonnées $\theta_j(a_j|pa_\delta(a_j))$ prennent des valeurs réelles. θ a la même structure que δ, et nous définissions le vecteur de coordonnées réelles $\bar{\theta}$ de manière analogue à $\bar{\delta}$ (tous les deux ont le même nombre de coordonnées).

Une fois reparamétré, le problème d'optimisation devient:

$$\underset{\bar{\theta}\in\mathbb{R}^N}{\text{maximiser}} V_{\bar{\theta}}^{R,T}(P^0) \tag{2}$$

où N est défini par l'Equation (1).

Utiliser une approche de montée de gradient pour résodure le problème de maximisation de l'Equation (2) nécessite d'être capable de calculer le gradient $\nabla_{\bar{\theta}} V_{\bar{\theta}}^{R,T}(P^0)$, soit analytiquement, soit numériquement. On peut montrer que les composantes $\frac{\partial V_{\bar{\theta}}^{R,T}(P^0)}{\partial \bar{\theta}_k}$ peuvent s'exprimer à partir de marginales d'un réseau bayésien dynamique, dérivé de celui définissant $V_{\bar{\theta}}^{R,T}(P^0)$. En théorie, le gradient peut donc être calculé de la même manière que la valeur $V_{\bar{\theta}}^{R,T}(P^0)$ elle même, en utilisant par exemple l'algorithme JT. Cependant, cela est souvent trop coûteux en pratique. Nous présentons donc une approximation du gradient par différences finies, que nous avons utilisée dans les expériences numériques: $\forall k = 1...N$,

$$\frac{\partial V_{\bar{\theta}}^{R,T}(P^0)}{\partial \bar{\theta}_k} \approx \frac{V_{\bar{\theta}^+}^{R,T}(P^0) - V_{\bar{\theta}}^{R,T}(P^0)}{\epsilon}, \tag{3}$$

où $\bar{\theta}_k^+ = \bar{\theta}_k + \epsilon$ et $\forall l \neq k\ \bar{\theta}_l^+ = \bar{\theta}_l$. $V_{\bar{\theta}}^{R,T}(P^0)$ et $V_{\bar{\theta}^+}^{R,T}(P^0)$ sont calculées de manière exacte ou approchée, en utlisant JT, LBP ou MC. ϵ est un pas petit et positif. Remarquons qu'un changement d'une seule coordonnée $\bar{\theta}_k$ de $\bar{\theta}$ induit un changement dans plusieurs coordonnées de $\bar{\delta}$. En effet, la politique $\bar{\delta}^+$ associée à $\bar{\theta}^+$ est donnée par:

$$\begin{aligned}
\bar{\delta}_k^+ &= \frac{\bar{\delta}_k e^\epsilon}{1 + (e^\epsilon - 1)\bar{\delta}_k} \\
\bar{\delta}_g^+ &= \frac{\bar{\delta}_g}{1 + (e^\epsilon - 1)\bar{\delta}_k}, \forall g \in G(k) \\
\bar{\delta}_g^+ &= \bar{\delta}_g, \forall g \notin G(k),
\end{aligned} \tag{4}$$

où $G(k)$ est défini comme suit: si $\bar{\delta}_k = \delta_j(a_j|pa_\delta(a_j))$, alors $g \in G(k)$ *si et seulement si* $\exists a'_j \neq a_j$, $\bar{\delta}_g = \delta_j(a'_j|pa_\delta(a_j))$. Une fois calculée l'approximation du gradient par différences finies, la mise à jour du vecteur de paramètres $\bar{\theta}$ est donnée par:

$$\bar{\theta}^{q+1} = \bar{\theta}^q + \eta_q \nabla_{\bar{\theta}^q} V_{\bar{\theta}^q}^{R,T}(P^0) \tag{5}$$

Il existe plusieurs façons de choisir le pas η_q utilisé dans l'Equation (5) pour mettre à jour $\bar{\theta}^q$ dans la direction du gradient à l'itération q (Luenberger & Ye, 2008). Il peut être *fixé* une fois pour toutes. Il peut aussi être choisi *optimal*, *i.e.* conduisant à une augmentation maximale de $V_{\bar{\theta}}^{R,T}(P^0)$. La méthode des *conditions de Wolfe* peut être vue comme une méthode intermédiaire de choix du pas qui fonctionne bien en pratique. Dans nos expériences numériques, nous avons comparé ces trois méthodes de recherche linéaire.

3.3.2 Algorithmes de montée par coordonnées (CA)

Dans le cas de variables d'action binaires, en utilisant le fait que $\forall j, \forall pa_\delta(a_j),\ \delta_j(a_j = 2|pa_\delta(a_j)) = 1 - \delta_j(a_j = 1|pa_\delta(a_j))$, nous pouvons construire directement un problème d'optimisation non contraint, dont le vecteur de paramètres est

$$\tilde{\delta} = \{\delta_j(a_j = 1|pa_\delta(a_j)), \forall j, \forall pa_\delta(a_j)\}.$$

Cela permet aussi de réduire le nombre de paramètres à optimiser à $N' = \frac{N}{2}$ et le problème d'optimisation devient:

$$\underset{\tilde{\delta}\in[0,1]^{\frac{N}{2}}}{\text{maximiser}} V_{\tilde{\delta}}^{R,T}(P^0) \tag{6}$$

L'algorithme de montée par coordonnées est un algorithme sans gradient. Il consiste à parcourir de manière cyclique les $\frac{N}{2}$ coordonnées $\delta_j(a_j = 1|pa_\delta(a_j))$ et essayer d'améliorer localement la valeur en

modifiant la coordonnée courante $\tilde{\delta}_k$. Les modifications peuvent consister à utiliser un *pas fixe* ou optimiser la valeur en modifiant uniquement la coordonnée courante (en utilisant l'algorithme *golden section search*, voir (Luenberger & Ye, 2008)). Si toutes les coordonnées sont restées inchangées après un cycle complet, un maximum local a été trouvé.

Les méthodes sans gradient sont généralement inapplicables à des problèmes avec plus de 1000 paramètres (Rios & Sahinidis, 2013) (nous dépassons largement ce nombre dans les problèmes que nous considérons). Cependant, l'algorithme de montée par coordonnées cyclique décrit ci-dessus a donné de bons résultats en pratique. Puisque les variables d'action sont binaires, les contraintes sont des contraintes 'de boîte' et l'algorithme de montée par coordonnées a la garantie de converger vers un maximum local de $V_{\tilde{\delta}}^{R,T}(P^0)$.

Si l'algorithme de montée de gradient permet de calculer les coordonnées du gradient en parallèle, l'algorithme de montée par coordonnées est forcément séquentiel.

4 Expériences numériques

Nous avons réalisé un certain nombre d'expériences numériques sur des problèmes de type PDMF-AF à horizon infini, ce qui est le cas le plus difficile pour notre algorithme. Dans toutes les expériences, nous avons utilisé un facteur d'amortissement de $\gamma = 0.9$.

Les différents algorithmes de résolution que nous avons testés empiriquement sont dénommés sous la forme *Optim-Eval*, où *Optim* est la méthode d'optimisation (GA ou CA) et *Eval* est la méthode d'évaluation, exacte (JT) ou approchée (LBP ou MC).

Notre code est en Matlab et utilise des appels à des fonctions de la librairie libDAI (Mooij, 2010) pour les algorithmes JT et LBP. Nous avons utilisé une implémentation Scilab de l'algortihme *Mean-Field Approximate Policy Iteration* (MF-API) pour les PDMG (Sabbadin *et al.*, 2012). Dans les PDMG, la transition, la récompense et la politique sont supposées avoir la même structure, et l'objectif est de trouver la meilleure politique déterministe factorisée. MF-API est basé sur une évaluation en champ moyen des politiques déterministes factorisées, *i.e.* une méthode 'plus approchée' que LBP. De plus, il n'y a pas de garantie avec cet algorithme d'améliorer la valeur de la politique à chaque itération. Les expériences ont été réalisées sur un serveur de 16 coeurs, et l'algorithme de montée de gradient a été parallélisé, en utilisant la toolobox Matlab de parallélisation.

Dans la Section 4.1, une comparaison des différents algorithmes d'évaluation est effectuée. Dans la Section 4.2, nous comparons les différents algorithmes d'itération de la politique approchée sur des petits problèmes ($n = 6$ variables d'état binaires, $m = 6$ variables d'action binaires, et $r = 6$ fonctions de récompense), pour lesquels une politique *globale* optimale peut être calculée. Nous avons utilisé la MDP toolobox[4] pour calculer les politiques globales optimales. Dans la Section 4.3, nous comparons les différents algorithmes d'itération de la politique approchée sur des problèmes de grande taille et de structure aléatoire (15 variables de chaque type, et des variables d'action binaires ou ternaires). Finalement, dans la Section 4.4, nous illustrons l'utilisation de nos algorithmes de type itération de la politique sur des problèmes d'épidémiologie agricole de grande taille (25 et 100 variables d'action binaires de chaque type). Nous n'avons trouvé aucune autre implémentation d'un algorithme dédié à la résolution de PDMF-AF permettant de traiter d'aussi grands problèmes. Dans la Section 4.2, nous avons utilisé une *structure de politique naturelle*, dans laquelle $pa_\delta(a_j)$ est l'union des variables d'état qui (i) soit apparaissent dans une même fonction de récompense R_α que a_j (ii) soit influencent une variable d'état conjointement avec a_j (*i. e.* apparaissent conjointement avec a_j dans un ensemble $pa_P(s_i^{t+1})$).

4.1 Méthodes d'évaluation des politiques

Dans le but de choisir une méthode d'évaluation approchée, la méthode d'évaluation des marginales est importante, mais aussi l'horizon d'approximation, $\hat{T}$. Les deux peuvent avoir un impact sur l'erreur d'évaluation et le temps de calcul. Nous avons testé l'impact de l'horizon d'approximation sur de petits PDMF^3, pour lesquels nous pouvons aller jusqu'à $\hat{T} = 100$ (Table 1).

L'erreur obtenue pour $\hat{T} = 20$ est significative (10%), mais du même ordre avec LBP qu'avec JT. MC donne de bons résultats pour $\hat{T} = 40$ (erreur de 1%), et toutes les méthodes sont assez précises pour $\hat{T} = 100$, mais trop lentes pour être incorporées dans un algorithme d'optimisation.

[4] http://www.inra.fr/mia/T/MDPtoolbox

Méthode d'évaluation	ERM	temps moyen (sec)
MC $\hat{T} = 40$	0.01	23.45
JT $\hat{T} = 20$	0.11	0.085
LBP $\hat{T} = 20$	0.11	0.049
MC $\hat{T} = 100$	6.8×10^{-4}	66.05
JT $\hat{T} = 100$	2.4×10^{-5}	1.1
LBP $\hat{T} = 100$	0.009	0.23

Table 1: Erreur relative moyenne et temps d'évaluation moyen de PSF sur 100 petits PDMF^3 aléatoires ($n = 6$ variables d'état binaires, $m = 6$ variables d'action binaires, $r = 6$ fonctions de récompense). Pour l'évaluation MC, la valeur moyenne est calculée sur 4000 trajectoires.

C'est pourquoi nous avons choisi d'utiliser l'algorithme LBP avec $\hat{T} = 20$ dans l'étape d'évaluation de nos algorithmes de type itération de la politique, puisqu'il offre un bon compromis entre temps de calcul et erreur d'approximation. L'approche MC avec $\hat{T} = 40$ (et 4000 trajectoires) sera utilisée pour comparer les valeurs des politiques retournées par ces algorithmes, quand une évaluation exacte est impossible.

4.2 PDMF^3 aléatoires de petite taille

Nous avons réalisé une série d'expériences sur de petits PDMF^3 pour lesquels la politique *globale* optimale peut être calculée, dans le but d'évaluer les politiques retournées par différentes instanciations *Optim-Eval* de notre algorithme (Table 2). La politique uniforme est utilisée pour initialiser les algorithmes. La valeur de η_q dans les algorithmes de montée par coordonnées est de 0.1, car les paramètres de la politique appartiennent à $[0, 1]$, tandis qu'elle est de 10 ou 20 dans les algorithmes de montée de gradient où les paramètres peuvent prendre n'importe quelle valeur réelle.

L'erreur relative moyenne (ERM) entre la politique optimale et les politiques approchées est donnée, ainsi que le temps de calcul moyen des politiques. Comme la politique optimale est globale, elle peut ne pas avoir la structure naturelle supposée dans nos algorithmes. L'ERM donnée est donc une borne supérieure de l'ERM par rapport à la politique factorisée optimale.

En utilisant les algorithmes CA-JT ou GA-JT, on obtient des politiques dont l'ERM par rapport à la politique globale optimale est proche de 2%. En utilisant les algorithmes CA-LBP ou GA-LBP avec différentes méthodes de recherche linéaire, on obtient une ERM de l'ordre de 12%. Nous avons aussi essayé l'algorithme GA-JT pour une structure de politique supposée générée aléatoirement, et de même taille que la structure naturelle. Nous avons alors obtenu une ERM de 0.048 (0.02 avec la structure naturelle) et un temps d'exécution moyen de 3.66 minutes (3.25 minutes avec la structure naturelle). Nous pouvons donc raisonnablement penser que, quand aucune structure de politique n'est donnée dans les contraintes du problème, la structure naturelle que nous avons proposée est un bon choix. Ici GA n'est pas parallélisé, sauf pour le cas de variables d'action ternaires où il commence à y avoir un gain en temps de calcul.

Plus généralement, plus un algorithme *Optim-Eval* donné est lent, meilleur il est en termes de qualité. La résolution exacte est plus rapide mais elle ne permet pas de trouver la meilleure politique factorisée avec la structure contrainte, tandis que CA-JT ou GA-JT fournissent une politique factorisée localement optimale présentant cette structure donnée.

4.3 PDMF^3 aléatoires de grande taille

Sur des PDMF^3 et des PDMG aléatoires, avec 15 variables de chaque type et 15 fonctions de récompense, la politique globale optimale ne peut plus être calculée. Nous avons comparé (i) CA-LBP, GA-LBP et MF-API sur des PDMG à variables binaires et (ii) CA-LBP, GA-LBP et une politique uniforme sur des PDMF^3 (Table 3). Pour les PDMF^3, nous nous sommes donné une structure de politique aléatoire, avec au plus 9 variables parent par variable d'action. Les algorithmes ont été initialisés avec la politique gloutonne dans le cas des PDMG, et avec la politique uniforme dans les autres cas. La politique gloutonne est la politique factorisée déterministe qui choisit les actions qui maximisent la récompense immédiate. Pour ces problèmes, la parallélisation de GA-LBP le rend beaucoup plus rapide que CA-LBP, au prix d'une légère perte de qualité.

Méthode - pas	ERM	temps moyen
Variables d'action binaires		
GA-JT - $\eta_q = 20$	0.0200	3.25 min
GA-LBP - $\eta_q = 20$	0.1228	1.88 min
GA-LBP - $\eta_q = 10$	0.1214	2.47 min
GA-LBP - pas de Wolfe	0.1216	2.48 min
GA-LBP - pas optimal	0.1217	2.63 min
CA-JT - $\eta_q = 0.1$	0.0222	5.14 min
CA-LBP - $\eta_q = 0.1$	0.1228	1.53 min
Exacte (politique globale)	-	1.95 min
Variables d'action ternaires		
GA-LBP - $\eta_q = 20$	0.1288	39.74s
Exacte (politique globale)	-	43.44s

Table 2: Erreur relative moyenne et temps de calcul moyen sur 100 petits PDMF^3 aléatoires ($n = 6$ variables d'état binaires, $m = 6$ variables d'action binaires, $r = 6$ fonctions de récompense). Les deux dernières lignes correspondent à 100 problèmes aléatoires avec $n = m = r = 5$ et des variables d'action ternaires.

Méthode - politique initiale - pas	Valeur (MC)	temps moyen
PDMG		
CA-LBP - gloutonne - $\eta_q = 0.1$	99.492	6.31 min
GA-LBP - gloutonne - pas de Wolfe	99.538	39.91 sec
CA-LBP - uniforme - $\eta_q = 0.1$	97.508	22.58 min
GA-LBP - uniforme - pas de Wolfe	99.472	1.25 min
MF-API - gloutonne	99.684	1.16 sec
PDMF^3, variables d'action binaires		
GA-LBP - uniforme - pas de Wolfe	85.65	9.18 min
CA-LBP - uniforme - $\eta_q = 0.1$	86.738	74.24 min
PDMF^3, variables d'action ternaires		
GA-LBP - uniforme - pas de Wolfe	76.47	14.97 min
CA-LBP	pas applicable	–
Uniform policy	65,542	–

Table 3: Evaluation Monte-Carlo des politiques solution obtenues pour de grand PDMF^3 et PDMG aléatoires (15 variables de chaque type). Les valeurs sont moyennées sur 5 problèmes aléatoires. GA est parallélisé (8 coeurs).

Sur les PDMG, nos algorithmes fournissent des PSF de bonne qualité si on compare à MF-API, mais ils sont plus lents que MF-API. Ces bons résultats en termes de qualité, tout comme ceux obtenus sur les problèmes de petite taille, nous encouragent à utiliser nos algorithmes sur des problèmes de grande taille n'appartenant pas à la sous-classe des PDMG.

4.4 Problèmes d'épidémiologie agricole

Nous considérons ici des PDMF^3 de très grande taille, inspirés par un problème d'épidémiologie agricole modélisé dans le cadre PDMG dans (Sabbadin *et al.*, 2012; Cheng *et al.*, 2013). Les parcelles sont distribuées sur une grille de taille 5×5 ou 10×10. Une variable d'état binaire (culture infectée ou saine) et une variable d'action binaire (système de culture normal ou parcelle traitée et laissée en jachère) sont attachées à chaque parcelle. La maladie peut se propager aux parcelles voisines uniquement. Dans le cas de la grille 10×10, le nombre total de paramètres décrivant une PSF est $N = 5184$.

Pour rendre le modèle PDMG initial plus réaliste, nous considérons que le traitement a lieu avant la propagation de la maladie et réduit la probabilité de contamination de parcelles voisines. Dans ce cas, le problème n'est plus un PDMG. La prbabilité d'infection devient:

$$P(\epsilon, p_1, n_1, p_2, n_2) = \epsilon + (1-\epsilon)(1-(1-p_1)^{n_1}(1-p_2)^{n_2}).$$

p_1 (resp. p_2) est la probabilité de contamination courte distance sans (resp. avec) traitement, n_1 (resp. n_2) est le nombre de parcelles voisines infectées et non traitées (resp. traitées), et ε est la probabilité de

Méthode - pas	Valeur (MC)	temps moyen
grille 5×5, départ uniforme		
Uniforme	10367	-
Gloutonne	13191	-
CA-LBP - $\eta_q = 0.1$	17264	1h16
GA-LBP - pas de Wolfe	13195	21min20
grille 10×10, départ uniforme		
Uniforme	40495	-
Gloutonne	52575	-
CA-LBP - $\eta_q = 0.1$	65068	43h42
GA-LBP - pas de Wolfe	52574	10h20

Table 4: Evaluation Monte-Carlo des politiques dans le problème d'épidémiologie agricole.

Méthode - pas - $\hat{T}$	Valeur (MC)	temps moyen
grille 5×5, $p = 0.2$, départ uniforme		
Uniforme	11393	-
Gloutonne	13953	-
Politique 1	18911	-
Politique 2	18349	-
CA-LBP - $\eta_q = 0.1$ - $\hat{T} = 20$	15055	47.25 min
GA-LBP - pas de Wolfe - $\hat{T} = 20$	14427	43.18 sec
GA-LBP - pas de Wolfe - $\hat{T} = 40$	14480	3.8 min

Table 5: Evaluation Monte-Carlo des politiques dans le problème d'épidémiologie agricole avec réseau de surveillance.

contamination longue distance. La forme de la fonction de récompense est la même que dans (Sabbadin *et al.*, 2012), avec une récompense maximale de $\rho = 100$.

La Table 4 donne les résultats obtenus pour $\epsilon = 0.01$, $p_1 = 0.6$, $p_2 = 0.4$ et $\hat{T} = 20$ (dans l'évaluation approché par LBP). En prenant $\hat{T} = 40$ nous avons obtenu des résultats similaires. Nous avons comparé les politiques issues de CA-LBP et GA-LBP avec la politique stochastique uniforme et la politique gloutonne. CA-LBP améliore la valeur de la politique gloutonne de 30% dans le cas de la grille 5×5, et de 23% dans le cas de la grille 10×10. GA-LBP converge quant à lui vers la politique gloutonne.

Enfin, nous avons considéré la situation d'un réseau de surveillance dans un problème d'épidémiologie agricole. Dans ce type de problèmes, les politiques expertes donnent de bons résultats. La transition et la récompense sont les mêmes que dans Sabbadin *et al.* (2012), mais les décisions des agriculteurs à l'échelle de la parcelle se prennent en fonction de la même information: 4 parcelles (la parcelle de coordonnées (2,2) et les trois parcelles symétriques) sont surveillées et leur état (sain ou infecté) au temps courant est connu. MF-API ne permet pas de résoudre ce problème, qui n'est pas un PDMG. Les politiques issues de CA-LBP et GA-LBP sont comparées à la politique uniforme, à la politique gloutonne et à des politiques expertes, consistant à traiter une parcelle (i) si une des parcelles monitorées parmi les plus proches est infectée (politique 1) ou (ii) si au moins trois des parcelles monitorées sont infectées (politique 2).

Les résultats montrent que CA-LBP et GA-LBP améliorent la politique uniforme ou gloutonne, GA-LBP étant plus rapide mais de qualité légèrement moins bonne que CA-LBP (Table 5). Cependant, les politiques expertes sont de meilleure valeur. Dans cet exemple, nous avons pu vérifier que les politiques expertes étaient des maxima locaux de $V_\delta^{R,T,LBP}(P^0)$ (ou des points critiques). Dans le cas général, quand des politiques expertes factorisées sont disponibles, initialiser CA-LBP ou GA-LBP avec elles est une bonne manière de les évaluer et de déterminer si elles peuvent être améliorées ou si elles sont des maxima locaux.

5 Conclusion

Nous avons proposé un cadre générique, appelé PDMF^3, et une famille d'algorithmes de résolution approchée pour des PDMF-AF de grande taille, basés sur le calcul de politiques stochastiques factorisées. Dans ce cadre, nous pouvons traiter des problèmes ayant jusqu'à 2^{100} états et actions.

Les expériences numériques ont montré que ces algorithmes retournent des politiques de valeur équivalente

ou meilleure que celles retournées par les algorithmes existants, quand ceux-ci sont disponibles. Quand les problèmes sont de trop grande taille pour être résolus avec les approches existantes, les politiques retournées par nos algorithmes améliorent des politiques arbitraires simples. Quand une connaissance experte est disponible à propos des politiques solutions, nos algorithmes peuvent utiliser ces politiques expertes comme initialisation, et soit les améliorer, soit montrer qu'elles ne peuvent être améliorées localement. Les politiques expertes peuvent également suggérer une structure de politique.

Une utilisation raisonnable de CA-LBP et GA-LBP est donc de les utiliser dans une boucle itérative Politique Experte - Optimisation - Interprétation Experte. La flexibilité de notre approche nous permet aussi d'utiliser successivement les différents algorithmes: (i) GA-LBP pour arriver rapidement à un optimum local puis (ii) CA-LBP (si les variables d'action sont binaires) pour éventuellement améliorer la valeur courante et (iii) CA-MC ou GA-MC pour optimiser une estimation asymptotiquement non biaisée de la valeur.

Le cadre que nous proposons ici admet certaines extensions naturelles. Par exemple, l'observabilité partielle est déjà traitée en partie, puisque les PSF n'ont accès qu'à une partie des variables d'état (voir la dernière expérience numérique par exemple). Il peut être étendu facilement au cas des Dec-POMDP à espace d'état factorisé, en considérant des observations pour lesquelles la probabilité d'observation est factorisée. Le principe de l'approche présentée ici serait inchangé, à condition que le graphe des dépendances formé par la transition, la PSF et la probabilité d'observation soit acyclique. Une extension plus difficile de ce travail serait de déterminer automatiquement une structure de politique satisfaisante. Déterminer la structure qui admet une PSF de valeur maximale revient à optimiser à la fois des variables discrètes (représentant la structure de la politique, avec des contraintes de parcimonie) et des variables continues (les paramètres de la PSF).

References

BERNSTEIN D., GIVAN R., IMMERMAN N. & ZILBERSTEIN S. (2002). The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, **27**(4), 819–840.

BUFFET O. & ABERDEEN D. (2009). The factored policy-gradient planner. *Artificial Intelligence*, **173**, 722–747.

CHENG Q., LIU Q., CHEN F. & IHLER A. (2013). Variational planning for graph-based MDPs. In *Advances in Neural Information Processing Systems 26*, p. 2976–2984.

DIBANGOYE J. S., AMATO C., BUFFET O. & CHARPILLET F. (2014). Exploiting separability in multiagent planning with continuous-state MDPs. In *Proceedings of the 13th international conference on Autonomous Agents and Multiagent Systems*.

DIBANGOYE J. S., AMATO C. & DONIEC A. (2012). Scaling up decentralized MDPs through heuristic search. In *Proceedings of the 28th Conference on Uncertainty and Artificial Intelligence*, p. 217–226.

FORSELL N. & SABBADIN R. (2006). Approximate linear-programming algorithms for graph-based markov decision processes. In *Proceedings of the 17th European Conference on Artificial Intelligence*, p. 590–594.

FREY B. & MACKAY D. (1998). A revolution: Belief propagation in graphs with cycles. In *Advances in Neural Information Processing Systems*, p. 479–485.

GUESTRIN C., KOLLER D. & PARR R. (2001). Multiagent planning with factored MDPs. In *Advances in Neural Information Processing Systems*, p. 1523–1530.

HOEY J., ST-AUBIN R., HU A. & BOUTILIER C. (1999). SPUDD: Stochastic planning using decision diagrams. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, p. 279–288.

KIM K.-E. & DEAN T. (2002). Solving factored MDPs with large action space using algebraic decision diagrams. In *Proceedings of the 7th Pacific Rim International Conference on Artificial Intelligence*, p. 80–89.

KUMAR A., ZILBERSTEIN S. & TOUSSAINT M. (2011). Scalable multiagent planning using probabilistic inference. In *Proceedings of the 22th International Joint Conference on Artificial intelligence*.

LITTMAN M., GOLDSMITH J. & MUNDHENK M. (1998). The computational complexity of probabilistic planning. *Journal of Artificial Intelligence Research*, **9**, 1–36.

LUENBERGER D. G. & YE Y. (2008). *Linear and Nonlinear Programming*. Springer, 3rd edition.

MOOIJ J. M. (2010). libDAI: A free and open source C++ library for discrete approximate inference in graphical models. *Journal of Machine Learning Research*, **11**, 2169–2173.

MURPHY K. (2002). *Dynamic Bayesian networks: representation, inference and learning*. PhD thesis, School of Computer Science, University of California, Berkeley.

OLIEHOEK F. A., WHITESON S. & SPAAN M. T. J. (2013). Approximate solutions for factored Dec-POMDPs with many agents. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems*.

PEYRARD N. & SABBADIN R. (2006). Mean field approximation of the policy iteration algorithm for graph-based markov decision processes. In *ECAI*, p. 595–599.

PUTERMAN M. (1994). *Markov decision processes*. John Wiley and Sons.

RAGHAVAN A., JOSHI S., FERN A., TADEPALLI P. & KHARDON R. (2012). Planning in factored action spaces with symbolic dynamic programming. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*.

RIOS L. M. & SAHINIDIS N. V. (2013). Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*, **56**, 1247–1293.

SABBADIN R., PEYRARD N. & FORSELL N. (2012). A framework and a mean-field algorithm for the local control of spatial processes. *International Journal of Approximate Reasoning*, **53**(1), 66–86.

SALLANS B. & HINTON G. E. (2004). Reinforcement learning with factored states and actions. *Journal of Machine Learning Research*, **5**, 1063–1088.

ST-AUBIN R., HOEY J. & BOUTILIER C. (2000). APRICODD: Approximate policy construction using decision diagrams. In *Advances in Neural Information Processing Systems*, p. 1089–1095.

YEDIDIA J. S., FREEMAN W. T. & WEISS Y. (2005). Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, **51**(7), 2282–2312.

Résultats structurels pour les modèles de contrôle décentralisés coopératifs⋆

Jilles Dibangoye[1], Olivier Buffet[2], Olivier Simonin[1]

[1] INSA Lyon / INRIA — CITI-Inria Lab., Lyon
jilles-steeve.dibangoye@insa-lyon.fr — http://jilles.dibangoye.net/
olivier.simonin@insa-lyon.fr — http://perso.citi.insa-lyon.fr/osimonin/

[2] INRIA / Université de Lorraine / CNRS — LORIA, Nancy
olivier.buffet@loria.fr — http://www.loria.fr/~buffet/

Les processus de décision markoviens partiellement observables décentralisés (Dec-POMDP) ont émergé comme le cadre standard pour la prise de décision séquentielle décentralisée Bernstein *et al.* (2002). Ce modèle général considère plusieurs agents avec des observations différentes qui coopèrent pour atteindre un objectif commun, mais ne peuvent pas communiquer entre eux. Malheureusement, sa complexité au pire cas a limité son applicabilité. Le cas à horizon fini est NEXP Bernstein *et al.* (2002).

Pour permettre une plus grande capacité de mise à l'échelle, beaucoup d'attention a été dévouée à des modèles avec des hypothèses restrictives concernant essentiellement la dynamique et les récompenses Goldman & Zilberstein (2004); Becker *et al.* (2004); Nair *et al.* (2005); Melo & Veloso (2011).

Ce travail propose une méthodologie générale, l'*analyse structurelle*, qui permet de concevoir, sans perte d'optimalité, des politiques et fonctions de valeur concises. Nous prouvons pour la première fois que les besoins en mémoire pour les politiques et les fonctions de valeurs peuvent être assymétriques, d'où des gains significatifs dans certains cas. Un autre résultat nouveau et important est la preuve que, sous de légères conditions sur les récompenses, la fonction de valeur optimale consiste en des fonctions linéaires des états cachés. Pour permettre une plus grande applicabilité, cette analyse structurelle repose sur une approche de résolution récente des Dec-POMDP qui passe par une ré-écriture sous la forme d'un MDP déterministe à espace d'état continu dont la fonction de valeur optimale est convexe et linéaire par morceaux Dibangoye *et al.* (2013).

Dans l'ensemble, il semble que l'analyse de complexité asymptotique fournisse une hiérarchie des problèmes, alors que l'analyse structurelle permette de guider la caractérisation de politiques et fonctions de valeurs (optimales) concises, ce qui peut conduire à développer une théorie et des algorithmes capables de passer à l'échelle, applicables et largement adaptables.

Références

Becker R., Zilberstein S., Lesser V. R. & Goldman C. V. (2004). Solving transition independent decentralized Markov decision processes. *JAIR*, **22**, 423–455.

Bernstein D. S., Givan R., Immerman N. & Zilberstein S. (2002). The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, **27**(4).

Dibangoye J. S., Amato C., Buffet O. & Charpillet F. (2013). Optimally solving Dec-POMDPs as continuous-state MDPs. In *IJCAI'13*.

Goldman C. V. & Zilberstein S. (2004). Decentralized control of cooperative systems : Categorization and complexity analysis. *JAIR*, **22**(1), 143–174.

Melo F. S. & Veloso M. M. (2011). Decentralized MDPs with sparse interactions. *AI journal*, **175**(11), 1757–1789.

Nair R., Varakantham P., Tambe M. & Yokoo M. (2005). Networked distributed POMDPs : A synthesis of distributed constraint optimization and POMDPs. In *AAAI'05*.

⋆. Ceci est un résumé d'un article à paraître à IJCAI 2015.

Plannification dans des domaines partiellement observables avec des états épistémiques flous et une dynamique probabiliste.

Nicolas Drougard[1], Florent Teichteil-Königsbuch[1], Jean-Loup Farges[1], Didier Dubois[2]

[1] Onera – The French Aerospace Lab
2 avenue Edouard Belin, 31055 Toulouse
prenom.nom@onera.fr and http://www.onera.fr/DCSD

[2] IRIT – Paul Sabatier University,
118 route de Narbonne, 31062 Toulouse
dubois@irit.fr and http://www.irit.fr/-Equipe-ADRIA-

Résumé : Une nouvelle traduction des PDMs partiellement observables (PDMPO) en PDM complètement observables est décrit dans cet article. Contrairement à la traduction classique, l'espace d'état du problème résultant est fini, permettant aux algorithmes de résolution des PDMs de résoudre cette version simplifiée du problème partiellement observable initial : cette approche représente les croyances de l'agent avec des mesures floues sur les états du système, menant à un PDM dont l'espace d'état est constitué d'un nombre fini d'états épistémiques. Après une brève description du modèle PDMPO ainsi que la présentation de notions de la Théorie de Possibilités, la traduction est décrite de manière formelle et avec des arguments sémantiques. Ensuite les calculs menant à cette transformation sont détaillés afin que le PDM final profite autant que possible de la structure factorisée du PDMPO initial : ces calculs permettent la réduction de la taille de l'espace d'état du PDM final, et font de ce dernier un PDM factorisé. Finalement cette réduction ainsi que la structure du PDM final sont illustrés sur un PDMPO simple.

Mots-clés : PDMPOs factorisés, Théorie des Possibilités, Intégrale de Choquet, Transformation pignistique, Discrétisation de l'espace des croyances

Processus décisionnels de Markov pour l'optimisation dans des systèmes d'argumentation probabilistes

Emmanuel Hadoux[1], Aurélie Beynier[1], Nicolas Maudet[1], Paul Weng[2], Anthony Hunter[3]

[1] Sorbonne Universités, UPMC Univ Paris 6, UMR 7606, LIP6, F-75005, Paris, France prenom.nom@lip6.fr

[2] SYSU-CMU Joint Institute of Engineering, SYSU-CMU Shunde International Joint Research Institute 5000 Forbes Avenue Pittsburgh, PA 15213-3890, USA paweng@cmu.edu

[3] Department of Computer Science, University College London, Gower Street, London WC1E 6BT, UK a.hunter@cs.ucl.ac.uk

Lorsque deux agents ont des points de vue contradictoires, chacun peut échanger des arguments afin d'essayer de convaincre l'agent adversaire. Les débats d'argumentation formalisent ce problème. Dans ce papier nous abordons le problème, pour un agent, de l'optimisation d'une séquence d'actions à effectuer dans un débat. Dans ce problème, nous supposons que l'adversaire agit de manière stochastique et qu'il dispose d'un état de croyances initial inconnu. Bien qu'une transformation naïve vers un modèle de Markov induise un nombre d'états ne nous permettant pas de résoudre en pratique le problème, nous montrons que l'exploitation de plusieurs propriétés des débats d'argumentation permet de le résoudre en pratique. En particulier : (1) les débats prenant place dans un espace public où les arguments sont échangés, ils peuvent être modélisés sous la forme de *Processus Décisionnel de Markov à Observabilité Mixte* (MOMDP), (2) les débats étant fortement structurés, nous pouvons appliquer des techniques d'optimisation afin de réduire la taille de l'instance initiale, sans en modifier l'expressivité ni la solution optimale. Différentes expérimentations mettent en évidence la pertinence et l'apport des MOMDP et des techniques d'optimisation présentées.

Mots-clés : Argumentation probabiliste, Prise de décision dans l'incertain, Modèles markoviens

Acknowledgments

Funded by the French National Research Agency under grant LARDONS ANR-10-BLAN-0215 and grant AMANDE ANR-13-BS02-0004.

Identification du meilleur bras dans le modèle des bandits linéaires

Marta Soare[1], Alessandro Lazaric[1], Rémi Munos[1,2]

[1] Inria Lille - Nord Europe

[2] Google DeepMind

Nous étudions le problème de l'identification du meilleur bras dans le modèle de bandit linéaire, où les récompenses des bras dépendent linéairement d'un paramètre inconnu θ^* et le but est de découvrir le bras ayant la plus grande récompense. Nous caractérisons la complexité de ce problème et nous proposons des stratégies d'allocation d'échantillons pour l'identification du meilleur bras sous une contrainte de précision donnée, tout en minimisant le nombre d'échantillons utilisés. Notamment, nous montrons l'importance de l'exploitation de la structure linéaire globale pour améliorer l'estimation des récompenses des bras presque optimaux. Nous analysons les strategies proposées et évaluons leurs performances empiriques.

Transfert séquentiel dans le modèle de bandit linéaire

Marta Soare[1], Ouais Alsharif[2], Alessandro Lazaric[1], Joelle Pineau[4]

[1] Inria Lille - Nord Europe

[2] Google

[3] McGill University

Pour s'attaquer aux tâches d'apprentissage automatique avec peu de données, on fait appel à des méthodes de transfert et méthodes multi-tâches pour exploiter l'information obtenue dans les tâches précédentes. Alors que les gains potentiels de ces approches ont été largement étudiés dans le batch learning, on en sait beaucoup moins sur l'impact du transfert dans la prise de décision séquentielle avec un feedback limité. Dans ce papier, nous étudions le transfert séquentiel d'échantillons dans un cadre où chaque tâche est un problème de bandit linéaire. Le but est d'améliorer le regret par tâche en transférant les données pertinentes des tâches précédentes. Nous proposons des algorithmes multi-tâches qui atteignent cet objectif et nous analysons leur performance théorique. Enfin, nous présentons une évaluation empirique préliminaire.

Génération de plans à base de connaissances

Anaëlle Wilczynski, Bruno Zanuttini, Jérôme Lang

Résumé : Les Knowledge-Based Programs (KBPs) associent représentation des connaissances et planification. Il s'agit de protocoles décrivant les actions à effectuer par un agent, en fonction de son état de connaissance, afin d'atteindre un but donné. Ces plans possèdent une grande expressivité, grâce à l'utilisation de la logique modale S5, et une plus grande compacité que les plans classiques. La question de la génération de tels plans n'a été que peu étudiée. Notre objectif est de combler ce manque. Nous proposons des algorithmes permettant de générer des KBPs à partir de la spécification d'un état initial, d'un but et d'un ensemble d'actions disponibles. Deux types d'algorithmes sont présentés, par progression, c'est-à-dire en partant de l'état initial pour aller vers le but, et par régression, en partant cette fois-ci du but. Dans les deux cas, un algorithme de recherche en largeur est décrit, ayant la propriété de fournir un plan optimal en nombre d'actions à effectuer dans le pire cas. De même, nous exposons dans les deux cas des algorithmes de recherche en profondeur, dans lesquels une action est choisie pour chaque état de connaissance, par le biais de différentes fonctions heuristiques dont nous montrons certaines propriétés. Nous proposons également des benchmarks adaptés, pour lesquels la planification classique est moins précise, et testons nos algorithmes sur ces problèmes.

TouIST (Toulouse Integrated Satisfiability Tool) : utilisation de la plateforme pour la planification

Khaled Skander Ben Slimane, Alexis Comte, Olivier Gasquet, Abdelwahab Heba, Olivier Lezaud, Frédéric Maris, and Maël Valais

IRIT - CNRS UMR 5505
Université Paul Sabatier – Toulouse 3
118 route de Narbonne, 31062 Toulouse Cedex 9
Frederic.Maris@irit.fr

Résumé : La plateforme de résolution TouIST (Ben Slimane *et al.*, 2015a,b) que nous avons développée, permet de coder facilement des problèmes combinatoires génériques sous la forme de bases de clauses ou de bases de contraintes et d'utiliser des solveurs SAT ou SMT pour les résoudre. Cet environnement permet donc de coder la résolution de problèmes de planification classique ou temporelle. Pour faciliter et automatiser cette résolution, nous avons développé un module externe PDDL2TouISTL qui permet de générer à partir d'un problème de planification classique ou temporelle décrit dans le langage PDDL, des règles permettant de décrire le codage de ce problème dans le langage de notre plateforme.

1 Démonstration de TouIST pour la planification

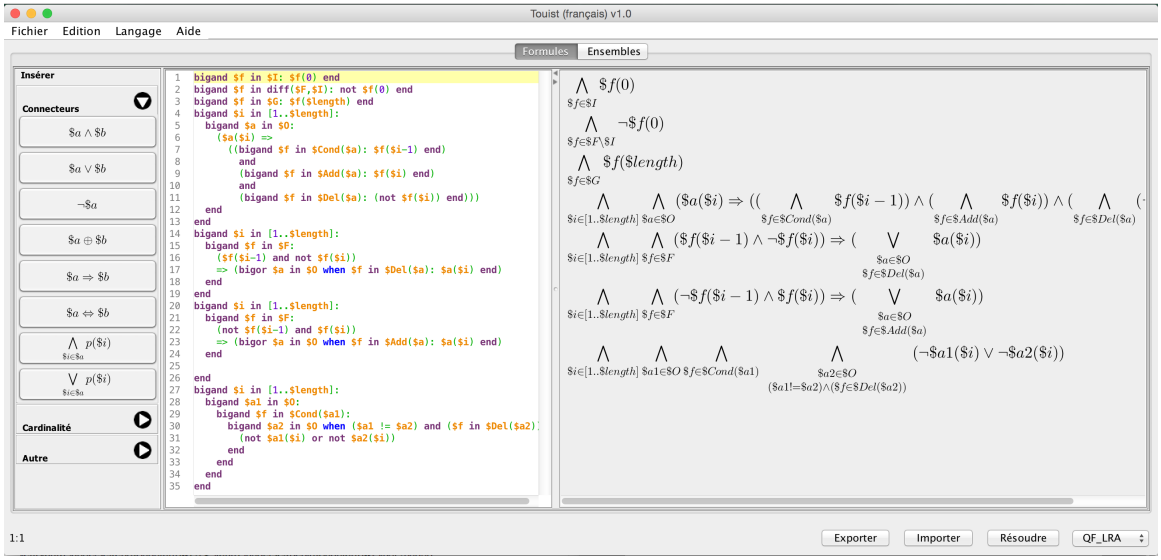


FIGURE 1 – Règles de codage (langage touistl à gauche et affichage LaTeX à droite).

La planification par satisfaction de base de clauses (planification SAT) a été introduite avec le planificateur SATPLAN (Kautz & Selman, 1992). Dans cette approche, on travaille directement sur un ensemble fini de variables propositionnelles. Deux actions identiques pouvant apparaître à des endroits différents d'un même plan doivent pouvoir être différenciées et on leur associe donc des propositions différentes. Comme on ne connaît pas à l'avance la longueur d'un plan solution d'un problème, on ne peut pas créer un codage unique permettant de le résoudre puisqu'il faudrait créer une infinité de variables propositionnelles pour représenter toutes les actions de tous les plans possibles. La solution la plus commune consiste alors à créer un codage représentant tous les plans d'une longueur k fixée. La base de clause ainsi obtenue est donnée en entrée à un

solveur SAT qui retourne, lorsqu'il existe, un modèle de cette base. Le décodage de ce modèle permet alors d'obtenir un plan-solution. Si la résolution du codage ne donne pas de modèle, la valeur de k est augmentée et le processus réitéré. Pour la complétude du procédé, tous ces modèles doivent correspondre exactement à tous les plans solutions d'une longueur fixée du problème. Notre plateforme de résolution de problèmes TouIST est capable de prendre en compte à la fois des formules logiques et des *ensembles de domaines*. Si notre objectif est de résoudre plusieurs problèmes de planification génériques, nous pouvons ainsi profiter de la flexibilité de TouIST qui permet à l'utilisateur de décrire une méthode générique de résolution avec des règles encodées comme formules et d'utiliser des ensembles de domaines pour décrire chaque problème de planification particulier. De nombreuses règles de codage pour la résolution de problèmes de planification ont déjà été proposées (Kautz & Selman, 1992; Mali & Kambhampati, 1999; Rintanen *et al.*, 2006). La figure 1 présente l'intégration de l'un de ces codages dans TouIST (codage dans les espaces d'états avec frame-axiomes explicatifs). La figure 2 illustre les ensembles de domaines définis par Pddl2TouistL pour un problème particulier du célèbre monde des cubes "BlockWorld" et la figure 3 donne la solution renvoyée par le solveur.

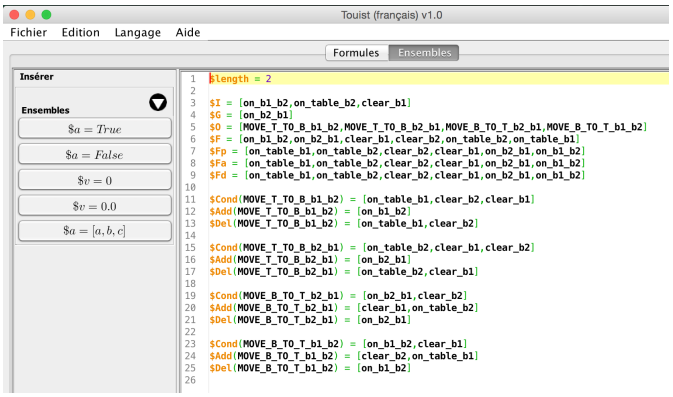


FIGURE 2 – Ensembles de domaines définis pour un problème de planification (BlocksWorld).

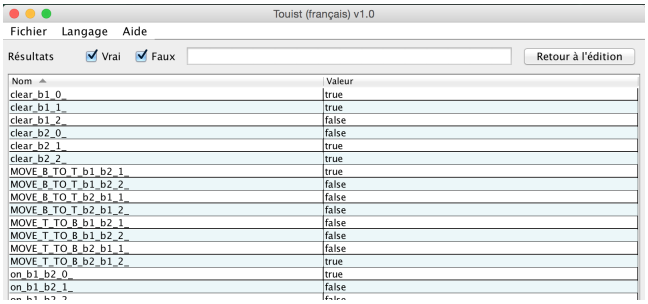


FIGURE 3 – Solution renvoyée par le solveur (BlocksWorld).

Pour résoudre des problèmes de planification réels, l'une des principales difficultés à surmonter consiste à prendre en compte la dimension temporelle. En effet, de nombreux problèmes du monde réel nécessitent, pour être résolus ou exécutés plus efficacement, la prise en compte de la durée des actions, des instants auxquels des évènements se produisent, ou encore la concurrence d'actions. En complément de SAT, notre plateforme TouIST est capable de gérer des théories comme la différence logique ou l'arithmétique linéaire sur les nombres entiers (QF_IDL, QF_LIA) ou les nombres réels (QF_RDL, QF_LRA), et d'appeler un solveur SMT pour trouver une solution. Pour être résolus, les problèmes de planification temporelle issus du monde réel nécessitent une représentation continue du temps, et donc, l'utilisation de nombres réels dans les codages logiques. TouIST peut être utilisé pour résoudre de tels problèmes impliquant des actions avec durée, des évènements exogènes et des buts temporellement étendus, par exemple avec les règles de codage proposées dans (Maris & Régnier, 2008).

Références

BEN SLIMANE S., COMTE A., GASQUET O., HEBA A., LEZAUD O., MARIS F. & VALAIS M. (2015a). La logique facile avec TouIST (formalisez et résolvez facilement des problèmes réel). In *Actes des 9es Journées d'Intelligence Artificielle Fondamentale (IAF'2015), 29 Juin-1er Juillet, 2015, Rennes, France*.

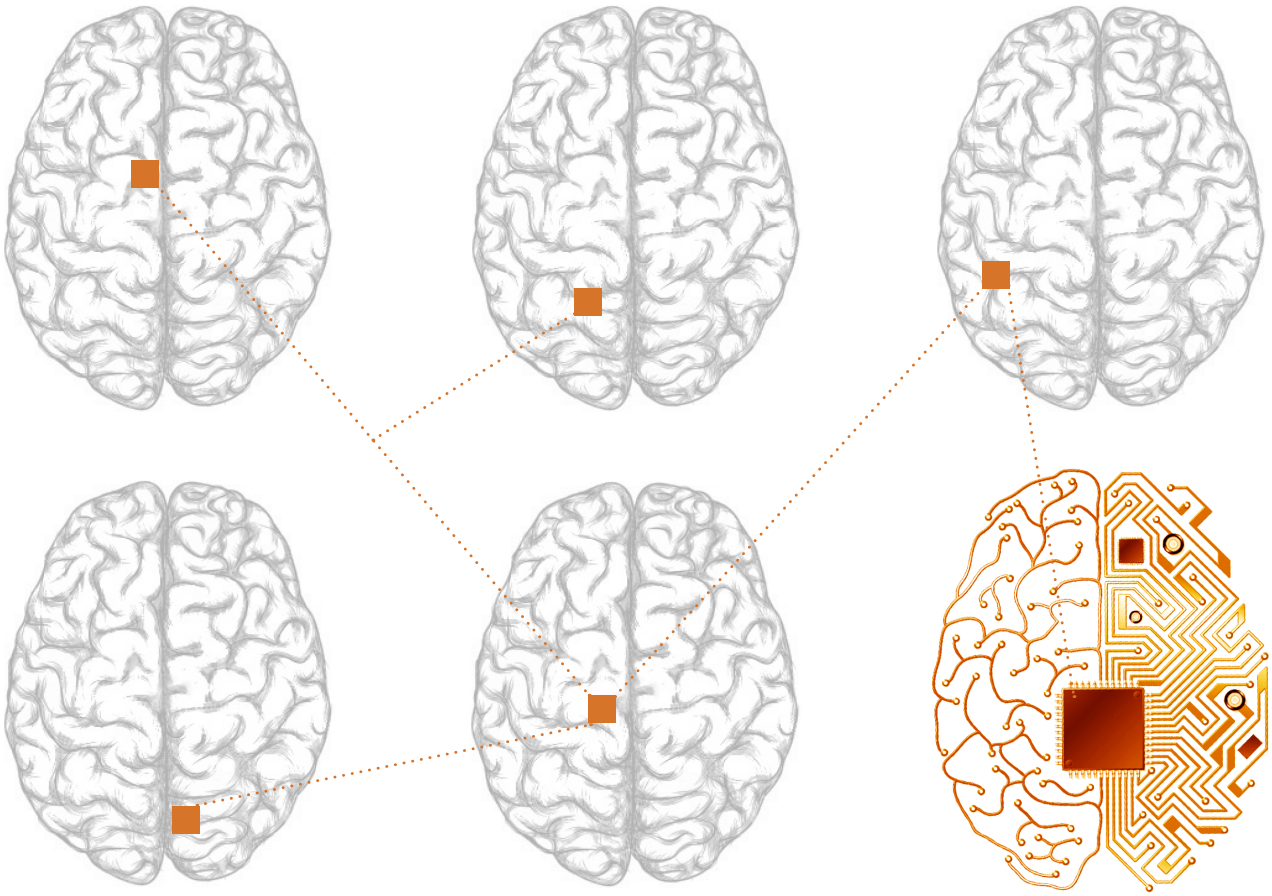
BEN SLIMANE S., COMTE A., GASQUET O., HEBA A., LEZAUD O., MARIS F. & VALAIS M. (2015b). Twist your Logic with TouIST. In *Proceedings of 4th International Conference on Tools for Teaching Logic (TTL'2015), June 9-12, 2015, Rennes, France*.

KAUTZ H. A. & SELMAN B. (1992). Planning as satisfiability. In *ECAI*, p. 359–363.

MALI A. D. & KAMBHAMPATI S. (1999). On the utility of plan-space (causal) encodings. In J. HENDLER & D. SUBRAMANIAN, Eds., *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, July 18-22, 1999, Orlando, Florida, USA.*, p. 557–563 : AAAI Press / The MIT Press.

MARIS F. & RÉGNIER P. (2008). TLP-GP : new results on temporally-expressive planning benchmarks. In *20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2008), November 3-5, 2008, Dayton, Ohio, USA, Volume 1*, p. 507–514 : IEEE Computer Society.

RINTANEN J., HELJANKO K. & NIEMELÄ I. (2006). Planning as satisfiability : parallel plans and algorithms for plan search. *Artif. Intell.*, **170**(12-13), 1031–1080.



PFIA 2015

http://pfia2015.inria.fr

Plate-forme Intelligence Artificielle

Rennes du 29 juin au 3 juillet 2015